Міністерство освіти і науки України
Чернівецький національний університет
імені Юрія Федьковича

Професійна практика програмної інженерії

Навчальний посібник

Чернівці
Чернівецький національний університет
2015

УДК 061
ББК 32.973-018п87.75
П 84

Рекомендовано до друку Міністерством освіти і науки України
(Гриф № 1/11-19279 від 08.12.2014 р.)

Рецензенти: ***Виклюк Я.І.***, доктор технічних наук, професор, в. о. ректора приватного вищого навчального закладу «Буковинський університет»;
Поворознюк А.І., доктор технічних наук, професор, професор кафедри «Обчислювальна техніка та програмування» Національного технічного університету «Харківський політехнічний інститут»;
Говорущенко Т.О., кандидат технічних наук, доцент, доцент кафедри системного програмування Хмельницького національного університету

П 84 **Професійна** практика програмної інженерії : навчальний посібник / укл. Жихаревич В.В. – Чернівці : Чернівецький національний університет, 2015. – 384 с.

У запропонованому виданні розглядаються питання професійної практики програмних інженерів. Висвітлюється: історія становлення програмної інженерії як професії, питання професійної етики програмних інженерів, професійні асоціації програмних інженерів та їх роль, освітній та культурний аспекти інженерії програмного забезпечення.

Для студентів вищих навчальних закладів, які навчаються за напрямком підготовки „Програмна інженерія”.

ББК 32.973-018п87.75
УДК 061

ЗМІСТ

ВСТУП

Ключовою метою будь-якої учбової програми в галузі інженерії є надання випускникам знань і початкового досвіду, необхідних для початку професійної інженерної діяльності. При цьому важливим керівним принципом є включення практичного досвіду професійної діяльності під час навчання студентів, що спеціалізуються з програмної інженерії.

Працедавці випускників, що спеціалізуються у сфері розробки програмного забезпечення, висувають певні вимоги до характеристики їх професійних навичок. Наприклад, щороку національна асоціація коледжів і працедавців США NACE (National Association of Colleges and Employers) проводить дослідження для визначення, які якості працедавці найбільше цінують у претендентах на працевлаштування [1]. У 2003 році працедавці оцінили важливість якостей і навичок кандидатів за п'ятибальною шкалою, де п'ятірка означала «надзвичайно важливо», а одиниця – «не важливо». Найбільш затребуваними характеристиками були вказані: навички комунікації (середній бал 4.7), чесність (4.7), навички роботи в команді (4.6), навички міжособистісних відносин (4.5), мотивація й ініціатива (4.5), строга етика роботи (4.5).

Проблема критичної залежності суспільства від якості і вартості програмного забезпечення в умовах відносної незрілості програмної інженерії робить питання професіоналізму ще важливішим для учбових планів з програмної інженерії, ніж для інших інженерних програм. Випускникам за фахом «Програмна інженерія» необхідно прийти на робочі місця підготовленими як до розв'язання реальних задач, так і до сприяння розвитку дисципліни програмної інженерії до професіональнішого і прийнятнішого рівня. Як й іншим фахівцям у галузі інженерії, розробникам програмного забезпечення необхідно скрізь, де це доречно і допустимо, уміти знаходити необхідну для ухвалення рішень кількісну інформацію, а також бути здатними ефективно функціонувати в умовах невизначеності й уникати невиправданих спрощень при моделюванні.

Програмна інженерія як професія має певні зобов'язання перед суспільством. Продукти, створені програмістами, впливають на життя і діяльність клієнтів та користувачів. Очевидно, що розробники програмного забезпечення повинні діяти етично і професійно. Преамбула до «Кодексу етичних норм професіонала в галузі програмної інженерії» [ACM (Association for Computing Machinery) 1998] формулює це так [2]:

Унаслідок специфіки своїх ролей у процесі створення програмних систем інженери з програмного забезпечення мають необмежені можливості приносити користь або заподіювати шкоду як самостійно, так і сприяючи іншим або впливаючи на інших. Інженери повинні взяти на себе зобов'язання зробити програмну інженерію корисною і поважаною професією, щоб бути упевненими в тому, що їх робота використовується на благо. Як наслідок даного зобов'язання інженери по програмному забезпеченню повинні строго дотримуватися «Кодексу етичних норм професіонала в галузі програмної інженерії».

Для сприяння в забезпеченні етично коректної і професійної поведінки викладачі програмної інженерії зобов'язані не тільки ознайомити студентів з «Кодексом», але і залучити їх до активного обговорення, що ілюструє і висвітлює вісім принципів «Кодексу», а також основні дилеми, з якими стикаються професійні інженери в типових робочих ситуаціях.

Особливості викладання програмної інженерії безпосередньо впливають на деякі чинники професійної діяльності (наприклад, на здатність працювати в команді, навички комунікації й аналітики), тоді як решта чинників (такі як строга робоча етика, упевненість у власних силах) є предметом тоншого впливу освіти на характер індивідуума, його особисті якості та зрілість.

В освітньо-методичній літературі існує ряд рекомендацій та ідей щодо методів включення матеріалу з професійної діяльності в учбовий план з програмної інженерії. Зокрема, проведено аналіз матеріалу, що безпосередньо стосується професійної діяльності (як-от технічні комунікації, етика, інженерна економіка тощо), а також ідеї моделювання робочих ситуацій (учбові приклади,

лабораторні роботи, колективно здійснювані учбові проекти тощо).

Існує велика кількість різних чинників, які істотно впливають на підготовленість студентів до професійної діяльності. Прикладами таких чинників є: участь у складанні учбового плану викладачів, що мають професійний досвід; досвід роботи студентів як практикантів; участь студентів у сумісних освітніх заходах; різноманітна позапрограмна діяльність, наприклад, відвідування семінарів, екскурсії на підприємства і участь у професійних студентських клубах і співтовариствах.

Студенти, які успішно прослухали курс „Професійна практика програмної інженерії", повинні вміти [1]:

- ухвалювати етичні рішення при зіткненні з етичними дилемами, посилаючись як на загальні етичні принципи, так і на етичні кодекси інженерії, комп'ютингу і програмної інженерії;
- приділяти увагу безпеці, захищеності і правам людини в інженерії та управлінні ухваленням рішень;
- знати основи історії інженерії, комп'ютингу і програмної інженерії;
- роз'яснювати і застосовувати законодавство, яке стосується програмної інженерії, включаючи законодавство з авторського права, патенти й інші аспекти інтелектуальної власності;
- описувати ефект, який зумовлюють рішення у сфері програмної інженерії на суспільство, економіку, соціальне середовище, їх замовників, керівництво, партнерів і на них самих;
- розуміти важливість різноманітних професійних співтовариств, визначальних для програмної інженерії на регіональному рівні, в країні, а також на міжнародній арені;
- розуміти роль стандартів і сукупностей знань, що визначають стандарти в галузі програмної інженерії;
- усвідомлювати потребу в постійному підвищенні своєї кваліфікації як інженера загалом, так й інженера з програмного забезпечення зокрема.

РОЗДІЛ 1. ІСТОРІЯ СТАНОВЛЕННЯ ТА ПЕРСПЕКТИВИ РОЗВИТКУ ПРОГРАМНОЇ ІНЖЕНЕРІЇ ЯК ПРОФЕСІЇ

Народження та еволюцію програмної інженерії як однієї з дисциплін комп'ютерної науки можна відстежити по розвитку і дозріванню погляду на програмування взагалі [3]. На зорі комп'ютерної епохи завдання програмування в основному розглядалося як послідовне вибудовування набору команд, що змушують комп'ютер зробити що-небудь корисне. Програмовані завдання були легко зрозумілі, наприклад, розв'язування диференційних рівнянь з фізики, які реалізовувалися самим фізиком, без посередника між ним і комп'ютером. Коли комп'ютери подешевшали і стали звичними, використовувати їх почали все більше і більше людей. У кінці 1950-х були винайдені мови високого рівня, які спрощували спілкування з машиною. Але, як і раніше, отримання віддачі від комп'ютера було справою фахівців, які писали програми для чітко визначеного завдання.

Саме тоді «програмування» почало набувати статусу професії: можна було долучити фахівця в галузі комп'ютерної техніки замість того, щоб писати програму самому. Такий підхід відокремив користувача від комп'ютера: користувач став формулювати завдання вже не у вигляді строгої комп'ютерної нотації, як було раніше. Далі фахівець читав і перекладав це завдання точною мовою машинних команд. При цьому поступово, з розвитком техніки, цей процес з монотонної роботи перетворився в інтелектуальну діяльність, порівнянну з мистецтвом. Виникла потреба в людях зі спеціальною підготовкою й особливими розумовими здібностями, яких називають програмістами. Окрім необхідних знань у галузі комп'ютерних наук та технологій програмування, професійні програмісти під час своєї роботи повинні активно застосовувати знання, які відносять до соціально-економічних дисциплін. При цьому професійне програмування почало набувати рис окремого інженерного напрямку діяльності [4].

1.1. Історія розвитку програмування

З глибокої давнини відомі спроби створити пристрої, які прискорюють та полегшують процеси обчислень [3]. Ще древні греки та римляни використовували пристрій, подібний до рахівниць, – абак. Такі пристрої були відомі й у країнах Стародавнього Сходу. У XVII ст. німецькі вчені В. Шиккард (1623 р.), Г. Лейбніц (1673 р.) та французький науковець Б. Паскаль (1642 р.) створили механічні обчислювальні пристрої – попередники відомого арифмометра. Обчислювальні машини вдосконалювались протягом кількох століть. Але при цьому не застосовувалося поняття «програма» та «програмування».

Лише на початку XIX ст. (1830 р.) англійський учений, професор математики Кембріджського університету Чарльз Беббідж, аналізуючи результати обробки перепису населення Франції, теоретично дослідив процес виконання обчислень та обґрунтував основи архітектури обчислювальної машини. Працюючи над проектом аналітичної машини – «Машини для обчислення різниць», Ч. Беббідж передбачив багато ідей і принципів організації та роботи сучасних ЕОМ, зокрема принцип програмного керування та програми, яка запам’ятовується. Загальна зацікавленість наукою дала Ч. Беббіджу та Аді Лавлейс (1815–1852 рр.) довгі роки плідної співпраці. У 1843 р. А. Лавлейс переклала статтю Менабреа за лекціями Ч. Беббіджа, де у вигляді детальних коментарів (за об’ємом вони перевищували основний текст) сформулювала головні принципи програмування аналітичної машини. Вона розробила першу програму (1843 р.) для машини Беббіджа, переконала його в необхідності використання у винаході двійкової системи числення замість десяткової, розробила принципи програмування, які передбачають повторення однієї і тієї ж послідовності команд при певних умовах. Саме вона запропонувала терміни «робоча комірка» та «цикл». А. Лавлейс склала перші програми для розв’язання системи двох рівнянь та обчислення чисел Бернуллі за досить складним алгоритмом та зробила припущення, що з часом аналітична машина буде

складати музичні твори, малювати картини та використовуватися в практичній і науковій діяльності. Час підтвердив її правоту та точність прогнозів. Своїми працями Ада Лавлейс заклала теоретичний фундамент програмування та по праву вважається першим у світі програмістом і засновником наукового програмування.

У 1854 р. англійський математик Джордж Буль опублікував книгу «Закони мислення», в якій описав алгебру висловлювань – Булеву алгебру. На її основі на початку 80-х років XIX ст. побудовано теорію релейно-контактних схем та конструювання складних дискретних автоматів. Алгебра логіки багатогранно вплинула на розвиток обчислювальної техніки, будучи інструментом розробки та аналізу складних схем, інструментом оптимізації великої кількості логічних елементів, з багатьох тисяч яких складаються сучасні ЕОМ.

Ідеї Ч. Беббіджа реалізував американський учений Г. Холлеріт, який за допомогою побудованої лічильно-аналітичної машини та перфокарт за три роки обробив результати перепису населення в США станом на 1890 р. У машині вперше було використано електрику. У 1896 р. Холлерітом було засновано фірму з випуску обчислювальних перфораційних машин і перфокарт.

У 1936 р. англійський математик Алан Тьюринг увів поняття машини Тьюринга як формального уточнення інтуїтивного поняття алгоритму. Вчений показав, що будь-який алгоритм у деякому сенсі може бути реалізовано на машині Тьюринга, і, як наслідок, доводив можливість побудови універсальної ЕОМ. Машина може містити початкові дані задачі, яка розв’язується, та програму її розв’язування. Машину Тьюринга можна вважати ідеалізованою моделлю універсальної ЕОМ.

У 40-х роках XX ст. механічна елементна база обчислювальних машин почала замінюватися електричними та електронними пристроями. Перші електромеханічні машини створені в Німеччині К. Цузе (Ц-3, 1941 р.) і в США під керівництвом професора Гарвардського університету Г. Айкена (Mark-1, 1944 р.). Перша електронна машина створена в США

групою інженерів під керівництвом професора Пенсільванського університету Дж. Мочлі та аспіранта Дж. Еккерта (ENIAC – Electronic Numerical Integrator and Calculator – електронний числовий інтегратор та калькулятор, 1946 р.). У 1949 р. в Англії було побудовано EDVAC (Electronic Discrete Variable Automatic Computer) – першу машину, яка володіла автоматичним програмним керуванням, внутрішнім запам'ятовуючим пристроєм та іншими необхідними компонентами сучасних ЕОМ.

Логічні схеми обчислювальних машин були розроблені в кінці 1940-х років Дж. фон Нейманом, Г. Гольдстайном та А. Берксом. Особливий внесок у цю справу зробив американський математик Джон фон Нейман, який брав участь у створенні ENIAC. Він запропонував ідею зберігання команд керування і даних у машинній пам'яті та сформулював основні принципи побудови сучасних ЕОМ. Машина із програмою, яка в ній зберігається, виявилася більш швидкодіючою та гнучкою, ніж створені раніше.

У 1951 р. у США було налагоджено перше серійне виробництво електронних машин UNIVAC (UNIVersal Automatic Computer – універсальна автоматична обчислювальна машина). В цей же час фірма IBM почала серійний випуск машини IBM/701.

У СРСР першими авторами ЕОМ, яку було створено в грудні 1948 р., є І. С. Брук і Б. І. Рамеєв. А перша радянська ЕОМ із програмою, яка в ній зберігається, створена у 1951 р. під керівництвом С. О. Лебедєва (МЭСМ – малая электронная счётная машина). У 1953 р. в СРСР розпочався серійний випуск машин, першими з яких були БЭСМ-1 та «Стрела».

З появою цифрових програмно-керованих машин з'явилася нова область прикладної математики – програмування. Як область науки та зародок нової професії вона виникла у 1950-х роках. На початку програми складались вручну на машинній мові (у машинних двійкових кодах). Програми були громіздкими, їх налагодження – дуже трудомістким. Для спрощення прийомів і методів складання та налагодження програм створили мнемокоди, що за структурою були близькими до машинної мови та використовували символьну адресацію. Спеціальні програми (assembler – асемблери) переводили програму, записану у

мнемокоді, на машинну мову. Асемблери, розширені макрокомандами, використовуються і сьогодні, особливо коли необхідно оптимізувати той чи інший фрагмент програмного коду. Далі були створені автокоди, які можна використовувати на різних машинах. Автокод – набір псевдокоманд для розв'язання спеціалізованих задач, наприклад наукових або інженерних. Для таких задач існує розвинута бібліотека стандартних програм.

До кінця 1950-х років основним елементом конструкції ЕОМ були електронні лампи (I покоління). У цей період розвиток ідеології та техніки програмування відбувався за рахунок досягнень американських вчених Дж. фон Неймана, який сформулював основні принципи побудови ЕОМ, та Дж. Бекуса, під керівництвом якого у 1954 р. створено Fortran (Formula Translation) – першу мову програмування високого рівня, яка в різних модифікаціях використовується до теперішнього часу. Так, у 1965 р. у Дартмутському коледжі Д. Кемені та Т. Куртцем було розроблено спрощену версію Фортрану – Basic. У 1966 р. комісія при Американській асоціації стандартів (ASA – American Standards Association) розробила два стандарти мови: Фортран та Базисний Фортран. Використовуються також подальші модифікації мови (наприклад, 1970-х та 1990-х років).

Досягнення в галузі електроніки та мікроелектроніки дозволили замінити елементну базу ЕОМ на більш досконалу. В кінці 1950-х років громіздкі електронні лампи замінюють напівпровідниковими мініатюрними транзисторами. З'являються ЕОМ II покоління; потім приблизно через 10 років – III покоління на інтегральних схемах; ще через 10 років – IV покоління на великих (ВІС) і надвеликих інтегральних схемах (НВІС). В Японії у 1990-х роках реалізовані перші проекти ЕОМ V покоління, в яких використані досягнення в галузі функціональної електроніки та нанотехнологій. Якщо об'єм оперативного запам'ятовуючого пристрою (ОЗП) однієї з найкращих машин у СРСР 1960-х років М-20, яку було створено під керівництвом С. О. Лебедєва у 1958 р., становив 4096 слів (8 Кбайт) та швидкодію 20 тисяч операцій на секунду, то сучасні персональні комп'ютери характеризуються ОЗП, який має об'єм десятки Гбайт та швидкодію порядку сотні

мільйонів операцій на секунду, що дозволяє розв'язувати найскладніші задачі.

Паралельно з розвитком апаратного забезпечення ЕОМ активно розвивалися нові погляди на організацію та розробку програмного забезпечення. Так, у 1953 р. А.А. Ляпуновим було запропоновано операторний метод програмування, який полягав у автоматизації програмування, а алгоритм рішення задачі представлявся у вигляді сукупності операторів, які утворюють логічну схему задачі. Схеми дозволяли розділити громіздкий процес складання програми, частини якої складались за формальними правилами, а потім об'єднувалися в єдине ціле. Для перевірки ідей операторного методу в СРСР у 1954 р. була розроблена перша програмуюча програма ПП-1, яка протягом наступних п'яти років постійно удосконалювалась.

У США в 1954 р. почав застосовуватись алгебраїчний підхід, який по суті збігався з операторним методом. У 1956 р. корпорацією IBM розроблена універсальна ПП Фортран для автоматичного програмування на ЕОМ IBM/704.

У цей період з накопиченням досвіду та теоретичним осмисленням удосконалювалися мови програмування. Протягом 1958–1960 років в Європі було створено Algol, який породив цілу серію алголоподібних мов, серед яких слід виділити: Pascal (Н. Вірт, 1970 р.), С (Д. Рітчі та Б. Керніган, 1972 р.), Ada (Ж. Ішбіа, 1979 р.). У 1961–1962 роках Дж. Маккарті в Массачусетському технологічному інституті було створено мову функціонального програмування Lisp, яка відкрила у програмуванні один з альтернативних напрямків, запропонованих Дж. фон Нейманом.

На початок 1970-х років існувало більше 700 мов високого рівня та близько 300 трансляторів для автоматизації програмування.

Ускладнення структури ЕОМ призвело (у 1953 р. для машин II покоління) до створення операційних систем (ОС) – спеціальних керуючих програм для організації та розв'язання задач на ЕОМ. Наприклад, моніторна система, створена у Массачусетському технологічному інституті, забезпечувала пакетну обробку, тобто

неперервне, послідовне проходження через ЕОМ багатьох груп (пакетів) завдань і використання бібліотек службових програм, які зберігаються в машині. Це дозволило сумістити операції запуску та виконання програм.

Для персональних ЕОМ на теперішній час розроблено велику кількість ОС, наприклад: MS DOS, Windows, OS/2, MacOS, Unix, Linux та багато інших. Найбільш широке розповсюдження отримала ОС Windows, яка має розвинутий графічний інтерфейс та широкий набір додатків, які дозволяють обробляти різноманітну інформацію з багатьох сфер людської діяльності.

Стосовно розвитку технологій програмування, то цей процес розпочався із 1965 р., коли італійські вчені Бом і Джакопіні запропонували використовувати як базові алгоритмічні елементи прямування, розгалуження та цикл. Майже в той самий час до аналогічних висновків прийшов голландський учений Е. Дейкстра, який заклав основи структурного програмування. У 1970-х роках ця методологія остаточно сформувалася і корпорація IBM повідомила про застосування в розробці програмного забезпечення «Удосконалених методів програмування», одним із компонентів яких була технологія низхідного структурного програмування, основу якого складає таке:

- складна задача розбивається на прості, функціонально керовані задачі, кожна задача має один вхід та один вихід; керуючий потік програми складається із сукупності елементарних функціональних підзадач;
- керуючі структури прості, тобто логічна задача повинна складатися з мінімальної, функціонально повної сукупності достатньо простих керуючих структур;
- програма розробляється поетапно, на кожному етапі розв’язується обмежена кількість точно заданих задач.

Чітко сформульовані основи низхідної розробки, структурного кодування та наскрізного контролю дозволяли перейти до промислових методів розробки програмного забезпечення.

Подальший розвиток отримала технологія модульного програмування, основа якого полягає ось у чому:

- функціональна декомпозиція (розбиття) задачі на самостійні підзадачі – модулі, які пов'язані лише вхідними та вихідними даними;

- модуль являє собою «чорну скриньку», яка дозволяє розробляти частини програм одного проекту на різних мовах програмування, а потім за допомогою компонувальних засобів об'єднувати їх в єдиний завантажувальний модуль;

- повинно бути ясне розуміння призначення всіх модулів задачі та їх оптимального сполучення;

- за допомогою коментарів повинно бути описане призначення всіх змінних модуля.

У період 1970–1980 років розвиток теоретичних досліджень оформило програмування як самостійну наукову дисципліну, яка займалася методами розробки програмного забезпечення.

В історії розвитку промислового програмування велику роль відіграв програміст та бізнесмен Білл Гейтс. Його історія дуже повчальна для програмістів-початківців. У 1972 р. Білл Гейтс та його шкільний товариш Пол Ален заснували компанію, яка займалася аналізом вуличного руху «Traf-O-Data» та використовувала для обробки даних комп'ютери з мікропроцесором 8008 – першим із відомого ряду мікропроцесорів компанії «Intel». Будучи студентом Гарвардського університету, у 1975 р. він сумісно з Аленом написав для комп'ютера Altair (фірми MITS) інтерпретатор – програму-перекладач з мови програмування на мову машинних кодів. Вони уклали з керівництвом фірми договір, згідно з яким їх програми розповсюджувалися разом із комп'ютерами. Товариші заснували компанію «Microsoft», в якій Б. Гейтсу належало 60 % акцій, П. Алену – 40 %. У 1976 р. Б. Гейтс увів у практику продаж ліцензій на свої програмні продукти безпосередньо виробникам комп'ютерів, що дозволило «вбудовувати» їх у комп'ютери. Це було велике досягнення в галузі маркетингу, яке принесло фірмі величезні прибутки. Фірма залучила таких нових замовників, як фірми «Apple», «Commodore», «Tendi». У 1980 р. фірма IBM запропонувала «Microsoft», в якій тоді працювало біля двох десятків чоловік, створити мови програмування для її нового

персонального комп'ютера, у подальшому відомого як IBM PC. У 1981 р. «Microsoft» придбала в розробника Т. Патерсона дискову ОС (DOS), і у серпні цього ж року IBM PC постачалася разом з операційною системою MS DOS. Успіх був настільки великий, що, крім величезних прибутків, привів до того, що і архітектура мікропроцесорів Intel, і комп'ютери IBM, і програми «Microsoft» фактично стали галузевими стандартами. У 1988 р. «Microsoft» створила свою ОС Windows із потужним графічним інтерфейсом, яка вдосконалюється кожного року й до теперішнього часу. У 1995 р. операційні системи, які випускалися фірмою, використовували 85 % персональних комп'ютерів, штат співробітників налічував 18 тисяч чоловік, річний випуск досяг 200 програмних продуктів, а прибуток склав мільярди доларів. У 1998 р. Б. Гейтс став найбагатшою людиною у світі, а наприкінці 1999 р. оголосив про своє рішення піти з посту голови компанії та зайнятися виключно програмуванням. Сьогодні Білл Гейтс – одна із найпопулярніших фігур комп'ютерного світу. Журнал «People» писав: «Гейтс у галузі програмного забезпечення відіграв таку ж роль, як Едісон у галузі електроосвітлення: частково новатор, частково підприємець, частково торговець, але, без сумніву, геній».

На початку 1990-х років професійне програмування вийшло на рівень технологій. При цьому методи розробки програмного забезпечення почали синтезувати в собі різноманітні аспекти:

- методи інженерних розрахунків для оцінки витрат та вибору рішень;
- математичні методи для складання алгоритмів;
- методи керування для визначення вимог до системи, організації робіт і прогнозування.

На зміну структурному програмуванню на початку 1990-х років прийшло об'єктно-орієнтоване програмування (ООП). Його можна розглядати як модульне програмування нового рівня, коли замість переважно випадкового, механічного об'єднання процедур і даних головним постає їх змістовий зв'язок. Об'єкт розглядається як логічна одиниця, яка містить дані і правила (методи) їх обробки. Об'єктно-орієнтована мова створює

«програмне оточення» у вигляді множини незалежних об'єктів, кожен з яких відрізняється своїми властивостями та способами взаємодії з іншими об'єктами. Програміст задає сукупність операцій, описуючи структуру обміну повідомленнями між об'єктами. Як правило, він «не заглядає» всередину об'єктів, але при необхідності може змінювати елементи всередині об'єктів або формувати нові.

ООП засноване на трьох найважливіших принципах (інкапсуляція, наслідування, поліморфізм), які надають об'єктам нові властивості. Інкапсуляція – об'єднання в єдине ціле даних та алгоритмів їх обробки. Тут дані – поля об'єкта, а алгоритми – об'єктні методи. Наслідування – властивість об'єктів породжувати своїх нащадків. Об'єкт-нащадок автоматично наслідує всі поля і методи, може доповнювати об'єкти новими полями, замінювати та доповнювати методи. Поліморфізм – властивість споріднених об'єктів розв'язувати схожі за змістом проблеми в різний спосіб.

Ідея використання програмних об'єктів досліджувалася протягом десятка років різними вченими. Однією з перших мов цього типу вважають Simula-67. А у 1972 р. з'явилась мова Smaltalk, розроблена Аланом Кеєм, яка затвердила статус об'єктно-орієнтованої мови.

З 1990-х років почали розвиватися інструментальні середовища та системи візуального програмування для створення програм на мовах високого рівня: Turbo Pascal, Delphi, Visual Basic, C++ Builder тощо.

Розвиток основних принципів об'єктно-орієнтованого програмування отримало з появою компонентного програмування (КП). КП – динамічний процес без жорстких правил, який виконується в основному для розподіленої розробки та програмування розподілених систем. Суть КП полягає в тому, що незалежні проектувальники-програмісти розробляють незалежні компоненти (окремі частини) єдиної системи, розподілені по множині вузлів великої мережі. Ці частини можуть належати різним власникам та керуватися організаційно незалежними адміністраторами.

У компонентному програмуванні компонент розглядається як сховище (у вигляді DLL- або EXE-файлів) для одного або кількох класів. Класи розповсюджуються в бінарному вигляді, а не у вигляді початкового коду. Надання доступу до методів класу здійснюється через строго визначені інтерфейси за протоколом. Це знімає проблему несумісності компіляторів, забезпечуючи без перекомпіляції зміну версій класів у різних додатках. Інтерфейси задають зміст сервісу та виступають посередником між клієнтом і сервером.

Фірма Microsoft створила технології для розподіленої розробки розподілених систем, такі як COM (Component Object Model), COM+, .NET. Розроблені й інші технології: CORBA (консорціуму OMG), JAVA (компанії Sun Microsystem) та інші.

Незалежність мов високого рівня від ЕОМ залучила до сфери алгоритмізації задач спеціалістів різноманітних галузей знань, дозволила використовувати численні стандартні типові програми, а програмістам – усувати дублювання при написанні програм для різноманітних типів ЕОМ і значно підвищити продуктивність праці.

Ідея перекласти на ЕОМ функції укладачів алгоритмів та програмістів дала нові можливості розвитку сфери штучного інтелекту, яка повинна була створювати методи автоматичного розв'язання інтелектуальних задач. Формалізація знань з різних галузей, наявних у професіоналів, накопичення їх у базах знань, реалізованих на ЕОМ, стали основою для створення експертних систем, зокрема різноманітних інтелектуальних роботів. Ці системи можуть не лише знайти розв'язок тієї чи іншої задачі, але й пояснити, як він отриманий. З'явилась можливість маніпулювати знаннями, мати знання про знання (метазнання). Знання, які зберігаються в системі, стали об'єктом її власних досліджень. Разом з тим, незважаючи на значні успіхи у сфері створення експертних систем, автоматизація процесу програмування – справа далекої перспективи, і поки що без копіткої праці професійних програмістів не обійтись.

На сучасному етапі програмування включає комплекс питань, пов'язаних з написанням специфікацій (умов задач),

проектуванням, кодуванням, тестуванням і функціонуванням програм для ЕОМ. Сучасне програмне забезпечення для ЕОМ має складну структуру і містить, як правило, операційну систему, транслятори з різноманітних мов, тестові програми контролю та діагностики, а також набір обслуговуючих (сервісних) програм. Наприклад, деякі вчені для проектування систем програмного забезпечення розробляють ідею «кільцевої структури» шести рівнів: 1-й (внутрішній) – програми для апаратури; 2-й – ядро ОС; 3-й – програми спряження; 4-й – частина ОС, орієнтована на користувача; 5-й – системи програмування; 6-й (зовнішній) – програми користувача. Згідно із цими проектами наукових досліджень, планується спростити процес створення програмних засобів шляхом автоматизації синтезу за специфікаціями початкових вимог на природних та інтуїтивно зрозумілих мовах.

Широке застосування структурних та об'єктно-орієнтованих методів програмування з використанням графічних моделей, які сприяють кращому розумінню структури та функціонуванню створюваного програмного забезпечення, гальмувалося відсутністю відповідних інструментальних засобів. Це породило потребу в розробці програмно-технологічних засобів спеціального класу – CASE (Computer Aided Software Engineering), які реалізують технологію створення та супроводу програмного забезпечення для різноманітних систем. Передумови для появи CASE-технологій виникли наприкінці 1980-х років. Початково термін «CASE» застосовувався лише до питань автоматизації розробки програмного забезпечення, але тепер програмна інженерія має більш широке значення для розробки систем у цілому. До CASE-технологій, які використовуються у процесі розробки програмного забезпечення, відносять засоби розробки та впровадження мов високого рівня, методів структурного та модульного програмування, мов проектування та засобів їх підтримки, формальних і неформальних мов опису системних вимог та багато інших аспектів програмної інженерії.

1.2. Умови виникнення інженерії програмного забезпечення

Програмна інженерія (промислове програмування) зазвичай асоціюється з розробкою великих і складних програм колективами розробників [3 – 15]. Становлення й розвиток цієї сфери діяльності було викликано низкою проблем, пов'язаних з високою вартістю програмного забезпечення, складністю його створення, необхідністю управління і прогнозування процесів розробки.

Великі програми спричинили великі проблеми, пов'язані з їх створенням і використанням. Зі збільшенням продуктивності, кількості обчислювальних машин і розширенням сфери їх застосування з'явилися програми двох типів. Перші створювалися і продавалися разом з машинами (транслятори, операційні системи, бібліотеки підпрограм). Програми другого типу виготовлялися на замовлення і призначалися для виконання завдань з різних галузей науки і техніки. Таким чином, з'явився замовник – організація, яка ставила завдання, призначала терміни, виділяла бюджет і оплачувала роботу. У зв'язку з цим дуже швидко з'явилися дві типових проблеми – необхідність документованого супроводу програми й усунення непорозумінь між розробником і замовником.

У 1960-х роках, унаслідок розширення розповсюдження застосувань комп'ютерів, зросла роль і затвердилося розуміння важливості програмного забезпечення. Цьому сприяло те, що почала з'являтися значна кількість проектів програмного забезпечення, які характеризувалися:

- наявністю замовника або ринкової ніші;
- великими розмірами й витратами;
- жорсткими вимогами до процесів реалізації і результатів.

Контекст, в якому розроблялося й використовувалося програмне забезпечення, сприяв особливому положенню програмного забезпечення в обчислювальній техніці й характеризувався таким:

- почало розроблятися дуже велике програмне забезпечення, характерним представником якого була тоді операційна система OS360 для ЕОМ серії IBM. Це програмне забезпечення включало

більше 500 тис. операторів і розроблялося значним для того часу колективом розробників (близько 1000). Досвід цього проекту був узагальнений і став відомим;

- програмне забезпечення розв'язувало настільки серйозні задачі, що виникла проблема з його супроводом, суть якого зводилася не тільки до виправлення помилок, допущених при розробці, але більшою мірою до модифікації програмного забезпечення у зв'язку зі зміною вимог замовника або середовища, в якому експлуатувалося програмне забезпечення, або внаслідок бажання розробника продовжити експлуатацію шляхом випуску вдосконалених версій;

- часті зриви термінів розробки й перевищення бюджету вимагали не тільки нового підходу до організації процесу розробки, але і нових методів і засобів, що забезпечують обґрунтований розрахунок параметрів проекту, які характеризували фінансування, терміни, обсяги програмного забезпечення, кількісний і якісний склад колективу розробників. Існуюча і широко використовувана одиниця вимірювання «людино-місяць» на таких масштабних проектах не працювала;

- досвід розробки програмного забезпечення, який був накопичений до цього періоду часу, показував, що дедалі рідше розроблялися принципово нові проекти. Тільки 15 % зі всіх проектів вимагали розробки «з нуля». Інші 85 % можна було віднести до повторюваних проектів. Тому актуальним стає використання досвіду, накопиченого в програмному забезпеченні. При цьому поступово стало зрозуміло, що повинно вирішуватися питання використання досвіду не тільки безпосередньо програмування, у вигляді частин програм, але і досвіду результатів виконання інших процесів, наприклад проектування. Для реалізації цього завдання у 1984 р. були широко розгорнені роботи з дослідження програмного забезпечення в аспекті повторного використання (reuse);

- техніка програмування і процеси, які були ефективні в 1950-х і ранніх 1960-х роках («програмування в малому») для розробки невеликих програм малими колективами, стали неефективними при розробці великого, складного програмного

забезпечення, що складається з мільйонів рядків коду, що вимагає кількох років роботи сотень фахівців різних спеціальностей. Була потрібна нова техніка, придатна для «програмування у великому». Почали з'являтися нові процеси, які вимагали певної організації (табл. 1.1). Ті, хто працював з великими проектами, швидко зрозуміли, що виробництво цих програмних систем значно відрізняється від розробки малих. Виникли принципові труднощі масштабування при спробі привнести прийоми написання малих програм у проектування великих. Отже, склалася ситуація, яка призвела до поняття кризи в програмному забезпеченні та необхідності пошуку шляхів виходу з неї, так званої срібної кулі [4].

Таблиця 1.1

ХАРАКТЕРИСТИКА ОСОБЛИВОСТЕЙ РОЗРОБКИ В ЧАСИ «КРИЗИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

Характеристика	1960±5 років програмування «any-wish-way»	1970±5 років «програмування в малому»	1980±5 років «програмування у великому
Об'єкти	Маленькі програми	Алгоритми і програми	Системна структура
Дані	Неструктурована інформація	Структури даних і типи	Бази даних
Інструменти і методи розробки	Асемблери	Компілятори, редактори, завантажувачі	Середовища, інтегровані інструменти, документи
Організаційне управління	Немає	Індивідуальні зусилля	Колективні зусилля, супровід

Виявилось, що проблеми в побудові великих комп'ютерних систем полягають зовсім не в тому, щоб зібрати разом машинні команди. Швидше, самі розв'язувані проблеми були недостатньо зрозумілі ані всім розробникам разом, ані кожному окремо. Розробники проекту змушені були витрачати силу-силенну часу на спілкування один з одним, замість того, щоб писати код. Іноді люди залишали проект, і це впливало не тільки на виконувану безпосередньо ними роботу, але і на роботу тих, хто від них залежав. Заміна розробника вимагала серйозної підготовки для

введення його в суть технічних умов проекту і розробки системи. Здавалося, що будь-яка зміна первинних вимог до системи впливає на велику кількість складових частин проекту, спричиняючи надалі затримку постачання готового продукту. На початках програмування проблем такого роду просто не існувало, а зараз вони з'явилися і вимагали нового підходу.

Було запропоновано та випробувано багато рішень. Одні розробники вважали, що вихід із ситуації, яка склалася, полягає в поліпшенні інструментів менеджменту. Інші пропонували інакше організувати команду розробників. Треті виступали за вдосконалення мов програмування і засобів розробки. Багато хто закликав до введення внутрішніх стандартів на зразок уніфікованих угод кодування. Дехто пропонував математичні і формалізовані підходи. Нестачі ідей не було. Остаточна думка була одностайною: програмне забезпечення потрібно будувати так само, як інженери будують інші великі складні системи – мости, нафтопереробні заводи, фабрики, кораблі і літаки. Питання було в тому, щоб розглядати кінцеву програмну систему як комплексний продукт, а його розробку – як інженерну роботу. Інженерний підхід вимагав управління, організації, інструментарію, теорій, методологій і технічних прийомів. Так народилася програмна інженерія.

У класичній статті 1987 р. Д. Брукс [9], перефразовуючи Арістотеля, стверджував, що в розробці програмного забезпечення є два роди проблем – найважливіші та другорядні. Другорядні стосуються поточного інструментарію і технологій, наприклад, проблема синтаксису, викликана використовуваною мовою програмування. Подібні перешкоди можна подолати шляхом застосування кращих засобів розробки і технологій. І навпаки, при розв'язанні першорядних задач новий інструментарій, швидше за все, не допоможе. Завдання комплексного проектування, скажімо, створення і представлення моделі прогнозування погоди або економічного розвитку, вимагають інтелектуальних зусиль, творчості та часових витрат. Брукс довів, що немає потреби у винайдені універсальних засобів

для виконання першорядних завдань, що стоять перед програмними інженерами.

Доказ Брукса розвінчує деякі помилкові припущення, пов'язані з терміном «програмна криза», уведеним через те, що програмні проекти весь час виходили за рамки бюджету й порушували терміни. Ця проблема здавалася тимчасовою, а її розв'язок вбачався в застосуванні кращих засобів розробки або методів управління. Насправді, проектанти порушували терміни через те, що виконуване завдання було складним і погано вивченим як замовниками, так і розробниками, і ні в кого не було анінайменшого уявлення про те, як оцінити складність завдання і розрахувати терміни його реалізації. Термін «програмна криза» і дотепер ще іноді використовується, проте існує загальна думка про те, що характерні труднощі розробки програмного забезпечення не мають швидких короткострокових рішень. Це стосується і нових складних галузей програмних застосувань, що відрізняються труднощами першого роду.

Інша тенденція розвитку зародилася всередині самої галузі та була заснована на посиленні погляду на розробку програм як на більш ніж просте "кодування". Натомість програмне забезпечення розглядалося як деякий об'єкт, що має повний життєвий цикл, який починається з появи концепції та проходить стадії проектування, розробки, введення в дію, супроводу і розвитку. Зміщення фокусу уваги з кодування породив розробку методологій та розвиненого інструментарію, що озброїв команди інженерів, зайнятих на всіх етапах життєвого циклу програмного забезпечення.

Наведений екскурс в історію підкреслює розвиток програмної інженерії, що виростає з програмування. Одначе були й інші технологічні віяння, що зіграли помітну роль у становленні галузі. Найбільш сильним виявився вплив змін цінового балансу між апаратним і програмним забезпеченням комп'ютерів. Якщо раніше вартість комп'ютеризованої системи зазвичай визначалася вартістю апаратури, а витрати на програми вважалися маловажливими, то тепер, навпаки, ціна програмної складової стає більш ніж домінуючим чинником у вартості

системи. Зниження цін на устаткування і зростання вартості програм змістили ціновий баланс у бік програмного забезпечення, надаючи дедалі більш економічного значення важливості програмної інженерії.

Можна простежити зв'язок між зростанням вартості програмного забезпечення та появою нових технологій програмування, зокрема модульного, структурного та об'єктно-орієнтованого програмування [10, 11].

На перших етапах становлення програмної інженерії (навіть коли вона так ще не називалася) було відмічено, що висока вартість програм пов'язана з розробкою однакових (або схожих) фрагментів коду в різних програмах. Викликано це було тим, що в різних програмах як частини цих програм виконувалися однакові (або схожі) завдання: розв'язання нелінійних рівнянь, розрахунок заробітної плати тощо. Використання при створенні нових програм раніше написаних фрагментів обіцяло істотне зниження термінів і вартості розробки. Це спричинило розвиток модульного програмування. Головний принцип модульного програмування полягав у виділенні спільних фрагментів програм та оформленні їх у вигляді модулів. Кожен модуль забезпечувався описом, в якому встановлювалися правила його використання – інтерфейс модуля. Інтерфейс задавав зв'язки модуля з основною програмою – зв'язки за даними і зв'язки по управлінню. При цьому можливість повторного використання модулів визначалася кількістю і складністю цих зв'язків, або наскільки ці зв'язки вдалося погоджувати з організацією даних і управлінням основної програми. Найбільш простими в цьому відношенні виявилися модулі розв'язання математичних задач: розв'язування рівнянь, виконання завдань оптимізації. До теперішнього часу накопичені та успішно використовуються великі бібліотеки таких модулів.

Для багатьох інших типів модулів можливість їх повторного використання виявилася проблематичною з причини складності їх зв'язків з основною програмою. Наприклад, модуль розрахунку зарплати, написаний для однієї фірми, може не підійти для іншої, оскільки зарплата в цих фірмах розраховується не в усьому

однаково. Повторне використання модулів зі складними інтерфейсами достатньо актуальне й нині. З цією метою розробляються спеціальні форми (структури) зображення модулів і організації їх інтерфейсів.

Наступний етап зростання вартості програмного забезпечення був пов'язаний з переходом від розробки простих програм до розробки складних програмних комплексів. До таких складних програм належать: системи управління космічними об'єктами, управління оборонним комплексом, автоматизації великої фінансової установи тощо. Складність таких комплексів оцінювалася такими показниками:

1. Великий об'єм коду (мільйони рядків).
2. Велика кількість зв'язків між елементами коду.
3. Велика кількість розробників (сотні чоловік).
4. Велика кількість користувачів (сотні та тисячі).
5. Тривалий час використання.

Для таких складних програм виявилось, що основна частина їх вартості визначається не процесом створення програм, а їх впровадженням та експлуатацією. За аналогією з промисловою технологією стали говорити про життєвий цикл програмного продукту як про послідовність певних етапів: етапу проектування, розробки, тестування, впровадження й супроводу.

Етап супроводу програмного комплексу включав дії з виправлення помилок у роботі програми і внесення змін відповідно до нових вимог користувачів. Основна причина високої вартості (а деколи і неможливості виконання) етапу супроводу полягала в тому, що програми були погано спроектовані – документація була незрозуміла й не відповідала програмному коду, а сам програмний код був дуже складний і заплутаний. Потрібна була нова технологія, яка забезпечить «правильне» проектування і кодування. Це спричинило розвиток структурного проектування і програмування. Основні принципи цієї технології полягали ось у чому:

1. Низхідне функціональне проектування, при якому в системі виділяються основні функціональні підсистеми, які потім можуть розбиватися на інші підсистеми.

2. Застосування спеціальних мов проектування і засобів автоматизації використання цих мов.

3. Порядок проектування і розробки: планування і документування проекту, підтримка відповідності коду проектній документації.

4. Структурне кодування без goto.

Наступна проблема зростання вартості програм була викликана тим, що зміна вимог до програми стали виникати не тільки на стадії супроводу, але і на стадії проектування – проблема замовника, який не знає, чого він хоче. Створення програмного продукту перетворилося на його перманентне перепроектування. Виникло питання, як проектувати і писати програми, щоб забезпечити можливість внесення змін у програму, не змінюючи раніше написаного коду.

Розв'язком цієї проблеми стало використання підходу або методу, який почали називати об'єктно-орієнтованим проектуванням і програмуванням. Суть підходу полягає в тому, що вводиться поняття класу як розвиток поняття модуля з певними властивостями і поведінкою, тобто характерними обов'язками класу. Кожен клас може породжувати об'єкти – екземпляри даного класу. При цьому працюють основні принципи (парадигми) об'єктно-орієнтованого програмування:

1. Інкапсуляція – об'єднання в класі даних (властивостей) і методів (процедур їх обробки).

2. Наслідування – можливість виведення нового класу зі старого з частковою зміною властивостей і методів.

3. Поліморфізм – визначення властивостей і методів об'єкта за його контекстом.

Проілюструвати можливості принципів об'єктно-орієнтованого програмування можна на такому прикладі (рис. 1.1). У організації, що складається з трьох відділів, треба нараховувати заробітну плату. У програмі кожен відділ представлений своїм модулем – об'єктом, а нарахування зарплати – об'єктом «Зарплата». При необхідності розрахунку зарплати об'єкта «Відділ» передається екземпляр об'єкта «Зарплата». Об'єкт «Відділ» передає об'єкту «Зарплата»

необхідні дані, а потім, за допомогою методів об'єкта «Зарплата», виконує необхідні розрахунки.

Нехай у відділі 3 частково змінилися правила нарахування зарплати. У цій ситуації при об'єктно-орієнтованому підході з класу «Зарплата» виводиться клас «Зарплата 1», який успадковує правила нарахування зарплати, що не змінилися, і перевизначає ті, що змінилися. Тут при розрахунку зарплати об'єктам «Відділ 1» і «Відділ 2» передаватиметься об'єкт «Зарплата», а об'єкту «Відділ 3» – об'єкт «Зарплата 1». При таких змінах:

1. Спрацьовує принцип наслідування: код «Зарплата», «Відділ 1» і «Відділ 2» залишаються без зміни, а код «Зарплата 1» змінюється рівно настільки, наскільки це необхідно.
2. Спрацьовує принцип поліморфізму: код «Відділ 3» також не змінюється – він продовжує вважати, що працює з об'єктом «Зарплата».

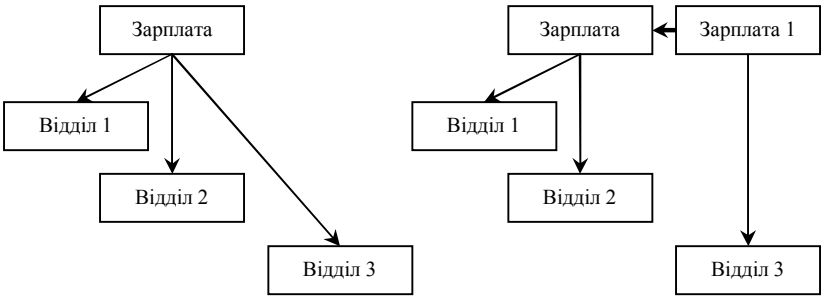

Рис. 1.1. Ілюстрація принципів об'єктно-орієнтованого програмування

Таким чином, програмна інженерія (або технологія програмування) як деякий напрям виникла та формувалася під тиском зростання вартості створюваного програмного забезпечення. Головна мета цієї галузі знань – скорочення вартості та термінів розробки, а також підвищення якості й надійності програмних продуктів.

Програмна інженерія пройшла кілька етапів розвитку, в процесі яких були сформульовані фундаментальні принципи і методи розробки програмних продуктів. Основний принцип програмної інженерії полягає в тому, що програми створюються в

результаті виконання кількох взаємозв'язаних етапів (аналіз вимог, проектування, розробка, впровадження, супровід), що становлять життєвий цикл програмного продукту. Фундаментальними методами проектування і розробки є модульне, структурне і об'єктно-орієнтоване проектування і програмування.

Незважаючи на те, що програмна інженерія досягла певних успіхів, перманентна криза програмування продовжується [10]. Пов'язано це з тим, що рубіж 1980–1990 років характеризується як початок інформаційно-технологічної революції, викликаної вибуховим зростанням використання інформаційних засобів: персональних комп'ютерів, локальних та глобальних обчислювальних мереж, мобільного зв'язку, електронної пошти, мережі Internet тощо.

Ціна успіху – криза програмування набуває хронічних форм:

- у США витрачається щорічно більше 200 млрд. доларів на більш ніж 170 тисяч проектів розробки програмного забезпечення у сфері інформаційних технологій;
- приблизно 30 % з них закриваються, так і не завершившись; майже половина проектів завершуються з перевищенням первинних оцінок бюджету або строків та володіють обмеженою функціональністю;
- втрати від недоотриманого ефекту впровадження програмного забезпечення вимірюються трильйонами.

Статистика щодо 30 тисяч проектів з розробки програмного забезпечення в американських компаніях (рис. 1.2) [10] показує такий розподіл між:

- Успішними проектами – вчасно і в рамках бюджету – був виконаний весь намічений фронт робіт.
- Проблемними проектами – порушення термінів, перевитрату бюджету і/або зроблено не все, що вимагалося.
- Проваленими проектами – не були доведені до кінця через перевитрати коштів бюджету та якості.

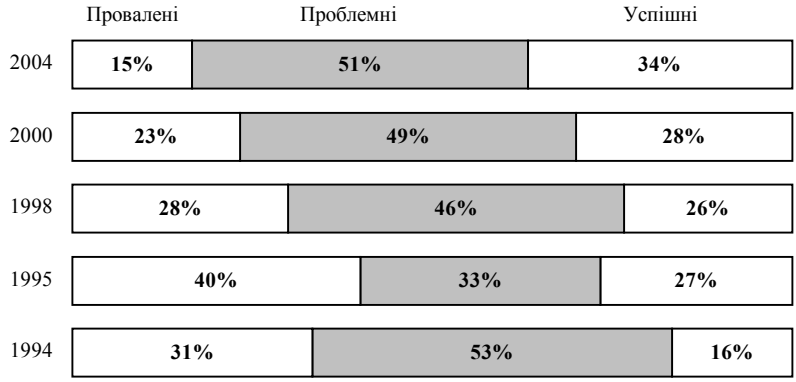

Рис. 1.2. Статистика успішності виконання проектів у американських компаніях протягом 1994–2004 років [10]

Таким чином, ми можемо очікувати продовження зростання значення програмної інженерії з цілого ряду причин. По-перше, світові витрати на програмне забезпечення постійно ростуть. Наприклад, у 1985 р. вони складали 140 млрд. доларів, а у 2000 р. перевищували 800 млрд. Уже один цей факт гарантує, що програмна інженерія ростиме як окрема галузь комп'ютерних знань. По-друге, програмне забезпечення чимраз ширше проникає в наше суспільство: дедалі більше програм використовується для управління найважливішими функціями самих різних машин, таких як літаки і медичні прилади, а також для підтримки глобальних процесів, подібних електронній комерції. Цей факт гарантує зростаючий інтерес суспільства до надійного програмного забезпечення, до розширення законодавчої основи для відповідних стандартів, вимог і процедур сертифікації. Немає сумніву, що залишиться важливим і питання навчання того, як у найкращий спосіб створювати ще більш довершене програмне забезпечення.

1.3 Визначальні події та особистості на шляху становлення інженерії програмного забезпечення як професійної галузі

Як уже зазначалося, в кінці 1960-х – на початку 1970-х років склалася ситуація, яка увійшла в історію як перша криза програмування. Особливість її полягала в тому, що вартість програмного забезпечення стала наближатися до вартості апаратного забезпечення комп'ютерів, а динаміка зростання цих вартостей дозволяла прогнозувати, що до середини 1990-х років все людство займатиметься розробкою програм для комп'ютерів. Тоді й заговорили про програмну інженерію (або технологію промислового програмування, як її називали на теренах СРСР) як про деяку дисципліну, метою якої є скорочення вартості програм.

Відтоді програмна інженерія пройшла достатньо бурхливий розвиток. Етапи розвитку програмної інженерії можна виділяти по-різному. Кожен етап пов'язаний із появою (або усвідомленням) чергової проблеми та знаходженням шляхів і способів її розв'язання [11].

Наприкінці 1990-х років знання і досвід, накопичені в індустрії програмного забезпечення за попередні 30–35 років, а також унаслідок більш ніж 15-річних спроб застосування різних моделей розробки, усе це, нарешті, оформилося в те, що прийнято називати дисципліною програмної інженерії – *Software Engineering*. Якоюсь мірою таке формування дисципліни на основі широко розповсюдженого практичного досвіду нагадує ті процеси, що відбувалися в управлінні проектами. Виникали і розвивалися професійні асоціації, спеціалізовані інститути, комітети зі стандартизації й інші утворення, які, зрештою, прийшли до спільної думки про необхідність зведення професійних знань з відповідних галузей і стандартизації відповідних програм навчання.

До 1990-х років існували два погляди на програмне забезпечення як галузь знань – точка зору комп'ютерних наук і точка зору інженерії програмного забезпечення (рис. 1.3). З погляду комп'ютерних наук, конструювання комп'ютерних програм – це математичні дії, подібні до розв'язування,

наприклад, диференціальних рівнянь. З іншої точки зору, інженерії програмного забезпечення, програмне забезпечення набуває рис конструктивного об'єкта інженерної сфери знань, а конструювання комп'ютерних програм стає частиною відповідної індустрії. Час показує, що в період становлення інженерії програмного забезпечення має місце гібридний погляд на галузь знань програмного забезпечення, для якого характерний перехід від першої точки зору, наприклад, у створенні невеликих програм («програмування в малому»), до другої точки зору, наприклад, у створенні програмного забезпечення з компонентів («програмування у великому»).

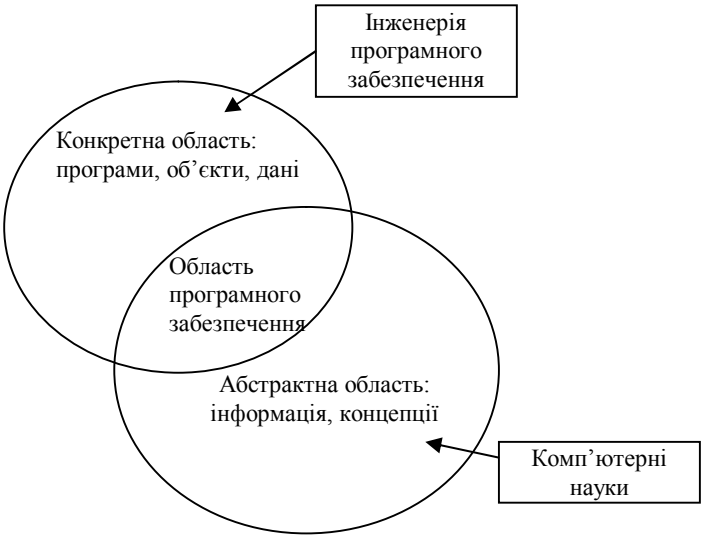

Рис. 1.3. Область програмного забезпечення

Сам термін – software engineering (програмна інженерія) – вперше був озвучений у жовтні 1968 року на конференції підкомітету NATO з науки і техніки (м. Гарміш, Німеччина) [10]. На конференції були присутні понад 50 професійних розробників програмного забезпечення з 11 країн. Учасники конференції обґрунтували наукову і практичну важливість нової дисципліни. Розглядалися проблеми проектування, розробки,

розповсюдження і підтримки програм. Там уперше і прозвучав термін «програмна інженерія» як деяка дисципліна, яку треба створювати і якою треба керуватися в процесі розв'язання зазначених проблем. При цьому, вперше акценти в методах, засобах і процесах розробки програмного забезпечення були зміщені, по-перше, з кодування програм на інші процеси їх розробки, а по-друге, з якісних аспектів убік кількісних, інженерних. Окрім цього, додатковий стимул одержали роботи економічного напряму і менеджменту проектів програмного забезпечення.

Незабаром після цього в Лондоні відбулася зустріч 22-х керівників проектів з розробки програмного забезпечення. На зустрічі аналізувалися проблеми і перспективи розвитку програмної інженерії. Особливо наголошувалося на зростаючому впливі програмного забезпечення на життя людей. Уперше серйозно заговорили про існуючу на той час кризу програмного забезпечення. Принципи, які застосовувалися, і методи розробки програм вимагали постійного вдосконалення. Саме на цій зустрічі була запропонована концепція життєвого циклу програмного забезпечення (SLC – Software Lifetime Cycle) як послідовності кроків-стадій, які необхідно виконати в процесі створення й експлуатації програм. Навколо цієї концепції точилося багато суперечок. У 1970 р. У.У. Ройс (W.W. Royce) провів ідентифікацію кількох стадій у типовому циклі та висловив припущення, що контроль виконання стадій приведе до підвищення якості програм і скорочення вартості їх розробки.

З 1975 року стали регулярно проводитися міжнародні конференції з інженерії програмного забезпечення (International Conference on Software Engineering), на які останніми роками надходить більше 400 заявок на доповіді і які збирають більше 1000 учасників, а з 1982 р. ще і Міжнародні симпозіуми ACM SIGSOFT з основ інженерії програмного забезпечення (ACM SIGSOFT International Symposium on the Foundations of Software Engineering). Дослідники вважають ці конференції кращими у галузі інженерії програмного забезпечення. Технічні заходи, присвячені програмній інженерії, регулярно проходять в Європі,

Азії, Індії, Південній Америці, країнах Тихоокеанського басейну і багатьох інших регіонів. Проводиться велика кількість дрібніших заходів, що часто присвячуються спеціальним напрямам програмної інженерії, зокрема конференції з навчання інженерії програмного забезпечення, конференції з супроводу, реверсивної інженерії та повторного використання програмного забезпечення.

Поряд із працями конференцій, розповсюдженню результатів досліджень сприяють журнальні публікації. До провідних журналів, що спеціалізуються на тематиці програмної інженерії, належать ACM Transactions on Software Engineering and Methodology, який видається з 1975 року (ACM – Association for Computing Machinery – Асоціація з обчислювальної техніки), а також IEEE Transactions on Software Engineering – праці з програмної інженерії, які видаються IEEE-CS з 1972 року (IEEE-CS – Computer Society of the Institute for Electrical and Electronic Engineers – Комп'ютерне Товариство Інституту Інженерів з Електрики та Електроніки). Програмній інженерії присвячуються й багато інших періодичних видань.

Перший цілісний погляд на цю сферу професійної діяльності з'явився у 1979 році, коли Комп'ютерне Товариство IEEE підготувало стандарт IEEE Std 730 щодо якості програмного забезпечення. Після 7 років напружених робіт, у 1986 році IEEE випустило IEEE Std 1002 "Taxonomy of Software Engineering Standards".

У 1990 р. почалося планування всеосяжних міжнародних стандартів, в основу яких лягли концепції та погляди стандарту IEEE Std 1074 і результатів роботи утвореної у 1987 р. спільної комісії ISO/IEC JTC 1 (ISO – International Organization for Standardization; IEC – International Electrotechnical Commission; JTC 1 – Joint Technical Committee 1, Information Technology).

У 1995 р. група цієї комісії SC7 "Software Engineering" випустила першу версію міжнародного стандарту ISO//IEC 12207 "Software Lifecycle Processes". Цей стандарт став першим досвідом створення єдиного загального погляду на програмну інженерію. Відповідний національний стандарт Росії – ГОСТ Р ИСО/МЭК 12207-99 [ГОСТ 12207, 1999] – містить повний

автентичний переклад тексту міжнародного стандарту ISO/IEC 12207-95 (1995 р.).

У свою чергу, IEEE і ACM, розпочавши спільні роботи ще у 1993 році з кодексу етики і професійної практики в даній галузі (ACM/IEEE-CS Code of Ethics and Professional Practice), до 2004 року сформулювали два ключових описи того, що ми сьогодні називаємо основами програмної інженерії – Software Engineering:

1. Guide to the Software Engineering Body Of Knowledge (SWEBOK), IEEE 2004 Version – Керівництво до збірки знань з програмної інженерії [7];
2. Software Engineering 2004. Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering – Навчальний план викладання програмної інженерії у вузах [1].

Обидва стандарти стали результатом консенсусу провідних представників індустрії і визнаних авторитетів у галузі програмної інженерії – за аналогією з тим, як був створений PMI PMBOK (A guide to the Project Management Body of Knowledge – Керівництво до збірки знань у галузі менеджменту проектів Інституту управління проектами).

Основоположною для інженерії програмного забезпечення слід вважати працю Ройса (W. Royce), в якій було введене поняття життєвого циклу програмного забезпечення. Завдяки цій роботі з'явилася можливість досліджувати окремі процеси і розробляти ресурси фаз життєвого циклу, а результати будувати як продукти інженерії. Науковець, починаючи з розгляду відомих на той час двох етапів – аналіз і кодування, увів нові, «наднормативні» етапи. Указується, що при існуючих підходах до розробки програм тільки на етапі тестування з'ясовується реальний час роботи системи, об'єм пам'яті, швидкість введення/виведення та інші кількісні параметри. Тому звертається увага на первинність процесу проектування, а не кодування і важливість документування програмного забезпечення.

Інші дослідження, які зробили принциповий внесок в інженерію програмного забезпечення, пов'язані з розвитком концепції життєвого циклу і його складових, а також із застосуванням економічних методів в інженерії програмного

забезпечення. Тут слід відзначити наукові виклади Б. Боема (B. Boehm). Результатом тривалих досліджень цих напрямів з'явилася розробка моделей життєвого циклу програмного забезпечення, методів і засобів, що забезпечують реалізацію відповідних процесів, а також моделей і засобів оцінки вартості програм. Останні моделі і засоби зараз широко використовуються в розробці проектів програмного забезпечення не тільки для визначення вартості розробки, але і для розрахунку інших параметрів, як життєвого циклу в цілому, так і окремих його фаз.

Окрім указаних учених (W. Royce, B. Boehm), слід нагадати про праці Ф. Брукса (F. P. Brooks), Х. Мілза (H. D. Mills), Д. Парнаса (D. L. Parnas), В. Хамфрі (W. S. Humphrey), Т. Маккейба (T. J. McCabe), Г. Буча (G. Booch), Т. Демарко (T. DeMarco), Д. Кнута (D. E. Knuth), С. Макконнелла (S. McConnell) та багатьох інших.

Найбільш авторитетною організацією з інженерії програмного забезпечення вважається Інститут програмної інженерії Software Engineering Institute (SEI), який знаходиться у структурі університету Карнегі-Меллона в Пітсбурзі США. Основним досягненням інституту є розробка та впровадження моделі оцінки зрілості програмного забезпечення (Capability Maturity Model for Software, SW-CMM). Відповідно до концепцій SW-CMM, зрілість процесу в організації визначає її здатність до розробки й випуску високоякісних програмних продуктів.

У колишньому Радянському Союзі перші роботи з інженерії програмного забезпечення були ініційовані О. П. Єршовим. Цю галузь тоді назвали «технологія програмування». Значний внесок в її дослідження зробив В. М. Глушков. Перші практичні результати одержані І. В. Вельбіцким. Окрім цього, слід відзначити праці Є. А. Жогольова, В. В. Ліпаєва, В. Н. Редько, Е. Х. Тиугу. Вони використовували термін «технологія програмування», розглядаючи його як деяке загальне базове поняття, яке, однак, часто виходить за рамки суто програмування. Тепер цей термін практично не використовується, хоча мають місце терміни «технологія розробки програмного забезпечення» і «програмна інженерія».

1.4. Перспективи та проблеми програмної інженерії в XXI столітті

Переважна більшість фахівців, які займаються розробкою та впровадженням програмних продуктів, вважають, що наступні кілька десятиліть зроблять XXI століття ерою програмного забезпечення. Саме воно стане основним елементом, що забезпечить людям необхідні можливості та якість життя, і фахівці, які знають, як розробляти високоякісні програмні системи, одержать найбільший шанс змінити світ на краще [4].

З одного боку, фахівці будуть дуже раді цій можливості, але, з іншого боку, на них ляже величезна відповідальність за забезпечення належної якості програмних систем, що розробляються, і сервісів, що надаються ними. Як вважає Баррі Боєм, основними проблемами, з якими зіткнуться в XXI столітті фахівці в галузі інженерії програмного забезпечення, є постійне прискорення змін, наявність невизначених і непередбачених ситуацій, функціональна надійність, диверсифікація і взаємозалежність.

Пройшли ті дні, коли люди все своє життя могли робити одне й те саме. У XXI столітті життєздатність організацій, які займаються розробкою програмного забезпечення, визначатиметься їх здатністю пристосуватися до частих і непередбачуваних змін та працювати швидше, ніж конкуренти. Проте за всі зміни доводиться платити. Людям подобаються результати змін, але зазвичай вони не люблять змінювати свою поведінку. Фахівці в галузі інженерії програмного забезпечення, що підвищують рівень зрілості процесів розробки, часто вважають, що досягнення найвищого рівня зрілості забезпечує належну зрілість і оптимальність процесів на всі часи. Цей підхід призводить до ризику надмірної оптимізації, він може зробити процеси розробки надмірно обтяженими і важко змінюваними.

Існує маса суперечностей між людьми та організаціями, які швидко пристосовуються до змін, і тими, хто вважає за краще не робити цього. Вдалим прикладом є труднощі, з якими зіштовхуються розробники програмного забезпечення, намагаючись пристосуватися до змін при дотриманні фіксованої

структури контракту та специфікації на розробку програмного забезпечення. Такого роду контракти визначаються адміністраторами, що дотримуються принципу «That's How We've Always Done It» – «ми завжди так робимо». У комерційному світі розробляються більш довершені структури контрактів, засновані, наприклад, на моделі «загальної долі» («shared destiny»). Застосування вдосконалених і загальноприйнятих видів контрактів сприятиме успішній розробці програмного забезпечення за наявності частих змін.

Багато джерел змін відносно непередбачувані, наприклад, ті, що пов'язані з виникненням нових технологій. Часто, коли користувачів просять наперед специфікувати бажаний для них спосіб взаємодії з новим додатком, вони відповідають «I'll Know It When I See It» – «це буде зрозуміло, коли ми його побачимо». У таких випадках використання послідовної каскадної моделі, в якій спочатку специфікуються всі вимоги, швидше за все, призведе до створення неповороткої системи, що вимагає великого обсягу робіт для внесення змін.

Для скорочення невизначеності найкраще використовувати стратегію «Buy Information To Avoid Risk» – «щоб уникнути ризику, купуйте інформацію». Ця стратегія передбачає витрачання додаткових коштів на прототипування, імітаційне моделювання, еталонне тестування, аналіз тенденцій ринку і т.д. для скорочення рівня невизначеності та ризиків. Окрім того, важливо організовувати майбутні проекти так, щоб залишалася можливість пристосуватися до появи нових джерел непередбачених змін. Це вимагає пристосованості до швидких змін у колективі розробників, в їх організації, в архітектурі програмного продукту і процесі його розробки.

Разом з розвитком можливості швидких змін у майбутніх проектах потрібно буде добиватися вищого рівня функціональної надійності (dependability), оскільки програмне забезпечення стає домінуючим чинником у конкурентних відмінностях продуктів і сервісу, які пропонуються компаніями. Одночасне досягнення здатності до швидких змін і функціональної надійності є однією з

найбільших проблем XXI століття для фахівців у галузі інженерії програмного забезпечення.

Для створення функціонально надійної системи важливо визначити, які зацікавлені сторони можуть найсильніше вплинути на успішність майбутньої системи, і чого вони понад усе потребують. Потреби різних зацікавлених сторін (кінцевих користувачів, адміністраторів, продавців), швидше за все, конфліктуватимуть. Наявність цих конфліктів означає, що потенційно проект може призвести до ситуацій, в яких виграш однієї зацікавленої сторони є програшем іншої сторони. Такі ситуації зазвичай перетворюються на ситуації, в яких не виграє жодна сторона: інвестори виявляють перевитрату бюджету, користувачі незадоволені продуктивністю, експлуатаційний персонал не влаштовує відсутність сумісності з іншими системами тощо.

Щоб уникнути подібної ситуації важливо вміти погоджувати очікування зацікавлених сторін, витрачати великі зусилля на забезпечення застосовності пропонованого рішення для всіх зацікавлених сторін, домовлятися про набір вимог, планів і ресурсів, що влаштовує всі сторони, до початку розробки. На ранній стадії узгодження вимог замість жорсткого терміна «вимога» краще використовувати терміни «завдання» або «мета».

«Безрозмірне» (one-size-uniformly-fits-all, OSUFA) визначення функціональної надійності в термінах часу безвідмовної роботи систем часто влаштовує не всі зацікавлені сторони. Висновок про непридатність підходу OSUFA стає ще явнішим за наявності тенденції до глобалізації, що охоплює різні культури. Наприклад, через відмінності в культурі країни тільки 17 з 380 компаній з розробки програмного забезпечення Таїланду зважилося використовувати запропоновану в США модель CMM (Capability Maturity Model), яка є стандартом з оцінки зрілості процесів розробки, що мають місце в компаніях.

Підхід OSUFA і стандартизація процесів розробки особливо погано відповідають ще одній тенденції XXI століття – появі великих, переважно програмних систем, що будуються з існуючих систем (software-intensive systems of systems, SISOS). Ця

тенденція означає наявність вимоги масштабування при розв'язанні проблеми одночасного забезпечення можливості швидких змін і функціональної надійності. Таким чином, у XXI столітті будуть вестися активні пошуки альтернативних підходів до принципу OSUFA для розробки програмних продуктів.

На сьогоднішній день ні в кого немає чарівного рецепту, як створити надійне програмне забезпечення, здатне успішно працювати в будь-яких проектах і організаціях, – є велика кількість неоднозначних питань, які, можливо, ніколи не одержать універсальної відповіді. За своєю складністю завдання створення надійного програмного забезпечення прирівняне сьогодні до «грандіозних викликів» (Grand Challenges) у технологіях [4].

Багато професійних розробників формують свій світогляд у рамках методологічного інструментарію лише одного постачальника, покликаного розв'язувати прикладні задачі строго певним способом. Показові зростання функціональних можливостей і, головне, зниження складності використання інструментарію для виконання типових завдань. Окрім зниження вартості розробки це опосередковано веде до зниження якості програмних систем, а отже, до підвищення вартості їх експлуатації внаслідок зростання кількості збоїв і помилок. Таке циклічне зчеплення явищ засноване на порочному колі: знижується престиж професії програміста і, як наслідок, руйнується цикл спадкоємності.

Програмісти, що мають значний досвід і високу кваліфікацію, за допомогою легшого у використанні інструментарію швидше реалізують завдання, які стоять перед ними. Скорочення циклу розробки веде до прискорення кар'єрного зростання досвідчених розробників, а в результаті потрібні нові програмісти замість тих, які пішли на підвищення. Проте базові навички знижуються (адже інструмент став «доброзичливіший»). Новачки знаходяться у скрутному становищі: як набути різностороннього практичного досвіду, якщо інструмент цього вже не дозволяє (він більш спеціалізований), а на розв'язання задачі відпущено дуже мало часу? Формується попит на ще «доброзичливіший» інструмент.

Багато навичок досвідчених розробників через непотрібність відмирають, а у нових програмістів вони так і не встигають сформуватися. Підростаючі в статусі (але не в навичках) програмісти йдуть на підвищення, залишаючи вакантні місця для нового набору, і ситуація повторюється.

Усе це призводить до досить небажаних наслідків:

- розробник інтерпретує завдання лише в термінах свого інструментарію і здатен виконати його тільки в рамках свого розуміння, межі якого визначаються інструментарієм;

- зростання простоти використання інструменту веде до еквівалентного зниження його гнучкості. Це зростання здійснюється переважно за рахунок поглиблення спеціалізації, а не за рахунок розширення навичок і світогляду розробника;

- чим більша відмінність між шаблонним рішенням з бібліотеки інструменту й оптимальним розв'язанням задачі, тим більше праці потрібно розробнику для боротьби із самим інструментом і його обмеженістю;

- чим менше навичок у розробника і складніше завдання боротьби з інструментом, тим менш остаточне рішення відповідає оптимальному.

Надмірне захоплення спрощенням завдань відзначав ще Эдгстер Дейкстра в передмові до «Дисципліни програмування»: «Присвятивши немало років свого наукового життя тому, щоб прояснити завдання програміста і зробити їх більш підвладними нашому інтелекту, я зі здивуванням (і роздратуванням) виявив, що моє прагнення внести ясність призводить до систематичних звинувачень у «внесенні труднощів у програмування». Але ці труднощі завжди в ньому були, і лише зробивши їх видимими, ми зможемо сподіватися, що навчимося розробляти програми з високим ступенем надійності, а не просто «ліпити команди»». Важко сказати краще.

Сучасні середовища програмування, такі як JBuilder, Delphi або Visual Studio, мають у своєму складі бази знань про виконані завдання, теоретичну складову й каркасні бібліотеки. Вони захищають програміста від пошуку неординарних рішень, жорстко задаючи русло розв'язання прикладних задач. А це, у

свою чергу, перешкоджає розширенню світогляду і підвищенню кваліфікації розробника.

Давнє прагнення зменшити вартість створення програмного забезпечення за рахунок зниження вимог до розробників, можливе при полегшенні використання інструментів, дається взнаки. Насіння, посіяне у 1980-ті та 1990-ті роки, починає рясно сходити, і це не може не насторожувати. Щоб побачити ці сходи, досить відзначити молодих програмістів, здатних у процесі самоствердження завдавати величезні збитки за допомогою кількох десятків рядків коду на макромові. Є кілька причин для неспокою, пов'язаних із «безальтернативністю» конвеєрного підходу до розробки програм.

По-перше, прагнення «не їсти, а жити за рахунок накопиченого жиру» порочно. Так не може продовжуватися довго, оскільки не відбувається повсюдна передача досвіду від старшого покоління молодшому, тобто відсутня спадкоємність. Старше покоління програмістів при переході на нові засоби ще якось тримає марку, користуючись одержаною раніше кваліфікацією. Але з молодим поколінням все інакше: не маючи можливості дізнатися, як насправді працюють програми, програміст перебуває в стані повної упевненості, що він все розуміє.

По-друге, прагнення понизити складність завдання призводить до того, що з її початкової постановки відкидаються якісь «малозначні» умови. Потім ще якісь. І так доти, поки завдання не вміщається «у двох рядках» і не стає доступним для розуміння «практично кожному». В результаті початкове завдання і те, яке доводиться виконувати, можуть мати дуже мало спільного.

По-третє, конвеєрний підхід до розробки програм передбачає масовість, але негативний вплив комунікацій між програмістами, що беруть участь у проекті, враховувати зазвичай забувають. Є цілий клас завдань, які можуть реалізовуватися виключно малим колективом розробників. Масовість при їх виконанні означає провал проекту.

Цю тему добре розкриває Фредерік Брукс у праці «Міфічний людино-місяць» [9]. До того ж за роки, які пройшли з моменту виходу книги, описувана в ній ситуація не тільки не поліпшилась, але і значно погіршилася. Відгородившись від дійсності новомодними теоріями та інструментами, керівники програмних проектів помилково стверджують про відсутність суттєвого зв'язку між персоналіями та їх робочими годинами. Такі керівники не усвідомлюють, що якщо одному кваліфікованому програмісту для виконання логічно складного завдання потрібно 40 годин, то п'яти «еквівалентним» розробникам деколи необхідні зовсім не вісім, а удвічі, а то й утричі більше. Не виключено, що завдання і зовсім не буде виконане, оскільки програмісти просто не зможуть домовитися.

По-четверте, у міру виходу людини на певний рівень знань і досвіду в неї не тільки з'являються додаткові можливості розв'язання складних задач, але й істотно підвищується самооцінка. Досвідчені фахівці вимогливі як до рівня оплати своєї праці, так і до тієї атмосфери зайнятості, яку конвеєрний підхід з його «зрівнялівкою» принципово не може дати, оскільки, в цьому випадку розробник – лише гвинтик у великому механізмі. Отже, істотно слабшають стимули появи подібних фахівців «усередині», а мотивувати їх «ззовні», окрім грошей, банально нічим. Масовість конвеєрного підходу до розробки програм автоматично передбачає зниження значення персоналій, що сильно б'є по професіоналізмові залучених у процес виконавців.

Урешті-решт, сенс поняття «спеціалізація» зводиться до формування якнайповніших знань і навичок у конкретній області на шкоду ширшим. Іншими словами, спеціалізація передбачає звуження світогляду дослідника з метою глибшого занурення в наочну сферу. Як показує досвід, найскладнішими, але найбільш важливішими функціями менеджера є правильна ідентифікація проблеми та її формування у вигляді питання (або ряду взаємозв'язаних питань), що має конкретне рішення. Крім того, менеджер повинен передавати завдання на виконання підлеглим і контролювати це виконання.

Маючи світогляд фахівця, менеджер знаходиться під впливом своєї наочної ділянки роботи, що заважає йому бути об'єктивним по відношенню до проблем, що знаходяться поза цією сферою. Але якщо проблема ідентифікується лише в рамках певної спеціалізації, немає надії на достовірний результат: вона неминуче ідентифікується хибно.

Це спричиняє такі наслідки:

- кероване інформаційне поле стає вузьконаправленим;
- менеджмент здатний вести діалог тільки на своїй мові, в рамках свого культурного середовища (спеціальності), ігноруючи або відчужуючи носіїв інших поглядів;
- з'являється або посилюється динамічна мінливість кадрів;
- просування по службі здійснюється на основі схильності індивідуума до пануючих поглядів;
- ігноруються або, в кращому разі, неправильно інтерпретуються проблеми, що не входять до спеціалізованої наочної сфери, накопичується багаж нерозв'язаних проблем.

Як результат, вибудовувана менеджментом інфраструктура має незгладимі відбитки панівних поглядів.

Отже, можна окреслити наслідки проблем сучасної програмної інженерії [4, 9–11]:

1. Частішає помилкове ставлення користувачів до використовуваних програм. Не можна явно перевірити достовірність рекламних тверджень унаслідок закритості початкових кодів переважної більшості комерційних програмних продуктів. А результати непрямих перевірок, проведених за допомогою навантажувальних та інших методик тестування, можуть виявитися недостовірними.

2. Росте вартість програм. По-перше, вона, як правило, значно вище від витрат розробника. По-друге, самі витрати можуть бути істотно понижені шляхом відмови від деяких «фундаментальних» міфів і застосування методик дослідження наочних сфер (проектування, розробки і тестування), очищених від цих міфів.

3. Зростає вартість інтеграції. Основний внесок в це зростання дає відсутність підтримки виробниками концепції відкритих систем: архітектура багатьох промислових систем закрита, що

істотно ускладнює інтеграцію замовниками продуктів різних виробників. Відсутність підтримки відкритих систем необхідна постачальникам для нав'язування своїх рішень. Вона легітимується за допомогою міфу про те, що «по-справжньому інтегровані системи можливі тільки в тому випадку, якщо вони побудовані з продуктів одного постачальника».

4. Росте «вагомість» програм. Для їх нормального функціонування дедалі більше необхідна підтримка значної кількості обчислювальних, технічних, людських та інших ресурсів. Союз виробників програмного забезпечення й апаратури змушує кінцевого користувача витрачати дедалі більше коштів: IT-бюджети компаній ростуть набагато швидше, ніж віддача від розв'язаних за допомогою IT бізнес-задач.

5. Знижується надійність. Це спричинює зростання кількості відмов та незапланованих простоїв, про що свідчить, зокрема, збільшення кількості шкідливих програм і комп'ютерних злочинів з використанням «вузьких місць» і «незапланованих можливостей» програмних систем.

6. Зростає небезпека фатальних помилок. Зниження надійності програм переходить з розряду негативних чинників у розряд катастрофічних. Вірогідність помилки, здатної призвести до техногенних катастроф національних масштабів, не зменшується. У таких умовах неадекватне ставлення до якості програмного забезпечення, що демонструється багатьма розробниками, з розряду халатності переходить у розряд злочинних дій.

Контрольні запитання та завдання

1. Розкажіть коротку історію розвитку програмування.

2. Дайте характеристику кризі програмного забезпечення 1970-х років. Які шляхи були намічені світовим професійним співтовариством для подолання кризи?

3. Викладіть у вигляді короткого реферату автобіографію та основні здобутки людини, яка зробила важливий внесок у розвиток та становлення програмної інженерії як самостійної галузі. Наприклад: Kent Beck, Barry Boehm, Fred Brooks тощо.

4. Вкажіть проблеми та перспективи програмної інженерії.

РОЗДІЛ 2. ХАРАКТЕРИСТИКА ПРОФЕСІЙНОЇ ІНЖЕНЕРНОЇ ДІЯЛЬНОСТІ РОЗРОБНИКІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Програмна Інженерія (Software Engineering) [11] – наука побудови комп'ютерних програмних систем, що містить у собі теоретичні концепції, методи і засоби програмування, технологію програмування, системи та інструменти їхньої підтримки, сучасні стандарти, зокрема процеси життєвого циклу, вимірювання, оцінювання якості розробки програмних продуктів. Головне призначення програмної інженерії – побудова програмних продуктів, починаючи з аналізу предметної області й закінчуючи виготовленням вихідного коду для виконання на комп'ютері. Фундаментальну основу побудови програмних продуктів становлять: теорія алгоритмів, математична логіка, теорія обчислень, теорія керування та ін.

Колективна розробка великих програмних проектів зумовила розвиток інженерних, технологічних методів і засобів регламентованого проектування програмних систем з урахуванням організаційних процесів життєвого циклу: інженерія вимог, керування ризиком і якістю, планування і регулювання ресурсів, оцінювання процесів життєвого циклу та показників якості, вартості і строків виготовлення програмного продукту.

2.1. Основні визначення та відмітні риси програмної інженерії

Інженерія програмного забезпечення – це сфера комп'ютерної науки, яка займається побудовою програмних систем, настільки великих або складних, що для цього потрібна участь команди розробників, деколи навіть кількох команд. Зазвичай такі системи існують довгі роки, розвиваючись від версії до версії, зазнаючи на своєму життєвому шляху певну кількість змін, таких як усунення помилок, поліпшення існуючих, додавання нових або видалення застарілих можливостей, адаптація для роботи в новому середовищі. Можна охарактеризувати програмну інженерію як «застосування принципів інженерної розробки до програмного забезпечення». Можна сформулювати й більш

строге визначення суті програмної інженерії як «застосування систематизованого, наукового і вимірюваного підходу до створення, функціонування і супроводу програмних продуктів».

Девід Парнас визначив інженерію програмного забезпечення як «колективне проектування багатоваріантного програмного забезпечення». У ньому не тільки відображена суть терміна, але і підкреслюється відмінність між програмуванням і програмною інженерією. Програміст пише програму цілком, а програмний інженер створює тільки програмний компонент, який входить у програмну систему, стикуючись із компонентами, створеними іншими такими ж інженерами. Компонент, написаний одним програмним інженером, може бути переписаний або використаний іншими розробниками для створення нових версій системи, навіть якщо автор компонента давно покинув проект. Програмування – це діяльність, здебільшого, індивідуальна, а програмна інженерія – завжди колективна.

Загалом на сьогоднішній день немає єдиного визначення поняття «Програмна інженерія». Наведемо кілька визначень, сформульованих провідними спеціалістами в цій галузі, або зафіксовані в документах провідних організацій [10, 11].

Програмна інженерія – це:

– встановлення і використання обґрунтованих інженерних принципів для економного отримання програмного забезпечення, яке є надійним і реально працює (Фрідріх Бауер, 1972);

– та форма інженерії, яка застосовує принципи комп'ютерних наук і математики для рентабельного розв'язання проблем програмного забезпечення (CMU/SEI-90-TR-003);

– застосування систематичного, дисциплінованого, вимірюваного підходу до розробки, використання й супроводу програмного забезпечення (IEEE, 1990);

– дисципліна, метою якої є створення якісного програмного забезпечення, яке завершується вчасно, не перевищує виділених коштів і задовольняє висунуті вимоги (Стівен Шах, 1999);

– це система методів, способів і дисциплін з планування, розробки, експлуатації та супроводу програмного забезпечення, призначених для промислового виробництва (SWEBOK);

– розділ комп'ютерної науки, який вивчає методи і засоби побудови комп'ютерних програм; відображає закономірності розвитку та узагальнює накопичений досвід програмування; оперує об'єктами (модулями, компонентами, програмними аспектами тощо) та визначає автоматизовані операції щодо їхнього застосування; виробляє правила, порядок інженерної діяльності і керування технологічним процесом побудови з простих об'єктів нових, більш складних, цільових об'єктів (програмного забезпечення, програмних систем, сімейств систем, програмних проектів тощо), а також методи вимірювання й оцінювання готового продукту.

На відміну від математичної або інших фундаментальних наук, метою яких є отримання нових знань для розв'язання відповідних задач, метою програмної інженерії є застосування знань для розробки складних програмних об'єктів, де знання – це уособлення загальної теорії побудови програм для комп'ютерів, орієнтованої на виготовлення продукту, впровадження якого принесе певну користь користувачеві.

Програмна інженерія як наукова дисципліна – це теоретичні, формальні методи та відповідні засоби побудови складних програмних об'єктів.

Програмна інженерія – це інженерна дисципліна, яка пов'язана зі всіма аспектами виробництва програмного забезпечення, від початкових стадій створення специфікації до підтримки системи після здачі в експлуатацію. У цьому визначенні є дві ключові фрази: *інженерна дисципліна* та *всі аспекти виробництва програмного забезпечення*.

Інженерна дисципліна. Інженери – це ті фахівці, які виконують практичну роботу й добиваються практичних результатів. Учений може сказати: проблема не може бути розв'язана в рамках існуючих теорій і це буде науковий результат, гідний публікації та захисту дисертації.

Для виконання завдання інженери застосовують відповідні теорії, методи і засоби, але вони застосовують їх вибірково і завжди намагаються знайти рішення, навіть у тих випадках, коли теорій або методів, відповідних даному завданню, ще не існує. У

цьому випадку інженер шукає метод або засіб для розв'язання задачі, застосовує його і несе відповідальність за результат – адже метод або засіб ще не перевірені. Набір таких інженерних методів або способів, можливо, теоретично не обґрунтованих, але таких, що отримали неодноразове підтвердження на практиці, відіграє велику практичну роль. У програмній інженерії вони отримали назву кращих практик (best practices).

Інженери працюють в умовах обмежених ресурсів: часових, фінансових і організаційних (обладнання, техніка, люди). Іншими словами, продукт має бути створений у встановлені терміни, в рамках виділених коштів, обладнання і людей.

Усі аспекти виробництва програмного забезпечення. Програмна інженерія займається не тільки технічними питаннями виробництва програмного забезпечення (специфікація вимог, проектування, кодування тощо), але і управлінням програмними проектами, включаючи питання планування, фінансування, управління колективом і так далі. Крім того, завданням програмної інженерії є розробка засобів, методів і теорій для підтримки процесу виробництва програм.

Програмні інженери застосовують систематичні й організовані підходи до роботи для досягнення максимальної ефективності та якості програмного забезпечення. Їх завдання полягає в адаптації існуючих методів і підходів для розв'язання конкретної проблеми.

Таким чином, програмна інженерія як інженерна галузь характеризується діяльністю, заснованою на таких принципах:

- *ефективність* – результати одержують заданими ресурсами, і вони відповідають заданим вимогам і стандартам;
- *практичність* – результати мають конкретних замовників;
- *фундаментальність* – результати одержують на основі знань фундаментальних наук;
- *наслідуваність* (повторне використовування) – результати одержують на основі накопиченого досвіду, виключаючи діяльність «з нуля»;
- *відчутність* – результати є відчутними продуктами, які можна застосовувати, руйнувати і досліджувати за допомогою емпіричних методів пізнання;

- *супроводжуваність* – результати, знаходячись в експлуатації, обов'язково супроводжуються (обслуговуються).

У процесі розвитку людства з'явилося багато інженерних галузей, але їх становлення проходило один і той самий шлях, в якому розрізняють три фази (рис. 2.1) [6].

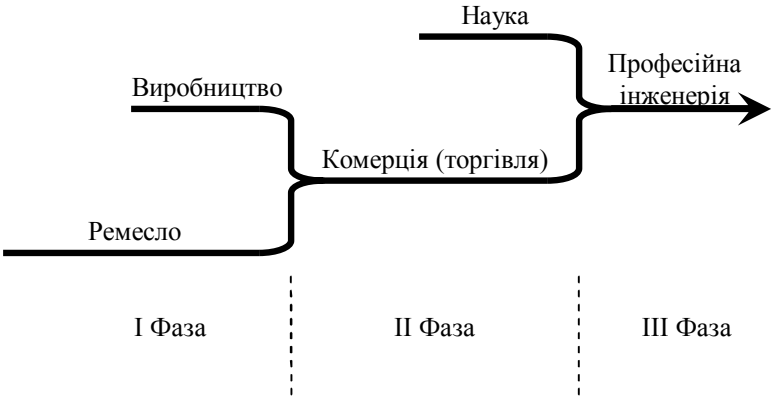

Рис. 2.1. Фази розвитку інженерної галузі [6]

Кожна фаза відрізняється виконавцями, ресурсами, методами реалізації і використання продуктів галузі (табл. 2.1) [6]:

- фаза I: виконавці – віртуози і талановиті одинаки; ресурси – інтуїція і власний досвід; методи – випадкова передача досвіду, екстравагантне застосування матеріалів; використання – виробництво для себе;

- фаза II: виконавці – майстерні виробники; ресурси – окремі інструменти; методи – механічний тренінг, облік економічних чинників у виборі матеріалів; використання – виробництво для продажу, утворення ринку;

- фаза III: виконавці – освічені професіонали; ресурси – машини і комплекси, які використовуються в технологіях; методи – теоретичні і емпіричні, передача знань шляхом диференційованого навчання, супровід; використання – сегментація ринку.

Таблиця 2.1

ХАРАКТЕРИСТИКИ ФАЗ РОЗВИТКУ ІНЖЕНЕРНОЇ ГАЛУЗІ

Характеристики / Фаза	Виконавці	Ресурси	Методи	Використання
Фаза I	Віртуози і талановиті одинаки	Інтуїція, груба сила і власний досвід	Випадкова передача досвіду, екстравагантне застосування матеріалів	Виробництво для себе
Фаза II	Майстерні виробники	Окремі інструменти	Механічний тренінг, облік економічних чинників у виборі матеріалів	Виробництво для продажу, утворення ринку
Фаза III	Освічені професіонали	Машини і комплекси, які використовуються в технологіях	Теоретичні та емпіричні, передача знань шляхом диференційованого навчання, супровід	Сегментація ринку

Характеристика вказаних фаз для інженерії програмного забезпечення наводиться в табл. 2.2 [6].

Таблиця 2.2

ХАРАКТЕРИСТИКИ ФАЗ РОЗВИТКУ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Фаза (початок) / Характеристика	I (1960 р.)	II (1970 р.)	III (1980 р.)
Особливості програмування	Програмування «абияк»	Програмування «в малому»	Програмування «у великому»
Підготовка кадрів	Практично відсутня	Прикладна математика	Комп'ютерні науки
Ресурси	Асемблери, машинні коди	Транслятори, лінкери, завантажувачі, програмні системи	Середовища розробки програм
Технології	Відсутні	HIPO, формалізовані технічні завдання	Системи PSL, SREM, SADT
Економіка	Відсутня	Інтуїтивна	Системи SCEP, SLIM
Ринок	Відсутній, замовні програми	Виробництво для продажу	Сегментація ринку

HIPO – Hierarchical-Input-Processing-Output – технологія розробки програмного забезпечення на базі ідеї структурного програмування.

PSL – Problem Statement Language – мова постановки задач.

SREM – Software Requirements Engineering Methodology – методологія розробки технічних вимог до програмного забезпечення.

SADT – Structured Analysis and Design Technique – структурний аналіз і проектування.

SCEP – Simple Certificate Enrolment Protocol – протокол реєстрації сертифікатів.

SLIM – Service Level Improvement Method – метод поліпшення рівня обслуговування.

Реалізація інженерної діяльності здійснюється інженерами в контексті технологій. Технологія – це організована сукупність процесів, спрямована на отримання з початкових матеріалів кінцевих продуктів за допомогою методів і засобів технологій. Інженери – це професіонали, чия освіта дозволяє їм, використовуючи знання фундаментальних наук і конкретних технологій, реалізовувати процеси, застосовуючи методи і засоби технології для створення надійних, широко використовуваних продуктів. Часто доброго програміста, що вміє писати структуровані програми, що складаються з багаторазово використовуваних компонентів, називають інженером з програмного забезпечення. На те, що це не відповідає дійсності, указує, наприклад, перелік спеціальностей і спеціалізацій, якими може володіти інженер з програмного забезпечення (табл. 2.3). Видно, що в цьому переліку немає професії «програміст».

Таблиця 2.3

ПЕРЕЛІК СПЕЦІАЛІЗАЦІЙ ПРОГРАМНИХ ІНЖЕНЕРІВ

Інженер з програмного забезпечення				
Спеціальність	Практичний інженер	Системний інженер	Проектний менеджер	Інженер з технічної підтримки
Спеціалізації	Архітектор, проектувальник (дизайнер), тестувальник, інтегратор, аналітик програмного забезпечення, аналітик корисності і людського чинника, специфікатор вимог, інженер з безпеки, секретності і надійності	Системний аналітик, прикладний архітектор, бізнес-аналітик, системний інтегратор, системний інженер, системний архітектор, специфікатор системних вимог	Проектний менеджер, укладач графіків, планувальник, інженер з оцінки вартості, ризик-менеджер, аналітик, керівник розробників, фінансовий менеджер	Документатор, менеджер конфігурацій, інженер з якості і гарантій, інженер із законодавчої і фінансової підтримки, інженер з безпеки, секретності і надійності

На даний час інженерія програмного забезпечення – це систематизований, регламентований і структурований підхід до розв’язання задач розробки, експлуатації, супроводу й утилізації програмного забезпечення. При цьому процеси та програмне

забезпечення повинні відповідати заданим технічним, економічним, соціальним і правовим вимогам.

Технічні вимоги обов'язково відображають те, що, як процеси, так і продукти життєвого циклу повинні повністю задовольняти вимогам, специфікованим замовником.

Економічні вимоги обов'язково відображають те, що проект програмного забезпечення повинен виконуватися в рамках заданого фінансового бюджету.

Соціальні вимоги обов'язково відображають те, що створювані програмні продукти повинні володіти властивістю корисності.

Правові вимоги обов'язково відображають те, що виконання програмного проекту повинне здійснюватися законними методами. Особливо це важливо, коли при розробці застосовується успадковане програмне забезпечення або компоненти багатократного використання.

Як і інші інженерні дисципліни, інженерія програмного забезпечення характеризується таким [10, 11]:

- *творчість* – інженерія концентрується на проблемах аналізу і проектування;

- *інструментальність* – ключові проблеми в інженерії – це вибір і використання інструментів та засобів;

- *стандартизація* – кращі практичні досягнення інженерії у вигляді інженерних принципів є основою створення стандартів;

- *спадкоємство* (повторне використання) – в інженерії, повторне використання знань і продуктів фаз життєвого циклу є найважливішим чинником підвищення продуктивності та якості;

- *професіоналізм* – інженерія програмного забезпечення – це професія.

Остання властивість характеризує інженерію програмного забезпечення не як академічну, а швидше, як практичну дисципліну. При цьому професійний інженер з програмного забезпечення відрізняється такими властивостями [10, 11]:

- ухвалює рішення, оцінюючи стан проблеми, й кожного разу вибирає підходи для виконання конкретних завдань і в конкретному контексті, знаходячи баланс між витратами й прибутком;

- оцінює та вибирає засоби аналізу тих чи інших параметрів програмного забезпечення та технологій їх створення;

- виконує в практичній діяльності одну з багатьох функцій, наприклад, дослідник, розробник, архітектор, виробник, тестувальник, експлуатаційник, керівник, продавець, консультант, викладач;

- результати праці інженера можуть бути різні, від пристроїв і систем, до процесів і структур;

- застосовує знання з інших дисциплін (на додаток до своїх власних), наприклад, з математики, базових наук і економіки. При цьому основними дисциплінами є дисципліни комп'ютерних наук, дискретна математика і групова динаміка;

- створює інструменти для різноманітних етапів розробки програмного забезпечення;

- працює дисципліновано й систематично;

- працює в колективах, разом з іншими професіоналами, розвиваючи навички взаємостосунків і колективної роботи;

- дотримується етичних і професійних принципів, захищаючи суспільство, замовників і себе;

- постійно поповнює свої знання, освоюючи нові методи, техніку й технології;

- спирається на специфічні знання і досвід, працюючи в межах специфічних галузей знань, враховуючи особливість відповідних специфічних рішень, виробництва, вимог тощо;

- уміє визначати, які частини програмного забезпечення можна повторно використати, а які необхідно розробляти наново.

Як уже зазначалося, термін «інженерія програмного забезпечення» був придуманий у кінці 1960-х рр., коли стало зрозуміло, що кількість накопичених уроків програмування не переходить в якість програмних систем. Незважаючи на величезний прогрес у програмуванні – систематизоване вивчення алгоритмів і структур даних, а також винахід «структурного програмування» – як і раніше, залишалися серйозні проблеми в побудові великих програмних систем. Програмні методи, які використовував фізик при розв'язуванні диференціальних рівнянь якого-небудь експерименту, не годилися для програміста,

що працює в команді розробників операційної системи або системи управління базами даних. Був потрібен класичний інженерний підхід: ясно визначити завдання, яке стоїть перед розробником, а потім використовувати й розвивати готові стандартні інструменти і методи для його реалізації.

Прогрес програмної інженерії з 1960-х рр. беззаперечний. Стандартні прийоми упорядковують цю сферу, перетворюючи її з мистецтва на ремесло, у те, що ми зазвичай розуміємо під технологією або інженерією. Звичайно, відмінності від традиційних технологій залишаються. Скажімо, при розробці підсилювача інженер-електрик може точно описати систему. Усі параметри і допустимі відхилення чітко визначені й однаково розуміються як замовником, так і розробником. Такі параметри для програмних систем дотепер невідомі. Ми не знаємо, ані які параметри визначати, ані як це робити.

У класичних інженерних дисциплінах у розробника є інструменти і математична підготовка для визначення параметрів виробу незалежно від процесу проектування. Наприклад, усе той же інженер-електрик спирається на математичні рівняння, щоб переконатися, що проект не спотворить розрахункову споживану потужність приладу. У програмній інженерії такий математичний інструментарій розвинений слабо, та і сама його застосовність усе ще під питанням. Типовий програмний інженер значно більше довіряє своєму досвіду і міркуванню, ніж математичним методам. Досвід і міркування, звичайно, необхідні, але і формальні засоби аналізу також доволі істотні в інженерній практиці.

Сукупність програмних засобів часто буває компонентом набагато більшої системи, а процес програмної розробки, відповідно, – частиною набагато об’ємнішої діяльності щодо проектування системи. У такому проектуванні вимоги до програмного забезпечення узгоджуються з вимогами до інших частин системи, що розробляється. Наприклад, у систему телефонної комутації входять комп’ютери, телефонні лінії і кабелі, телефони, інше устаткування, таке як супутники, і, нарешті, програмне забезпечення, що управляє різними

компонентами. Передбачається, що саме комбінація цих складових задовольняє вимоги до системи в цілому.

Такі вимоги, як: «система не повинна не діяти більше однієї секунди протягом 20 років» або «коли телефонна слухавка знята, звучить гудок протягом півсекунди», – можуть бути задоволені комбінацією налаштування устаткування, програмного забезпечення і пристроїв спеціального призначення. Остаточне рішення щодо якнайкращої відповідності всім вимогам засноване на ухваленні компромісних рішень. Системи управління електростанцією або дорожнім рухом, банківські системи, системи адміністрування лікарень – ось додаткові приклади, що демонструють необхідність розглядати програмне забезпечення як компонент більшої системи.

Програмне забезпечення все більше і більше вбудовується в різні системи – від телевізорів до літаків. Робота з такими проектами вимагає від програмного інженера широкого погляду на загальні завдання системного проектування. Програмному інженеру необхідно брати участь у формуванні вимог для всієї системи, а також спробувати зрозуміти прикладну область ще до початку розробки абстрактних інтерфейсів, вимогам яких повинно буде відповідати програмне забезпечення. Наприклад, якщо у приладу, що відіграє роль інтерфейсу з користувачем, є тільки примітивні можливості введення даних, то навряд чи в такій системі знадобиться розвинений текстовий процесор.

Розглядаючи інженерію програмного забезпечення як частину системотехніки, ми виявляємо важливість компромісу як відмітної ознаки будь-якої інженерної дисципліни [10]. Класичний компроміс передбачає вибір між тим, що повинно бути зроблено в програмному забезпеченні, і тим, що повинно бути зроблене в апаратурі. Програмна реалізація забезпечує гнучкість, тоді як апаратна – продуктивність.

Розвиток галузі програмування визначив роль програмного інженера, а також вимоги до його досвіду й освіти. Він, звичайно ж, повинен бути добрим програмістом, упевнено розбиратися в структурах даних і алгоритмах, вільно володіти однією або більшою кількістю мов програмування. Це його, так би мовити,

«програма-мінімум», що грубо визначається як уміння написати порівняно невелику програму цілком або частину складної програмної системи. Але є і «програма-максимум», що вимагає від програмного інженера ще більшого.

Програмний інженер повинен бути знайомий з кількома способами проектування, знати, як перевести розпливчасті вимоги і побажання замовника в чітке технічне завдання, й уміти розмовляти з користувачем системи в термінах конкретної сфери роботи, а не на «програмному сленгові». Такі здібності вимагають, у свою чергу, гнучкості і відвертості, щоб схопити суть наочної області різних додатків і стати в ній фахівцем. Програмний інженер повинен володіти можливістю переходити від одного рівня абстракції до іншого на різних стадіях проекту: від особливих процедур і вимог додатку до абстракцій програмної системи, до специфіки дизайну системи і, нарешті, до рівня детального кодування.

Як і в будь-якій інженерній галузі, програмний інженер повинен розвивати вміння, що дозволяють побудувати цілий набір моделей, і оцінити ці моделі, управляючи вибором компромісів, що виникають у процесі розробки програмного забезпечення. Різні моделі використовуються на етапі визначення вимог до проектованої системи, в розробці архітектури програмного забезпечення і на стадії реалізації проекту. На деяких стадіях модель може бути використана для відповіді на питання про поведінку системи та її продуктивність.

Програмний інженер – це член команди, тому повинен володіти навичками спілкування і міжособистісних відносин, а також уміти планувати не тільки свою роботу, але і роботу інших.

Як уже згадувалося, програмний інженер відповідає за цілий спектр проблем. Часто в організаціях обов'язки розподіляють між кількома фахівцями, що мають різні посади. Наприклад, просто аналітик відповідає за формулювання вимог, взаємодію із замовником і вивчення конкретної сфери роботи додатка, а аналітик з ефективності – за аналіз продуктивності системи. Іноді один і той самий інженер відіграє різні ролі на різних етапах проекту або в різних проектах.

У чому ж відмінності програмної інженерії від напрямку комп'ютерних наук та яка особливість програмної інженерії в порівнянні з іншими інженерними галузями?

Розглянемо першу частину питання [4]. Комп'ютерна наука займається теорією і методами обчислювальних і програмних систем, тоді як програмна інженерія займається практичними проблемами створення програмного забезпечення. Комп'ютерна наука складає теоретичні основи програмної інженерії, отже, інженер з програмного забезпечення повинен знати основні поняття комп'ютерних наук. Так само, як інженер з електроніки повинен знати фізику. У ідеалі, програмна інженерія має бути підтримана якимись теоріями інформатики, але це не завжди так. Програмні інженери часто використовують прийоми, які прийнятні до застосування тільки в конкретних умовах і не можуть бути узагальнені на інші випадки, а елегантні теорії комп'ютерних наук та інформатики не завжди можуть бути застосовані до реальних великих систем.

І нарешті, комп'ютерні науки та інформатика – це не єдиний теоретичний фундамент програмної інженерії, оскільки коло проблем, що стоять перед програмним інженером, значно ширше, ніж просто написання програм. Це ще управління фінансами, організація робіт у колективі, взаємодія із замовником тощо. Розв'язання цих проблем вимагає фундаментальних знань, що виходять за рамки інформатики та комп'ютерних наук.

Щодо особливостей програмної інженерії порівняно з іншими інженерними галузями слід зазначити таке. Перш за все, життєвий цикл продукту будь-якої інженерії у спрощеному вигляді включає фази проектування, створення зразка, випробування, виробництва, експлуатації.

Комп'ютерна програма – це (на відміну від об'єктів інших інженерій) не матеріальний об'єкт. Звідси випливають певні відмінності. Фаза виробництва полягає в копіюванні зразка на інші носії. Вартість фази незначна. Якщо кодування вважати елементом проектування (що дуже близько до істини), то відсутня також і фаза створення зразка (будується компілятором). Тому вартість програми – це вартість лише її проектування.

Друга істотна відмінність полягає в тому, що програма – штучний об'єкт. Тобто для програми немає об'єктивних законів, яким би підкорялася її поведінка. Наприклад, у інженера-будівельника є об'єктивні закони будівельної механіки: рівноваги моментів і сил, стійкості механічних систем тощо. Інженер-будівельник може перевірити свої архітектурні рішення на відповідність цим законам і тим самим забезпечити успіх проекту. Ці закони об'єктивні, вони діятимуть завжди. У програмного інженера, на перший погляд, також є типові, перевірені часом архітектурні рішення (наприклад, клієнт-серверна архітектура). Але ці рішення визначаються рівнем розвитку обчислювальної техніки (і адекватним рівнем вимог). Із появою техніки з принципово новими можливостями програмному інженерові доведеться шукати нові рішення.

Прямим наслідком відсутності можливості «теоретичного» контролю проекту є те, що тестування продукту – це єдиний спосіб переконатися в його якості. Саме тому вартість тестування складає істотну вартість програмного забезпечення. До речі, будівельний інженер, як правило, позбавлений можливості такого «тестування» свого продукту перед здачею його в експлуатацію.

Ну і нарешті, програмна інженерія – молода дисципліна, досвід якої налічує всього кілька десятків років. У порівнянні з досвідом будівельної інженерії (тисячоліття) це дуже мало. Програмну інженерію іноді порівнюють з ранньою будівельною, коли закони будівельної механіки ще не були відомі й будівельні інженери діяли методом проб і помилок, накопичуючи безцінний досвід. Попри молодий вік, програмна інженерія також накопичила певний досвід, який дозволяє (при розумному його застосуванні) робити вдалі проекти. Цей досвід виражений в основних поняттях, принципах, моделях і методах програмної інженерії.

2.2. Характеристика програмного забезпечення як головного результату діяльності програмних інженерів

Основні поняття програмної інженерії це: дані та їхні структури (прості і складні), функції і композиції, базові об'єкти (модуль, об'єкт, компонент, каркас, контейнер, повторно використовуваний компонент тощо) і цільові об'єкти, що будуються [10, 11].

Цільовим об'єктом програмної інженерії є програмне забезпечення.

Програмне забезпечення – це набір комп'ютерних програм, процедур і даних та пов'язаної з ними документації (ISO/IEC 12207). Погляд на програмне забезпечення як тільки на програму, що «сидить у комп'ютері», дуже вузький. Справа в тому, що продається (постачається) не тільки програма, але ще і документація, в якій можна прочитати, як встановити програму і як нею користуватися, а також дані для установки програми в різних умовах (конфігураційні файли). Тому програмне забезпечення іноді називають програмним продуктом. Тобто програмний продукт (програмне забезпечення) – це не тільки програми, а також уся пов'язана з ними документація і конфігураційні дані, необхідні для коректної роботи програми. А фахівці з програмного забезпечення розробляють програмні продукти, тобто таке програмне забезпечення, яке може бути продане споживачеві.

У залежності від того, для кого розробляються програмні продукти (конкретного замовника або ринку), програмні продукти бувають двох типів [10]:

- «коробочні» продукти (generic products – загальні продукти або shrink-wrapped software – упаковані програмні продукти), прикладами яких можуть служити системи керування базами даних, текстові процесори, графічні пакети, засоби керування проектами тощо;

- замовні продукти (bespoke – виготовлені на замовлення або customized products – налаштовані програмні продукти), прикладами яких можуть бути системи керування для електронних пристроїв, системи підтримки різноманітних

виробничих або бізнес-процесів, системи керування повітряним транспортом тощо.

Важлива різниця між ними полягає в тому, хто ставить задачу розробки продукту (визначає та специфікує вимоги). У першому випадку це роблять самі розробники на основі аналізу ринку (маркетингу) – і при цьому ризикують самі. У другому – замовник, і при цьому ризикує, що розробник не зможе реально виконати всі вимоги в строк і при виділеному бюджеті.

З чого ж складається вартість програмних продуктів? Перш за все, структура вартості істотно залежить від типу програмного забезпечення, методів його розробки та методів оцінки. Так, багато спеціалістів відзначають високу частку вартості етапу супроводу. Для деяких типів програмних продуктів вона може складати 60 і більше відсотків від загальної вартості. Тим часом, етап супроводу включає виконання двох видів робіт: виправлення помилок у програмі (невідповідностей початковим вимогам) і внесення змін у програму (додавання нових вимог). При іншому підході до оцінки можна вважати, що етап супроводу не варто оцінювати окремо, оскільки виправлення помилок можна віднести до продовження тестування, а внесення змін – до нового проекту.

Типовий розподіл вартості між основними етапами (без супроводу) виглядає так [10]:

- 15% – специфікація – формулювання вимог і умов розробки;
- 25% – проектування – розробка і верифікація проекту;
- 20% – розробка – кодування і тестування компонент;
- 40% – інтеграція і тестування – об'єднання і тестування продукту в зібраному вигляді.

Відхилення від цієї схеми залежно від типу програмного забезпечення виглядає ось як. Для загальних продуктів характерна більш висока частка тестування за рахунок скорочення насамперед частки специфікації (до 5%). Розподіл вартості замовних продуктів залежить від їх складності. Для складного програмного забезпечення також зростає частка інтеграції та тестування, але за рахунок скорочення частки проектування та розробки. Частка специфікації може зростати.

Скорочення частки проектування та розробки досягається за рахунок використання апробованих проектних рішень та повторного використання готових компонент.

Застосування апробованих рішень і готових компонент при створенні загальних продуктів дозволяє підвищити якість та скоротити строки розробки.

Розглянемо питання властивостей якісного програмного продукту. Перш за все якісна програма повинна робити те, що очікує від неї замовник, тобто задовольняти вимоги замовника. Такі вимоги називають функціональними. Але, крім функціональних вимог, існує ще ряд загальних характеристик, якими в тією чи іншою мірою повинна володіти кожна програма. Ці характеристики прийнято називати нефункціональними вимогами. До нефункціональних вимог відносять [10]:

1. Супроводжуваність (maintainability). Супроводжуваність означає, що програма має бути написана з розрахунком на подальший розвиток. Це критична властивість системи, оскільки зміни програмного забезпечення неминучі внаслідок зміни бізнес-середовища. Супровід програми виконують, як правило, не ті люди, які її розробляли. Супроводжуваність включає такі елементи, як наявність і зрозумілість проектної документації, відповідність проектної документації початковому коду, зрозумілість початкового коду, простота змін початкового коду, простота додавання нових функцій.

2. Надійність (dependability). Надійність програмного забезпечення включає такі елементи, як:

• відмовостійкість – можливість відновлення програми і даних у разі збоїв у роботі;

• безпека – збої в роботі програми не повинні призводити до небезпечних наслідків (аварій);

• захищеність від випадкових або навмисних зовнішніх дій (захист від неадекватних дій користувача, вірусів, спаму тощо).

3. Ефективність (efficiency). Програмне забезпечення не повинно даремно витрачати системні ресурси, такі як пам’ять, процесорний час, канали зв’язку. Тому ефективність програмного забезпечення оцінюється такими показниками: час виконання

коду, завантаженість процесора, обсяг необхідної пам'яті, час відгуку і тому подібне.

4. Зручність використання (usability). Програмне забезпечення повинно бути легким у використанні, причому саме тим типом користувачів, на яких воно розраховане. Це включає інтерфейс користувача й адекватну документацію. Причому інтерфейс має бути не інтуїтивно, а професійно зрозумілим користувачеві.

Слід зазначити, що реалізація нефункціональних вимог часто вимагає більших витрат, ніж реалізація функціональних. Так, супроводжуваність потребує значних зусиль з підтримки відповідності проекту вихідному коду і застосування спеціальних методів створення програм, які передбачають модифікації, надійність – додаткових засобів відновлення системи при збоях, ефективність – пошуку спеціальних архітектурних рішень і оптимізації коду, а зручність – проектування не «інтуїтивно» зрозумілого, а професійно зрозумілого інтерфейсу користувача.

Труднощів при створенні програмного забезпечення достатньо багато. Усі вони так чи інакше пов'язані з головною проблемою програмної інженерії – пошуком універсального методу і процесу, придатного для створення програмного продукту будь-якого типу в будь-яких умовах. Тут головна проблема – змінні умови. У зв'язку з цим розробники програмного забезпечення зіштовхуються з такими труднощами [10]:

1. Спадкоємство раніше створеного програмного забезпечення (legacy systems). Існує достатньо багато систем, створених багато років тому, морально застарілих, але які продовжують працювати. Проблема спадкоємства таких систем полягає в їх супроводі – підтримці та розвитку.

2. Різнорідність програмних систем. Програмне забезпечення повинно працювати в розподілених мережах, на різнорідному устаткуванні, в різних середовищах, під управлінням різних операційних систем.

3. Скорочення часу на розробку. Згідно із запитами ринку вимоги до програмних систем змінюються дуже швидко. Суть проблеми полягає в тому, щоб скоротити час розробки програмного забезпечення без зниження його якості.

Ці труднощі часто пов'язані між собою. Завдання розробки мережного варіанта старої локальної бази даних в обмежені терміни досить типове, а універсального методу реалізації цього завдання на сьогоднішній день немає.

2.3. Огляд методів і моделей програмної інженерії

Розглянемо існуючі методи програмної інженерії [10, 11]. Загалом метод програмної інженерії – це структурний підхід до створення програмного забезпечення, який сприяє виробництву високоякісного продукту ефективним в економічному аспекті способом. У цьому визначенні є дві основні складові: по-перше, створення високоякісного продукту, а по-друге, економічно ефективним способом. Іншими словами, метод – це те, що забезпечує розв'язання основної задачі програмної інженерії: створення якісного продукту при заданих ресурсах часу, бюджету, устаткування, людей.

Починаючи з 1970-х років створено достатньо багато методів розробки програмного забезпечення. Найбільш відомі [11]:

- метод структурного аналізу і проектування (Том де Марко, 1978);
- метод сутність-зв'язок проектування інформаційних систем (Пітер Чен, 1976);
- метод об'єктно-орієнтованого аналізу (Граді Буч, 1994; Джеймс Рамбо, 1991).

Методи програмної індустрії засновані на ідеї створення моделей програмного забезпечення з поетапним перетворенням цих моделей у програму – остаточну модель розв'язуваної задачі. Так, на етапі специфікацій створюється модель – опис вимог, яка далі перетвориться в модель проекту програмного забезпечення, проект – у програмний код. При цьому важливо, щоб моделі методу зображувалися графічно за допомогою визначеної мови представлення моделей.

Методи повинні включати такі компоненти [11]:

- опис моделей системи і нотація, використовувана для опису цих моделей (наприклад, об'єктні моделі, кінцево-автоматні моделі тощо);

- правила і обмеження, які треба виконувати і враховувати при розробці моделей (наприклад, кожний об'єкт повинен мати однакове ім'я);

- рекомендації – евристики, що характеризують оптимальні прийоми проектування в даному методі (скажімо, рекомендація про те, що в кожного об'єкта не має бути більше семи підоб'єктів);

- керівництво по застосуванню методу – опис послідовності робіт (дій), які треба виконати для побудови моделей (усі атрибути мають бути задокументовані до визначення операцій, пов'язаних з цим об'єктом).

Немає ідеальних методів, усі вони застосовні тільки в тих або інших випадках. Немає абсолютних методів – вживані на практиці методи можуть включати елементи різних підходів. Вибір найбільш підходящого в конкретному випадку методу є однією із задач фахівця з програмної інженерії.

Цілком очевидно, що всі методи розробки програмного забезпечення реалізуються з використанням комп'ютерної техніки. Такий підхід прийнято називати CASE-технологіями.

CASE – Computer Aided System Engineering – різного роду інструментальні програми, що використовуються для підтримки процесу створення програм. CASE-засоби можуть бути класифіковані за кількома ознаками [16]:

• За рівнем застосування:

- Upper CASE – засоби аналізу вимог;
- Middle CASE – засоби проектування;
- Low CASE – засоби розробки додатків.

• Спеціалізовані:

- засоби проектування баз даних;
- засоби реінжинірингу (відновлення) моделі (наприклад, формування діаграм на основі аналізу програмних кодів).

• Допоміжні:

- планування і управління проектом;
- конфігураційного управління;
- тестування.

• Інтегровані CASE-засоби, які охоплюють усі етапи і процеси створення програмного забезпечення, від аналізу вимог до тестування й випуску документації. Інтегровані CASE-засоби виступають, як правило, у вигляді набору узгоджених за інтерфейсом засобів, призначених для підтримки окремих етапів процесу.

На даний час існує дуже багато CASE-засобів і фірм-виробників, що спеціалізуються на їх розробці. При виборі CASE-засобів слід керуватися основним принципом: спочатку метод створення програмного забезпечення, а потім – CASE-засоби, застосовні для цього методу. Зворотний вибір може призвести до поганих наслідків.

Стосовно процесів створення програмного забезпечення, то одним з основних понять програмної інженерії є поняття життєвого циклу програмного продукту і програмного процесу.

Життєвий цикл – безперервний процес, що починається з моменту ухвалення рішення про створення програмного забезпечення і закінчується зняттям його з експлуатації. Життєвий цикл розбивається на окремі процеси.

Процес – сукупність дій і завдань, що мають на меті досягнення значущого результату. Основними процесами (іноді називають етапами або фазами) життєвого циклу є [7]:

1. Розробка специфікації вимог (результат – описи вимог до програми, які обов'язкові для виконання, – опис того, що програма повинна робити).

2. Розробка проекту програми (результат – опис того, як програма працюватиме).

3. Кодування (результат – початковий код і файли конфігурації).

4. Тестування програми (результат – контроль відповідності програми вимогам).

5. Документування (результат – документація до програми).

Окрім основних, існує багато додаткових і допоміжних процесів, пов'язаних не зі створенням продукту, а з організацією робіт (нефункціональні процеси): створення інфраструктури,

управління конфігурацією, управління якістю, навчання, розв'язання протиріч.

Повне встановлення процесу передбачає [7]:

• опис процесу – детальний опис дій і операцій процесу;

• навчання процесу – проведення занять з персоналом з освоєння процесу;

• уведення метрик – встановлення кількісних показників ходу виконання;

• контроль виконання – вимірювання метричних показників і оцінка ходу виконання;

• удосконалення – зміна процесу за змінних умов застосування.

Застосування повних (важких) процесів вимагає додаткових ресурсів (часто істотних) і далеко не завжди окупається отриманим результатом. Тому вибір складу процесів у кожному конкретному випадку може бути зроблений по-різному відповідно до вибраної моделі процесу.

Модель програмного процесу – це спрощений опис програмного процесу, представлений з деякої точки зору [7]. Модель задається у вигляді практичних етапів, необхідних для створення програмного забезпечення. Модель визначає, що і як робити. Тобто які процеси, з яким ступенем конкретизації і яка послідовність виконання. Вибір моделі процесу є першим кроком при створенні програмного забезпечення. Правильний вибір моделі дуже важливий, оскільки багато в чому визначає успіх проекту. Вибір важких процесів може «потопити» проект, а занадто легковажне ставлення до процесів – спричинити втрату контролю над ходом виконання.

Відповідно до двох типів процесів – основних і додаткових – можна говорити про два типи моделей процесу: моделі процесу розробки (моделі життєвого циклу) і моделі організації робіт щодо виконання розробки.

До перших типів моделей (моделі життєвого циклу) належать моделі, в яких описується порядок виконання дій зі створення продукту. До найбільш відомих моделей відносять такі [22–25]:

1. Модель водоспаду (каскадна) – процес розбивається на послідовне виконання стадій; кожна стадія починається після повного завершення попередньої, продукт створюється завершенням останньої стадії і повинен повністю відповідати встановленим на початку вимогам.

2. Спіральна (циклічна) модель – процес також розбивається на стадії, але стадії виконуються циклічним повторенням. На першому циклі створюється прототип продукту, що виконує частину вимог. Подальші цикли пов'язані з нарощуванням прототипу до повного задоволення вимог.

3. Компонентна модель передбачає збірку продукту із заздалегідь написаних частин – компонент. Основний наголос робиться на інтеграцію і сумісне тестування компонент.

4. Формальна модель заснована на формальному описі вимог з подальшим перетворенням (трансляцією) вимог у вихідний код. Застосування формальної моделі гарантує відповідність коду описаним вимогам.

Відмінності між цими моделями існують тільки в теорії. На практиці спіральна модель може бути доповнена елементами каскадної і компонентної. Завдання програмного інженера – підібрати правильну їх комбінацію, орієнтуючись тільки на кінцевий результат.

До другого типу моделей, моделей організації робіт, належать:

1. Модель потоку робіт (workflow model) – показує послідовність дій, що виконуються людьми на різних етапах розробки програмного забезпечення. Для кожної дії вказують входи, виходи (результати) і зв'язки по входах і виходах.

2. Модель потоків даних (dataflow model) – зображує процес у вигляді послідовного перетворення даних. Кожне перетворення може виконуватися однією або кількома діями.

3. Ролева модель (role model) – показує ролі людей, які беруть участь у програмному процесі, а також дії і результати, за які вони відповідають.

2.4. Інфраструктурна модель компонентів професії Форда–Гіббса. Аналіз зрілості компонентів професії інженерії програмного забезпечення

Окрім моделей процесів розробки програмного забезпечення, які використовують різноманітні компанії, не менш важлива модель, яка характеризує коло питань, пов'язаних зі співробітниками цих компаній – професійних програмних інженерів. Основна мета побудови та аналізу такої моделі – відповідь на питання, чи є програмна інженерія професією [4, 6].

З урахуванням величезного розриву між кращими і гіршими компаніями, що займаються розробкою програмного забезпечення, завдання сьогоднішнього дня полягає не стільки в тому, щоб розвинути передові методики, скільки в тому, щоб підтягнути до них середній рівень розробки програмного забезпечення. Традиційний спосіб підвищення середнього рівня в галузі, що безпосередньо зачіпає життєдіяльність суспільства, полягає у створенні офіційної професії.

Хоча саме слово «професія» часто вживається недоречно, поняття, що стоїть за ним, має давно усталене юридичне значення. Згідно зі стандартними визначеннями професійної діяльності, особу, яка виконує «професійні обов'язки», відрізняють кілька характеристик.

Як правило, професійна робота вимагає глибоких знань в якій-небудь галузі науки, що отримуються в процесі тривалого спеціального навчання і підготовки. Але тут слід підкреслити відмінності між професійною підготовкою і загальною науковою освітою, а також підготовкою до роботи згідно зі стандартними процедурами, розумовою або фізичною працею. Крім того, професійна робота може бути творчою і художньо-образотворчою. Це залежить головно від винахідливості, уяви або таланту виконавця.

Професійна робота вимагає послідовної реалізації власних думок і права вибору. Їй властивий інтелектуальний характер і зміна форм. Тобто слід відрізняти професійну діяльність від рутинної розумової, ручної або фізичної праці.

Багато розробників програмного забезпечення бачать у наведеному визначенні професії характеристики своєї власної роботи. Ця робота, поза сумнівом, вимагає глибоких знань (або принаймні докладних технічних відомостей), і її можливості пов'язані зі спеціальним навчанням і підготовкою. Розробка програмного забезпечення містить значний елемент творчості і, звичайно, вимагає застосування думок і право вибору. Таким чином, робота, що виконується розробниками програмного забезпечення, відповідає класичному визначенню «професійної роботи».

Крім цього, реальна професійна практика інженерної діяльності наповнює визначення професії дещо іншим змістом. Згідно з ним, професія – це [4]:

• Глибокі пізнання і серйозна підготовка.

• Кодекс поведінки, що встановлює вищі етичні стандарти, ніж ті, які зазвичай допускаються в інших сферах життєдіяльності.

• Система дисциплінарних покарань для професіоналів, що порушують цей кодекс.

• Висока соціальна відповідальність у порівнянні з особистою вигодою і відповідний обов'язок дотримуватися професійної етики.

• Необхідність отримання ліцензії або сертифікату перед початком практичної діяльності.

Наскільки ж відповідає цим критеріям розробка програмного забезпечення? Перед тим, як дати відповідь на це питання, охарактеризуємо модель професії.

Усі професіонали повинні пройти інтенсивний курс спеціалізованого навчання – отримати початкову професійну освіту. Але перш ніж приступити до виконання професійних обов'язків необхідно не тільки володіти знаннями, набутими на етапі початкової освіти. Досить важливим фактором є надбання та розвиток початкових навичок професійної діяльності. Професіоналізм передбачає також активний професійний розвиток, тобто додаткове навчання, що має місце під час професійної практики.

Будь-яка професія вимагає наявності методів оцінки і гарантії адекватності освіти та компетентності окремих професіоналів. Загальні форми цих компонентів – акредитація професійних учбових програм і видача свідоцтв (сертифікатів або ліцензій) окремих професіоналів.

Професії повинні мати гарантії, що знання і навички окремих професіоналів та їх практична діяльність є соціально відповідальні. Існує загальна форма цього професійного компонента – морально-етичний кодекс. Іноді цей компонент можна назвати процесуальним кодексом або нормами поведінки.

Нарешті, будь-яка професія має самоідентичність, тобто професіонали бачать себе як частину співтовариства однаково мислячих індивідуумів, які піклуються про якість їх професійної практики. Ця ідентичність зазвичай проявляється у формі професійного суспільства (асоціації або співтовариства). Більшість професіоналів належать до певної асоціації, яка, здебільшого через добровільні зусилля, просуває розвиток і взаємодію інших компонентів професії.

Таким чином, модель професії можна зобразити у вигляді двох рівнів: рівня практикуючого фахівця та рівня інфраструктури.

Таблиця 2.4

РІВНІ ТА КОМПОНЕНТИ МОДЕЛІ ПРОФЕСІЇ

Рівень практикуючого фахівця	**Рівень інфраструктури**
Професіоналізм Знання Професійна практика	Початкова професійна освіта Акредитація Розвиток навичок Сертифікація Ліцензування Професійний розвиток Етичний кодекс Професійні суспільства

Гері Форд (Gary Ford) і Норман Е. Гіббс (Norman E. Gibbs) з Інституту програмної інженерії (SEI – Software Engineering

Institute) проаналізували елементи сформованої «зрілої» професії відносно програмної інженерії [6].

Зобразимо інфраструктурну модель компонентів професії Форда–Гіббса, яка окреслює типовий шлях для людини, що прагне бути професіоналом. На рис. 2.2 блоки представляють діяльність та організаційні компоненти, а пунктирні еліпси – компоненти гарантії якості.

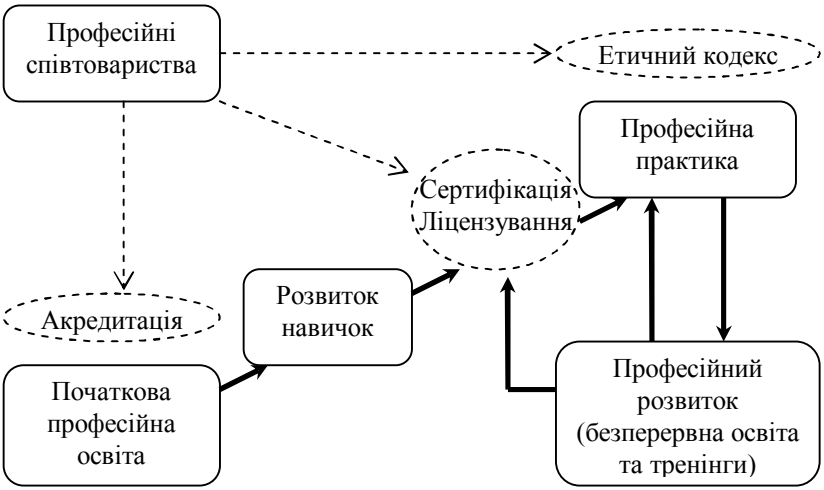

Рис. 2.2. Багаторівнева інфраструктурна модель компонентів професії (модель Форда–Гіббса, 1996, SEI) [6]

Акредитація гарантує якість початкової професійної освіти, яка забезпечується університетами. Далі, щоб стати професіоналом, людина повинна розвивати навички в додаток до початкової освіти (через університетські програми кооперації з компаніями, навчання за місцем роботи, стажування або інші засоби). Сертифікація і / або ліцензування гарантує компетентність людини під час професійної практики. Крім того, можливі періоди професійного розвитку, які приводять до необхідності повторної видачі свідоцтва або повторного ліцензування. Професія гарантує, що практикуючі фахівці

перебувають у відповідальній формі, дотримуючись морально-етичного кодексу. Професійні співтовариства допомагають гарантувати, що решта компонентів взаємодіють адекватно.

Взаємодії серед компонентів професії фактично значно складніші, ніж наведені в моделі Форда–Гіббса. Наприклад, вимоги до акредитації можуть справити істотній ефект на етапі початкової професійної освіти. Професійне співтовариство може керувати процесом видачі сертифікатів або процесом ліцензування, а також розвивати етичний кодекс. Директиви видачі сертифікатів або ліцензування можуть впливати на зміст професійного розвитку. Рис. 2.3 показує деякі з цих взаємодій. Основний цільовий ефект взаємодії компонентів професії – поліпшення якості професійної практики інженерів [6].

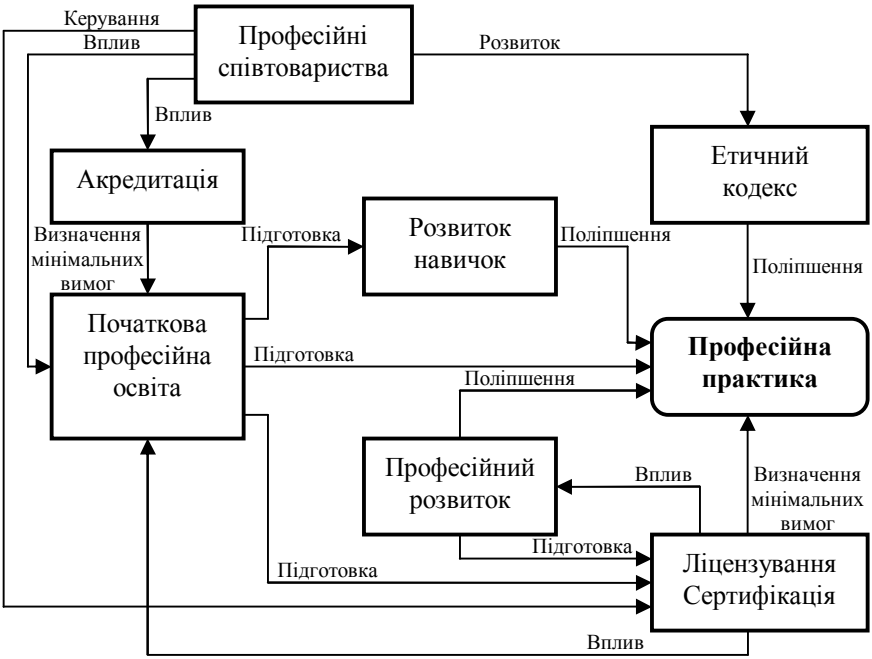

Рис. 2.3. Приклад взаємодії компонентів професії [6]

Охарактеризуємо компоненти професії [4, 6].

Початкова професійна освіта. Професійні працівники зазвичай починають професійну кар'єру з проходження університетської програми у вибраній ними галузі – в медичному, інженерному, юридичному вузі тощо.

Акредитація (атестація). Атестація університетських програм здійснюється органами нагляду, які визначають ступінь відповідності програм цілям освіти. Цим забезпечується наявність необхідної підготовки у випускників вузів, що дає можливість почати професійну діяльність після його закінчення. Рада з акредитації програм у галузі техніки і технології (Accreditaton Board for Engineering and Technology – ABET) здійснює нагляд за програмами технічних дисциплін у США. Аналогічна організація – Інженерна акредитаційна рада (Canadian Engineering Accreditation Board – CEAB) – діє в Канаді.

Розвиток практичних навичок і прийомів. У більшості професій для оволодіння професійними навичками однієї лише освіти недостатньо. Професіоналам-початківцям необхідна практика застосування своїх знань, перш ніж взяти на себе відповідальність за виконання робіт у вибраній сфері. У США для лікарів передбачено три роки інтернатури. Бухгалтери повинні один рік відпрацювати в затвердженій спеціальним комітетом організації, щоб одержати ліцензію. Професійні інженери повинні мати принаймні 4 роки досвіду практичної роботи. Обов'язковий період навчання та практики гарантує, що професією володіють люди, що мають досвід виконання роботи на необхідному рівні компетентності.

Сертифікація. Завершивши освіту і розвинувши практичні навички, професіонал повинен скласти один або кілька іспитів, щоб довести, що він оволодів необхідним мінімумом знань. У США лікарі складають іспити у відповідних професійних радах. Бухгалтери складають іспит на звання (Certified Public Accountant – CPA). Інженери спочатку складають іспит з основ інженерії після закінчення коледжу, а потім, приблизно через чотири роки, – іспит за фахом. У деяких професіях потрібно підтверджувати сертифікат через певні періоди часу.

Ліцензування. Ліцензування аналогічно сертифікації, за тим винятком, що воно має обов'язковий характер і здійснюється державними органами.

Професійний розвиток. Від професіоналів вимагається актуалізувати свою спеціальну освіту. Безперервна професійна освіта підтримує на необхідному рівні або поліпшує знання й уміння працівників, після того як вони приступили до професійної діяльності. Вимога професійного розвитку особливо сильна в тих спеціальностях, де область технічних знань схильна до швидких змін. Тут пальму першості утримує медицина – через постійне вдосконалення лікарських препаратів, терапії, медичного устаткування, методів діагностики й лікування. Вимога професійного розвитку допомагає забезпечити збереження мінімально необхідного рівня компетенції впродовж всієї кар'єри.

Професійні співтовариства. Професіонали вважають себе членами співтовариств, що складаються з подібних їм індивідуумів, які ставлять професійні стандарти вище за особисті інтереси або інтереси працедавців. Спочатку професійні співтовариства сприяють обміну досвідом і знаннями, але з часом їх функції розширюються і включають визначення критеріїв сертифікації, управління її програмами, формулювання стандартів атестації і етичних кодексів, а також заходів дисциплінарної дії на порушників цих кодексів.

Кодекси поведінки. У кожній професії існує етичний кодекс, який регулює поведінку людей, що належать до даного співтовариства професіоналів. У кодексі не просто вказується, що можна робити, а що ні. За його порушення професіонали можуть бути виключені зі своїх професійних співтовариств або позбавлені ліцензії на професійну діяльність. Дотримання визнаного етичного кодексу поведінки допомагає професіоналам відчувати свою належність до поважаного співтовариства, а заходи із забезпечення етичних стандартів допомагають підтримувати мінімально прийнятний рівень поведінки.

Окрім восьми характерних елементів професії, виділених Фордом і Гіббсом, у багатьох професіях з'являється дев'ятий,

який швидше стосується цілих організацій, ніж окремих працівників.

Сертифікація організацій. У багатьох галузях ліцензію повинні одержувати не тільки окремі професіонали, їх організації теж повинні бути сертифіковані. Бухгалтерські фірми проходять перевірки такими ж організаціями. Лікарні й університети теж атестуються. У таких складних сферах діяльності, як бухгалтерський облік, освіта і медицина, сертифікація організацій гарантує належний рівень виконання роботи, який не може бути досягнутий за рахунок компетентності лише індивідуумів. Якості, що характеризують саму організацію, можуть впливати на цей рівень так само сильно, як і характеристики окремих працівників.

Форд і Гіббс указують, що багато непрофесійних видів діяльності володіють певною кількістю перерахованих елементів-характеристик. Наприклад, у штаті Каліфорнія (США) потрібно мати ліцензії оббивальникам, боксерам-любителям, приватним детективам і жокеям у перегонах на мулах, але ці види діяльності не вимагають більшості інших елементів професійної роботи. Всі звичайні професії володіють майже всіма описаними елементами.

Для кожного з елементів, що формують професію, Форд і Гіббс також визначили кілька рівнів зрілості [6]:

- «*Відсутність*». Даний елемент професії просто не існує;

- «*За потребою*». Елемент існує, але тільки в окремих невзаємопов'язаних випадках;

- «*Сформований*». Елемент існує і чітко визначається в конкретній (специфічній) професії;

- «*Зрілий*». Елемент існує протягом тривалого часу, активно підтримується й удосконалюється яким-небудь професійним органом.

Зрілість професії означає, що її елементи досягли необхідної стадії. Зрозуміло, поняття зрілості постійно змінюється. Деякі елементи, що здавалися зрілими років 30 тому, сьогодні вже не видаються такими, а нинішня зрілість інших не визнаватиметься через кілька десятиліть.

У табл. 2.5 описується зрілість професії інженерії програмного забезпечення. В основному професія сформувалася, але деякі елементи відстали, а деякі просунулися до стадії зрілості [4].

Таблиця 2.5

ЗРІЛІСТЬ ЕЛЕМЕНТІВ ПРОФЕСІЇ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Елемент	**Поточний стан**
Початкова професійна освіта	У проміжку між «*за потребою*» і «*сформований*». Ступінь бакалавра комп'ютерної науки, електротехніки, математики тощо, складає звичайну підготовку для вступу до професії. Є десятки програм для отримання ступеня магістра інженерії програмного забезпечення. Останніми роками ініційовані численні програми для студентів старших курсів, які підтримуються професійними компаніями
Акредитація (Атестація)	«*Сформований*». Вказівки щодо атестації з 2004 року формулюються робочою групою від IEEE Computer Society і ACM, але ще не реалізовані
Розвиток практичних навичок і умінь	«*Сформований*». Розроблені методичні вказівки по навиках і уміннях, якими повинен володіти інженер програмного забезпечення, щоб почати займатися цією професією
Сертифікація	«*Сформований*». Комерційні постачальники програмного забезпечення, такі як Microsoft, Novell і Oracle, вже багато років пропонують програми сертифікації, пов'язані з їхніми технологіями. З 2002 р. IEEE Computer Society пропонує «Свідоцтво сертифікованого професійного розробника» як підтвердження загального статусу в інженерії програмного забезпечення
Ліцензування	«*За потребою*». Реалізовано в окремих випадках. Наприклад, у штаті Техас (США) професійні розробники ліцензуються згідно з

Продовження табл. 2.5

	положенням, прийнятим в 1998 р. У провінціях Британська Колумбія і Онтаріо (Канада) реєстрація професійних інженерів з розробки програмного забезпечення почалася з 1999 р.
Професійний розвиток	«*За потребою*», з тенденцією наближення до «*сформованого*». Деякі організації опублікували методичні вказівки з професійної розробки програмного забезпечення. (Див. основні напрями неперервної освіти IEEE Computer Society на сайті: *www.computer.org/certification*)
Професійні товариства	«*Сформований*», з тенденцією наближення до «*зрілості*». Існують IEEE Computer Society, ACM (Association for Computer Machinery) й інші професійні асоціації. Ці співтовариства прямо указують, що представляють інженерію програмного забезпечення. Весь комплекс продуктів і послуг, необхідних для підтримки інженерів-програмістів як професіоналів, ці співтовариства не пропонують. Вони не можуть вживати дисциплінарні заходи до порушників етичного кодексу у сфері програмної інженерії
Кодекс етичної поведінки	«*Сформований*». ACM і IEEE Computer Society прийняли кодекс етичної поведінки для інженерів з програмного забезпечення. Цей документ поки не прийнятий у галузі й не одержав широкого визнання.
Сертифікація організацій	«*Сформований*», з тенденцією наближення до «*зрілості*». Інститут програмної інженерії (SEI) сформулював модель зрілості в розробці програмного забезпечення, яку він активно просуває і удосконалює. З 1987 р. модель використовувалася для оцінки багатьох організацій, але повсюдно поки не застосовується. Сертифікація згідно з ISO 9000-9004 широко визнана, особливо в Європі

Інженерія програмного забезпечення поки не повністю відповідає визначенню професії [4]. Ще тільки формується система відповідної початкової освіти. Сертифікація набула поширення лише з 2002 р. Ліцензування доступне лише крихітній частці зайнятих у розробці програмних продуктів. Етичний кодекс існує, але дотримання його положень не забезпечується застосуванням дисциплінарних заходів. Ведеться велика робота щодо прискорення просування програмної інженерії до стадій «сформований» і «зрілий».

Застосувавши науковий метод Френсіса Бекона до програмної інженерії, можна визначити три кроки, які необхідно зробити, щоб допомогти інженерії вийти на рівень зрілості [4]:

Звільнити мислення від забобонів. Галузь програмної інженерії повинна відмовитися від згубної пристрасті до розробки продуктів за принципом «написати і виправити», про який давно відомо, що він нікому не вигідний.

Систематично накопичувати спостереження і досвід. У фірмах почали систематизувати дані щодо ефективності практичних методик розробки програмного забезпечення й оцінювати, які з них приносять найбільший успіх. Причому деякі фірми досягли вражаючих результатів. Інші повинні наслідувати їх приклад.

Зупинитися, подивитися, що одержано, і зробити висновки.

Сфера розробки програмного забезпечення підійшла до одного з поворотних пунктів. Можна залишитися в зручному маленькому світі програмування за принципом «напишемо і виправимо», не намагаючись вирватися за межі сьогодення і не прагнучи добитися величезних переваг, які обіцяють дослідження лідерів програмної інженерії. Але можна сміливо просунутися вперед до нової професії інженерії програмного забезпечення і почати стверджуватися у світі високої продуктивності, зниження витрат, укорочених термінів розробки і вищої якості.

У зв’язку з цим наведемо характеристику розвитку компонентів професії програмної інженерії від рівня «за потребою» до «зрілості», сформульовану Фордом і Гіббсом [6].

Таблиця 2.6

ЕВОЛЮЦІЯ КОМПОНЕНТІВ ПРОГРАМНОЇ ІНЖЕНЕРІЇ ЯК ПРОФЕСІЇ

Рівні зрілості / Компонент	За потребою	Сформованість	Зрілість
Початкова професійна освіта	Достатність отримання ступеня бакалавра комп'ютерних наук, інженерії, математики тощо як загальної підготовки для започаткування професійної діяльності	Існування визнаної форми початкової професійної освіти в галузі програмної інженерії, але відсутність стандартного навчального плану	Наявність навчального плану, який відображає найкращі практичні досягнення; має місце загальнонаціональне сприйняття моделі навчального плану; модель навчального плану регулярно переглядається та модифікується
Акредитація освіти	Акредитація базується на критеріях комп'ютерних наук або інженерії	Акредитація базується на критеріях програмної інженерії: об'єднання ABET та CSAB	Нормативи акредитації постійно переглядаються та модифікуються
Розвиток навичок	Деякі студентські проекти або програми кооперації з певними компаніями; деякі програми тренінгів, які організовують компанії для нових робітників	Виникнення специфічних вимог для розвитку навичок, необхідних у галузі програмної інженерії для початку професійної діяльності	Впровадження механізмів розвитку навичок та їх широке використання (наприклад, програми навчання та інженерних тренінгів); впроваджуються та розвиваються навички для конкретних спеціалізацій
Сертифікація	Сертифікати ICCP, ASQC; Комерційна сертифікація, пов'язана з пакетами програм та технологіями їх виготовлення	Сертифікація програмного інженера; Національне визнання стандартів сертифікації	Сертифікація у специфічних областях у рамках програмної інженерії; Національне визнання специфічних стандартів сертифікації
Ліцензування	Державне ліцензування професійного інженера, виходячи з існуючих статутів	Державне ліцензування, спрямоване на перевірку специфічних навичок програмних інженерів	Ліцензування базується на проведенні відповідних експертиз; Сумісна робота NSPE та NCEE

Продовження табл. 2.6

Професійний розвиток	Індивідуальні заняття професійним розвитком у разі необхідності	Поява нормативів професійного розвитку (навчальні плани)	Визначення кар'єрного росту для спеціалістів програмної інженерії; національне визнання навчальних планів і нормативів навчальних тренінгів
Етичний кодекс	Етичний кодекс організацій професійних інженерів ACM, IEEE, ASQC, ICCP;	Етичний кодекс спеціально для програмних інженерів	Широко використовуваний та адаптований етичний кодекс; професійні утворення мають механізми впливу на порушників кодексу
Професійні суспільства	ACM, IEEE Computer Society та інші	Поява суспільств, які репрезентують виключно програмну інженерію	Суспільства мають відповідний асортимент продуктів та сервісу для програмних інженерів

ABET – Accreditation Board for Engineering and Technology – Акредитаційна рада для інженерії та технології
ACM – Association for Computing Machinery – Асоціація з обчислювальної техніки
ASQC – American Society for Quality Control – Американська спілка з контролю якості
CSAB – Computing Sciences Accreditation Board - Акредитаційна рада комп'ютерних наук
ICCP – Institute for the Certification of Computing Professionals – Інститут сертифікації професіоналів з обчислювальної техніки
IEEE – Institute of Electrical and Electronics Engineers – Інститут інженерів з електротехніки та електроніки
NCEE – National Council of Engineering Examiners – Національна рада експертів з інженерії
NSPE – National Society of Professional Engineers – Національна спілка професійних інженерів

Контрольні запитання та завдання

1. Дайте визначення інженерії програмного забезпечення.
2. Опишіть етапи становлення інженерії програмного забезпечення як професії.
3. Дайте порівняльну характеристику напрямкам програмної інженерії та комп'ютерної інженерії.
4. Наведіть відомі вам методи та моделі програмної інженерії.
5. Опишіть взаємодію компонентів професії програмного інженера.
6. Опишіть процес розвитку основних компонентів професії програмного інженера.
7. Охарактеризуйте зрілість елементів професії інженерії програмного забезпечення на даний момент часу.

РОЗДІЛ 3. ПРОФЕСІЙНІ ТОВАРИСТВА ТА АСОЦІАЦІЇ ПРОГРАМНИХ ІНЖЕНЕРІВ. ПРОВІДНІ УСТАНОВИ В ОБЛАСТІ ПРОГРАМНОЇ ІНЖЕНЕРІЇ ТА ЇХ ДІЯЛЬНІСТЬ

Професійні асоціації, такі як комп'ютерне співтовариство IEEE Computer Society, асоціація спеціалістів з комп'ютерних технологій ACM та інші, є важливим атрибутом зрілої професії. Вони відкривають можливість професіоналам зі схожим мисленням збиратися разом, обмінюватися ідеями в особистому спілкуванні, через статті, групи за інтересами і на конференціях. Професіональні організації підтримують різноманітні структуровані способи обміну хитрощами та прийомами професії, які необхідні інженерам з програмного забезпечення, щоб працювати на високому професійному рівні [4].

3.1. Історія виникнення та розвитку співтовариств професійних інженерів

Світовий досвід об'єднання інженерів у громадські організації налічує близько 200 років. Стрімке накопичення знань, постійна поява нових великих відкриттів і винаходів у різних галузях науки сприяли розвитку інженерного співтовариства. Перша і друга промислові революції зумовили бурхливе зростання інженерної діяльності, збільшення кількості і якості підготовки інженерних кадрів, що заклало основу появи професійного науково-інженерного співтовариства і перших інженерних суспільних об'єднань [3, 4].

Інженерний рух в Європі. Одне з перших національних суспільних об'єднань інженерів у світі було створене у Великобританії в 1818 році й називалося Інститутом цивільних інженерів (Institution of Civil Engineers, ICE). Сьогодні членами ICE є більше 80000 англійських інженерів – практично всі прогресивні члени інженерного співтовариства Великобританії. Мету своєї діяльності ICE вбачає в розвитку професії як такої, сприянні професійній інженерній освіті, підтримці престижу професії і соціального статусу інженера в суспільстві, підтримці передових ідей і талановитих інженерів, забезпеченні зв'язків між

професійним співтовариством і державними органами, розвитку міжнародних контактів. Створення першого суспільного об'єднання саме у Великобританії зумовлене тим, що країна завжди надавала особливе значення підтримці науково-інженерного співтовариства, що, безумовно, сприяло розвитку будівництва, промисловості і винахідницької справи. Завдяки розвитку інженерної справи вже в 1825 році у Великобританії була побудована перша залізниця протяжністю 61 кілометр між містами Стоктон і Дарлінгтон. Розвиток промисловості вимагав великої кількості кадрів. Зріс попит на інженерів, конструкторів, винахідників, і за вагомою урядовою підтримкою стали масово створюватися інженерні школи і професійні асоціації. Мета створення професійних асоціацій полягала в розвитку інтересу до інженерних професій. Вони здійснювали самоорганізацію на принципах самоврядування і представляли інтереси інженерного співтовариства перед урядом, а також звертали увагу уряду на проблеми розвитку співтовариства, забезпечували просування передових ідей і висунення вперед кращих представників інженерного співтовариства, щоб підтримка уряду була ефективною й адресною.

Зростання кваліфікованої робочої сили, урядова підтримка інженерної справи і освіти стала базовим чинником зростання кількості винаходів і їх практичного застосування. Саме тому Англія задала тон першому етапу промислової революції, головними підсумками якого стали винахід парових машин і прядильних верстатів, що призвело до могутнього промислового ривка галузей легкої промисловості. Промисловості, що бурхливо розвивається, були потрібні все нові і нові винаходи для забезпечення масового машинного виробництва деталей, це спричинило появу нових винаходів, ключовим з яких став фрезерний верстат. У 1844 році знов великий винахід – бетон, який через кілька років став застосовуватися в будівництві міських систем каналізації.

Великобританія стала першовідкривачем такого популярного і дешевого сучасного транспорту, як метро. 10 січня 1863 року в Лондоні відкрилася перша ділянка метро, а в Радянському Союзі

метрополітен почав свою роботу лише 15 травня 1935 року, коли метро вже стало широко розповсюджуватися по всьому світу, як інструмент розв'язання гострих транспортних проблем, пов'язаних із розвитком територій і переміщенням людей.

Англійський уряд сфокусувався на підтримці інженерного співтовариства, оскільки вигода від його розвитку була цілком очевидна. Інститут цивільних інженерів Великобританії став ефективним мостом, що забезпечує постійні комунікації між урядом та інженерним співтовариством. У результаті цієї взаємодії з'явилися ефективні програми підтримки найбільш обдарованих інженерів, видатних інженерних проектів і підприємців-промисловців.

Усі винаходи й досягнення першої промислової революції призвели до другої промислової революції, період якої припав на кінець XIX – початку XX століття, але в ній головну роль зіграли вже Сполучені Штати Америки. При цьому британці не зменшили темпу розвитку інженерної справи і продовжили представляти світу великих винахідників. У 1930 роки британський інженер Френк Уайтлл зареєстрував перший патент на реактивний двигун. Британці зробили свій внесок і у винахід сучасних комп'ютерів, прообраз якого в 1936 році запропонував англієць Алан Тьюрінг. Найбільш продуктивно його ідеї розвинули в США, які згодом стали лідером комп'ютерної галузі.

Сьогодні багато англійських компаній, діяльність яких заснована на інженерних досягненнях, очолюють світові рейтинги найдорожчих (капіталізованих) компаній. Ці компанії вже зосередили у своїх руках не тільки значну частку продуктивних сил і капіталу, але і велику частину наукових досліджень і розробок у сфері нових матеріалів і технологій. Серед них – British Aerospace, що займає 9-те місце у світі за обсягом продажів серед авіаційних і космічних компаній. Вона ж – друга у світі з виробництва літальних апаратів. Одні з кращих двигунів у світі випускає компанія Rolls-Royce. У царині чорної металургії лідирує британська сталева монополія British Steel, у області кольорової металургії глобальне лідерство утримує британсько-австралійська компанія Rio Tinto, інший найбільший

гравець ринку кольорової металургії – це Johnson Mattney. У хімічній галузі була широко відома британська Imperial Chemical Industries, проте в 2008 році вона повністю увійшла до складу голландської Akzo Nobel (провідний хімічний концерн Нідерландів).

Англійська промисловість досягла високого рівня розвитку завдяки активній підтримці держави, яка багато десятиліть стимулювала промисловий розвиток країни і підтримувала експансію національних компаній на міжнародні ринки.

Увага до провідних німецьких підприємств з боку держави і їх підтримка були рушійною силою розвитку інженерної справи в Німеччині. Найважливішим стимулом розвитку, науки, техніки й інженерної справи в Німеччині стало об'єднання німецьких земель і утворення Німецької імперії. До кінця XIX ст. Німеччина випереджає Францію і стає другою після Великобританії найбільш промислово розвиненою країною Європи. Специфічна риса розвитку німецької промисловості – наявність різних концернів, особливо в хімічній промисловості, які підтримувалися державою й активно впроваджували у виробництво новітні технічні і технологічні розробки того часу. До початку XX століття Німеччина мала найрозвиненішу у світі хімічну промисловість, три німецькі концерни – BASF, Bayer і Hoechst – контролювали 90 % світової продукції синтетичних фарбників. І зараз німецькі компанії продовжують відігравати провідну роль у світовій хімічній промисловості (крім перерахованих, це Henkel Group). Німеччина також на весь світ славиться своїми автомобільними концернами – Daimler, Volkswagen, BMW. Німецькі компанії E.ON і RWE – одні з найбільших енергетичних компаній як Німеччини, так і світу. Провідні позиції німецьких компаній у світовій економіці і промисловому виробництві впродовж уже двох століть не випадкові, всі вони стали можливі завдяки підтримці інженерної освіти і професійного науково-інженерного співтовариства, завдяки налагодженим механізмам залучення німецьких інженерів до промислового виробництва.

Найкрупнішим об'єднанням інженерів у Німеччині на сьогодні є Асоціація німецьких інженерів (Association of German Engineers), яка об'єднує 139000 професіоналів. Вона була утворена в 1857 році в маленькому містечку Алексисбаде, зараз штаб-квартира організації знаходиться в Дюсельдорфі. Асоціація німецьких інженерів сприяє технологічному розвитку країни і просуває інтереси інженерів та інженерного бізнесу в Німеччині. Завдяки Асоціації німецьких інженерів уряд Німеччини поінформований щодо проблем інженерного співтовариства країни, допомагає їх розв'язувати, а також успішно інвестує бюджетні кошти в інженерні розробки. В результаті економіка Німеччини росте і розвивається за якнайкращим сценарієм, країна одержує величезну експортну виручку від продажу високотехнологічної продукції. Рівень життя в Німеччині один із найвищих у світі.

Сьогодні Німеччина готується до нового технологічного стрибка, залучає на свою територію кваліфіковані кадри й молодих талановитих інженерів. Для цього спільно з Асоціацією німецьких інженерів розроблена урядова програма, яка полегшує прибуття на територію Німеччини нових кадрів та їх інтеграцію в економіку країни.

Бурхливо розвивалася інженерна справа і у Франції, поряд з Великобританією однією з провідних морських і колоніальних імперій Європи. Услід за Великобританією 4 березня 1848 року, за спільним рішенням інженерів Франції, було засноване Центральне товариство цивільних інженерів (Societe Centrale des Ingenieurs Civils). У подальші роки воно зіграло велику роль у формуванні професійних федерацій і союзів.

Взагалі, Франція завжди славилася своїми військовими інженерами і прекрасною зброєю, проте в XIX столітті Франція переорієнтувалася на цивільне будівництво, на розробках у цій галузі концентруватимуться французькі інженери. Проте військовій справі, як і раніше, приділяється висока увага. Франція володіє власним ядерним арсеналом, сама виготовляє літаки, військові кораблі й підводні човни. Французькі винахідники також прославилися відкриттями, що змінили сучасний світ. Наприкінці XIX століття у Франції брати Люмьери

винайшли кіноапарати для зйомки і проекції рухомих фотографій, ці винаходи стали прообразом сучасного кінематографу, а Франція дотепер є одним з лідерів у кіноіндустрії.

У 1992 році три федерації – Національна рада інженерів Франції (Conseil National des Ingenieurs Francais), Федерація асоціацій і союзів дипломованих інженерів Франції (Federation des Associations et Societes Francaises d'Ingenieurs Diplomes) і Союз інженерів і учених Франції ISF (Societe des Ingenieurs et Scientifiques de France) – об'єдналися в Національну раду інженерів і учених Франції. Ця Рада налічує 180 об'єднань на національному рівні, 160000 членів, а головну свою мету бачить у підтримці іміджу та репутації професії інженера, забезпеченні зв'язку інженерного співтовариства з французьким урядом.

Сьогодні Франція відома своєю автомобільною промисловістю, багато автомобільних гігантів Європи мають прописку у Франції – це концерни Renault, Citroen, Peugeot. Французька нафтогазова компанія Total – четверта за обсягами видобутку у світі. Провідну роль в економіці Франції і в даний час продовжують відігравати державні монополії – в електроенергетиці це «Electricite de France». Французькі компанії входять до найбільших європейських холдингових структур, як-от Air France-KLM. Ці приклади наочно демонструють, що промисловість і промислові корпорації, у тому числі за значної державної участі, продовжують відігравати провідну роль в економіці країни. А залишатися успішними їм допомагає не тільки уряд, але і постійне впровадження нових технологій і розробок, чого не відбулося б, якби компанії не зробили ставку на сучасні інженерні рішення і науково-інженерне співтовариство. Тому можна з упевненістю сказати, що інженерні фахівці та їх розробки є ключовим чинником розвитку цих високотехнологічних галузей французької промисловості.

Помітну роль в європейському інженерному русі зіграла і невелика Швейцарія. Один із найстаріших в Європі – Швейцарський союз інженерів і архітекторів (Swiss Society of Engineers and Architects, SIA), який був заснований в 1837 році.

На сьогоднішній момент Швейцарський союз інженерів і архітекторів – це провідне професійне об'єднання фахівців у галузі будівництва, інноваційних технологій і охорони навколишнього середовища у Швейцарії. Маючи у своєму арсеналі близько 15000 кваліфікованих інженерів і архітекторів, Союз є високопрофесійною міжгалузевою мережею, головне завдання якої полягає в екологічному облаштуванні й підвищенні якості навколишнього середовища Швейцарії.

На сучасному етапі діяльність швейцарських інженерних співтовариств і об'єднань безпосередньо пов'язана з розв'язанням соціальних задач: створення сприятливої інфраструктури, розвиток всіх комунальних систем, будівництво екологічного житла і розвиток зелених технологій. Швейцарія – одна з тих країн, в якій соціальна спрямованість діяльності сучасних інженерів виражена найяскравіше.

Інженерні асоціації в Північній Америці. Активний розвиток інженерні асоціації одержали й на північноамериканському континенті. У США в 1852 році було засноване Американське товариство інженерів (American Society of Civil Engineers, ASCE). На даний момент це найстаріше національне об'єднання інженерів у США, що налічує більше 140000 членів. Услід за ASCE інженерні асоціації стали відкриватися і в інших американських штатах, з'явилися більш вузькоспеціалізовані об'єднання інженерів. У 1888 році в США, в штаті Вірджинія, утворене Американське товариство інженерів військово-морського флоту. У 1894 році в Лос-Анджелесі створюється Асоціація інженерів і архітекторів (Engineers & Architects Association, EAA).

У цілому США стають основним лідером другої промислової революції, тому зрозуміла поява великої кількості професійних інженерних асоціацій і об'єднань у країні. У 1859 році США першими почали видобувати нафту в Пенсільванії і одержувати гас, а це у свою чергу зумовило зародження автомобільної промисловості. Перший автомобіль Генрі Форда був зроблений в 1896 році, компанія його імені заснована в 1903 році, а всього

через 10 років після цього виробництво автомобілів стало конвеєрним, а автомобіль – масовим засобом пересування.

Після другої світової війни, а також активізації різних меншин у боротьбі за свої права, в США з'являються достатньо специфічні інженерні співтовариства. У 1950 році створюється Товариство жінок-інженерів (Society of Woman Engineers), а в 1975 році – Національна асоціація афроамериканских інженерів (National Association of Black Engineers), що допомагає працевлаштуватися чорношкірим американцям.

Практично одночасно із США розвивалися інженерні організації і в Канаді. У 1887 році було утворене Канадське товариство інженерів цивільного будівництва (Canadian Society of Civil Engineering, CSCE), яке успішно функціонує дотепер і є найбільшим у країні. Товариство всіляко сприяє розвитку цивільного будівництва, роблячи особливий акцент на таких галузях: інженерна геологія, проектування будівель і споруд, гідротехнічний інжиніринг, захист навколишнього середовища, транспортний інжиніринг і геофізика.

Сучасні інженерні товариства і асоціації в Північній Америці розвиваються за активної підтримки не тільки державних структур, але і професійних інститутів і університетів. У США і Канаді досягнутий той необхідний ступінь кооперації між студентами, вченими і державними органами, який дозволяє оперативно впроваджувати у виробництво найважливіші наукові розробки.

США – це лідер світової економіки. Усі найбільші світові корпорації розташовані саме тут. За рівнем технологічної і технічної оснащеності, ступенем капіталізації, інноваційним потенціалом американські компанії випереджають компанії інших розвинених країн, не кажучи вже про країни, що розвиваються. Серед гігантів американської промисловості можна назвати General Electric – провідну машинобудівну компанію світу, розробника величезної кількості технічних новинок, General Motors – найбільшу автомобільну компанію світу, Ford Motors – одну з найстаріших автомобільних компаній світу. Безумовно, визначальну роль відіграють компанії

високотехнологічної галузі – IBM, Microsoft, які мають безпосереднє відношення до програмної інженерії. Американську прописку має й найбільша приватна нафтова компанія світу і один з лідерів у галузі з впровадження інновацій – Exxon Mobil. Провідну роль в американській і світовій економіці відіграють також дві інші нафтові корпорації – Chevron і ConocoPhilips.

Інженерні асоціації в Латинській Америці. Одне з найстаріших і найбільших інженерних об'єднань у Латинській Америці – це Корпорація інженерів Венесуели (El Colegio de Ingenieros de Venezuela, CIV). Вона була утворена в 1861 році указом тодішнього президента Мануеля Феліппе Товару. Спочатку цим об'єднанням займалося Міністерство оборони і міністерство військово-морського флоту, що особливо підкреслює значущість Корпорації інженерів Венесуели для суспільства і уряду, а пізніше з 1861 року діяльність Корпорації інженерів Венесуели керує Міністерство освіти Венесуели. Нині у Корпорації перебувають більше 120000 інженерів і архітекторів.

Інженерні асоціації в Австралії і Океанії. Найкрупніше професійне об'єднання інженерів в Австралії – це Об'єднання інженерів Австралії (Institution of Engineers Australia, IEAus), що налічує 90000 членів. Стимулом до створення інженерного об'єднання в Австралії слугував початок промислової революції, проте перше засідання було проведено тільки через 20 років після оголошення створення організації, в 1919 році. Об'єднання інженерів Австралії бачить своєю метою просування австралійських інженерів та їх визнання не тільки всередині Австралії, але і по всьому світу.

Існують інженерні асоціації і в Новій Зеландії. Найбільше об'єднання – це Інститут професійних інженерів у Новій Зеландії (Institution of Professional Engineers New Zealand), що має у своєму складі на даний час 12000 інженерних фахівців. Інститут займається професійним розвитком інженерних кадрів, впровадженням професійних стандартів в інженерній діяльності.

Інженерні асоціації в Азії. Найкрупнішим національним інженерним об'єднанням є Об'єднання інженерів Індії (Institution of Engineers India, IEI), що налічує більше півмільйона чоловік.

Об'єднання інженерів Індії було створено в 1920 році в Мадрасі (сучасний Ченай). Професійна організація об'єднує інженерів з 15 галузей, усередині Індії і за її межами об'єднання має 99 відділень. 2 рази на рік Об'єднання інженерів Індії проводить міжнародні зустрічі інженерів. Крім того, Об'єднання інженерів Індії представляє інженерне співтовариство Індії на різних міжнародних форумах.

Окремо необхідно виділити Японію, найбільш високорозвинену країну сучасної Азії, що досягла феноменального прогресу в лічені десятиліття. У Японії немає великих територій, немає корисних копалини і немає величезних людських ресурсів, проте у неї прекрасні інженерні кадри. Географічне і геополітичне положення Японії зумовлює необхідність впровадження найбільш передових досягнень у галузі цивільного будівництва, тому вага інженерних об'єднань у сфері цивільного будівництва в Японії дуже висока, а найстарше і найавторитетніше інженерне об'єднання Японії створене саме в цій галузі. Товариство інженерів цивільного будівництва Японії (Japan Society of Civil Engineers, JSCE) створене в 1914 році. Спочатку його членами були всього 443 професійних інженери, зараз воно налічує 39000 японських професіоналів. Товариство просуває інтереси інженерів цивільного будівництва, проводить дослідження, сприяє міжнародному науковому обміну, підтримує кращі проекти.

Розвиток інженерної справи, підтримка інженерів та інженерної справи з боку держави зіграли величезну роль у процесі перетворення Японії у високорозвинену високотехнологічну країну. Електротехнічна промисловість і робототехніка стали локомотивами економіки Японії, світову популярність набула японська техніка таких фірм: Panasonic Corporation, Sony Corporation, Toshiba. Японія славиться і своєю автомобільною промисловістю: Toyota Motors, Honda Motors – найпопулярніші марки автомобілів. Унікальне поєднання рис національної вдачі, грамотної урядової політики, спрямованої на підтримку та розвиток технологій і технічних, інженерних

спеціальностей дозволили Японії стати однією з найбільш високорозвинених і економічно успішних країн світу.

Серед впливових інженерних об'єднань в Азії можна виділити Гонконзьке об'єднання інженерів (Hong Kong Institution of Engineers, HKIE). Історія організації починається в 1947 році, первинна назва Engineering Society of Hong Kong. У 1975 році товариство одержало свою сучасну назву. Нині його членами є більше 20000 професійних інженерів з 19 галузей. Створення інженерного товариства в Гонконзі зумовило процес успішної індустріалізації, яку Гонконг провів у 50–70-ті роки XX століття. За рахунок промислового експорту економіка спеціального адміністративного району КНР росла на 8–9 % в рік. З кінця 80-х років провідну роль в економіці Гонконгу починає відігравати сектор послуг. Проте Гонконг продовжує залучати інженерів, проводити й упроваджувати навчальні програми, підвищувати кваліфікацію фахівців інженерного співтовариства.

Міжнародні інженерні об'єднання. З середини XX століття національні громадські організації інженерів по всьому світі стали об'єднуватися в міжнародні союзи. Деякі союзи були галузевими, інші входили в національні інженерні асоціації. Так, в 1954 році була заснована Європейська асоціація інженерної індустрії ORGALIME (Organisme de Liaison des Industries Métalliques Européennes), до якої увійшли компанії з 22 європейських країн, що представляють машинобудування, металообробку, електроенергетику й електроніку.

У 1951 році була сформована Європейська федерація національних інженерних асоціацій (European Federation of National Engineering Associations, FEANI), яка об'єднувала інженерні асоціації з 32 європейських країн. З 1997 року штаб-квартира асоціації знаходиться в Брюсселі (Бельгія). Сьогодні серед членів асоціації налічується близько 3,5 млн. професійних європейських інженерів, саме їх інтереси асоціація і відстоює, прагнучи того, щоб національна кваліфікація інженерів відповідала загальним європейським стандартам і визнавалася в усіх країнах.

Національні інженерні об'єднання є найважливішою складовою суспільного життя всіх провідних країн світу і одним з ключових чинників їх економічного зростання.

Одними із найбільш затребуваних на сьогодні є комп'ютерні фахівці: комп'ютерні інженери, програмні інженери тощо. Інтернет відіграє ключову роль у розповсюдженні інформації, з розвитком інтернет-технологій зовсім іншої ролі набувають системи безпеки передачі інформації. Без перебільшення можна сказати, що інтернет абсолютно змінив сучасні комунікації. Усі ці зміни зумовили появу асоціацій інженерів у галузі IT-технологій. Одна з перших подібних асоціацій, Міжнародна асоціація інженерів (International Association of Engineers, IAENG), була створена ще в 1968 році. Нині до складу IAENG входять керівники науково-дослідних центрів, декани факультетів, департаментів, професори, учені, інженери, що займаються науково-дослідною діяльністю, а також випускники університетів, студенти і аспіранти із понад 100 країн світу.

Одним з найбільших галузевих об'єднань інженерів є Об'єднання інженерів з електротехніки й радіоелектроніки (The Institute of Electrical and Electronics Engineers, IEEE). Об'єднання є некомерційною професійною асоціацією зі штаб-квартирою в Нью-Йорку (США). Всього членами Об'єднання є більше 400000 інженерних фахівців по всьому світу, проте більше 50 % з них – громадяни США. Об'єднання створено в 1963 році в результаті злиття Інституту радіоінженерів (Institute of Radio Engineers) і Американського інституту інженерів з електротехніки (American Institute of Electrical Engineers). Об'єднання інженерів з електротехніки й радіоелектроніки також є розробником промислових стандартів. Ним було розроблено більше 900 стандартів у галузях енергетики, біомедичних технологій і охорони здоров'я, інформаційних технологій, безпеки, телекомунікацій, транспорту, аерокосмічної галузі і нанотехнологій.

Сучасною тенденцією розвитку всіх світових інженерних асоціацій і рухів є активна увага до навколишнього середовища. Величезної ваги набувають ті інженерні асоціації, які займаються

медичними і біологічними дослідженнями, розробляють сучасне медичне устаткування. Саме ці області досліджень отримують сьогодні максимальну державну підтримку.

Асоціації і об'єднання студентів-інженерів. Розглянемо особливості студентських об'єднань у провідних світових інженерних вузах, більшість з яких розташована в Сполучених Штатах Америки і Великобританії. Згідно з рейтингом The world University Rankings, у 2011–2012 рр. перше місце серед провідних інженерних вузів розділили два американські вузи: Каліфорнійський технологічний інститут і Массачусетський технологічний інститут. У рейтингу, куди увійшли 500 провідних вузів світу, українські вищі навчальні заклади відсутні.

Лідерство двох американських університетів пояснюється потужною дослідницькою базою, багато винаходів у галузі робототехніки, штучного інтелекту були здійснені за найактивнішої участі студентського і викладацького складу університетів, що вже є абсолютно звичайною практикою. Відійшли в далеке минуле часи, коли цінний винахід припадав на другу половину життя інженера-дослідника, коли він ставав досить поважною людиною у віці. Нова апаратура і сучасна дослідницька база, нова техніка і технології дозволяють вже у студентські роки робити досить серйозні проривні винаходи. Особливо хотілося би відзначити лідерство інженерної школи Массачусетського інституту в усіх світових рейтингах протягом уже чверті століття, а саме з 1988 року.

У останні десятиліття XX століття в багатьох університетах і інститутах створюються факультетські інженерні об'єднання і асоціації. У 1989 році в невеликому нідерландському містечку Делфт була утворена Міжнародна асоціація студентів інженерів (International Association of Civil Engineering Students, IACES). За 20 років своєї діяльності асоціація встановила зв'язки з 50 вузами всередині Європи і за її межами, які мають свої інженерні школи. Головне завдання асоціації – міжнародний обмін досвідом між студентами, знайомство зі специфікою викладання інженерної справи в інших країнах, культурний обмін, встановлення контактів з майбутніми працедавцями.

Отже, взаємодія національних інженерних об'єднань і урядових структур є поширеною світовою практикою, яка своїм корінням сягає початку позаминулого сторіччя. Така взаємодія дає високі результати, позитивно впливає на розвиток будівництва, промисловості, транспорту, медицини, телекомунікацій, що у свою чергу сприяє розв'язанню багатьох соціальних задач, дає імпульс розвитку нових галузей економіки.

Новим імпульсом розвитку інженерного співтовариства слугувало закінчення другої світової війни. Саме інженери стали відігравати одну з визначальних ролей в економічному розвитку багатьох країн, адже потрібно було в короткі терміни відновлювати зруйноване війною господарство. Це, у свою чергу, стимулювало розвиток нової техніки і технологій та впровадження їх у виробництво.

Інженерні рішення багато в чому визначають розвиток сучасних економік, постійно з'являються нові сфери діяльності, де інженерна праця затребувана не менше, ніж у військовій справі, цивільному і промисловому будівництві тощо. Серед таких нових галузей – інформаційні технології, біоінженерні технології, медичні розробки, пошук нових джерел енергії тощо.

Створення союзів і асоціацій інженерів набуло значного поширення в усьому світі. У результаті сьогодні у світі діє більше 120 великих національних інженерних союзів на всіх п'яти континентах. Діє кілька десятків міжнародних інженерних асоціацій. У багатьох країнах Латинської Америки створюються не свої національні інженерні співтовариства, а окремі національні представництва найбільших міжнародних інженерних асоціацій, що засвідчують процеси глобалізації і тіснішої економічної взаємодії між країнами.

Набуло широкого світового розповсюдження створення інженерних громадських організацій з окремих вузьких напрямків. Інженерні союзи й асоціації почали утворюватися в різних куточках світу, але, як і раніше, провідна роль в інженерному русі належить США і Європі.

Окремо необхідно відзначити ту підтримку, яку уряди розвинених країн та країн, що розвиваються (наприклад, Індії),

надають інженерам, інженерній освіті, як активно вони стимулюють інженерні розробки й усіляко сприяють їх швидкому впровадженню в життя.

3.2. Огляд діяльності комп'ютерного співтовариства інституту інженерів з електроніки та електротехніки (IEEE-CS)

Рис. 3.1. Емблема інституту інженерів з електроніки та електротехніки

Інститут інженерів з електроніки та електротехніки IEEE (Institute of Electrical and Electronics Engineers) є однією з найстаріших і найкрупніших інженерних організацій. IEEE сприяє технологічному процесу створення, розробки, інтеграції, сумісному використанню і застосуванню знань з електротехнічних та інформаційних технологій і новітніх наукових досліджень у цих напрямках. Інститут IEEE бере участь у створенні та розповсюдженні стандартів, які мають відношення до всіх електротехнічних питань у багатьох галузях промисловості.

IEEE володіє подвійною структурою: географічною та технічною, тобто з одного боку інститут розподілено по всьому світу, тобто наявний географічний поділ, а з іншого боку в IEEE йде паралельна робота в різних технічних галузях.

До сфери інтересів IEEE входить ряд питань, пов'язаних з електротехнікою, радіоелектронікою, обчислювальною технікою, інформатикою, а також деякими розділами фізики і математики. Основні напрями роботи цієї організації:

- проведення спеціалізованих професійних конференцій;
- публікація спеціалізованих видань;
- підтримка освітньої діяльності;
- підтримка інноваційних технічних і методичних розробок у різних сферах;

- розробка і розповсюдження технічних стандартів.

До складу IEEE входять 9 регіональних відділень, 36 професійних товариств, 4 ради і 1450 студентських відділень. Поточне управління діяльністю на верхньому рівні здійснюється Радою директорів і Виконавчим комітетом, роботу яких очолюють Президент і Виконавчий директор.

IEEE – це одна з найбільших у світі асоціацій, що займається стандартизацією в галузі електротехніки і електроніки. IEEE організує роботу щодо стандартизації в комп'ютерній, авіаційній, космічній, а також телекомунікаційній промисловості. Головна мета IEEE полягає у сприянні технічним інноваціям на благо всього людства.

IEEE випустила велику кількість відомих у всьому світі стандартів, зокрема: IEEE 802.3 – Ethernet, IEEE 802.11 –WiFi, IEEE 802.16 – WIMAX, IEEE 1294 – USB та багато інших.

Свою історію IEEE починає з 1884 року, ще на зорі використання електрики в промисловості. IEEE був організований в Нью-Йорку на зустрічі працівників електротехнічної промисловості як American Institute of Electrical Engineers (AIEE) – Американський інститут інженерів з електрики. Його метою була допомога в роботі професійних інженерів у цій новій галузі, а також розповсюдження інформації й обмін досвідом. У 1912 році, у зв'язку з появою нової галузі промисловості – радіозв'язку, була організована інша організація – Institute of Radio Engineers (IRE) – Інститут радіоінженерів. Свою теперішню структуру і назву IEEE одержав у 1963 році після об'єднання AIEE та IRE. До того моменту в інституті вже було більше 150000 учасників, 140000 з яких були громадянами США. На початку XXI століття IEEE вже об'єднував 38 різних організацій, випускав 130 журналів, проводив більше 300 конференцій щорічно й розробляв близько 900 стандартів. Наприкінці 2010 року IEEE об'єднував 395000 учасників, включаючи більше 80000 студентів 160 країн. Він фінансує 40 товариств і рад, організованих у підрозділи, які займаються такими питаннями:

Підрозділ I – схеми і прилади

- Товариство з розробки схем і систем.
- Товариство з розробки компонентів і промислової технології.
- Товариство з проектування електронних приладів.
- Товариство з розробки лазерів та електрооптики.
- Рада з проектування датчиків.
- Товариство з розробки напівпровідникових схем.

Підрозділ II – промислові застосування

- Товариство з розробки діелектриків і електроізоляційних матеріалів.
- Товариство з розробки промислових застосувань.
- Товариство з проектування контрольно-вимірювальних приладів і проведення вимірювань.
- Товариство з розробки елементів силової електроніки.

Підрозділ III – техніка зв'язку

- Товариство із забезпечення комунікацій.

Підрозділ IV – електромагнетизм і радіація

- Товариство з проектування антен і розповсюдження сигналів.
- Товариство з розробки технології ретрансляції.
- Товариство з розробки приладів побутової електроніки.
- Товариство із забезпечення електромагнітної сумісності.
- Товариство з дослідження магнетизму.
- Товариство з розробки мікрохвильової теорії і методів.
- Товариство, що забезпечує виконання досліджень у сфері теорії ядра і плазми.
- Рада з дослідження надпровідності.

Підрозділ V – комп'ютери

- Комп'ютерне товариство.

Підрозділ VI – інжиніринг і людське суспільство

- Товариство з питань освіти.
- Товариство з питань менеджменту процесу інжинірингу.
- Товариство професійних комунікацій.
- Товариство із забезпечення надійності.
- Товариство з вивчення питань соціальної значущості технологій.

Підрозділ VII – енергія і енергетика

- Товариство з вивчення проблем у галузі енергетики.

Підрозділ VIII – сигнали і застосування

- Товариство з питань розробки аерокосмічних і електронних систем.
- Товариство з геофізики та дистанційного збору даних.
- Товариство з океанічної техніки.
- Товариство з обробки сигналів.
- Товариство з ультразвукового, сегнетоелектричного і частотного контролю.
- Товариство з мобільних технологій.

Підрозділ IX – системи і контроль

- Товариство з систем контролю і управління.
- Товариство з техніки у медицині і біології.
- Товариство промислової електроніки.
- Товариство з інформаційної теорії.
- Рада з інтелектуальних систем передачі даних.
- Рада зі штучних нейронних мереж.
- Товариство з робототехніки і автоматизації.
- Товариство «Системи, людина та кібернетика».

Розглянемо більш детально діяльність Комп'ютерного товариства (Computer Society) IEEE, яке є найбільшим серед 36 технічних товариств IEEE та найстарішим у світі об'єднанням професіоналів з комп'ютерних технологій. Комп'ютерне товариство IEEE, утворене в 1946 році, об'єднує програмних та комп'ютерних інженерів усього світу і є товариством, яке розвивається надзвичайно високими темпами. Товариство створене для розвитку теорії, практики і застосування комп'ютерів (комп'ютерних технологій) і технології обробки інформації. Комп'ютерне товариство IEEE є однією з провідних організацій комп'ютерних професіоналів у світі. Штаб-квартира знаходиться у Вашингтоні; ще два центральних офіси – у Брюсселі та Токіо.

Рис. 3.2. Емблема Комп'ютерного товариства Інституту інженерів з електроніки та електротехніки

Робота товариства надзвичайно різноманітна. Все, що може допомогти комп'ютерним професіоналам у роботі є предметом діяльності товариства. Згідно зі стратегічним планом, товариство IEEE Computer Society повинно стати провідним виробником і постачальником інформації та різноманітних послуг для професіоналів у всьому світі.

Комп'ютерне товариство реалізує свою місію шляхом організації конференцій, видань, технічних комітетів, груп з підготовки стандартів і локальних студентських відділень. Воно спонсорує (повністю або частково) більше 140 щорічних конференцій, симпозіумів і нарад зі всього спектра тематики інформаційних технологій. Уся діяльність максимально децентралізована і спирається на роботу регіональних комітетів і підкомітетів. Річний доход товариства у 2012 році досяг 30 млн. доларів, причому членські внески складають менше 10 % з цієї суми. Діє спеціальна програма «Distinguished Visitors Program», в рамках якої відомі науковці виступають по всьому світу з лекціями щодо сучасного стану комп'ютерної науки і технології.

IEEE Computer Society є лідером комп'ютерної індустрії в розробці широко застосовуваних, технічно довершених стандартів. Існує більше 200 робочих груп, що належать до 11 комітетів зі стандартизації IEEE Computer Society. Керівники робочих груп постійно запрошують членів товариства взяти участь у розробці стандартів, і тисячі професіоналів відгукуються на цей заклик.

Співтовариство IEEE Computer Society затверджує колегіальні (університетські) програми з комп'ютерних наук (інформатики) та техніки в Сполучених Штатах. Як приклад розглянемо SWEBOK, розроблений комітетом IEEE Computer Society – Software Engineering Coordinating Committee за підтримки одинадцяти провідних організацій у сфері розробки програмного забезпечення [7]. SWEBOK – Software Engineering Body of Knowledge – Зібрання знань з програмної інженерії – містить опис складу знань за 10 розділами (галузями знань) програмної інженерії. Необхідно відзначити спільну діяльність IEEE Computer Society і ACM (Association for Computing Machinery – Асоціації з обчислювальної техніки) щодо створення моделі учбового плану в галузях обчислювальної техніки та інформаційних технологій («Model Curricula for Computing»).

На даний час створені модельні навчальні плани з напрямків інформатики, інженерії програмного забезпечення, інженерії обчислювальної техніки, інформаційних систем. Наприклад, ACM/IEEE CC2001 – Computing Curricula 2001 – Академічний освітній стандарт у галузі комп'ютерних наук. Виділені чотири основні розділи: Комп'ютерні науки (Computer science), Комп'ютерна інженерія (Computer engineering), Програмна інженерія (Software engineering) та Інформаційні системи (Information systems). У кожному розділі дається опис відповідних сфер знань, їх склад і плани рекомендованих курсів [1].

Для професіоналів, охочих підвищити свою кваліфікацію та одержати відповідні сертифікати, IEEE Computer Society пропонує майже 3500 курсів, що стосуються розвитку технологій, ведення бізнесу, розвитку професійних навичок. Ці курси передбачають більше 10000 годин навчання і можуть допомогти всім охочим членам співтовариства підготуватися до іспиту для отримання стандартних сертифікатів, включаючи Microsoft, Cisco, Sun, IT-безпеки й управління проектами. Багато курсів доступні на кількох мовах.

Є можливість також одержати свідоцтва, які підтверджують високий професіоналізм – сертифікати IEEE CS: «Certified

Software Development Associate» (CSDA) і «Certified Software Development Professional» (CSDP).

Крім того, комп'ютерне товариство IEEE Computer Society оголосило про відкриття програми сертифікації розробників програмного забезпечення (Certified Software Development Associate program), або, скорочено, CSDA program.

Диплом CSDA підтверджує високий рівень знань і дозволяє претендувати на вищі позиції при працевлаштуванні, успішнішу кар'єру і вищу зарплату в процесі професійної діяльності.

Дипломи CSDA або CSDP для багатьох інженерів уже стали першим кроком на шляху вступу до лав професіоналів.

Відносно професійних програмних інженерів дипломи CSDA та CSDP забезпечують такі якості:

• є розуміння основних принципів розробки програмного забезпечення, що допомагає у кар'єрному зростанні;

• скорочується розрив між знаннями, одержаними в інституті, та реальними потребами галузі;

• прискорюється процес навчання, що сприяє зростанню професіоналізму;

• демонструється прагнення до професійного зростання.

Відносно роботодавців, що приймають на роботу спеціалістів із дипломами CSDA та CSDP, забезпечуються такі гарантії:

• прийом на роботу фахівців, які допомагають добитися успіху в умовах зростаючої конкуренції;

• гарантії того, що співробітники здатні виконувати складні проекти;

• скорочується час навчання і підготовки персоналу до виконання нових проектів;

• дається об'єктивна оцінка при виборі кандидата на підвищення, на роль лідера проекту.

Комп'ютерне товариство IEEE публікує, поширює і зберігає більше 20 періодичних видань, серед них – популярні журнали Computer, IT Professional, IEEE Internet Computing, IEEE Software, Transaction on Computing, Transaction on Software engineering. Журнали пропонують читачам рецензовані статті та дослідницькі роботи у всіх сферах інформатики, включно зі штучним

інтелектом, апаратним забезпеченням комп'ютерів, комп'ютерною графікою, мережними технологіями, інформаційними технологіями, розробкою програмного забезпечення, мультимедійними технологіями тощо. Видавництво IEEE Computer Press видає велику кількість книг, матеріалів конференцій і навчальних курсів. Журнал Computer, який неодноразово завойовував різноманітні нагороди, надсилається кожному члену товариства. Computer – це щомісячне видання, яке інформує членів IEEE-CS про останні технологічні новинки, тенденції та з інших питань, важливих для професіоналів.

Члени товариства отримують безкоштовний доступ до такої технічної інформації:

- журнал «Computer magazine»;
- необмежений прямий доступ до 600 книг Safari R;
- необмежений прямий доступ до 3000 модулів електронних курсів, включаючи такі теми, як Java, Cisco, Microsoft, управління проектами, безпека тощо;
- вільний прямий доступ до книг і технічних статей суспільства.

Крім того, існують:

- знижки на підписку провідних періодичних науково-технічних видань IEEE;
- знижки на сертифікаційні іспити та підготовчі курси для професіоналів з розробки програмного забезпечення;
- символічні реєстраційні внески для участі в конференціях, які спонсоруються товариством IEEE;
- знижки при купівлі книжок та збірників тез конференцій;
- безкоштовне членство в місцевих групах;
- можливості роботи на громадських засадах в одному з багатьох підрозділів і комітетів товариства;
- можливість отримання нагород товариства.

Електронна бібліотека Комп'ютерного товариства є об'ємною електронною колекцією, що включає випуски 18 періодичних видань товариства (після 1988 року), а також праці більше 850 різних конференцій (після 1995 року). 68000 статей, що зберігаються в ній, роблять цифрову бібліотеку цінним

дослідницьким інструментом для професіоналів у будь-якій області інформатики.

Технічні комітети товариства IEEE Computer Society – це глобальні колективи професіоналів зі спільними інтересами в конкретній області інформатики. Члени цих колективів взаємодіють через електронні засоби спілкування, зустрічаються на конференціях тощо. Більше 30 технічних комітетів об'єднують членів товариства за їх професійними інтересами. Кожний з комітетів регулярно проводить конференції і робочі зустрічі, як очні, так і заочні, видає інформаційні бюлетені. Розмір комітетів варіює від 500 до 10000 членів. Існує 4 технічних комітети зі спеціалізаціями в таких сферах, як архітектура ЕОМ, операційні системи, Інтернет, програмна інженерія, безпека та інші. Більшість технічних комітетів видають і безкоштовно поширюють для своїх членів інформаційні бюлетені.

Одним з основних підрозділів IEEE Computer Society, що спеціалізуються на питаннях інформаційної безпеки, є Технічний комітет з безпеки і захисту приватної інформації – "IEEE Computer Society Technical Committee on Security and Privacy". У його складі функціонують три підкомітети:

1. Підкомітет зі стандартів (Subcommittee on Standards);

2. Підкомітет з академічної роботи (Subcommittee on Academic Affairs);

3. Підкомітет зі спеціалізованих конференцій (Subcommittee on Security Conferences).

Основними заходами, які проводить цей комітет, є:

- щорічний симпозіум з безпеки та захисту приватної інформації (IEEE CS Symposium on Security and Privacy);

- щорічний семінар з основ інформаційної безпеки (Computer Security Foundations Workshop).

Комітет веде роботу зі збору та узагальнення актуальної інформації про події співтовариства фахівців з інформаційної безпеки: оголошення про заплановані конференції, звіти щодо минулих конференцій та семінарів, огляди літератури і періодики, посилання на ресурси в мережі Інтернет і т.п. Спеціальний інформаційний бюлетень з цією інформацією – «Cipher» –

розсилається членам співтовариства в середньому один раз на два місяці.

Для заохочення членів товариства, що досягли видатних результатів, IEEE Computer Society спонсорує дієву і престижну програму нагород. Ці нагороди відзначають як технічні досягнення, так і служіння професії й суспільству.

Керівники IEEE Computer Society ініціювали створення великої кількості іменних премій за високі досягнення в галузі комп'ютерних наук (Computer Science) і комп'ютерної індустрії (Computer Industry) в цілому, тобто за успіхи як у науковій, так і у виробничій сфері. Одна з найпочесніших нагород – звання «Fellow Member of IEEE» із врученням золотого значка присуджується щорічно за високі досягнення у професійній діяльності й активну громадську роботу.

Загалом можна відзначити такі нагороди співтовариства IEEE Computer Society:

- премія імені Harry H. Goode присуджується за видатні досягнення у сфері обробки інформації;
- премія імені W. Wallace McDowell присуджується за видатні досягнення останніх років з теорії, конструювання, освіти, практики та інновацій у галузі комп'ютерів і комп'ютерних технологій;
- премія «Technical Achievement Award» присуджується за видатний та інноваційний внесок у галузі комп'ютерної інформатики, конструювання і комп'ютерних технологій за роботи, виконані протягом останніх 10–15 років. За цією номінацією вручається п'ять нагород;
- премія «Computer Entrepreneur Award» є однією із найпрестижніших нагород IEEE Computer Society, яка вручається провідним менеджерам і керівникам компаній, що здійснили значний внесок у зростання одного із сегментів комп'ютерної індустрії: від створення комп'ютерів і комп'ютерних технологій до конкретних застосувань;
- найвища нагорода IEEE Computer Society – премія «Computer Pioneer Award» – вручається за видатний основоположний внесок у розвиток комп'ютерної індустрії та комп'ютерних технологій.

3.3. Огляд діяльності асоціації з обчислювальної техніки (ACM)

Рис. 3.3. Емблема асоціації з обчислювальної техніки

Асоціація з обчислювальної техніки ACM (Association for Computing Machinery) [2] є однією з найстаріших організацій, пов'язаних з інформаційними технологіями, – вона була заснована в 1947 році, на початку розвитку комп'ютерної техніки. Організована через рік після появи першої обчислювальної машини загального призначення ENIAC (Electronic Numerical Integrator and Computer), ACM була заснована математиками та інженерами, що об'єдналися для розвитку теорії і практики інформаційних технологій. Джон Маучлі (John William Mauchly), один з винахідників ENIAC, був серед засновників ACM.

Основні завдання ACM – підвищення технічної компетентності фахівців у галузі комп'ютерних технологій, підтримка освітніх проектів у сфері інформаційних технологій, організація науково-практичних конференцій, симпозіумів і семінарів, суспільно-політична робота, пов'язана з інформаційними технологіями, публікація періодичних видань і збірок наукових праць, присвячених проблемам сучасних інформаційних технологій, підтримка електронного архіву таких публікацій, розробка і впровадження стандартів, а також інша подібна діяльність. Під егідою ACM проводяться щорічні міжнародні студентські олімпіади з програмування. Основним керуючим органом цієї організації є Рада ACM, в яку входить 16 чоловік, зокрема президент і віце-президент. Управління поточними справами Асоціації здійснюють чотири профільні

комітети. Штаб-квартира ACM, в якій працюють основні виконавчі органи, розташовується в Нью-Йорку з 1960 року. Членами асоціації є понад 90000 осіб – викладачів, дослідників, практиків, менеджерів та інженерів.

ACM стала тією ареною, на якій відбувається плідний обмін ідеями, на якій члени асоціації, та інші вчені-фахівці з комп'ютерної техніки розповідають про вже досягнуте й обговорюють перспективи розвитку даної галузі. Діяльність ACM розвивалася разом з розвитком самої галузі. Асоціація відігравала одну з головних ролей у підвищенні якості комп'ютерної техніки і забезпеченні її ефективного використання.

Однією з основ організації роботи ACM є розділення всього співтовариства членів асоціації на так звані групи спеціальних інтересів (Special Interests Group – SIG) – підрозділи, що спеціалізуються на окремих відносно вузьких проблемах розвитку інформаційних технологій. Усього ACM об'єднує 34 робочих групи, що займаються різними питаннями розробки й використання програмного забезпечення, апаратних засобів і телекомунікацій. Кожна з груп самостійно визначає для себе межі своєї діяльності, а їх політика і фінансові питання координуються одним з комітетів.

Одна з цих груп – Special Interest Group on Security, Audit and Control (SIGSAC, Група спеціальних інтересів з питань безпеки, аудиту і контролю) – спеціалізується на питаннях інформаційної безпеки. Основним завданням даної групи є організація роботи спеціалізованих науково-практичних конференцій, таких як:

- Симпозіум з технологій і моделей управління доступом (SACMAT: ACM Symposium on Access Control Models and Technologies), що проводиться щорічно починаючи з 1995 року;
- Конференція з безпеки комп'ютерів і комунікацій (CCS: ACM Conference on Computer and Communications Security), що проводиться щорічно починаючи з 1993 року.

Крім того, питання інформаційної безпеки прямо або опосередковано торкаються роботи інших спеціалізованих груп Асоціації, наприклад Special Interest Group on Electronic Commerce (Група з проблем електронної комерції).

Відзначимо ще чотири спеціальні групи по інтересах:

- SIGART – група зі штучного інтелекту (the ACM Special Interest Group on Artificial Intelligence). Сфера інтересів SIGART – дослідження інтелекту й реалізація штучного інтелекту в комп'ютерних системах;

- SIGARCH – група з області архітектури комп'ютерів (the ACM Special Interest Group on Computer Architecture). Сфера інтересів SIGARCH – архітектура апаратних засобів та їх взаємодія з компіляторами й операційними системами;

- SIGITE – група з освіти в галузі інформаційних технологій (the ACM Special Interest Group on Information Technology Education). Місія SIGITE полягає в тому, щоб забезпечити форум для взаємодії практиків, педагогів та інших науковців у області IT-освіти. Обговорюються питання учбових планів, застосування інноваційних технологій в освіті, педагогічні аспекти IT-освіти;

- SIGMIS – Управління інформаційними системами (the ACM Special Interest Group on Management Information Systems). Дана група розвиває передові технології та дослідження в царині управління інформаційними системами та управління інформаційними технологіями. SIGMIS – засновник мережі ISWorld і спонсор кількох конференцій з інформаційних систем і технологій.

Щодо видавничої діяльності асоціації з обчислювальної техніки, то слід відзначити підрозділ ACM Press, який публікує престижний академічний журнал Journal of the ACM (JACM) і популярні журнали для комп'ютерних професіоналів, Communications of the ACM (також відомий як Communications або CACM) і Queue. Інші публікації ACM включають:

• ACM Crossroads, популярний студентський комп'ютерний журнал у США;

• Ряд журналів, специфічних для підгалузей комп'ютерної науки, які називаються ACM Transactions. Найбільш відомі з них:

- ACM Transactions on Computer Systems;
- ACM Transactions on Database Systems;
- ACM Transactions on Programming Languages and Systems.

Асоціація з обчислювальної техніки запровадила премію Тюрінга (Turing Award) – найпрестижнішу премію в галузі комп'ютерних наук.

Премія названа на честь Алана Тюрінга – британського математика і криптографа. Вона присуджується щорічно з 1966 року за теоретичні і практичні досягнення в інформаційних технологіях і вважається найпрестижнішою премією в цій сфері. На сьогодні розмір премії складає 250 тисяч доларів, спонсорами є компанії Intel і Google.

Уперше Премію Тюрінга було присуджено Алану Перлісу в 1966 році за розвиток технології створення компіляторів. Пізніше її отримали Ніклаус Вірт та Пітер Наур за видатний внесок у розробку мов програмування.

Слід відзначити активну роботу ACM зі студентською молоддю по всьому світу, зокрема проведення щорічних міжнародних студентських олімпіад з програмування. Загалом, заходи, які проводяться асоціацією серед молодих програмістів вищих учбових закладів, дозволяють студентам розширити свої знання, набути нових професійних навичок і цінного досвіду роботи в команді, що допоможе їм стати в майбутньому глобальними лідерами наступного покоління. Асоціація ACM, проводячи олімпіади з програмування, надає талановитим і творчо мислячим молодим людям можливість зібратися разом, взяти участь в інтелектуальному конкурсі та продемонструвати свої здібності в мистецтві програмування й оперативного розв'язання складних практичних задач.

Олімпіади сприяють виявленню найбільш обдарованих учнів, дозволяють правильно зорієнтувати їх у виборі майбутньої професії, пропагують науково-технічні знання серед молоді.

Олімпіада з програмування – це інтелектуальне змагання із виконання різноманітних завдань, для чого необхідно придумати або застосувати який-небудь алгоритм і/або реалізувати даний алгоритм на одній з мов програмування. Олімпіади з програмування проводяться з метою виявлення найбільш талановитих і здібних людей у галузі програмування, а також

підвищення інтересу до програмування з боку підростаючого покоління.

Одним з популярних і престижних видів змагань молодих програмістів сьогодні є студентські змагання зі спортивного програмування. Чемпіонат світу з програмування (International Collegiate Programming Contest, ICPC), організовуваний асоціацією з обчислювальної техніки, – найпрестижніше змагання молодих програмістів, що має майже тридцятирічну історію. Ці турніри вимагають від учасників не тільки високого рівня знань, але і швидкості мислення, реакції та вміння орієнтуватися в нестандартних ситуаціях, повної емоційної і психологічної віддачі. Крім того, формат даних змагань передбачає командну участь, що виховує в молоді навички взаємодії й уміння працювати в команді.

«Енергія, свіжі ідеї і талант цих суперзірок здатні допомогти суспільству в розв'язанні багатьох проблем і змінити світ на краще, – підкреслює д-р Уїльям Біл Паучер (Dr. Bill Poucher), професор університету Бейлора (Baylor University) і виконавчий директор ICPC. – Вони уміють працювати в команді, і вони доб'ються успіху, значно розширюючи можливості, які ми використовуємо сьогодні для взаємодії один з одним».

Уперше міжнародний командний чемпіонат з програмування було проведено в Техаському університеті в 1970 році. Свій нинішній вигляд чемпіонат набув у 1977 році, коли перший фінал був проведений у рамках щорічної конференції ACM з інформатики, і відтоді проводиться щорічно.

Починаючи з 1989 року організацією змагань займається університет Бейлора. У різний час спонсорами змагань ставали такі компанії, як Apple, AT&T і Microsoft, проте з 1997 року по теперішній час генеральним спонсором є компанія IBM.

З 1977 по 1989 в олімпіаді переважно брали участь команди вузів із США і Канади. На сьогоднішній день олімпіада перетворилася на всесвітнє змагання: у 2009 році в ній взяло участь 7109 команд з 88 країн, 100 з яких зійшлися в боротьбі за головний трофей у фінальному турнірі. Кількість команд продовжує рости на 10–20 % у рік.

3.4. Огляд організацій, які підтримують професійну розробку програмного забезпечення

Наведемо загальну інформацію про організації, які мають відношення до професійної розробки програмного забезпечення, а також короткі відомості про ресурси, що пропонуються цими організаціями керівникам програмних проектів [23]. У табл. 3.1 перераховані організації, діяльність яких описуватиметься далі.

Таблиця 3.1

ОРГАНІЗАЦІЇ, ЩО ПІДТРИМУЮТЬ РОЗРОБКУ ПЗ

Скорочена назва	Організація	Основний напрямок діяльності	Який інтерес представляє для програмних інженерів
SEI	Інститут програмної інженерії (Software Engineering Institute)	Розробка програмного забезпечення	Система і модель оцінки зрілості програмних засобів (Capability Maturity Model for Software, SW-CMM)
ASQ	Американське товариство якості (American Society of Quality)	Поліпшення якості програмних продуктів	Основи знань у сфері забезпечення якості ПЗ (Software Quality Engineering Body of Knowledge, CSQEBOK)
ISO	Міжнародна організація зі стандартизації (International Organization for Standardization)	Міжнародні стандарти	Стандарти з якості ISO серії 9000 (ISO 9000 Quality Standards); ISO/IEC 12207 IT-стандарти, що визначають процес життєвого циклу розробки ПЗ
ANSI	Американський національний Інститут стандартів (American National Standards Institute)	Національні стандарти для США	Керівництво по застосуванню ISO/IEC 12207, що забезпечує управління програмними проектами
NIST	Національний Інститут стандартів і технологій (National Institute of Standards and Technology)	Технології, системи вимірювань і стандарти для промисловості США	Національна премія за досягнутий рівень якості імені Малькольма Балдріджа (Malcolm Baldrige National Quality Award Performance Excellence, NBNQA)
PMI	Інститут управління проектами (Project Management Institute)	Загальне управління	Основи знань з управління проектами (Project Body of Knowledge, PMBOK)

Інститут програмної інженерії (Software Engineering Institute, SEI) – це науково-дослідний інститут, що фінансується федеральним урядом при університеті Карнегі–Меллона в Пітсбурзі (Carnegie Mellon University, Pittsburgh). Він фінансується міністерством оборони США канцелярією заступника міністра оборони у справах закупівель, технологій і логістики (Office of the Under-Secretary of Defense for Acquisition, Technology and Logistics). У грудні 1984 року контракт SEI був удостоєний нагороди університету Карнегі–Меллона.

Місія інституту програмної інженерії полягає в такому:

- поліпшення якості комп'ютерних систем залежно від їх цільового призначення;
- прискорення введення сучасних методів програмного інжинірингу;
- забезпечення використання сучасних методів в організаціях, пов'язаних з програмним забезпеченням залежно від їх цільового призначення;
- уведення стандартів відмінної якості для програмного забезпечення.

Рис. 3.4. Емблема інституту SEI

Інститут програмної інженерії фінансує декілька спеціальних дослідницьких програм у таких сферах:

- процеси, методи і системи програмного забезпечення;
- освіта;
- передача технології;
- підтримка Ada і STARS;
- управління ризиками.

Для програмних інженерів великий інтерес являє модель оцінки зрілості програмного забезпечення (Capability Maturity Model for Software, SW-CMM). На рис. 3.5 показано графічне зображення п'яти рівнів зрілості технологій, які визначають відповідну якість програм [13].

Модель SW-CMM наочно демонструє важливість процесів технології створення програмного забезпечення. Велика частина роботи, що виконується інститутом програмної інженерії згідно з моделлю CMM, спрямована саме на поліпшення якості програмного забезпечення.

Продовжуючи зусилля компанії IBM щодо оцінювання якості програмного забезпечення, які було розпочато в середині 80-х років XX століття, команда, очолювана Уотсом Хамфрі (Watts Humphrey), розробила модель CMM з метою створення структури

з поліпшення процесу розробки програмних засобів. Узявши за основу концепцію повторюваності (стабілізація, а потім поліпшення процесу) Эдварда Демінга (W. Edwards Deming) і Джозефа Джурана (Joseph Juran), а також сітку зрілості управління якістю Філіпа Кросбі (Philip Crosby), Хамфрі та його колеги запропонували п'ятирівневу модель зрілості організації для процесу розробки програмного забезпечення.

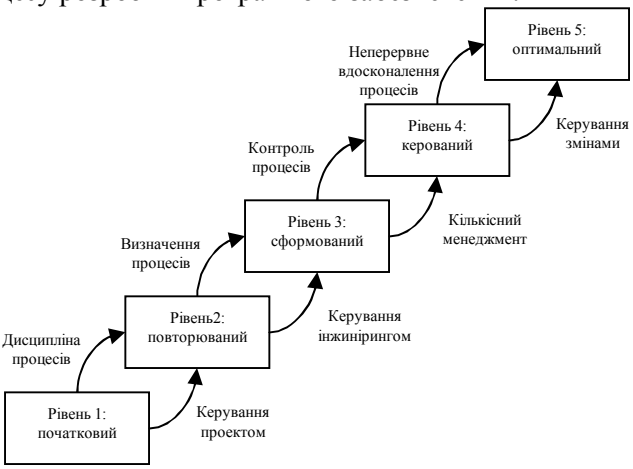

Рис. 3.5. Рівні зрілості CMM інституту програмної інженерії [13]

Зібравши серію зі 150 оціночних питань щодо характеристик програмного забезпечення, Уотс заявив, що концепції якості Демінга і модель етапів Кросбі «досягли своєї мети», породивши класифікацію оціночних питань CMM у тому вигляді, в якому ми її знаємо сьогодні.

Кожен рівень указує на вищий ступінь зрілості та має свою орієнтацію. Чим вищий рівень зрілості, тим вищий статус підприємства. Щоб досягти вищого ступеня зрілості, організації необхідно продемонструвати свої можливості в ключових завданнях. Нижче в таблиці 3.2 показані концентровані та ключові завдання процесу для кожного рівня зрілості.

Відповідно до концепцій SW-CMM, зрілість процесу в організації визначає її здатність до розробки й випуску високоякісних програмних продуктів.

Таблиця 3.2

ХАРАКТЕРИСТИКИ РІВНІВ ЗРІЛОСТІ CMM

Рівень	Акцент	Ключові завдання процесу
Рівень 5: оптимальний	Безперервне поліпшення процесу	- Запобігання дефектам - Управління змінами технології - Управління змінами процесу
Рівень 4: керований	Якість продукту і процесу	- Кількісне управління процесом - Управління якістю ПЗ
Рівень 3: сформований	Певний технологічний процес	- Акцент на організаційному процесі - Визначення організаційного процесу - Експертні оцінки - Програма навчання - Міжгрупова координація - Інтегроване управління ПЗ
Рівень 2: повторюваний	Менеджмент проекту і процес розподілу зобов'язань	- Планування програмного проекту - Відстеження програмного проекту - Управління субконтрактами з розробки ПЗ - Забезпечення якості ПЗ - Менеджмент конфігурації ПЗ - Управління вимогами
Рівень 1: початковий	Головні учасники проекту	- Розробка програмного продукту

Процес – це послідовність дій і завдань, які необхідно виконати для досягнення поставленої мети.

Здатність забезпечення якості складних програмних засобів – це діапазон результатів, які очікуються в результаті виконання даного технологічного процесу.

Наведемо споріднені моделі зрілості, які використовуються інститутом SEI при розробці, розповсюдженні та супроводі [13]:

CMMI – інтегрована модель CMM (CMM Integration);

SW-CMM – модель зрілості можливостей для ПЗ (Capability Maturity Models for Software);

P-CMM – модель зрілості персоналу (People Capability Maturity Model);

SA-CMM – модель зрілості характеристик у процесі придбання ПЗ (Software Acquisition Capability Maturity Model);

SE-CMM – модель зрілості характеристик інжинірингу систем (Systems Engineering Capability Maturity Model);

IPD-CMM – модель зрілості характеристик при розробці інтегрованого продукту (Intergrated Product Development Capability Maturity Model).

Американське товариство якості (American Society for Quality, ASQ)

Американське товариство якості, створене в 1946 році шляхом злиття кількох локальних товариств боротьби за якість, які були широко розповсюджені по всій території США, спочатку займалося наданням інформації про статистичний контроль якості випускникам технологічних вузів, які під час другої світової війни пройшли навчання з метою поліпшення і підтримки якості матеріалів оборонного значення.

Відтоді організація ASQ розрослася, і зараз до неї входять більше 1000 локальних організацій із загальною кількістю членів понад 130000 чоловік. Члени ASQ є ініціаторами більшості методів управління якістю, які сьогодні використовуються по всьому світі. До них належать: статистичний контроль процесу, вартість вимірювання і контролю якості, тотальне управління якістю і метод нульових дефектів (відсутність дефектів).

Організація ASQ бачить свою місію у сприянні забезпечення високої якості в результаті індивідуальної та організаційної діяльності по всьому світі шляхом надання можливостей для навчання, поліпшення якості та обміну знаннями. До її основних функцій відносять організацію процесу призначення національної премії у сфері якості імені Малькольма Балдріджа (Malcolm Baldrige National Quality Award) і сертифікацію фахівців, що забезпечують високу якість у процесі розробки програм.

Рис. 3.6. Емблема організації ASQ

Зазвичай при визначенні лауреатів національної премії якості імені Малькольма Балдріджа в ролі експертів виступають професіонали із забезпечення якості, що працюють у місцевих і державних організаціях, які вирішують питання забезпечення якості програмного забезпечення. Оскільки критерії оцінки

міняються щороку, навчання і перепідготовка експертів вимагає постійного контролю з боку ASQ.

Протягом багатьох років ASQ визначає перелік документів (Body of Knowledge, BOK) для сертифікації професіоналів із забезпечення якості в процесі програмного інжинірингу (Certification of Professionals in Software Quality Engineering), проводить іспити та сертифікує фахівців-професіоналів щодо забезпечення якості програмного забезпечення. Нижче наведені пропоновані ASQ види сертифікації:

- сертифікований співробітник з поліпшення якості (Certified Quality Improvement Associate, CQIA);
- сертифікований аудитор із забезпечення якості (Certified Quality Auditor, CQA);
- сертифікований інженер із забезпечення якості (Certified Quality Engineer, CQE);
- сертифікований інженер із забезпечення надійності (Certified Reliability Engineer, CRE);
- сертифікований фахівець із забезпечення якості (Certified Quality Technician, CQT);
- сертифікований технічний інспектор (Certified Mechanical Inspector, CMI);
- сертифікований менеджер із забезпечення якості (Certified Quality Manager, CQM);
- сертифікований інженер із забезпечення якості ПЗ (Certified Software Quality Engineer, CSQE).

Зазвичай місцеві відділення американського суспільства якості і міжнародні організації проводять іспити двічі на рік – у червні та грудні. В даному випадку йде мова про іспити «з відкритою книгою», тобто при підготовці відповіді дозволяється користуватися довідниковою літературою. У всіх програмах сертифікації акцент зроблений на певну складову якості. Для отримання сертифікату кожен кандидат повинен скласти письмовий іспит, у ході якого необхідно вибрати правильну відповідь з кількох пропонованих варіантів. Для керівників програмних проектів існує програма сертифікації CSQE (сертифікований інженер з якості ПЗ), де наведено необхідні

відомості, які повинен знати інженер, що займається забезпеченням якості програмного забезпечення. Екзамен сертифікації інженера із забезпечення якості ПЗ складається зі 160 питань, він розрахований на 4 години і приймається тільки англійською мовою.

Збірник документів CSQE (Body of Knowledge CSQE) охоплює широкий набір питань, які впливають на якість ПЗ при його розробці. Усі вони дуже важливі для менеджерів проектів, особливо ті, що стосуються управління проектом, а саме питання планування, відстежування і виконання програмного проекту.

Документ Body of Knowledge CSQE складається з восьми частин (рис. 3.7).

Рис. 3.7. Основи знань ASQ

Іншим важливим сервісом ASQ є видання щомісячних і щоквартальних спеціалізованих журналів. Нижче наведені їх назви. Для менеджерів проектів особливий інтерес являє журнал Software Quality Professional, який робить акцент на темах з CSQE Body of Knowledge. У таблиці 3.3 наводиться відповідний перелік публікацій американського суспільства боротьби за якість, що викликають зацікавленість у програмних інженерів.

Таблиця 3.3

ПУБЛІКАЦІЇ ASQ З УПРАВЛІННЯ ПРОГРАМНИМИ ПРОЕКТАМИ

Quality Progress	Включає всеосяжні статті, написані кваліфікованими фахівцями-практиками. Статті присвячені застосуванню інноваційних методів в таких сферах, як менеджмент знань, поліпшення процесу і організаційна поведінка
Quality Management Journal	Публікує важливі дослідження у сфері управління якістю, надає форум для обговорення досліджень з визначення якості, виконаних як академічними фахівцями, так і фахівцями-практиками
Quality Engineering	У статтях наведені приклади застосування науки якості до менеджменту і діючих процесів у державному управлінні, бізнесі та промисловості
Journal of Quality Technology	Публікує статті, в яких зроблений акцент на практичному застосуванні нових методів, наведені приклади-інструкції існуючих методів і результати минулих досліджень
Software Quality Professional	Пояснює читачам практику забезпечення якості при розробці ПЗ, як це визначено в збірці матеріалів (BOK) для програми ASQ «Сертифікований інженер із забезпечення якості ПЗ», яка довела свою ефективність у багатьох галузях промисловості
Technometrics	Видається спільно з американською статистичною асоціацією (American Statistical Association, ASA). Публікації присвячені проблемам розробки й застосування статистичних методів у фізиці, хімії та інженерних науках
The Informed Outlook	Щомісячний інформаційний бюлетень зі стандартів, в якому наводиться своєчасна і точна інформація з таких стандартів, як ISO 9000, ISO 14000, QS-9000, TL 9000 і AS 9100

Опишемо більш детально національну премію якості імені Малькольма Балдріджа як винагороду за впровадження високих стандартів у сферу виробництва товарів та послуг, у тому числі в області інженерії програмного забезпечення.

У відповідь на загрозу домінування зарубіжних товарів вищої якості уряд Сполучених Штатів ухвалив рішення сфокусувати увагу нації на якості як основній зброї в конкурентній боротьбі, що загострилася наприкінці 80-х років XX століття. Національна премія якості імені Малькольма Балдріджа (Malcolm Baldrige National Quality Award, MBNQA) була заснована після ухвалення 20 серпня 1987 року цивільного закону 100-107, що дає право на створення нового державно-приватного об'єднання, яке було назване Фондом з присудження національної премії за якість імені Малькольма Балдріджа. Створений у 1988 році, фонд сприяє популяризації премії та процесу її присудження.

Нагорода була названа на честь Малькольма Балдріджа, який займав пост міністра торгівлі Сполучених Штатів з 1981 по 1987 рік, аж до своєї смерті в результаті нещасного випадку під час родео. Завдяки своєму управлінському таланту, він зробив величезний внесок у довгострокове підвищення ефективності роботи уряду.

Американське суспільство якості надає допомогу Національному інституту стандартів і технологій (National Institute of Standards and Technology, NIST), розглядаючи документи, висунені на здобуття премії, готуючи документи для нагороди, висвітлюючи процес нагородження і надаючи зацікавленим особам інформацію, що стосується забезпечення якості. Експертів вибирають серед місцевих професіоналів у сфері забезпечення якості на рівні міста і штату.

Існує сім критеріїв відбору претендентів для нагородження Національною премією Малькольма Балдріджа. На рис. 3.8 показані взаємозв'язки між цими критеріями.

Рис. 3.8. Критерії присудження Національної премії якості імені Малькольма Балдріджа

Основні критерії присудження Національної премії якості імені Малькольма Балдріджа:

1) лідерство;

2) стратегічне планування;

3) орієнтація на споживача і ринок;

4) інформація та аналіз;

5) орієнтація на людські ресурси;

6) управління процесами;

7) кінцеві результати.

Ці області пов'язані одна з одною та утворюють систему, за допомогою якої лідерство приводить до результатів. Такі критерії, як лідерство, стратегічне планування і орієнтація на ринок і споживача визначають напрям і планування бізнесу, тоді як орієнтація на людські ресурси, управління процесами і кінцеві результати належать до виконавчої сторони бізнесу. А інформація та аналіз відіграють, в основному, «підтримуючу» роль.

Але ці сім критеріїв є ширшими поняттями, що включають ключові параметри, якими також керуються експерти, оцінюючи претендентів на отримання даної премії.

До цих параметрів відносять:

1) якість, орієнтовану на замовника;

2) лідерство;

3) безперервне вдосконалення;

4) активну участь всіх співробітників компанії, споживачів, постачальників;

5) швидке реагування на потреби споживачів та короткий часовий цикл;

6) якісний дизайн і запобігання дефектам;

7) довгострокові перспективи;

8) управління на основі фактів;

9) розвиток взаємовигідної співпраці;

10) публічну відповідальність.

Уміле управління проектами «пронизує» всю організацію (а не просто програмні проекти) і знаходить своє віддзеркалення в чіткому виконанні та управлінні на основі фактів. Керуючись вищепереліченими принципами відбору, експерти виставляють претенденту оцінку в балах. Номінанти премії, які набрали найвищу кількість балів, удостоюються її нагородою.

Міжнародна організація зі стандартизації (International Organization for Standardization, ISO)

Міжнародна організація зі стандартизації заснована в 1946 році. Вона неурядова, а її штаб-квартира знаходиться в Женеві (Швейцарія). В рамках цієї організації працюють фахівці з розробки національних стандартів з майже 130 країн. Місія організації полягає у сприянні розвитку всесвітньої уніфікованої стандартизації (головно, технічної) в різних сферах виробничої діяльності та надання послуг з метою полегшення міжнародного обміну товарами і послугами. Американський національний інститут стандартів (American National Standards Institute, ANSI) є членом цієї організації і представляє в ній Сполучені Штати Америки.

Організація ISO сприяє співпраці різних країн в інтелектуальній, науковій, технологічній і економічній сферах діяльності. Результатом роботи ISO є міжнародні угоди, що публікуються як міжнародні стандарти. Назва ISO не є абревіатурою – вона походить від старогрецького слова *isos*, що означає «рівний», «рівносильний».

Рис. 3.9. Емблема ISO

Організація ISO має високий ступінь децентралізації з великою кількістю технічних комітетів, підкомітетів і робочих груп, до складу яких входять майже 30000 кваліфікованих представників промисловості, науково-дослідних інститутів, урядових установ, організацій споживачів тощо.

Головним органом управління ISO є щорічна Генеральна Асамблея, що ухвалює стратегічні рішення, які стосуються розвитку всієї організації. Підготовкою матеріалів для ухвалення таких рішень займається Рада ISO, збори якої проходять двічі на рік. Безпосередньо розробкою стандартів займаються технічні комітети і підкомітети, в роботі яких беруть участь представники

зацікавлених країн. За розробку кожного документа в підкомітеті відповідає спеціально створювана для цього робоча група. Проекти міжнародних стандартів, прийняті технічними комітетами, розсилаються в національні організації для голосування; документ набуває статусу міжнародного стандарту, якщо за нього проголосувало не менше 75% членів, що брали участь у голосуванні.

На рис. 3.10 зображена структура ISO. Учасники ISO представлені 158 національними органами зі стандартизації, які мають такий статус: 103 комітет-членів, 46 членів-кореспондентів, 9 членів-абонентів. Технічні комітети представлені 3041 технічним органом, з них: 193 технічних комітети, 540 підкомітетів, 2244 робочих групи, 64 спеціальних дослідницьких групи. Значення абревіатур комітетів, наведених на рис. 3.10, такі: CASCO – Committee on conformity assessment (комітет із сертифікації); COPOLCO – Committee on consumer policy (комітет з політики із замовниками); DEVCO – Committee on developing country matters (комітет у справах з країнами, що розвиваються); REMCO – Committee on reference materials (комітет з еталонних матеріалів).

Працівники технічного секретаріату включають представників тридцяти семи національних комітетів, що мають статус «комітет-член», які надають адміністративні та технічні послуги для секретаріатів технічних комітетів і підкомісій. Приблизно 500 працівників долучаються на постійній основі. Центральний секретаріат у Женеві задіює на постійній основі 153 працівників. Двадцять три країни координують міжнародну діяльність ISO.

Для реалізації діяльності фінансованих секретаріатів технічних комітетів і підкомітетів (Technical Committee / Sub- Committee, TC/SC) щорічно плануються 120 млн. швейцарських франків. Причому 37 національних комітетів, що мають статус «комітет-член», підтримують секретаріати технічних комітетів і підкомітетів. 60% потрібних засобів надходять від оплати членських внесків і 40% складають доходи від публікацій та інших послуг.

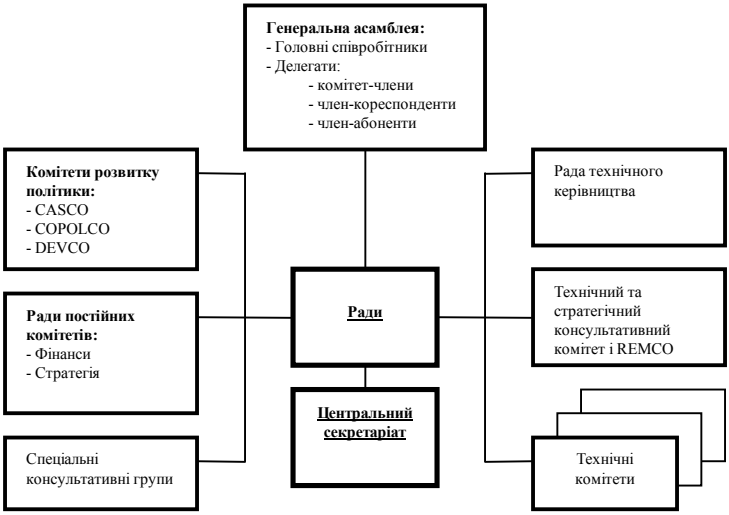

Рис. 3.10. Структура ISO

Міжнародний Стандарт – результат угоди між організаціями-учасниками ISO. Міжнародні стандарти розробляються технічними комітетами (ТС) і підкомітетами (SC) ISO в ході процесу, що складається з шести стадій (рис. 3.11):

1. Стадія заявки.
2. Попередня стадія.
3. Стадія розгляду в комітетах.
4. Стадія запиту.
5. Стадія схвалення.
6. Стадія публікації.

Якщо документ з певним ступенем зрілості доступний уже на початку процесу стандартизації (наприклад, стандарт, розроблений іншою організацією), то можна опускати деякі стадії. У «процедурі найкоротшого шляху» документ представляється організаціям-учасникам ISO (четверта стадія) – безпосередньо для схвалення як проект міжнародного стандарту (Draft International Standard, DIS). Якщо документ був розроблений міжнародною організацією зі стандартизації та визнаний Радою ISO (п’ята стадія) як завершальний проект міжнародного стандарту (Final DIS, FDIS), то він не проходить

через попередні стадії. На рис. 3.11 наведені різні маршрути затвердження міжнародних стандартів і результати роботи на кожній стадії розгляду.

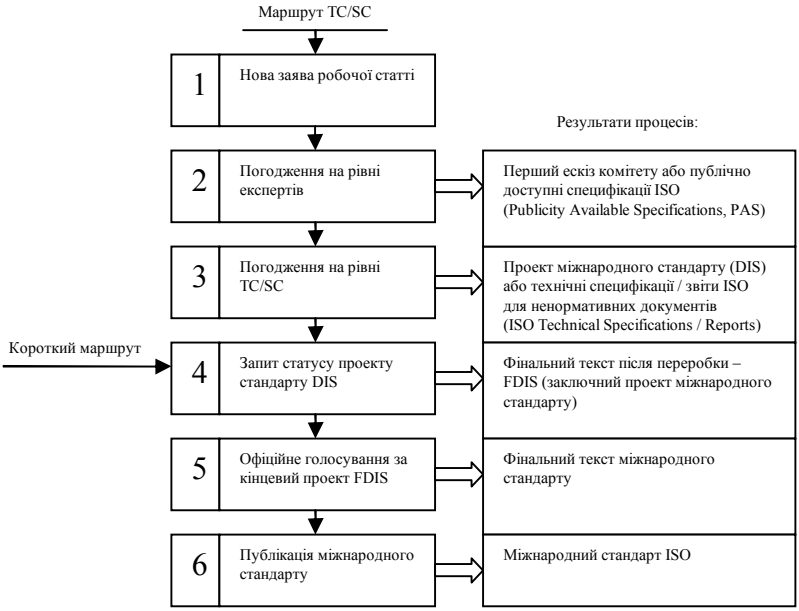

Рис. 3.11. Схематичне зображення результату процесів ухвалення стандартів ISO

Особливий інтерес для інженерів з розробки програмного забезпечення являє серія стандартів якості ISO 9000, прийнятих у 1987 році. Серія ISO 9000 була затверджена як серія Q90-Q94 Американського товариства якості ANSI; ISO 9001 – стандарт Великобританії BS5750 частина 1 і європейський стандарт EN29001. На даний час в основному використовують серію стандартів ISO, переглянутих у 1994 році.

Стандарти ISO 9000 – це набір міжнародних стандартів з управління якістю і керівних принципів для застосування в багатьох видах промисловості та бізнесу. З часу першої публікації в 1987 році ці стандарти завоювали глобальну репутацію як основу для створення мінімальних стандартів, що визначають функціонування систем управління якістю (Quality Management

Systems, QMS). Три з нинішніх стандартів, ISO 9001, ISO 9002 та ISO 9003, широко використовують як базу для виконання незалежної сертифікації системи якості. Ухвалення стандартів привело до сертифікації більше 200000 організацій по всьому світу, багато організацій знаходяться в процесі введення і виконання систем управління якістю ISO 9000. Стандарти ISO 9000 містять пункти з описом характеристик, які повинна мати система управління якістю.

Набір стандартів ISO 9000 редакції 1994 року застосовується по відношенню до різних видів бізнесу, залежно від обсягу продукції, об'єму постачань і тривалості циклу технічної підтримки. Похідні стандарти ISO 9000 були сформульовані для певних галузей промисловості. Наприклад, стандарти QS 9000 застосовуються в автомобільній промисловості. Стандарти QS 9000 фактично відповідають стандарту ISO 9000, в який внесені доповнення, що стосуються Американської промислово-автомобільної групи (American Industry Automotive Group, AIAG). Усі найпопулярніші стандарти ISO для програмних інженерів наведені в табл. 3.4.

Таблиця 3.4

СТАНДАРТИ ISO

ISO 9000-1	Керівні положення управління якістю і використання стандартів контролю якості	У стандарті пояснюються фундаментальні поняття щодо забезпечення якості, а також наведено керівництво для вибору і використання стандартів ISO 9001, 9002, 9003 і 9004
ISO 9000-3	Стандарти якості Керівні положення для розробки ПЗ	У стандарті описано застосування стандартів ISO 9000 для ПЗ
ISO 9001	Модель контролю якості при проектуванні, розробки, виробництві, установці та обслуговуванні	Стандарти ділової діяльності, використовувані на всіх фазах життєвого циклу розробки і супроводу продукту
ISO 9002	Модель контролю якості при виробництві, установці і обслуговуванні	Стандарти ділової діяльності, використовувані на всіх фазах життєвого циклу продукту, за винятком розробки
ISO 9003	Модель контролю якості при остаточній перевірці і тестуванні	Стандарти для бізнесу, використовувані на всіх фазах життєвого циклу продукту, за винятком проектування і розробки, а також виробництва
ISO 9004-1	Керівні положення менеджменту якості і елементів системи якості	

Наведемо для прикладу двадцять пунктів стандарту систем забезпечення якості ISO 9001 (редакція 1996 року):

1. Відповідальність менеджменту.
2. Система забезпечення якості.
3. Оцінка контракту.
4. Управління розробкою проекту.
5. Дії з управління документацією та даними.
6. Закупівля продукції.
7. Управління продуктом, що поставляється споживачу.
8. Ідентифікація продукту і можливість відстежування.
9. Управління процесом.
10. Перевірка і тестування.
11. Контрольне, вимірювальне і випробувальне устаткування.
12. Статус процесів перевірки і тестування.
13. Управління продуктами, що не відповідають вимогам.
14. Корегуючі та профілактичні дії.
15. Вантажні роботи, зберігання, упаковка і постачання.
16. Реєстрація даних про якість.
17. Внутрішня перевірка якості.
18. Підготовка кадрів.
19. Технічне обслуговування.
20. Статистичні методи.

Американський національний інститут стандартів (American National Standards Institute, ANSI)

Американський національний інститут стандартів з 1918 року виконує функції адміністратора і координатора системи добровільної стандартизації. Він був заснований п’ятьма інженерними суспільствами і трьома державними установами. На даний час інститут залишається приватною некомерційною організацією, яка підтримується різними приватними і державними організаціями на спонсорській основі.

Рис. 3.12. Емблема ANSI

Місія ANSI: глобальне сприяння політиці стандартизації Сполучених Штатів Америки.

Цілі ANSI:

- створення глобальних стандартів, які відображають інтереси Сполучених Штатів Америки;
- розробка американських стандартів, які використовуються за кордоном;
- зміцнення позицій США (політичних і технічних), які прийняті міжнародними і регіональними організаціями із забезпечення стандартів;
- забезпечення дотримання міжнародних стандартів, прийнятих як національні, коли це необхідно, для задоволення потреб суспільства споживачів.

Інститут ANSI самостійно не розробляє національні стандарти. Він сприяє досягненню консенсусу при їх розробці. Цей інститут є одним з п'яти постійних членів адміністративної Ради Міжнародної організації зі стандартизації.

Особливий інтерес для менеджерів проектів з розробки ПЗ являють деякі стандарти, які ANSI допомагає створювати спільно з іншими організаціями-розробниками. У табл. 3.5 подана інформація з технологічних стандартів.

Таблиця 3.5

ДОКУМЕНТИ, ОПУБЛІКОВАНІ ANSI
ДЛЯ ПРОГРАМНИХ ІНЖЕНЕРІВ

Номер документа	Назва документа
ISO/IEC12207:1995	Інформаційні технології – процеси життєвого циклу розробки програмного забезпечення (ПЗ)
ISO/IEC 12207.0–1996	Стандарт з інформаційних технологій – процеси життєвого циклу розробки ПЗ
ISO/IEC 12207.1–1997	Керівництво з інформаційних технологій – процеси життєвого циклу ПЗ; дані життєвого циклу
ISO/IEC 12207.2–1997	Керівництво з інформаційних технологій – процеси життєвого циклу ПЗ; розгляд процесу виконання
ISO/IEC TR 16326:1999	Програмний інжиніринг – керівництво для застосування ISO/IEC 12207 в менеджменті проекту
ISO/IEC TR 15271:1998	Інформаційні технології – керівництво по ISO/IEC 12207 (процеси життєвого циклу розробки ПЗ)
IEEE 1012a–1998	Стандарт IEEE для атестації та верифікації ПЗ – зміст наведений в IEEE 12207.1

Примітка: IEC – International Electrotechnical Commission – міжнародна електротехнічна комісія.

Американський національний інститут стандартів і технології (National Institute of Standards and Technology, NIST)

Ще одним національним органом зі стандартизації в США є Американський національний інститут стандартів і технології (NIST). NIST – неурядова некомерційна організація, яка координує роботи з добровільної стандартизації в приватному секторі економіки. Інститут здійснює керування діяльністю організацій-розробників стандартів та приймає рішення щодо надання статусу національного стандарту (якщо в ньому зацікавлені різні фірми і стандарт набуває міжгалузевого характеру).

Рис. 3.13. Емблема NIST

NIST не розробляє стандарти, але є єдиною організацією в США, що приймає (затверджує) національні стандарти. Це відповідає основному завданню NIST – сприяння розв'язанню проблем, що мають загальнодержавне значення (економія енергоресурсів, захист навколишнього середовища, забезпечення безпеки життя людей і умов виробництва).

Інститут розробляє цільові програми. Програмно-цільове планування охоплює виробництво і транспортування палива, постачання електроенергією, застосування ядерної, сонячної та інших видів енергії.

Значно менше уваги приділяється розробці стандартів на готову продукцію, оскільки в цій сфері діють фірмові нормативні документи.

Національні (федеральні) стандарти містять обов'язкові до виконання вимоги, що стосуються в основному аспектів безпеки. Поряд з обов'язковими федеральними стандартами в США діють технічні регламенти, які затверджуються органами державного управління.

Розробляють федеральні стандарти авторитетні організації, акредитовані Американським національним інститутом стандартів (ANSI).

На сьогоднішній день членами NIST є понад 1200 фірм, понад 250 виробничих і торговельних компаній, науково-технічних та інженерних товариств.

Структура фінансування інституту засвідчує його незалежність від державного бюджету: 37 % складають внески організацій-членів (6 %) і фірм-членів (31 %); розробка спеціальних програм на замовлення зацікавлених організацій – 15 %; надходження від продажу різних видань – 48 %.

Очолює інститут NIST Рада директорів. До її функцій відносять: вибори президента, трьох віце-президентів, виконавчого віце-президента і виконавчого комітету. Останній керує інститутом у період між засіданнями Ради директорів і контролює виконання бюджету. Рада директорів планує роботу інституту, розробляє пріоритетні напрямки стандартизації тощо.

Раді директорів підпорядковуються: Рада організацій-членів, Рада компаній-членів і Рада із захисту інтересів споживачів. Рада організацій-членів складається з представників різних спілок, об'єднань, а також представників федерального уряду або уряду окремих штатів (при їх зацікавленості). Рада організацій затверджує національні стандарти; аналізує прийняті іншими організаціями стандарти щодо їх прийнятності як національних; планує оновлення та створення нових нормативних документів; керує участю країни в міжнародній стандартизації. Цей орган має по одному представнику в двох інших Радах.

Рада компаній-членів включає представників усіх зацікавлених фірм із різних галузей економіки. Цей орган має п'ять представників у Раді захисту прав споживачів і одного – в Раді організацій. Рада компаній розробляє програми стандартизації з урахуванням галузевих інтересів; залучає нових членів; визначає потреби в нових національних або міжнародних стандартах; займається дослідницькою роботою, спрямованою на підвищення ефективності виробництва і торгівлі через стандартизацію.

У Раді захисту інтересів споживачів представлені по п’ять членів від двох інших рад, п’ять представників державних органів, представники споживчих спілок. Основні завдання цієї Ради визначаються цілями її роботи – стежити за дотриманням інтересів споживачів у національних стандартах. Рада виявляє сфери, де необхідно домагатися поліпшення якості товарів (послуг) за допомогою стандартизації; займається роз’яснювальною діяльністю, пропагуючи роль стандартизації в розвитку економіки і захисті прав та інтересів споживачів. Рада широко залучає самих споживачів до перевірок якості та безпеки товарів на місцях. Для цього створюються добровільні контролюючі групи, які під керівництвом представників Ради збирають інформацію і надсилають відгуки та рекомендації. Ці коментарі та рекомендації розглядаються і нерідко враховуються при оновленні або створенні нового нормативного документа.

Інститут управління проектами (Project Management Institute, PMI)

Інститут управління проектами заснований у 1969 році у Філадельфії (штат Пенсільванія, США) з метою надання допомоги фахівцям, що працюють над розробкою й виконанням проектів, а також надають інформацію про стадії цього процесу інженерам-практикам. PMI об’єднує понад 100000 у 125 країнах світу. PMI – некомерційна організація, що просуває, пропагує та розвиває проектний менеджмент у різних країнах. PMI розробляє стандарти проектного менеджменту, займається підвищенням кваліфікації фахівців.

Рис. 3.14. Емблема Інституту управління проектами (PMI)

Одним з найголовніших результатів роботи PMI є розробка збірника «Основи знань у області менеджменту проектів Інституту управління проектами». Даний збірник є результатом неформальної співпраці PMI із функціональними менеджерами з

будівельної, військової та аерокосмічної галузей промисловості у 60–70-х роках XX століття. У 1987 році Інститут управління проектами склав загальний збірник, опублікований під назвою «Project Management Body of Knowledge». Тут наводився повний опис матеріалів, необхідних для менеджерів проектів у будь-якій галузі промисловості. У 1996 році в цей документ були внесені зміни, і його перейменували в «A Guide to the Project Management Body of Knowledge» (на даний час він відомий під назвою PMBOK Guide). Основний висновок, який можна зробити після його вивчення, полягає в тому, що фактичний обсяг знань, яким необхідно володіти менеджеру проекту, величезний і його не можна адекватно представити в одній книзі, що складається зі 176 сторінок. У 2000 році була опублікована остання редакція цього документа з метою подальшого вдосконалення і розширення деяких визначень.

У 1998 році Американський національний інститут стандартів ANSI офіційно визнав за Інститутом управління проектами PMI статус акредитованого розробника стандартів (Accredited Standard Developer). У жовтні 1999 року документ PMI «A Guide to the Project Management Body of Knowledge» (PMBOK Guide) був затверджений як Американський національний стандарт. У лютому 1999 року Інститут інженерів з електротехніки та електроніки (IEEE) прийняв його як керівництво, що дозволяє реалізувати управління проектами. Стандарт у сфері управління розробкою програмних проектів SWEBOK, прийнятий асоціаціями IEEE-CS та ACM, заснований на положеннях збірника знань PMBOK.

У документі PMBOK представлена інформація з 9 розділів (сфери знань), що є основою набору з управління проектами. Ці сфери показані на рис. 3.15.

Ще однією сферою діяльності Інституту управління проектами є сертифікація за програмою «Професіонали у області менеджменту проектів» (PMI's Project Management Professional (PMP) Certification). При цьому необов'язково бути менеджером програмного проекту, але наявність сертифікату, звичайно ж, не завадить в умовах жорсткої конкуренції на ринку праці.

Починаючи з 1984 року Інститут PMI провів сертифікацію більше 10000 практиків-професіоналів по всьому світу згідно з програмою PMP Certification. Кількість сертифікованих фахівців продовжує швидко рости, оскільки, з одного боку, люди чимраз більше усвідомлюють важливість високого ступеня професіоналізму, а з іншого боку, зростання зумовлюється спрощенням процесу сертифікації, яке постійно удосконалюється PMI з 1999 року.

Рис. 3.15. Основи знань у сфері управління проектами [23]

Сертифікація вимагає від фахівця відповідності таким мінімальним стандартам:

- в освіті – наявність документа, що підтверджує базову професійну освіту (наприклад, диплом бакалавра);
- у досвіді – перебування на посаді, яка прямо або опосередковано стосується сфери управління проектами (наприклад, 4500 годин оплачуваної роботи);
- в обслуговуванні – демонстрація результату деякої роботи із застосуванням принципів управління проектами (наприклад, неоплачувана творча робота у відділі).

Крім того, фахівець із розробки проектів, складаючи професійний іспит, повинен показати добре знання документа «Project Management Body of Knowledge».

Окрім розглянутих організацій, які підтримують професійну розробку програмного забезпечення та забезпечують високі стандарти цього процесу, існує велика кількість організацій, основним завданням яких є не розробка стандартів, а їх просування. Як приклад можна навести:

- Асоціацію професіоналів у галузі інформаційних технологій (Association of Information Technology Professionals, AITP);
- Асоціацію в галузі інформаційних систем (Association for Information Systems, AIS).

Цілі і завдання асоціацій багато в чому схожі. Зокрема, визначені такі важливі завдання: проведення зустрічей, видання книг, журналів та інших матеріалів; співпраця з іншими організаціями, зацікавленими у просуванні і практичному застосуванні інформаційних систем; стимулювання досліджень; просування передових професійних стандартів і поліпшення професіоналізму фахівців. Організація досліджень, навчання фахівців та їх сертифікація є тим середовищем, що безперервно розвивається, в якому формується єдиний світогляд у сфері інформаційних технологій та інформаційних систем.

У світі існує велика кількість організацій, які активно працюють над розробкою програмних продуктів, удосконаленням інформаційних технологій, інформаційних систем, інформаційної безпеки, менеджменту, управління інформаційними системами і даними тощо. При цьому значуща розробка будь-якої організації може бути розвинена до рівня стандарту та стати еталоном.

Контрольні запитання та завдання

1. Чим зумовлена необхідність виникнення професійних асоціацій та співтовариств?
2. Опишіть у вигляді короткого реферату історію становлення та основні напрямки діяльності професійної асоціації програмних інженерів (наприклад: British Computer Society, Australian Computer Society, Software Engineering Society, Computer Professionals for Social Responsibility тощо). Зверніть увагу на конференції та семінари, які проводять асоціації, а також наукову та навчально-методичну діяльність у професійній сфері.

РОЗДІЛ 4. ПРОФЕСІЙНА ОСВІТА В ГАЛУЗІ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Уведення професіоналізму в програму навчання зумовлена потребами реального світу, такими як зростаючий попит на високоякісні продукти, зростаючий рівень відповідальності розробників програмного забезпечення та необхідність постійного підвищення кваліфікації після закінчення начального закладу [4, 5]. У більшості випадків студенти вступають до вузу без розуміння цих питань, що створює труднощі для викладачів і для майбутніх працедавців. Чим більше професійної практики одержують студенти, тим привабливіше стає для них навчання й реальніше майбутня робота. Відповідно, професійна практика в учбовій програмі може служити своєрідним каталізатором пробудження і підтримки інтересу студентів до інформатики.

На сьогоднішній день досить актуальне завдання – зробити навчання професійному підходу до програмної інженерії рівноправною складовою учбової програми. Розуміння важливості професіоналізму обов'язкове для більшості студентів, оскільки основна частина випускників реалізує себе в індустрії.

І приватний, і державний сектор, безумовно, зацікавлені в навчанні професіоналізму. Студенти, знайомі з реаліями професійної діяльності, розуміють значення навичок ефективного спілкування з колегами і клієнтами, докладають усіх зусиль для того, щоб робити свою роботу якісно, прагнуть до постійного підвищення своєї кваліфікації та вдосконалення власної фірми.

На даний час практично в усіх великих вузах розвивається інноваційна діяльність, що базується на комерціалізації наукоємнісних технологій. Відомо, що фундаментальні наукові дослідження – одне з головних джерел розвитку науково-педагогічних шкіл. Проте вузи відчувають серйозний дефіцит бюджетного фінансування і велику конкуренцію в конкурсах на отримання коштів з фондів. Вузівська громадськість стурбована цією ситуацією і гостро ставить питання про збільшення бюджетних асигнувань на вузівську науку. У цих умовах при багатьох кафедрах створюються самостійні інжинірингові,

консалтингові та випробувальні компанії, які займаються консультуванням продавців, покупців, виробників з широкого кола питань господарської діяльності підприємств, організацій, установ. Вони надають послуги з прогнозування ринку товарів, ліцензій, ноу-хау, світових цін на товари, дають оцінку торгово-політичних умов, розробляють техніко-економічні обґрунтування на об'єкти міжнародної співпраці тощо. Мета інноваційної діяльності не зводиться тільки до отримання прибутку. Її вирішальна роль полягає у використанні інновацій для вдосконалення інженерної освіти шляхом доручення до цієї діяльності студентів. Отже, у студентів є можливість одержати реальні навички підприємницької діяльності, розширюються можливості тренінгу. Крім того, інноваційна діяльність на кафедрах – ефективна форма підвищення кваліфікації викладацького складу.

Для системи інженерної освіти досить актуальна і в цьому сенсі інноваційна проблема формування концепції відкритого простору інженерної освіти. При цьому найбільш важливі два положення [4]:

1. Свобода професійної діяльності.
2. Вдосконалення інженерної освіти.

Свобода професійної діяльності є дуже важливим моментом при формування концепції відкритого простору інженерної освіти. Сучасний світ характеризується регулярними змінами економіки, великим потоком технологічних інновацій і значною динамікою робочої сили як у територіальному, так і в освітньому аспектах.

Оскільки інженери є однією з найбільш динамічних соціальних груп, то увага інженерних співтовариств до проблеми переміщень інженерів постійно росте. Кожен інженер повинен мати впевненість у тому, що його освіта і рівень компетентності, одержані в одній країні, не «девальвуються» при переїзді в інші країни. При цьому повинні бути певні гарантії в тому, що він зможе продовжувати своє професійне зростання і службову кар'єру, і від нього не зажадають підтвердження його кваліфікації

або повторення яких-небудь етапів навчання. Для розв'язання цієї проблеми можливі два шляхи [4]:

1. Державна нострифікація (ознака еквівалентності) дипломів інженерної освіти.

2. Співпраця інженерних співтовариств різних країн, що передбачає присудження на певних умовах інженерного звання і надання сертифікату, які підтверджують якість підготовки.

Забезпечення професійного визнання – це насамперед питання якості підготовки. Тому людина, що вибирає інженерну професію, повинна мати повну інформацію про те, які вузи забезпечують високий рівень підготовки, на яку кар'єру може розраховувати їх випускник.

У цьому плані можна послатися на міжнародний досвід Європейської федерації національних асоціацій інженерів (Fédération Européenne d'Associations Nationales d'Ingénieurs, FEANI), яка підтримує всіма доступними їй засобами на території своїх членів єдине інженерне звання, що підтверджується сертифікатом «єдиний диплом інженера». Особи з таким сертифікатом вносяться в регулярно публікований реєстр, який є джерелом інформації про рівень підготовки інженера для його потенційного працедавця в будь-якій з держав – членів федерації.

Вдосконалення інженерної освіти включає постійну діяльність з оновлення змісту, а також форм організації та методик процесу навчання (підготовки) інженерів, необхідних для досягнення і підтримки рівня інженерної освіти, що відповідає світовим стандартам.

Концепція відкритого простору інженерної освіти націлена не просто на збереження, відновлення і розвиток зв'язків у сфері освіти, які, наприклад, були в СРСР, але і на приєднання до європейських і світових інтеграційних процесів. Звідси випливає необхідність у дуже строгому відборі потенційних учасників відкритого простору. Досвід FEANI та акредитованих вузів США показує важливість розробки систем вимог (стандартів) як до вузівських програм, так і до професійної діяльності інженера. Наприклад, вимоги до «євроінженера» стосуються не тільки знань і спеціальних навичок. Тут також високі вимоги до:

1) рівня відповідальності;

2) рівня інноваційності;

3) професійної етичності інженера;

4) здатності працювати над міждисциплінарними проектами;

5) екологічної компетентності;

6) вільного володіння однією з європейських мов, окрім рідної.

Процедури експертизи, що проводяться при акредитації і сертифікації, а також отримані результати можуть стати основою для модифікації змісту існуючих програм і поштовхом до розробки нових.

У розробці стандартів активну участь беруть вузи, професійні інженерні асоціації, державні органи, наукові організації тощо.

Усі стандарти і список акредитованих вузів, як правило, доступні всім зацікавленим суб'єктам. Важливою складовою у вдосконаленні інженерної освіти є такі форми, як зв'язки інженерних вузів, зв'язки вузів з наукою і виробництвом, проведення сумісних досліджень, підвищення кваліфікації і перепідготовка кадрів.

Співпраця інженерних вузів, професійних співтовариств, наукових і промислових організацій у концепції відкритого простору освіти виявляється у двох напрямах: як мета і як умова.

У першому випадку співпраця означає визнання першорядної ваги інженерної освіти та її зв'язків з науковим і виробничим сектором економіки.

У другому випадку відображаються такі реалії:

1) високий ступінь диференціації науки і виробництва;

2) неоднорідність у розміщенні освітніх ресурсів, що передбачає постійний розвиток внутрішніх і зовнішніх зв'язків освіти;

3) неможливість повноцінної інженерної підготовки в рамках тільки учбового процесу (наприклад, для отримання диплома «європейського інженера» необхідний досвід роботи зі своєї спеціальності в умовах сучасного виробництва);

4) необхідність взаємного підвищення кваліфікації фахівців виробництва на базі вузів і викладачів – на базі передового виробництва.

4.1. Загальні питання інженерної освіти

Система інженерної освіти постійно знаходиться у полі зору провідних фахівців світу [3, 4], велика кількість яких є членами Європейського товариства інженерної освіти, де акумулюється основний досвід, накопичений у світі провідними системами освіти, – британської, американської, німецької, італійської, японської. Окрім них, сьогодні заявляють про себе порівняно молоді системи інженерної освіти країн, що розвиваються, створені на базі західноєвропейських і північноамериканських науково-педагогічних шкіл. Не останнє місце займають країни східної Європи, що мають власні вікові традиції інженерної освіти. Останніми роками вони вступають у міжнародні зв'язки з країнами заходу на основі фінансування зі спеціальних міжнародних освітніх фондів. На різноманітних міжнародних конференціях, присвячених проблемам інженерної освіти, періодично обговорюються як концептуальні проблеми інженерної освіти, так і конкретніші аспекти підготовки інженерних кадрів.

До концептуальних проблем інженерної освіти, які на сьогодні активно обговорюються, можна віднести такі [4]:

1. Яким повинен бути інженер прийдешнього століття? Які вимоги до нього потрібно пред'являти у світлі швидкого і значного ускладнення технологій, техніки, соціально-економічних змін в окремих країнах і світовій економіко-політичній системі?

Інженер, як творець нової складної техніки, принципово не може бути вузьким фахівцем. Його діяльність пов'язана з міждисциплінарним характером роботи. Інженер XXI століття повинен досконало володіти інформаційними технологіями, у сфері яких відбуваються значні зміни за рахунок постійної наростаючої потужності комп'ютерних систем. Він має глибоко розуміти екологічні проблеми не тільки з погляду вже завданого навколишньому середовищу збитку, але і з погляду прогнозування наслідків діяльності інженерного співтовариства. Логіка розвитку суспільства показує, що інженери у XXI столітті повинні бути ширше долучені до управління наукою і

технологіями, до розв'язання різних соціально-економічних проблем. Інженерна справа стане свого роду гуманітарною діяльністю.

2. Як радикально змінити саму систему інженерної освіти, щоб врахувати міждисциплінарну природу інженерної діяльності, її наростаючу складність і відповідальність перед цивілізацією? Як підвищити її ефективність з погляду поліпшення якості життєвих умов людей у глобальному масштабі?

У публікаціях учених, що займаються цією проблемою, наголошується, що радикальна реформа інженерної освіти неминуча. Жодна з існуючих систем не задовольняє потреб тих соціально-економічних інститутів, які вона обслуговує.

3. Яким повинен бути професорсько-викладацький склад?

Наголошується, що професорсько-викладацький склад – головний ресурс освітньої системи. Як правило, він має потужний фундамент у своїй сфері спеціалізації та дуже важко сприймає необхідність кардинальних змін у системі інженерної освіти в цілому (консерватизм освіти). Причини цього – в складному, системному характері проблеми реформи. Базис, стрижень реформи полягає в тому, що жоден з одержаних рівнів інженерної освіти не є його закінченням. Реформа повинна бути безперервною, продовжуватися протягом усієї професійної діяльності. Однаковою мірою ця теза стосується й професорсько-викладацького складу. У багатьох країнах ведуться пошуки нової системи, що стимулює підвищення кваліфікації викладачів, розвиток їх ерудиції, розширення сфери компетентності, оскільки викладач повинен володіти тими якостями, які він прагне прищепити своїм студентам. У деяких країнах (наприклад, в Австралії) в технічних вузах уведена нова спеціальність «Інженерна освіта». Вона розрахована на осіб, що мають інженерну освіту, й охочих стати викладачами не тільки у вузах, але і в промисловості, а також у низці освітніх центрів. Існують ідеї через кожні 4 – 5 років (так званий «період напіврозпаду технічних знань») радикально оновлювати навчальні курси або переключатися на викладання інших дисциплін.

4. Якими повинні бути освітньо-професійні програми та технології навчання за ними?

Загальне визнання сьогодні одержала диверсифікована (тобто різносторонньо розвинена, багатогалузева) система багаторівневої безперервної освіти, в якій домінує положення про те, що базова вища інженерна освіта не повинна бути уніфікованою ні за якими істотними ознаками (регіональними, галузевими, часовими тощо). Тому учбові програми повинні бути максимально гнучкими, такими, що дають велику свободу вибору для всіх споживачів системи інженерної освіти.

5. Як оцінювати якість інженерної освіти та керувати нею?

Ця проблема дуже актуальна для світової системи освіти. Вона розуміється системно в широкому плані та базується на концепції інтегрованої якості, широко використовуваній у менеджменті (сукупність засобів, принципів, методів і форм управління виробництвом для досягнення максимальної ефективності його роботи). Головний критерій оцінки якості – задоволення потреб споживачів у кінцевому продукті. Це означає, що оцінка якості повинна даватися не у самому кінці освітнього процесу, а проводитися безперервно. Необхідно враховувати не тільки рівень знань, умінь, навичок студентів і випускників, але також вплив зовнішніх і внутрішніх чинників на якість інженерної освіти. Це підвищить престиж інженерної професії, поліпшить ставлення молоді до інженерної кар'єри, підніме рівень роботи середньої школи, рівень діяльності вузів на всіх напрямках і якість викладання.

Завдання підвищення якості інженерної освіти реалізується як урядом, так і неурядовими організаціями й вузами. Важливим моментом є раціональний розподіл і використання бюджетних і позабюджетних коштів для підтримки і розвитку інженерної освіти. В умовах ринкової економіки до системи інженерної освіти може бути застосований підхід, заснований на обліку потреб тих, кого обслуговує система освіти, тобто:

1) молодих людей, що починають свою кар'єру;

2) людей, що вже почали її і хочуть для більшого успіху здобути нову професійну освіту або підвищити кваліфікацію;

3) промислові і комерційні підприємства, яким потрібні як високоосвічені фахівці, так і фахівці для рядової інженерно-технічної роботи;

4) націю, державу, зацікавлених у зростанні національного багатства країни за рахунок інженерної діяльності і розвитку державної інфраструктури.

Виникає питання: чи задоволені споживачі діяльністю системи освіти? Відповідь очевидна: ні! Звідси – головна мотивація необхідності радикальної реформи.

Проаналізуємо причини цієї незадоволеності.

1. Незадоволеність молодих людей виявляється в тому, що інтерес до інженерної освіти впав по всьому світі. Багато молодих фахівців після отримання освіти відмовляються від інженерної кар'єри або незадоволені своєю долею.

2. Промисловість також не задоволена випускниками вузів, оскільки вона потребує швидше інженерів широкого профілю, ніж вузьких фахівців. Її не влаштовує некомунікабельність випускників, відсутність у них управлінських навичок і мотивацій. На жаль, часто висококваліфіковані інженери використовуються як «гвинтики» налагодженого технологічного процесу та виконують монотонну роботу.

3. Є велика кількість людей, які також не задоволені діяльністю інженерів, вважаючи їх ворогами навколишнього середовища та відповідальними за техногенні катастрофи, що періодично відбуваються в різних куточках світу. Держави фінансують інженерну освіту, виходячи не з національних інтересів, а з контингенту студентів інженерних спеціальностей, що формується, переважно, стихійно.

У зв'язку з окресленими проблемами в галузі професійної інженерної освіти виникає усвідомлення необхідності її реформування.

На основі аналізу соціальних та промислових потреб інженерної діяльності викристалізовуються основні напрями реформи відповідної професійної освіти, які передбачається проводити держструктурами, неурядовими органами і вузами.

Головні з них:

1) гуманізація інженерної справи й наближення її до гуманітарної діяльності;

2) поворот інженерної освіти в бік екології, економіки, менеджменту, соціології, психології;

3) надання інженерній освіті максимальної гнучкості для швидкого реагування на зміну соціально-економічних потреб.

Один зі шляхів надання гнучкості – підготовка в рамках одного вузу інженерів різного рівня кваліфікації і функціонального призначення.

У Великобританії, наприклад, є три різновиди інженерів:

1) інженер-технік;

2) інкорпоративний (зареєстрований) інженер;

3) професійний (дипломований) інженер.

При цьому вибудовуються три моделі діяльності інженерів:

1) інновації (нововведення);

2) виробництво;

3) обслуговування.

Кожна з них передбачає виконання різних функцій (табл. 4.1).

Таблиця 4.1

ФУНКЦІЇ ІНЖЕНЕРІВ

Інноваційна діяльність	Дослідження, розробка і проектування
Виробнича діяльність	Управління виробництвом, виробничі системи, управління проектуванням
Обслуговуюча діяльність	Інженерний маркетинг, обслуговування устаткування, управління якістю, випробування і вимірювання

Вимоги до інженерів розділяються на кілька груп (табл. 4.2).

Таблиця 4.2

ВИМОГИ ДО ІНЖЕНЕРІВ

Інженерні знання	Знання специфічних інженерних технологій, широкі знання у сфері інженерної справи та ерудиція
Творчі здібності	Здатність застосовувати теорію на практиці, творчі навички і уява, здатність розв’язувати проблеми
Особисті якості	Здатність до кооперації і роботи в групах, комунікабельність (усна і письмова), здатність формувати завдання, рішучість у досягненні мети

На основі аналізу потреб формуються стандарти в інженерній освіті. У Великобританії, наприклад, ці стандарти відображають:

1) фундаментальні знання, потрібні для досягнення ефективних результатів у роботі;

2) здатність переносити компетентність з одного робочого місця на інше;

3) здатність конструктивно реагувати на зміни, що піддаються передбаченню, в технології, методах роботи на ринку, сфері трудової зайнятості.

Одним з основних критеріїв оптимальності стандартів навчання в сучасному вузі є вимоги, що ставляться до випускників інженерно-технічних спеціальностей вузів з боку підприємців-працедавців. Як правило, вузи при підготовці інженерних кадрів спираються на державні освітні стандарти. Проте останнім часом однією з гострих проблем стала невідповідність знань, умінь і навичок молодих інженерів тим вимогам, які диктує сфера виробництва. Це пов'язано з такими причинами:

1) невідповідністю вимог освітніх стандартів і вимог, що пред'являються до професійно-кваліфікаційних характеристик;

2) обмеженням норм часу викладання професійних дисциплін;

3) використанням традиційних форм і методів навчання;

4) відсутністю здійснення проміжних форм контролю реальних знань, умінь і навичок (компетенцій) підприємствами-працедавцями;

5) відсутність ситуативних тренінгів для психологічної адаптації учнів до виробничого й учбового середовища вузу.

Крім того, при навчанні в технічному вузі практично не враховується перспектива подальшої діяльності тих молодих людей, які надалі займатимуся науково-педагогічною діяльністю.

З метою вирішення деяких зі згаданих вище проблем багато учених у своїх науково-педагогічних дослідженнях намагаються створити різні моделі інженерних фахівців, заснованих на різних педагогічних методах.

Тією чи іншою мірою всі існуючі моделі інженерів включають такі параметри:

- вимоги до фахівця, що пред'являються його робочим місцем і характером розв'язуваних виробничих задач;

- необхідні для їх актуалізації знання і уміння;

- специфічні соціальні і психологічні якості особи, що забезпечують ефективність реалізації перших двох аспектів її діяльності.

Останнім часом однією з основних вимог до випускника будь-якого ступеня навчання інженерно-технічних вузів є необхідна професійна компетентність, яка надалі повинна перерости у високий професіоналізм. При цьому слід зазначити, що «високий професіоналізм» – це компетентність фахівця та його досвід, пов'язаний з даною професією. Досвід приходить з часом, а компетентність (у тому числі й навички) інженер повинен одержати в процесі навчання у вузі.

Існує велика кількість класифікацій професійних компетенцій інженера. Цим питанням займалися багато дослідників професійної інженерної освіти. Більшість з них дійшла висновку, що компетентність для інженера – це сума кваліфікацій, які визначаються набором знань, умінь і навичок, тобто професійних компетенцій та соціальної поведінки (рис. 4.1).

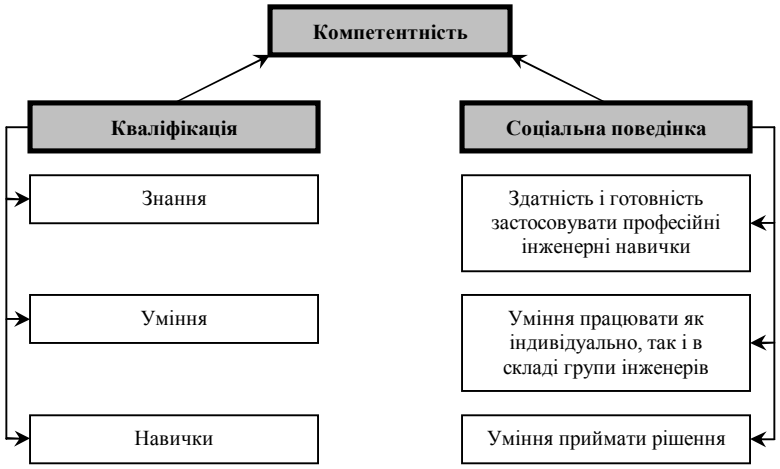

Рис. 4.1. Схема професійних компетенцій інженерів [4]

Професійні компетенції можна розділити на такі групи:

- інструментальні та системні;
- соціально-особові;
- спеціальні.

Причому перші дві групи компетенцій загальні і для масових інженерів, і для інженерів вищого ступеня освіти. Третя група компетенцій повинна розрізняти компетенції для бакалаврів, фахівців і магістрів (рис. 4.2).

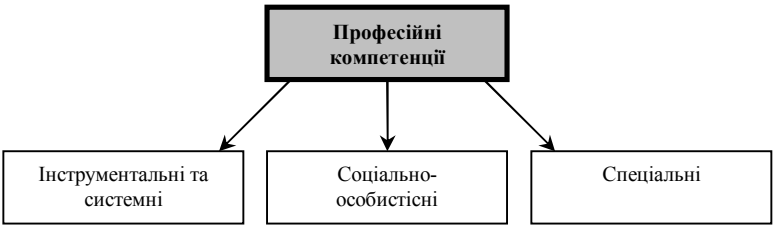

Рис. 4.2. Види професійних компетенцій інженерів [4]

До *інструментальних та системних компетенцій* відносять такі:

- здатність і готовність до аналізу і синтезу;
- здатність і готовність до організації і планування;
- здатність і готовність до використання навичок роботи з різним програмним забезпеченням;
- здатність і готовність до використання навичок управління інформацією (уміння знаходити й аналізувати інформацію);
- готовність до розв'язання проблем і до ухвалення рішень;
- здатність застосовувати знання на практиці;
- здібність до креативності;
- відповідальність за якість своєї роботи.

Соціально-особові компетенції:

- здатність до критики і самокритики;
- уміння і готовність працювати в команді;
- навички міжособистісних відносин;
- прагнення до успіху, лідерства, прояв ініціативи;
- готовність спілкуватися з фахівцями з інших галузей;

- здатність працювати в міжнародному середовищі;
- здатність і готовність навчатися протягом усього життя;
- здатність швидко адаптуватися до нових ситуацій;
- прихильність до етичних цінностей;
- здатність працювати самостійно.

Спеціальні компетенції інженерів різних ступенів навчання (різного рівня кваліфікації) можуть підрозділятися на економічні та організаційно-управлінські, а також професійно-профілюючі. Причому доцільно економічні й організаційно-управлінські об'єднати в одну групу компетенцій, оскільки в більшості виробничих ситуацій управління яким-небудь процесом передбачає знання управлінського (правового) характеру й економічного напряму (для оцінки економічної ефективності).

4.2. Особливості акредитації інженерних освітніх програм

У багатьох розвинених країнах існує двоступінчата система пред'явлення вимог до якості інженерної підготовки і визнання інженерних кваліфікацій [4]. Перший ступінь – оцінка якості освітніх програм бакалаврів у галузі техніки і технологій через процедуру їх акредитації. Другий ступінь – визнання професійних кваліфікацій інженерів через їх сертифікацію і реєстрацію.

Такі системи реалізуються в кожній країні національними, як правило, неурядовими професійними організаціями – інженерними радами, що мають у своєму складі органи з акредитації освітніх програм і сертифікації фахівців: ABET (США), ECUK (Великобританія), CCPE (Канада), IEAust (Австралія) тощо. Міжнародне визнання якості освітніх програм і кваліфікацій інженерів (Professional Engineer) проходить також у два етапи: шляхом укладання договорів, спрямованих на взаємне визнання національних критеріїв і процедур акредитації освітніх програм, таких як Вашингтонська угода (Washington Accord, 1989 рік), і договорів про взаємне визнання національних систем реєстрації професійних інженерів (Engineers Mobility Forum, 1997 рік, APEC Engineering Register, 2000 рік).

У більшості країн Європи поки відсутні системи акредитації інженерних освітніх програм. Проте відома діяльність European Federation of National Engineering Associations, FEANI щодо реєстрації професійних інженерів з присвоєнням статусу європейського інженера (European Engineer – EurIng).

Розглянемо вимоги і критерії якості підготовки фахівців у галузі техніки і технологій в університетах, а також вимоги до кваліфікації професійних інженерів з боку суспільно-професійних організацій, що акредитують освітні програми і сертифікують фахівців у різних країнах.

Accreditation Board for Engineering and Technology, ABET, США. Рада з акредитації в галузі техніки і технологій (ABET) є найбільш авторитетною в Сполучених Штатах і у всьому світі професійною організацією, що займається оцінкою якості інженерних освітніх програм в університетах. Рада була створена на базі існуючої у США з 1932 року організації Engineers' Council for Professional Development (Рада інженерів з професійного розвитку). На даний час ABET є федерацією, що складається з понад 30 професійних інженерних і технічних товариств. Радою акредитовано понад 2500 освітніх програм, що реалізовуються в більш ніж 550 університетах і коледжах Сполучених Штатів. За межами США на основі процедури оцінки істотної еквівалентності «substantial equivalence evaluation» ABET визнає відповідність програм зарубіжних вузів американським аналогам. На сьогодні більше 70 програм університетів Німеччини, Голландії, Туреччини, Сінгапуру, Мексики та інших країн визнані ABET.

Цілями акредитації програм з боку ABET є:

- підтвердження адекватної підготовки випускників університетів, що освоїли освітні програми, до ведення інженерної діяльності;
- вдосконалення інженерної освіти;
- ініціалізація нових творчих підходів до інженерної освіти;
- підтвердження необхідності та корисності кожної конкретної освітньої програми для суспільства.

ABET є світовим лідером у галузі розробки нових критеріїв, процедур і методів оцінки якості освітніх програм. Розроблені ABET Criteria 2000 на даний час використовуються акредитуючими організаціями багатьох країн як основа при розробці власних критеріїв національних систем акредитації.

У критеріях ABET сформульовані обов'язкові загальні вимоги до випускників університетів, що освоїли інженерні програми. Відповідно до цих вимог у результаті навчання випускники повинні набувати здатності:

- застосовувати природничо-наукові, математичні та інженерні знання;
- планувати і проводити експеримент, аналізувати й інтерпретувати дані;
- проектувати системи, їх компоненти або процеси відповідно до поставлених завдань;
- працювати в колективі з міждисциплінарної тематики;
- формулювати й розв'язувати інженерні проблеми;
- усвідомлювати професійні та етичні обов'язки;
- ефективно спілкуватися;
- демонструвати широку ерудицію, необхідну для розуміння глобальних і соціальних наслідків інженерних рішень;
- розуміти необхідність та вміти вчитися постійно;
- демонструвати знання сучасних проблем;
- застосовувати навички і сучасні інженерні методи, необхідні для інженерної діяльності.

Окрім загальних вимог до випускників університету в критеріях ABET, залежно від напряму інженерної підготовки, сформульовані також програмні вимоги. Структура учбового плану за конкретною програмою, згідно з вимогами ABET, повинна забезпечувати як широту, так і глибину підготовки з інженерних дисциплін відповідно до спеціалізації.

Загальні та програмні критерії використовуються ABET при проведенні професійної акредитації освітніх програм у галузі техніки і технологій в університетах США. На відміну від інституційної акредитації університетів, у центрі уваги

професійної акредитації, як правило, знаходиться тільки змістовна сторона процесу навчання:

- фундаментальні знання;
- спеціальні знання;
- практичні навички;
- навички проектування;
- використання комп'ютерів.

Якщо при інституційній акредитації деякі недоліки діяльності університету можуть компенсуватися за рахунок інших переваг, то професійна акредитація дотримується принципу: освітня програма сильна настільки, наскільки сильна її найслабша ланка. Програма акредитується тільки в тому випадку, якщо всі її блоки відповідають критеріям.

Основними завданнями професійної акредитації є:

- надання допомоги абітурієнтам у виборі університету, що реалізує якісні освітньої програми;
- надання допомоги працедавцям у прийомі на роботу якісно підготовлених фахівців, що освоїли акредитовані програми;
- надання допомоги урядовим організаціям в ухваленні рішень щодо підтримки університетів, які реалізують якісні освітні програми;
- надання допомоги приватним підприємствам і організаціям в ухваленні рішень щодо розміщення капіталу в університетах, які реалізують якісні освітні програми.

Аналогічні функції виконує суспільно-професійна акредитація освітніх програм в інших розвинених країнах, таких як Канада, Японія, Великобританія, Австралія тощо. Розглянемо вимоги до підготовки інженерів, наприклад, у Канаді та Японії.

Canadian Engineering Accreditation Board, CEAB, Канада. Канадська рада з акредитації в галузі техніки і технологій є структурним підрозділом Канадської ради професійних інженерів (Canadian Council of Professional Engineers, CCPE), що відповідає за акредитацію інженерних програм відповідно до його вимог. Випускники університетів, що освоїли програми, акредитовані CEAB, володіють правами і необхідною кваліфікацією для того, щоб одержати ліцензію професійного інженера в Канаді.

Канадська рада професійних інженерів – національна організація, яка об'єднує 12 провінційних і територіальних асоціацій, що регулюють інженерну діяльність у Канаді. У країні налічується більш ніж 160 тис. ліцензуючих професійних інженерів. CCPE розробляє національні вимоги, що забезпечують відповідні стандарти інженерної освіти і професійних кваліфікацій.

Крім того, що CCPE виражає позицію організацій, які входять до неї, з різних державних і міжнародних питань, вона також координує розробку національних принципів, процедур і керівництва, що стосується інженерної професії. CCPE сприяє кращому розумінню ролі професійних інженерів у розвитку економіки і суспільства, здійснює взаємодію з урядом від імені організацій, які входять до неї.

Вимоги CEAB до випускників університетських освітніх програм у галузі техніки і технологій полягають у такому:

- випускники повинні набувати основної для інженерного проектування здатності застосовувати відповідні знання з метою перетворення, використання й оптимального управління ресурсами за допомогою ефективного аналізу, інтерпретації даних і ухвалення рішень;

- випускники повинні бути такими, що добре адаптуються, творчими, винахідливими і чутливими до змін у суспільстві, технологіях і вимогах до інженерної професії;

- випускники повинні розуміти роль і обов'язки професійного інженера в суспільстві, усвідомлювати вплив інженерної діяльності в усіх її проявах на навколишнє середовище і суспільство;

- випускники повинні вміти ефективно працювати в команді та спілкуватися, як у рамках своєї професії, так і в суспільстві в цілому.

Japan Accreditation Board for Engineering Education, JABEE, Японія. Японська рада з акредитації інженерної освіти була заснована в 1999 році з метою забезпечення міжнародної еквівалентності освітніх програм у галузі техніки і технологій, що реалізуються вищими учбовими закладами Японії. Рада

співпрацює з університетським академічним співтовариством і промисловістю, сприяючи розвитку суспільства й економіки шляхом удосконалення інженерної освіти та підготовки професійних інженерів.

JABEE визначає критерії і процедури акредитації, проводить оцінку освітніх програм і публікує результати акредитації. Вимоги JABEE до випускників освітніх програм у галузі техніки і технологій полягають у тому, що випускники повинні набувати:

- здатність розглядати різні аспекти інженерних проблем з глобальної точки зору;
- розуміння результатів дії та впливу технологій на суспільство і навколишнє середовище, а також відповідальності та зобов'язань інженера перед суспільством відповідно до законів професійної етики;
- знання математики, природничих наук та інформаційних технологій, а також здатність застосовувати ці знання;
- знання зі спеціалізації та здатність застосовувати ці знання для виконання професійних завдань;
- здатність проектувати й ухвалювати інженерні рішення для задоволення потреб суспільства, використовуючи різні галузі науки, а також різні види технологій та інформації;
- комунікативні навички, включаючи усну й письмову мову, навички ведення дискусій рідною мовою і базові навички ефективного спілкування іноземною мовою;
- здатність вчитися самостійно і постійно;
- здатність виконувати й організовувати роботу відповідно до заданих обмежень.

Розглянуті вимоги стосуються не тільки основної природничо-наукової, математичної, професійної і спеціальної підготовки інженерів, й додаткових навичок, таких як здатність до комунікації, командної роботи, володіння іноземною мовою, та інших так званих «transferable skills», або «soft skills». Вважається, що вони важливі з погляду професійної діяльності інженерів і мають велике суспільне значення. Аналогічні вимоги ставляться до випускників інженерних програм і з боку суспільно-професійних організацій інших розвинених країн.

The Washington Accord. Вашингтонську угоду було підписано в 1989 році організаціями, відповідальними за професійну акредитацію освітніх програм у галузі техніки і технологій у країнах-учасницях (США, Канада, Великобританія, Ірландія, Австралія, Нова Зеландія, Південна Африка, Гонконг). Учасники угоди визнають істотну еквівалентність програм, акредитованих за узгодженими критеріями в національних системах, і підтверджують високу якість підготовки фахівців з акредитованих програм при їх працевлаштуванні на інженерні посади в будь-якій з країн-учасниць угоди.

Як мовиться в тексті угоди: «Сторони обмінялися інформацією і проаналізували відповідні процедури, політику і процеси, пов'язані з акредитацією освітніх програм у області техніки і технологій, і винесли Висновок про їх відповідність. Ґрунтуючись на Вашингтонській угоді, що включає даний Висновок, а також Зведення Правил і Процедур, сторони визнають істотну еквівалентність програм відповідно до академічних вимог, що ставляться до інженерної діяльності на професійному рівні».

На даний час учасниками Вашингтонської угоди є такі професійні організації: Accreditation Board for Engineering and Technology (США), Canadian Engineering Accreditation Board of the Canadian Council of Professional Engineers (Канада), Engineering Council (Великобританія), Institution of Engineers of Ireland (Ірландія), Institution of Engineers, Australia (Австралія), Institution of Professional Engineers New Zealand (Нова Зеландія), Engineering Council of South Africa (Південна Африка), Hong Kong Institution of Engineers (Гонконг).

Асоційованими членами Угоди є Japan Accreditation Board for Engineering Education (Японія), Accreditation Agency for Study Programs in Engineering, Informatics, Natural Sciences and Mathematics (Німеччина), Engineering Accreditation Council of Malaysia (Малайзія) і The Institution of Engineers, Singapore (Сінгапур).

Глобалізація світової економіки, що значно підвищила мобільність робочої сили у всіх сферах діяльності, зокрема інженерної, викликала необхідність розробки єдиних вимог до

освіти і професійних компетенцій інженерів. Розглянемо вимоги до «професійних інженерів» з боку міжнародних організацій APEC, FEANI і EMF.

Звання «професійний інженер» (Professional Engineer) означає, що його володар здатний вести самостійну професійну діяльність і має ліцензію одного або більше урядових органів на надання професійних інженерних послуг як незалежний практик.

Сфери діяльності професійного інженера включають розробку і виконання проектів, проведення досліджень, виконання розрахунків, аналіз, випробування, контроль, діагностику, технічну оцінку, технічний арбітраж та інші види діяльності.

У більшості країн для реєстрації як професійний інженер кандидат повинен:

- закінчити університет, навчаючись за акредитованою інженерною програмою;
- бути зареєстрованим у професійній інженерній організації;
- мати досвід практичної інженерної діяльності (від 3 до 7 років, залежно від країни);
- скласти професійний іспит.

Закінчення університету з навчанням за акредитованою інженерною програмою є першою необхідною умовою для реєстрації професійним інженером як у національних, так і в міжнародних організаціях, що ведуть відповідні реєстри.

APEC Engineering Register. Реєстр створений у рамках організації Азіатсько-тихоокеанського економічного співробітництва (Asia-Pacific Economic Cooperation, APEC), заснованого в 1989 році з метою розвитку економіки, торгівлі та інвестицій в Азіатсько-тихоокеанському регіоні. У APEC входить двадцять одна країна, зокрема США, Канада, Китай, Японія, Австралія, Нова Зеландія, Росія та інші.

Реєстрація інженерів в APEC Engineering Register означає визнання їх статусу Professional Engineer і підвищення конкурентоспроможності на міжнародному ринку праці в країнах-членах APEC. Учасниками APEC Engineering Register є країни, що мають свої національні системи сертифікації «професійних інженерів»: Австралія, Гонконг, Індонезія, Канада,

Малайзія, Нова Зеландія, Республіка Корея, США, Таїланд, Філіппіни, Японія. Україна, на жаль, не бере участь в APEC Engineering Register, оскільки не має національної системи сертифікації «професійних інженерів».

Для реєстрації як APEC Engineer необхідно задовольняти такі вимоги, затверджені Координаційним комітетом інженерів APEC (APEC Engineer Coordinating Committee):

- бути випускником університету за акредитованою інженерною програмою;
- одержати у своїй країні право на ведення самостійної професійної інженерної діяльності;
- мати не менше семи років досвіду практичної інженерної діяльності після закінчення університету;
- мати не менше двох років досвіду роботи на відповідальній керівній посаді при виконанні крупного інженерного проекту;
- постійно підтримувати й розвивати свою професійну кваліфікацію;
- діяти в рамках кодексу професійної етики, прийнятого APEC.

Federation Europeenne d'Associations Nationales d'Ingenieurs, FEANI є федерацією європейських інженерних організацій. Членами FEANI є 27 європейських країн: Австрія, Бельгія, Швейцарія, Кіпр, Чехія, Німеччина, Данія, Естонія, Іспанія, Фінляндія, Франція, Великобританія, Греція, Угорщина, Ірландія, Ісландія, Італія, Люксембург, Мальта, Нідерланди, Норвегія, Польща, Португалія, Румунія, Швеція, Словенія, Словаччина. FEANI об'єднує більше вісімдесяти національних інженерних асоціацій, через які представляє інтереси приблизно двох мільйонів інженерів в Європі. На жаль, Україна не має членства в FEANI.

FEANI є одним із засновників Всесвітньої федерації інженерних організацій (World Federation of Engineering Organizations, WFEO) і співпрацює з багатьма іншими організаціями, що займаються інженерними, технологічними проблемами та інженерною освітою. FEANI офіційно визнана Європейською комісією представником інтересів інженерної професії в Європі, має консультативний статус в UNESCO,

Організації з промислового розвитку ООН і Раді Європи. Своїми діями, особливо присвоєнням звання «Європейський інженер» (EurIng), FEANI сприяє взаємному визнанню інженерних кваліфікацій в Європі, а також посиленню позиції, ролі та відповідальності інженерів у суспільстві.

Власники звання EurIng вносяться в FEANI Register, який у 2003 році налічував більше 27 тис. професійних інженерів. Для включення в FEANI Register, що гарантує підвищення конкурентоспроможності інженера на європейському ринку інтелектуальної праці, необхідно відповідати певним вимогам. Одним з основних критеріїв є інженерна підготовка. За стандартом FEANI Formation, мінімальна складова освіти, одержаної в країнах-членах FEANI, – (*B+3U*), а мінімальна складова інженерного досвіду – *2E*. Основна формула, що описує вимоги FEANI до інженерної підготовки, має вигляд:

$C = B+3U+2U+2E$,

або

$C = B+3U+2T+2E$,

або

$C = B+3U+2E+2E$,

де *C* – термін інженерної підготовки;

B – високий рівень середньої освіти, підтверджений одним або кількома сертифікатами;

U – один рік вищої освіти (теоретична частина) в університеті або іншій освітній установі університетського рівня, визнаній FEANI і занесеній до спеціального списку освітніх установ і програм (FEANI Index);

T – один рік практики, метою якої є набуття додаткових знань за допомогою роботи в технічній сфері: будівництві, на підприємстві, в лабораторії, офісі або на іншому робочому місці, визначеному, схваленому і контрольованому університетом як практична частина інженерної програми;

E – один рік інженерного досвіду, оціненого і схваленого органом, визнаним FEANI. Під інженерним досвідом розуміється надбання знань за допомогою роботи після отримання освіти.

Для претендентів на звання EurIng, що здобули освіту поза країною-членом FEANI, існують інші умови. Претендент повинен здобути освіту в університеті, визнаному країною-членом FEANI еквівалентним освітнім установам, внесеним до FEANI Index. У цьому випадку мінімальні вимоги описуються формулою

B+Освіта+4E.

Претенденти, що одержали ступінь у галузі математики або природничих наук в освітній установі, занесеній до FEANI Index, або університеті, визнаному FEANI еквівалентним, також мають право на реєстрацію з присвоєнням звання EurIng, якщо їх вік складає не менше 35 років і вони мають не менше восьми років відповідного професійного інженерного досвіду, тобто формула для них буде такою:

B+Освіта+8E.

У особливих випадках претенденти у віці не менше 35 років, що не задовольняють вищеперелічені стандарти, мають право на реєстрацію в статусі EurIng за наявності не менше 15 років професійного інженерного досвіду, визнаного FEANI.

У критеріях FEANI сформульовані такі вимоги до професійних інженерів:

- розуміння суті професії інженера й обов'язку служити суспільству, професії і зберігати навколишнє середовище за допомогою проходження кодексу професійної поведінки FEANI;
- наявність високого рівня розуміння принципів інженерії, заснованих на математиці та інших наукових дисциплінах, що мають відношення до спеціалізації;
- загальні знання про інженерну діяльність у сфері спеціалізації і характеру сучасного виробництва, включаючи використання матеріалів, компонентів і програмного забезпечення;
- здатність застосовувати відповідні теоретичні і практичні методи до аналізу й розв'язання інженерних проблем;
- уміння використовувати існуючі та перспективні технології, що належать до спеціалізації;
- знання інженерної економіки, методів забезпечення якості, уміння використовувати технічну інформацію і статистику;

- уміння працювати в команді над міждисциплінарними проектами;

- здатність бути лідером, включаючи адміністративні, технічні, фінансові та особові аспекти;

- комунікативні навички і підтримка необхідного рівня компетенції за допомогою безперервного професійного розвитку (continuous professional development, CPD);

- знання стандартів і правил, що відповідають спеціалізації;

- реагування на постійні технічні зміни і творчий пошук у рамках професії;

- вільне володіння європейськими мовами, достатнє для спілкування при роботі в Європі.

Engineers Mobility Forum, EMF. Міжнародна організація мобільності професійних інженерів EMF була створена в 1997 році. Форум об'єднує національні асоціації з реєстрації «професійних інженерів». Учасники EMF погодили між собою вимоги до «професійних інженерів» і визначили міжнародні стандарти присудження даного звання. Це дає фахівцям право отримання рівнозначного статусу в країнах-учасниках Форуму, що забезпечує їх міжнародну професійну мобільність.

Країнами-учасниками EMF є США, Канада, Великобританія, Ірландія, Австралія, Нова Зеландія, Японія, Малайзія, Гонконг і Республіка Корея. Форум заснував Міжнародний реєстр професійних інженерів EMF, куди включаються зареєстровані «міжнародні професійні інженери EMF» (EMF Registered International Professional Engineers), що пройшли оцінку Комітету з моніторингу (EMF Monitoring Committee) і відповідають критеріям EMF Agreement і Memorandum of Understanding, підписаним країнами-учасниками Форуму.

Критерії для реєстрації як «міжнародний професійний інженер EMF» включають:

- наявність інженерної освіти, одержаної в університеті за акредитованими на основі міжнародних критеріїв Washington Accord програмами;

- здатність до самостійної професійної інженерної діяльності;

- досвід практичної діяльності не менше семи років, включаючи два роки роботи на відповідальній керівній посаді при виконанні великого інженерного проекту;

- безперервне професійне вдосконалення;

- відповідальність і згода діяти в рамках відповідних кодексів професійної етики інженера EMF.

Болонський процес формування єдиної зони вищої освіти в Європі, учасником якого стала й Україна, передбачає узгодження вимог до якості освіти й підготовки фахівців, зокрема за рахунок створення єдиної системи критеріїв оцінки якості, акредитації освітніх програм і сертифікації фахівців.

Україна бере активну участь у виконанні проекту EURRACE (EURopean ACcredited Engineer), що фінансується Європейською комісією, метою якого є вироблення пропозицій зі створення загальноєвропейської системи акредитації програм у сфері техніки і технологій. У виконанні проекту беруть участь авторитетні міжнародні інженерні організації, такі як FEANI, SEFI, CESAER, EUROCADRES, ENQHEEI, а також ряд національних акредитуючих агентств провідних європейських країн: ASIIN (Німеччина), CTI (Франція), ECUK (Великобританія), CoPI (Італія) та інші.

Створення загальноєвропейської системи акредитації освітніх програм у сфері техніки і технологій необхідне для того, щоб:

- поліпшити якість інженерних освітніх програм;

- полегшити процедуру міжнародного визнання ступенів шляхом вироблення загальних критеріїв для оцінки освітніх програм;

- сертифікувати випускників відповідно до загального стандарту щодо акредитації освітніх програм;

- сприяти визнанню і мобільності фахівців-професіоналів відповідно до EU Directives;

- сприяти угодам про взаємне визнання.

Основними етапами виконання проекту є:

- вироблення загальних критеріїв оцінки якості інженерної освіти;

- апробація узгоджених вимог у ряді країн шляхом проведення «пілотної» акредитації освітніх програм;

- корегування і вдосконалення критеріїв оцінки якості;

- повторне тестування системи і підготовка звіту.

У результаті виконання проекту на черговій зустрічі міністрів освіти країн-учасниць Болонського процесу в Бергені (Норвегія) у 2005 році було представлено проект загальноєвропейської системи акредитації освітніх програм у сфері техніки і технологій. Участь у цьому проекті України є важливим чинником успішного входження нашої країни в Болонський процес.

Розглянуті міжнародні вимоги до якості підготовки фахівців у сфері техніки і технологій, що висуваються професійними інженерними організаціями – головними зацікавленими сторонами, служать основою для проектування освітніх програм у зарубіжних університетах. Ці вимоги трансформуються в цілі та результати освітніх програм, яких планується досягти при навчанні студентів у вищих навчальних закладах. Для успішної конкуренції на світовому ринку освітніх послуг і для забезпечення національної економіки висококваліфікованими кадрами українській вищій школі слід активніше впроваджувати й розвивати аналогічні підходи і вимоги як при проектуванні освітніх стандартів та освітніх програм, так і при розвитку в Україні національної системи гарантій якості вищої освіти.

4.3. Особливості організації навчального процесу для програмних інженерів

Становлення навчання фахівців будь-якої нової галузі проходить три такі періоди [6]:

- «окрема дисципліна» – в цьому періоді, в навчальному плані відповідного напряму навчання з'являється одна дисципліна, що відображає поточний стан знань у новій галузі, мета якої полягає в тому, щоб ознайомити студентів із галуззю;
- «спеціальні курси» – в цьому періоді, ті організації, які мають потребу у фахівцях нової галузі, займаються підготовкою випускника навчального закладу шляхом організації у своїх учбових центрах спеціальних курсів, реалізовуючи тим самим спеціалізацію. Одночасно в учбових планах вузів, окрім основної, починають з'являтися й інші дисципліни («курси»), що більш глибоко розглядають окремі розділи основної дисципліни;
- «учбовий план» – у цьому періоді нова галузь дозріває настільки, що підготовка фахівців для неї не може забезпечуватися учбовим планом, який містить одну дисципліну і курси; потрібна розробка окремого плану, при цьому, залежно від форми освіти, створюється новий бакалаврат і спеціальності або нова спеціальність і спеціалізації.

Очевидно, що третій період повинен бути підготовлений об'єктивно і знаковими для цього будуть такі події: потреба у фахівцях нової галузі, що зростає у суспільстві; накопичення деякої кількості фахівців – «самоучок», що виконують завдання галузі та здатні навчати; створення деякого обсягу літератури (бажано навчальних посібників, довідників, монографій), який може лягти в основу навчально-методичного забезпечення учбового плану.

Найбільш близьким напрямком, у рамках якого зараз готуються професіонали з програмного забезпечення, є напрям «комп'ютерні науки» [1].

У світовій практиці навчання за напрямком «комп'ютерні науки» здійснюється на основі документа Computing Curricula 2001 (CC2001). Перелік дисциплін, які рекомендує вивчати цей документ, наведений у таблиці 4.3. Видно, що в переліку

знаходиться одна дисципліна, пов'язана з даною темою – «Інженерія програмного забезпечення».

Таблиця 4.3

ПЕРЕЛІК РЕКОМЕНДОВАНИХ ДИСЦИПЛІН ЗА CC2001

№ пп	Назва учбової дисципліни	Мінімальна кількість годин вивчення дисципліни
1.	Дискретні структури (математика)	108
2.	Основи програмування	216
3.	Алгоритми і складність	108
4.	Мови програмування	54
5.	Архітектура комп'ютерів	108
6.	Операційні системи	108
7.	Людино-машинна взаємодія	54
8.	Графіка, візуалізація і мультимедіа	54
9.	Інтелектуальні системи	54
10.	Інформаційний менеджмент (управління базами даних)	54
11.	Мережні обчислення	54
12.	Інженерія програмного забезпечення	108
13.	Наукові обчислення	54
14.	Соціальні, етичні і професійні питання	54

Для навчання комп'ютерним наукам характерні [10, 11]:

- підготовка фахівців з різних галузей комп'ютерних технологій;
- орієнтація на одну мову програмування (зазвичай C++, Object Pascal або Java);
- кодування програм «з нуля»;
- виконання програмування раніше проектування;
- програмування «в малому»;
- орієнтація на індивідуальну діяльність;
- відсутність у фахівців професійних інженерних навичок, таких як: дотримання стандартів, планування та оцінка своїх дій, просування на ринок і супровід програмного забезпечення;
- ігнорування вимог безпеки, надійності, економіки.

Спроби готувати професіоналів із програмного забезпечення, ґрунтуючись на структурно-логічній схемі напряму комп'ютерних наук, на даний час недостатньо ефективні з об'єктивних і суб'єктивних причин. З одного боку, ці спроби призводять до суперечностей з практикою індустріальної розробки програмного забезпечення великими колективами, що передбачає дотримання заданих вимог і стандартів, просування програмного забезпечення на сегментований ринок, його ефективний супровід і ліквідацію. З іншого боку, комісія, яка працювала над документом CC2001, визнала, що вона не готова сформулювати такий документ, який охопив би комп'ютерні науки в цілому. Розв'язанням цієї проблеми стало звернення комісії до спонсоруючих організацій з проханням розширити проект CC2001, і першим кроком у цьому напрямі став проект Computing Curricula – Software Engineering (CCSE) [1].

В Україні навчання комп'ютерних наук було розпочато у 1995 р. з уведенням триступінчатої форми освіти («молодший фахівець», «бакалавр», «магістр»). На даний час діє другий галузевий стандарт цього напряму. Рекомендований стандартом перелік учбових дисциплін для основних циклів наведений у таблиці 4.4. Порівнюючи цей перелік з тим, який рекомендує CC2001 (табл. 4.3), бачимо, що між ними немає принципових суперечностей, а в переліку стандарту також є тільки одна дисципліна, пов'язана з програмною інженерією («Технологія програмування і створення програмних продуктів»). Проте вказаний перелік в Україні ще доповнюють учбові дисципліни за вибором, які залежать від характеру університету і спеціальності. В український учбовий план також входять дисципліни, що на практиці не пов'язані за змістом з комп'ютерними науками, але мають загальний інженерний характер – економіка і організація виробництва, менеджмент, основи екології, охорона праці.

Таблиця 4.4

ПЕРЕЛІК РЕКОМЕНДОВАНИХ ДИСЦИПЛІН ЗА УКРАЇНСЬКИМ СТАНДАРТОМ ВІД 1995 Р.

№ п/п	Назва учбової дисципліни	Мінімальна кількість годин вивчення дисципліни
	Цикл природничо-наукової підготовки	
1.	Вища математика	540
2.	Фізика	324
3.	Інженерна і комп'ютерна графіка	108
4.	Основи дискретної математики	162
5.	Теорія ймовірності, імовірнісні процеси і математична статистика	162
6.	Основи програмування і алгоритмічні мови	162
7.	Об'єктно-орієнтоване програмування	162
8.	Системний аналіз і проектування комп'ютерних інформаційних систем	162
9.	Основи екології	54
10.	Безпека життєдіяльності	54
Разом по циклу		1890
	Цикл професійної і практичної підготовки	
11.	Основи електротехніки і електроніки	108
12.	Комп'ютерна схемотехніка (елементи і схеми комп'ютерних систем)	108
13.	Системне програмування і операційні системи	216
14.	Технологія програмування і створення програмних продуктів	108
15.	Організація баз даних і знань	108
16.	Методи і засоби комп'ютерних інформаційних технологій	108
17.	Системи штучного інтелекту	108
18.	Моделювання систем	135
19.	Числові методи в інформатиці	135
20.	Економіка і організація виробництва	162
21.	Менеджмент	54
22.	Основи охорони праці	54
Разом по циклу		1404

Навчання інженерії програмного забезпечення принципово відрізняється від навчання комп'ютерних наук, принаймні, у двох аспектах [1].

По-перше, для вивчення інженерії програмного забезпечення необхідні знання зі сфер трьох типів (рис. 4.3):

- «проблем» – конкретні наочні сфери, для яких створюється програмне забезпечення;
- «розв'язків» – власне інженерія програмного забезпечення;
- «проблем і розв'язків» – застосування методів інженерії програмного забезпечення в конкретних наочних сферах.

При цьому наголошуються такі особливості вказаних знань:

- знання сфери проблем, яких достатньо багато, принципово відрізняються від знань сфери розв'язків;
- знання сфери розв'язків знаходяться в різноманітних керівництвах, системній документації, фірмових технічних бібліотеках, тому недоступні широкому колу фахівців;
- надбання знань сфери «проблем і розв'язків» вимагає від викладача значних зусиль, оскільки спочатку отримуються знання сфери проблем, потім сфери розв'язків і, нарешті, придбані знання з'єднуються на практичній основі.

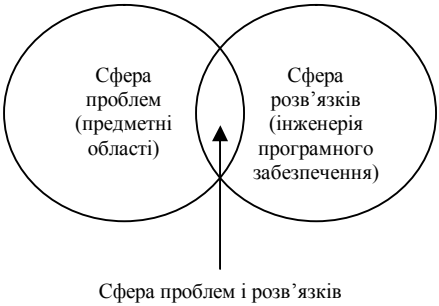

Рис. 4.3. Сфери інженерії програмного забезпечення

Окрім цього, існує окрема велика сфера знань, пов'язана з управлінням процесами розробки програмного забезпечення, яку відносять до інфраструктури інженерії програмного забезпечення, але яку студенти також повинні знати.

По-друге, множина учбових дисциплін і професія інженера програмного забезпечення, подібно до інших інженерних галузей, знаходяться під загальним впливом характеру інженерії. Інженерні дисципліни виникають з практики експлуатації технологій, побудованих на основі «зріючої» науки. Інженер, володіючи науковими знаннями і керуючись принципами аналізу, синтезу, проектування, застосовує знання до матеріалів і, використовуючи методи та засоби, здійснюючи відповідні процеси, створює продукти. Тому існує певний набір властивостей інженера, які є загальними для будь-якої інженерії та настільки важливими, що їх використовують як фундамент практичної підготовки в конкретній інженерії. Відомо близько 130 навичок, якими повинен володіти інженер з програмного забезпечення. Нижче наведені тільки деякі з них, відповідно до яких інженер повинен володіти такими властивостями [1]:

- показувати майстерність у знаннях і навичках, необхідних для початку практики розробника програмного забезпечення;
- працювати індивідуально і в групі над створенням програмного забезпечення;
- погоджувати суперечливі вимоги, знаходячи компроміси у витратах, часі, знаннях, існуючих засобах та організації робіт;
- розробляти прийнятні рішення в проблемних галузях, застосовуючи інженерні методи, враховуючи етичні, соціальні, юридичні й економічні аспекти;
- демонструвати розуміння і застосування сучасних теорій, моделей, техніки і методик, що становлять основу завдань визначення і аналізу, проектування, розробки, реалізації, валідації і верифікації програмного забезпечення;
- вести переговори, брати на себе лідерство, підтримувати зв’язок із зацікавленими колами в типових умовах розробки програмного забезпечення;
- вивчати все нове, що з’являється в галузі інженерії програмного забезпечення, прикладних сферах і постійно професійно розвиватися.

Особливе місце в навчанні інженерії програмного забезпечення відводиться математичній підготовці студентів, яка

повинна насамперед забезпечувати розуміння процесів, що відбуваються в області рішень. Тому особливу роль відіграють формалізми, які використовуються для опису даних, компонентів, процесів, архітектури, поведінки програмного забезпечення. Важливо вивчати моделювання і моделі даних, інтерфейсів, станів, викликів. Навпаки, розділи математики, які потрібні для розуміння різних проблем, а це: алгебра, геометрія, диференціальне та інтегральне числення, теорія чисел, розділи математичної фізики і механіки тощо, які повинні виноситися на факультативне вивчення.

Підготовка професіоналів з інженерії програмного забезпечення, на відміну від комп'ютерних наук, характеризується таким зсувом акцентів у навчальних планах і програмах дисциплін [1]:

- від програмування до аналізу і проектування;
- від програмування «в малому» до програмування «у великому», спираючись на готову архітектуру і компоненти;
- від індивідуальної до групової діяльності;
- від розробки до супроводу;
- від ігнорування стандартів до їх обов'язкового застосування;
- від недокументованої до документованої розробки;
- від недоказових до доказових дій;
- від випадкових до планованих дій;
- від загальних фахівців до ліцензованих фахівців-інженерів.

У 1980-х роках навчанням у галузі інженерії програмного забезпечення займався Інститут інженерії програмного забезпечення (SEI). Інститут був ініціатором і спонсором першої конференції з навчання у даній області (1987 р.). У 1995 році SEI створив Робочу групу з освіти і тренінгу в галузі інженерії програмного забезпечення Working Group on Software Engineering Education and Training (WGSEET), результатом діяльності якої став звіт, що пропонує принципи проектування і реалізації університетських програм з інженерії програмного забезпечення. У 1993 р. професійні асоціації IEEE-CS та ACM створили комітет IEEE-CS/ACM для затвердження інженерії програмного

забезпечення як професії. Пізніше цей комітет був замінений комітетом Software Engineering Coordinating Committee (SWECC). Він координував розробку трьох документів: кодексу етики і професійної практики; проекту з освіти – Software Engineering Education Project (SWEEP) з критеріями акредитації університетських програм; керівництва із системи знань у галузі інженерії програмного забезпечення – Guide to the Software Engineering Body of Knowledge (SWEBOK) [7]. Результатом спільної діяльності IEEE-CS та ACM став документ Computing Curriculum Software Engineering 2003 (CCSE 2003), який після доопрацювання у 2004 р. був затверджений для використання [1].

SWEEP містить стратегії і вимоги для акредитації учбових планів і програм (для США і Європи). SWEBOK містить рекомендації, суть яких полягає в такому [7]: знання повинні бути організовані у вигляді чотирьох областей – ядро, загальна область, основа та область забезпечення; кожна область – у вигляді множини компонентів. Область «ядро» включає знання, що відображають суть інженерії програмного забезпечення, і структурована відповідно до фаз життєвого циклу: проектування, конструювання, супровід, управління проектуванням. Загальна область включає знання, які є загальними для ядра: етика і професіоналізм, процеси, якість, моделювання, вимірювання, інструменти і середовища, документування. Основа включає знання, які охоплюють ядро і загальну область: основи обчислень, людські чинники, проблемні питання. Область забезпечення включає знання, які стосуються організації навчання інженерії програмного забезпечення, що не увійшли до вказаних трьох областей.

У документі CCSE 2004 наводяться зразки учбових планів (шаблони), які будуються на основі модулів, що складаються з учбових тем і утворюють сфери знань (табл. 4.5) [1].

Таблиця 4.5

СФЕРИ ЗНАНЬ ДОКУМЕНТА CCSE 2004

№ п/п	Сфера знань	Теми
1	Основи обчислень (обробки інформації)	Основи комп'ютерних наук. Технології проектування. Інструменти для розробки. Методи формальної побудови
2	Математичні основи інженерії	Основи математики (дискретної). Інженерні основи програмного забезпечення. Прикладна економіка програмного забезпечення
3	Професійна практика	Динаміка /психологія поведінки колективу. Навички спілкування. Професіоналізм
4	Моделювання і аналіз програмного забезпечення	Основи моделювання. Типи моделей. Основи аналізу. Виявлення вимог. Деталізація і документування вимог. Затвердження вимог
5	Розробка програмного забезпечення	Концепції проектування. Стратегії розробки. Проектування інтерфейсів. Деталізоване проектування. Інструментарій і оцінка проектування
6	Перевірка і затвердження програмного забезпечення	Термінологія і принципи верифікації і валідації. Рецензії. Тестування. Оцінка і тестування інтерфейсу програми. Аналіз і складання звітів щодо проблем
7	Еволюція програмного забезпечення	Процеси еволюції. Показники еволюції
8	Програмний процес	Концепції процесу. Реалізація процесу. Адаптація процесу
9	Якість програмного забезпечення	Культура і концепція якості програмного забезпечення. Стандарт якості програмного забезпечення. Процеси якості програмного забезпечення. Страхування процесу. Страхування продукту

Підкреслюється міжнародна спрямованість учбових планів, тому всі вони носять рекомендаційний характер. Розглядаються альтернативні умови викладання – дистанційне навчання і вільне відвідування занять. У документі для розробки конкретних учбових планів і програм наводяться такі директиви [1]:

- розробники планів і програм, викладачі при навчанні повинні виходити з кінцевого результату (продукту);
- розробники повинні витримувати баланс між учбовим матеріалом, гнучкістю планів і програм, які передбачають нововведення;
- студенти повинні знати сфери застосування інженерії програмного забезпечення;
- викладачі повинні підкреслювати прикладну природу інженерії програмного забезпечення;
- викладати інженерію програмного забезпечення потрібно як дисципліну, спрямовану на розв'язання задач;
- увага повинна приділятися основним принципам інженерії програмного забезпечення, а при розгляді технологій – аспектам їх внутрішньої організації;
- викладання повинно будуватися так, щоб студенти могли набувати досвіду, користуючись сучасними інструментами, навіть якщо детальний розгляд цих інструментів не є метою навчання;
- учбовий матеріал повинен ґрунтуватися на ретельних дослідженнях, математичній теорії та широко вживаній практиці;
- учбовий план повинен мати ґрунтовний практичний базис і передбачати обговорення етичних аспектів професіоналізму;
- навчання інженерії програмного забезпечення у XXI столітті повинно вийти за межі простої лекційної форми викладання, тому необхідно активно впроваджувати різноманітність підходів до навчання.

Реалізовуючи практичну спрямованість наведених рекомендацій, можна, наприклад, використовувати досвід, який пропонує двоетапне вивчення інженерії програмного забезпечення – на першому етапі студенти вивчають теоретичні

питання, а на другому відпрацьовують практичні навички колективної роботи.

В Україні починаючи з 2007 року введений у дію бакалаврат «Програмна інженерія», учбовий план «бакалаврат» розробляється на основі матеріалів CCSE 2004. Основою для створення плану став шаблон N2S-1c, рекомендований для вищих учбових закладів Північної Америки. Шаблон показує порядок розташування дисциплін, що покривають необхідні модулі знань (SEEK) інженерії програмного забезпечення.

Відмітною особливістю шаблону N2S-1c є початок вивчення інженерії програмного забезпечення з другого курсу, що дозволяє легко адаптувати його для факультетів комп'ютерних наук, що працюють згідно із семестровою системою. Дисципліни, що становлять ядро, мають у позначенні префікс «SE» перед кодом.

Деякі комірки шаблона залишені невизначеними для того, щоб забезпечити його гнучкість і можливість адаптації до умов конкретного університету. Гнучкість шаблона також забезпечується можливістю вибору широкого набору дисциплін комп'ютерних наук, що покривають відповідні області SEEK. Дисципліни комп'ютерних наук мають у позначенні префікс «CS» перед кодом.

У шаблон включені нетехнічні дисципліни для покриття відповідних сфер SEEK. Нетехнічні дисципліни мають у позначенні префікс «NT» (Not Technical) перед кодом. Керівництвом CCSE 2004 рекомендується починати викладання дисципліни «Групова динаміка і комунікації» (NT 181), яка носить психологічний характер, перед дисципліною етичного характеру «Професійна практика програмної інженерії» (NT 291), оскільки студенти повинні бути підготовлені для сприйняття останньої.

4.4. Огляд освітнього стандарту SWEBOK

З 1993 р. професійні асоціації IEEE-CS та ACM координують свої роботи в рамках спеціального спільного комітету – Software Engineering Coordinating Committee (SWECC). Проект SWEBOK (Software Engineering Body of Knowledge) був ініційований цим комітетом у 1998 р. [7]. Оцінений можливий обсяг змісту SWEBOK та інші фактори привели до того, що було рекомендовано проводити роботи з реалізації проекту не тільки силами добровольців з рядів експертів індустрії та представників найбільших споживачів і виробників програмного забезпечення, але і на основі принципу «повної зайнятості». Базовий комплекс робіт, відповідно до спеціального контракту, був переданий у Software Engineering Management Research Laboratory Університету Квебек у Монреалі (Universite du Quebec a Montreal).

Серед компаній, що підтримали цей унікальний проект, були Boeing, MITRE, Raytheon, SAP. У результаті проекту, здійсненого за фінансової підтримки цих та інших компаній і організацій, а також з урахуванням його значення для індустрії, SWEBOK Advisory Committee прийняв рішення зробити SWEBOK загальнодоступним на сайті http://www.swebok.org/. Сьогоднішня «публічність» (загальнодоступність) результатів проекту стала можлива, завперш, саме завдяки підтримці SWEBOK Industrial Advisory Board – структури, що поєднує представників компаній, які підтримали проект.

Проект SWEBOK планувався у вигляді трьох фаз: Strawman (“солом’яна людина”), Stoneman (“кам’яна людина”) та Ironman (“залізна людина”). До 2004 р. була випущена версія SWEBOK третьої фази – Ironman, тобто максимально наближена до остаточного варіанта і схвалена IEEE у лютому 2005 р. до публікації як Trial-версія. Основна мета цієї «пробної» версії SWEBOK – поліпшити подання, цілісність і корисність матеріалу на основі збору й аналізу відгуків на дану версію для того, щоб випустити фінальну редакцію документа.

На сьогоднішній день документ SWEBOK містить 10 галузей знань (knowledge areas) [7]. При цьому, що справедливо і для PMBOK, додавання нових галузей знань у SWEBOK досить

прозоре. Усе, що для цього потрібно, зрілість (чи, принаймні, явний та швидкий процес досягнення зрілості) та загальновизнаність відповідної галузі знань, якщо це не призведе до серйозного ускладнення SWEBOK.

Важливо розуміти, що програмна інженерія є дисципліною, яка розвивається. Більше того, дана дисципліна не стосується питань конкретизації застосування тих чи інших мов програмування, архітектурних рішень або рекомендацій щодо деяких конкретних технологій.

Керівництво до збірника знань, а таким є SWEBOK, включає базове визначення й опис галузей знань (наприклад, конфігураційне управління – configuration management) і, безумовно, недостатнє для охоплення всіх аспектів створення програмного забезпечення, але водночас потрібне для їхнього розуміння.

Однією з найважливіших цілей SWEBOK є саме визначення тих аспектів діяльності, що складають суть професії інженера-програміста.

Опис галузей знань у SWEBOK побудовано за ієрархічним принципом як результат структурної декомпозиції. Така ієрархічна побудова зазвичай нараховує два-три рівні деталізації, прийнятих для ідентифікації тих чи інших загальновизнаних аспектів програмної інженерії. При цьому структура декомпозиції галузей знань деталізована тільки до того рівня, який потрібен для розуміння природи відповідних тем і можливості пошуку джерел компетенції й інших довідникових даних і матеріалів. У принципі, вважається, що «збірник знань з програмної інженерії» як такий представлений не в обговорюваному керівництві (SWEBOK), а в першоджерелах, як зазначених у ньому, так і представлених за його рамками.

SWEBOK описує 10 галузей знань [7]:

1. Software requirements – програмні вимоги;
2. Software design – дизайн (архітектура);
3. Software construction – конструювання програмного забезпечення;
4. Software testing – тестування;

5. Software maintenance – експлуатація (підтримка) програмного забезпечення;

6. Software configuration management – конфігураційне управління;

7. Software engineering management – управління в програмній інженерії;

8. Software engineering process – процеси програмної інженерії;

9. Software engineering tools and methods – інструменти і методи;

10. Software quality – якість програмного забезпечення.

На додаток до них SWEBOK також включає огляд суміжних дисциплін, зв'язок з якими представлений як фундаментальний, важливий та обґрунтований для програмної інженерії [7]:

1. Computer engineering – комп'ютерна інженерія;
2. Computer science – комп'ютерні науки;
3. Management – управління;
4. Mathematics – математика;
5. Project management – управління проектами;
6. Quality management – управління якістю;
7. Software ergonomics – ергономіка програмного продукту;
8. Systems engineering – інженерія систем (системотехніка).

Прийняті розмежування між галузями знань (areas), їх компонентами (subareas) та іншими елементами нежорсткі. При цьому, на відміну від PMBOK, галузі знань SWEBOK не передбачають «входи» і «виходи». Деякою мірою така декомпозиція пов'язана з тим, що SWEBOK не асоційовано з тією чи іншою моделлю (наприклад, життєвого циклу) чи методом. Хоча, на перший погляд, перші п'ять галузей знань у SWEBOK представлені в традиційній послідовній (каскадній – waterfall) моделі, це не більше ніж дотримання прийнятої послідовності висвітлення відповідних тем. Інші галузі і структура декомпозиції галузей подані за абеткою.

Для наочного подання понятійного апарату галузей знань SWEBOK наведемо умовне розбиття галузей на основні (п'ять для проектування програмного забезпечення, рис. 4.4) і додаткові організаційні методи й підходи, які відображають інженерію

керування проектуванням програмного забезпечення (конфігурацією, проектами, якістю – рис. 4.5) [7].

Рис. 4.4. Основні галузі знань SWEBOK [7]

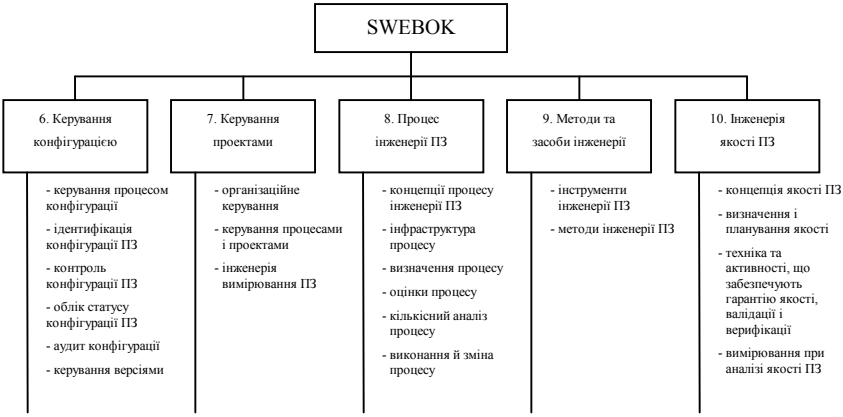

Рис. 4.5. Організаційні галузі знань SWEBOK [7]

1. Інженерія вимог. Вимоги до ПЗ – сукупність властивостей, які повинно мати ПЗ. Призначені для адекватного визначення

функцій, умов і обмежень виконання ПЗ, а також обсягів даних, технічного забезпечення і середовища його виконання.

Вимоги відбивають потреби людей (замовників, користувачів, розробників), зацікавлених у створенні ПЗ. Замовник і розробник спільно виявляють вимоги, аналізують, переглядають, визначають необхідні обмеження й умови, а також описують їх. Розрізняють вимоги до продукту і до процесу, а також функціональні, нефункціональні та системні вимоги. Вимоги до продукту і до процесу визначають умови виконання і режими роботи ПЗ в операційному середовищі, обмеження на структуру й пам'ять комп'ютерів і принципи взаємодії програм.

Функціональні вимоги визначають призначення та функції системи, а нефункціональні – умови стосовно виконання ПЗ, його перенесуваності й доступу до даних. Системні вимоги описують вимоги до програмної системи, яка складається з взаємозалежних програмних та апаратних підсистем і різних застосувань. Вимоги можуть бути кількісні (наприклад, кількість оброблених запитів на секунду, середній показник помилок і т.п.). Значна частина вимог стосується атрибутів якості (безвідмовність, надійність тощо), а також захисту і безпеки як ПЗ, так і даних.

Область знань «Вимоги до ПЗ (Software Requirements)» складається з таких розділів:

- інженерія вимог (Requirement Engineering),
- виявлення вимог (Requirement Elicitation),
- аналіз вимог (Requirement Analysis),
- специфікація вимог (Requirement Specification),
- валідація вимог (Requirement validation),
- керування вимогами (Requirement Management).

Інженерія вимог до ПЗ – це дисципліна аналізу й документування вимог до ПЗ, що полягає в перетворенні запропонованих замовником вимог до системи на опис вимог до ПЗ та їх валідації. Інженерія базується на моделі процесу визначення вимог і діяльності осіб, що забезпечують керування і формування вимог, а також на методах досягнення показників якості.

Модель процесу визначення вимог – це схема процесів життєвого циклу, що виконуються від початку проекту і доти, поки не будуть визначені й погоджені вимоги. Таким процесом може бути маркетинг і перевірка виконання вимог у даному проекті.

Керування вимогами до ПЗ полягає у контролі виконання вимог і плануванні використання ресурсів (людських, програмних, технічних, часових, вартісних) у процесі розроблення проміжних робочих продуктів на етапах життєвого циклу і продукту в цілому.

Якість і процес поліпшення вимог – це процес формулювання характеристик і атрибутів якості (надійність, реактивність тощо), які повинно мати ПЗ, методи їх досягнення на етапах життєвого циклу та оцінювання отриманих результатів.

Виявлення вимог – це процес витягування інформації з різних джерел (договорів, матеріалів аналітиків з декомпозиції задач і функцій системи тощо), проведення технічних заходів (співбесід, збирання пропозицій тощо) для формування окремих вимог до продукту і до процесу розробки. Вимоги погоджуються із замовником.

Аналіз вимог – процес вивчення потреб і цілей користувачів, класифікація і перетворення їх на вимоги до системи, апаратури і ПЗ, встановлення й усунення конфліктів між вимогами, визначення пріоритетів, меж системи і принципів взаємодії із середовищем функціонування.

Специфікація вимог до ПЗ – процес формалізованого опису функціональних і нефункціональних вимог, вимог до характеристик якості відповідно до стандарту якості ISO/IEC 9126, які будуть відпрацьовуватися на етапах життєвого циклу ПЗ. У специфікації вимог відбивається структура ПЗ, вимоги до функцій, якості й документації, а також задається архітектура системи і ПЗ, алгоритми, логіка керування і структура даних. Специфікуються також системні вимоги, нефункціональні вимоги і вимоги до взаємодії з іншими компонентами і платформами.

Валідація вимог – це перевірка викладених у специфікації вимог, що виконується для того, щоб шляхом відстеження джерел

вимог переконатися, що вони визначають саме дану систему. Замовник і розробник ПЗ проводять експертизу сформованого варіанта вимог для того, щоб розробник міг далі продовжувати проектування ПЗ. Один з методів валідації – прототипування, тобто швидке відпрацьовування окремих вимог на конкретному інструменті та дослідження масштабів зміни вимог, вимірювання обсягу функціональності та вартості, а також створення моделей оцінки зрілості вимог.

Верифікація вимог – це процес перевірки правильності специфікацій вимог щодо їх відповідності потребам, несуперечності, повноти і можливості реалізації, а також узгодженості зі стандартами. Як результат перевірки вимог складається погоджений вихідний документ, що встановлює повноту і коректність вимог до ПЗ, а також можливість продовження проектування ПЗ.

Керування вимогами – це керування процесами формування вимог на всіх етапах життєвого циклу, а також змінами й атрибутами вимог, проведення моніторингу – відновлення джерела вимог. Керування змінами виникає після того, як ПЗ починає працювати в заданому середовищі та виявляє помилки трактування вимог, невиконання деякої окремої вимоги тощо. Невід'ємною складовою процесу керування є трасування вимог для відстеження правильності встановлення та реалізації вимог до системи і ПЗ на етапах життєвого циклу, а також зворотний процес відстеження в отриманому продукті реалізованих вимог. Для уточнення деяких вимог або додавання нової вимоги складається план зміни вимог, що погоджується із замовником. Внесені зміни спричиняють і зміни у створеному продукті або в окремих його компонентах.

2. Проектування програмного забезпечення. Проектування ПЗ – це процес визначення архітектури, набору компонентів, їх інтерфейсів, інших характеристик системи і кінцевого складу програмного продукту.

Область знань «Проектування ПЗ (Software Design)» складається з таких розділів:

- базові концепції проектування ПЗ (Software Design Basic Concepts),
- ключові питання проектування ПЗ (Key Issue in Software Design),
- структура й архітектура ПЗ (Software Structure and Architecture),
- аналіз і оцінка якості проектування ПЗ (Software Design Quality Analysis and Evaluation),
- нотації проектування ПЗ (Software Design Notations),
- стратегія і методи проектування ПЗ (Software Design Strategies and Methods).

Базова концепція проектування ПЗ – це методологія проектування архітектури за допомогою різних методів (об'єктного, компонентного тощо), процеси життєвого циклу (стандарт ISO/IEC 12207) і техніки – декомпозиція, абстракція, інкапсуляція тощо. На початкових стадіях проектування предметна область декомпонується на окремі об'єкти (при об'єктно-орієнтованому проектуванні) або на компоненти (при компонентному проектуванні). Для подання архітектури програмного забезпечення вибираються відповідні артефакти (нотації, діаграми, блок-схеми і методи).

Ключові питання проектування – це декомпозиція програм на функціональні компоненти для незалежного й одночасного їхнього виконання, розподіл компонентів у середовищі функціонування та їх взаємодія між собою, забезпечення якості та живучості системи тощо.

Проектування архітектури ПЗ проводиться архітектурним стилем, заснованим на визначенні основних елементів структури – підсистем, компонентів, об'єктів і зв'язків між ними.

Архітектура проекту – високорівневе подання структури системи і специфікація її компонентів. Архітектура визначає логіку системи через окремі компоненти системи настільки детально, наскільки це необхідно для написання коду, а також визначає зв'язки між компонентами. Існують також інші види подання структур, що базуються на проектуванні зразків, шаблонів, сімейств програм і каркасів програм.

Один з інструментів проектування архітектури – патерн (шаблон). Це типовий конструктивний елемент ПЗ, що задає взаємодію об'єктів (компонентів) проектованої системи, а також ролі та відповідальності виконавців. Основна мова опису – UML. Патерн може бути структурним, що містить у собі структуру типової композиції з об'єктів і класів, об'єктів, зв'язків тощо; поведінковим, що визначає схеми взаємодії класів об'єктів та їх поведінку, задається діаграмами адитивностей, взаємодії, потоків керування тощо; погоджувальним, що відображає типові схеми розподілу ролей екземплярів об'єктів і способи динамічної генерації структур об'єктів і класів.

Аналіз і оцінка якості проектування ПЗ – це заходи щодо аналізу сформульованих у вимогах атрибутів якості, функцій, структури ПЗ, з перевірки якості результатів проектування за допомогою метрик (функціональних, структурних тощо) і методів моделювання та прототипування.

Нотації проектування дозволяють подати опис об'єкта (елемента) ПЗ і його структуру, а також поведінку системи. Існує два типи нотацій: структурна, поведінкова та множина їх різних зображень.

Структурні нотації – це структурне, блок-схемне або текстове подання аспектів проектування структури ПЗ з об'єктів, компонентів, їх інтерфейсів і взаємозв'язків. До нотацій відносять формальні мови специфікацій і проектування: ADL (Architecture Description Language), UML (Unified Modeling Language), ERD (Entity-Relation Diagrams), IDL (Interface Description Language) тощо. Нотації містять у собі мовний опис архітектури й інтерфейсу, діаграм класів і об'єктів, діаграм сутність-зв'язок, конфігурації компонентів, схем розгортання, а також структурні

діаграми, що задають у наочному вигляді оператори циклу, розгалуження, вибору і наступності.

Поведінкові нотації відбивають динамічний аспект роботи системи та її компонентів. Ними можуть бути діаграми потоків даних (Data Flow), діяльності (Activity), кооперації (Colloboration), послідовності (Sequence), таблиці прийняття рішень (Decision Tables), передумови і постумови (Pre-Post Conditions), формальні мови специфікації (Z, VDM, RAISE) і проектування.

Стратегія і методи проектування ПЗ. До стратегій відносять: проектування вгору, вниз, абстрагування, використання каркасів тощо. Методи є функціонально-орієнтовані, структурні, які базуються на структурному аналізі, структурних картах, діаграмах потоків даних тощо. Вони орієнтовані на ідентифікацію функцій та їх уточнення знизу-вгору, після цього уточнюються діаграми потоків даних і проводиться опис процесів.

В об'єктно-орієнтованому проектуванні ключову роль відіграє спадкування, поліморфізм та інкапсуляція, а також абстрактні структури даних і відображення об'єктів. Підходи, орієнтовані на структури даних, базуються на методі Джексона і використовуються для подання вхідних і вихідних даних структурними діаграмами. Метод UML призначений для опису сценаріїв роботи проекту в наочному діаграмному вигляді. Компонентне проектування ґрунтується на використанні готових компонентів (reuse) з визначеними інтерфейсами та їх інтеграції в конфігурацію, як основи розгортання компонентної системи для її функціонування в операційному середовищі.

Формальні методи опису програм ґрунтуються на специфікаціях, аксіомах, описах деяких попередніх умов, твердженнях і постумовах, що визначають заключну умову одержання правильного результату програмою. Специфікація функцій і даних, якими ці функції оперують, а також умови і твердження – основа доведення правильності програми.

3. Конструювання програмного забезпечення. Конструювання ПЗ – створення ПЗ із конструкцій (блоків, операторів, функцій) і його перевірка методами верифікації та тестування. До інструментів конструювання ПЗ віднесені мови конструювання, програмні методи й інструментальні системи (компілятори, системи керування базами даних, генератори звітів, системи керування версіями, конфігурацією, тестуванням тощо). До формальних засобів опису процесу конструювання ПЗ, взаємозв'язків між людиною і комп'ютером з урахуванням середовища оточення належать структурні діаграми Джексона.

Область знань «Конструювання ПЗ (Software Construction)» містить у собі такі розділи:

- зниження складності (Reduction in Complexity),
- попередження відхилень від стилю (Anticipation of Diversity),
- структуризація перевірок (Structuring for Validation),
- використання стандартів (Use of External Standards).

Зниження складності – це мінімізація, зменшення і локалізація складності конструювання.

Мінімізація складності – це обмеження на обробку складних структур і великих обсягів інформації протягом тривалого періоду часу. Вона досягається, зокрема, використанням у процесі конструювання простих елементів, а також рекомендацій стандартів.

Зменшення складності в конструюванні ПЗ досягається шляхом створення простого коду, що легко читається і спрощує тестування, підвищує продуктивність і впливає на досягнення інших характеристик і обмежень проекту. Зменшення складності спрощує процеси верифікації і тестування результатів конструювання елементів ПЗ.

Локалізація складності – це спосіб конструювання із застосуванням об'єктно-орієнтованого підходу, що лімітує інтерфейс об'єктів, спрощує їхню взаємодію, перевірку правильності самих об'єктів і зв'язків між ними. Локалізація призначена для внесення змін, пов'язаних з виявленими помилками в коді, або коли джерелом помилок є середовище, у якому виконується код.

Щодо попередження відхилень від стилю, то для розв'язання різних задач конструювання застосовуються різні стилі конструювання (лінгвістичний, формальний, візуальний).

Лінгвістичний стиль заснований на використанні словесних інструкцій і виразів для подання окремих елементів (конструкцій) програм. Він призначений для конструювання нескладних конструкцій і зводиться до вигляду традиційних функцій і процедур або реалізується методами логічного і функціонального програмування.

Формальний стиль використовується для точного й однозначного визначення компонентів системи, мінімальної кількості помилок, що можуть виникнути у зв'язку з неоднозначністю визначень або невдалих узагальнень об'єктів конструювання ПЗ.

Візуальний стиль – найбільш універсальний для конструювання прикладного ПЗ. Він дозволяє подавати елемент конструювання у наочному вигляді. Візуальна мова проектування UML надає розробнику набір діаграм для подання статичної і динамічної структур ПЗ. При його застосуванні створюється текстовий і діаграмний описи конструктивних елементів ПЗ, які виводяться на екран дисплея для перегляду і коригування.

Структуризація перевірок передбачає, що побудова ПЗ структурована так, що спрощується пошук помилок, дефектів і різних збоїв у процесі перевірок як на стадії незалежного тестування, так і в процесі експлуатації. Структуризації перевірок сприяють огляд, інспектування, спільний перегляд, модульне тестування із застосуванням автоматизованих засобів тестування тощо.

Щодо зовнішніх стандартів, то конструювання ПЗ залежить від використання стандартів, пов'язаних з мовами програмування, інструментальними засобами й інтерфейсами. При конструюванні має бути визначений достатній набір стандартів для керування і забезпечення координації між визначеними видами діяльності і групами операцій, мінімізації складності, внесення змін, аналізу ризиків тощо.

До таких стандартів відносять: мови програмування (Java, Ada 95, C++ тощо), інтерфейси мов програмування і прикладні інтерфейси платформ Windows (COM, DCOM), CORBA тощо. При конструюванні використовують стандарти мов опису даних (XML, SQL тощо), засобів комунікації (COM, CORBA тощо), інтерфейсних мов (POSIX, IDL, APL, UML тощо).

Перелічені вище розділи області знань «Конструювання ПЗ» у ядрі знань SWEBOK об'єднуються в групу «Основи конструювання». Крім того, розглядаються групи розділів «Керування конструюванням» та «Практичні міркування». Опишемо першу з них детальніше.

Керування конструюванням – це керування процесом конструювання ПЗ, планування, оцінка виконання плану й розроблення заходів щодо внесення змін.

Моделі конструювання містять у собі набір операцій, послідовність дій і результатів. Види моделей визначаються стандартом життєвого циклу, методологіями і практиками. Деякі стандарти життєвого циклу за своєю природою орієнтовані на конструювання типу екстремального програмування і раціонального уніфікованого процесу – RUP (Rational Unified Process).

Планування – це визначення порядку операцій, термінів і рівня виконання заданих умов у процесі конструкторської діяльності за моделлю життєвого циклу, що містить у собі задачі та дії зі створення, перевірки й оцінки показників якості. Виконавці розподіляються за процесами і виконують відповідні задачі з реалізації проміжного й кінцевого продуктів. Остаточний результат вимірюється за обсягом коду, ступенем повторного використання, кількістю помилок і дефектів, а також оцінюються показники якості ПЗ.

Внесення змін пов'язане з помилками, виявленими при перевірці та тестуванні, проводиться з метою збереження функціональної цілісності системи. У випадку виявлення помилок на етапі супроводження приймається рішення про внесення змін або заміну коду в цілому.

4. Тестування програмного забезпечення. Тестування ПЗ – це процес перевірки готової програми в статиці (перегляди, інспекції, налагодження вихідного коду) і в динаміці (прогін на наборі тестових даних) з метою перевірки різних шляхів виконання програми і порівняння отриманих результатів із заздалегідь заданими.

Існують дві форми перевірки коду – модульна й інтеграційна. Спочатку використовують стандарти (IEEE 829:1996 та IEEE 1008:1987) з перевірки і тестування модулів. Потім проводиться інтеграційне тестування модулів системи та їх інтерфейсів у динаміці виконання. Під час різних видів перевірок збираються дані про помилки, дефекти, відмови тощо й оформляється відповідна документація (таблиці типів помилок, частоти і часу виявлення відмов тощо). Зібрані дані використовуються при оцінюванні характеристик якості готового ПЗ, наприклад надійності.

Область знань «Тестування ПЗ (Software Testing)» містить у собі такі розділи:

- основні концепції і визначення тестування (Testing Basic Concepts and definitions),
- рівні тестування (Test Levels),
- техніки тестування (Test Techniques),
- метрики тестування (Test Related Measures),
- керування процесом тестування (Managing the Test Process).

Дана область знань SWEBOK визначає методи перевірки правильності ПЗ: це верифікація, валідація, тестування. Наводяться типи, рівні та техніки тестування ПЗ, методи планування процесу тестування, розроблення тестових наборів даних для прогону ПЗ у режимі випробування модулів або системи в цілому і наступною оцінкою результатів тестування.

Основна концепція тестування – це базові терміни, ключові проблеми та їхній зв’язок з іншими галузями знань. Тестування визначається як процес перевірки правильності програми в динаміці її виконання за тестовими даними. При тестуванні виявляються недоліки: відмови (faults) і дефекти (defects) як причини порушення роботи програми, збої (failures) як небажані

ситуації, помилки (errors) як наслідки збоїв тощо. Базове поняття тестування – тест, що виконується в заданих умовах і за наборами даних. Тестування вважається успішним, якщо знайдений дефект або помилка, і вони відразу усуваються. Ступінь тестованості визначається критерієм покриття системи тестами, перевірки всіх можливих шляхів виконання програм та імовірності припущення стосовно того, що може з'явитися збій або помилкова ситуація в системі.

Рівні тестування:

- тестування окремих елементів – це перевірка окремих, ізольованих і незалежних одна від одної частин ПЗ;

- інтеграційне тестування орієнтоване на перевірку зв'язків і взаємодії компонентів (інтерфейсів), що можуть розміщуватися на різних архітектурних платформах розподіленого середовища;

- тестування системи – це перевірка правильності функціонування системи, пошук і виявлення відмов і дефектів у системі та їхнє усунення. При цьому контролюється виконання сформульованих нефункціональних вимог (безпека, надійність тощо) у системі, правильність подання і взаємодія зовнішніх інтерфейсів системи із зовнішнім середовищем.

На всіх рівнях тестування застосовуються методи:

- функціонального тестування, які забезпечують перевірку реалізації функцій, що визначені у вимогах, а також правильності їх виконання;

- регресійного тестування, що орієнтоване на повторне вибіркове тестування системи або її компонентів після внесення в них змін на тих самих тестах, що й до модифікації;

- тестування ефективності – це перевірка продуктивності, пропускної здатності, максимального обсягу даних і системних обмежень відповідно до вимог;

- стрес-тестування – це перевірка поведінки системи при максимально допустимому навантаженні або в разі його перевищення;

- альфа- і бета-тестування – це тестування системи групою тестування організації-розробника (альфа) і тестування системи «зовнішніми» користувачами (бета);

- конфігураційного тестування – перевірка структури й ідентифікації системи, а також роботи системи на різних конфігураціях апаратури й устаткування.

Тестуванню підлягають також перевірка реалізації вимог і забезпечення параметрів налаштовування й розміщення компонентів ПЗ на заданій кількості та типах комп'ютерів та в умовах конкретного середовища.

Техніки тестування базуються на певних теоретичних і практичних положеннях щодо проектування (компонентного, об'єктно-орієнтованого, сервісного тощо), а також на таких даних:

- інформація про структуру ПЗ або системи в документації («біла скринька»);

- підбір тестових наборів даних для перевірки правильності роботи компонентів і системи в цілому без знання їх структури («чорна скринька»);

- аналіз граничних значень, таблиць прийняття рішень, потоків даних, статистики відмов тощо;

- блок-схеми побудови програм і складання наборів тестів для покриття системи цими тестами;

- виявлені та зафіксовані в таблицях системи дефекти, передумови і постумови виконання, структурні характеристики системи (кількість модулів, обсяг даних тощо).

Для вимірювання результатів тестування ПЗ й оцінки якості використовуються метрики. Вимір як частина планування і розробки тестів базується на розмірі програм, їх структурі та кількості виявлених помилок і дефектів. Метрики тестування – це вимірювання процесу планування, проектування і тестування, а також результатів тестування на основі таксономії відмов і дефектів, покриття границь тестування, перевірки потоків даних тощо.

Процес тестування документується і, відповідно до стандарту IEEE 829:1995, містить у собі опис тестових документів, їх зв'язку між собою та із задачами тестування. Без документації на процес тестування неможливо провести сертифікацію продукту за моделями зрілості, зокрема моделлю CMM. Після завершення

тестування оцінюється вартість і ризики ПЗ, викликані збоями або недостатньою надійністю роботи системи. Вартість тестування – одне з обмежень, на основі якого приймається рішення про його припинення або продовження.

Керування тестуванням:

- планування процесу тестування (складання планів, тестів, наборів даних) та оцінювання показників якості готового продукту;
- проведення тестування компонентів повторного використання і патернів як основних об'єктів складання ПЗ;
- генерація необхідних тестових сценаріїв, що відповідають середовищу виконання ПЗ;
- верифікація правильності реалізації системи і валідація реалізації вимог до ПЗ;
- збирання даних щодо відмов, помилок і виявлених непередбачуваних ситуацій при виконанні програмного продукту;
- підготовка звітів за результатами тестування й оцінка характеристик системи.

Відповідно до стандарту ISO/IEC 12207, тестування ПЗ розглядається як невід'ємна частина життєвого циклу.

5. Супровід програмного забезпечення. Супровід ПЗ – сукупність дій із забезпечення його роботи, внесення змін при виявленні помилок, адаптації ПЗ до нового середовища функціонування, а також підвищення продуктивності або поліпшення деяких характеристик ПЗ. Супровід, відповідно до стандартів ISO/IEC 12207 та ISO/IEC 14764, проводиться з метою виконання й модифікації програмного продукту в процесі експлуатації за умови збереження його цілісності.

Область знань «Супровід ПЗ (Software Maintenance)» складається з таких розділів:

- основні концепції (Basic Concepts),
- процес супроводу (Process Maintenance),
- ключові питання супроводу ПЗ (key Issue in Software Maintenance),
- техніки супроводу (Techniques for Maintenance).

Супровід розглядається з точки зору задоволення вимог споживача в готовому ПЗ, коректності його виконання, процесів навчання й оперативного обліку.

Основні концепції – це базові визначення і термінологія, підходи до еволюції і супроводу ПЗ, до оцінки вартості супроводу тощо. До основних концепцій можна віднести життєвий цикл ПЗ (стандарт ISO/IEC 12207) і складання документації. Головне призначення цієї сфери знань полягає у виконанні готової програмної системи, фіксації помилок, що виникають при виконанні, дослідженні їх причин, аналізі необхідності модифікації системи з метою усунення помилок, оцінці вартості робіт із проведення змін функцій і системи в цілому. Розглядаються проблеми, пов'язані з ускладненістю продукту при великій кількості змін, і методи її подолання.

Процес супроводу містить у собі моделі процесу супроводу і планування діяльності людей, що проводять запуск ПЗ, перевірку правильності його виконання і внесення у нього змін. Цей процес, згідно зі стандартом ISO/IEC 14764, проводиться шляхом:

- коригування, тобто зміни продукту для усунення виявлених помилок або нереалізованих задач;
- адаптації, тобто налаштовування продукту в умовах експлуатації, що змінилися, або в новому середовищі виконання;
- поліпшення, тобто еволюційної зміни продукту для підвищення продуктивності або рівня супроводу;
- перевірки ПЗ, пошуку і виправлення помилок при експлуатації системи.

Ключові питання супроводу ПЗ – це управлінські, вимірювальні та вартісні.

Суть управлінських питань – контроль ПЗ при модифікації й удосконаленні функцій і недопущення зниження продуктивності системи. Питання вимірювання пов'язане з оцінкою характеристик системи після її модифікації, а також повторного тестування для оцінки показників якості. Вартісні питання пов'язані з оцінкою витрат на супровід залежно від його типу, кваліфікації персоналу, платформи тощо.

Відомий фахівець у сфері ПЗ Дж. Леман (1970 р.) запропонував розглядати супровід як еволюційну розробку програмних систем, оскільки здана в експлуатацію система не завжди цілком завершена, її треба змінювати протягом терміну експлуатації. Внаслідок змін система стає більш складною і погано керованою. У зв'язку з цим виникає проблема зменшення її складності. До технологій еволюції ПЗ відносять реінженерію, реверсну інженерію та рефакторинг.

Реінженерія – це вдосконалення застарілого ПЗ шляхом його реорганізації або реструктуризації, а також перепрограмування окремих елементів або налаштовування параметрів на іншу платформу чи середовище виконання зі збереженням зручності його супроводу.

Реверсна інженерія полягає у відновленні специфікації (графів викликів, потоків даних тощо) за отриманим кодом системи для її аналізу на більш високому рівні. Відновлюється ідентифікація компонентів і зв'язків між ними для забезпечення перепрограмування системи на нову платформу. Найчастіше реверсна інженерія застосовується після того, як у код ПЗ було внесено багато змін і воно стало некерованим або змінилася платформа комп'ютера.

Рефакторинг – це реорганізація коду для поліпшення характеристик і показників якості об'єктно-орієнтованих і компонентних програм без зміни їх поведінки. Цей процес реалізується шляхом поступової зміни окремих операцій над текстами, інтерфейсами, середовищем програмування і виконання ПЗ, а також налаштування або внесення змін в інструментальні засоби підтримки ПЗ. Якщо при зміні зберігається формат існуючої системи, то рефакторинг – один з варіантів реверсної інженерії.

6. Керування конфігурацією. Керування конфігурацією – це ідентифікація компонентів системи, визначення функціональних, фізичних характеристик системи, апаратного і програмного забезпечення для контролю виконання, внесення змін і трасування конфігурації. Процес керування визначений як один з допоміжних процесів життєвого циклу (ISO/IEC 12207),

виконуваний технічним і адміністративним менеджментом проекту. При цьому складаються звіти про зміни, внесені у конфігурацію, і ступінь їхньої реалізації, а також проводиться перевірка відповідності внесених змін заданим вимогам.

Конфігурація системи – це сукупність функцій, програмного і технічного забезпечення системи, можливі їх комбінації залежно від наявності устаткування, загальносистемних засобів і вимог до продукту.

Конфігурація ПЗ складається з набору функціональних і технічних характеристик ПЗ, заданих у технічній документації і реалізованих у готовому продукті. Це сполучення різних елементів продукту із заданими процедурами збирання компонентів і налаштовування на середовище. Вихідними елементами конфігурації є графік розробки, проектна документація, вихідний виконуваний код, бібліотека компонентів, інструкції з установки і розгортання системи.

Область знань «Керування конфігурацією ПЗ (Software Configuration Management – SCM)» складається з таких розділів:

- керування процесом конфігурації (Management of SMC Process);
- ідентифікація конфігурації ПЗ (Software Configuration Identification);
- контроль конфігурації ПЗ (Software Configuration Control);
- облік статусу (поведінка або стани) конфігурації ПЗ (Software Configuration Status Accounting);
- аудит конфігурації ПЗ (Software Configuration Auditing);
- керування версіями ПЗ і доставкою (Software Release Management and Delivery).

Керування процесом конфігурації – це діяльність з контролю еволюції і цілісності продукту при ідентифікації, змінах і забезпеченні звітною інформацією, що стосується конфігурації. Вона містить у собі:

- систематичне відстеження внесених змін в окремі складові частини конфігурації, виконання аудиту змін і автоматизованого контролю за внесенням змін у конфігурацію системи або в ПЗ;

- підтримку цілісності конфігурації, її аудит і забезпечення внесення змін в елементи конфігурації;

- ревізію конфігурації з метою перевірки наявності розроблених програмних або апаратних засобів і узгодження версії конфігурації із заданими вимогами;

- трасування змін у конфігурації на етапах супроводу й експлуатації ПЗ.

Ідентифікація конфігурації ПЗ полягає в документуванні функціональних і фізичних характеристик елементів конфігурації, а також в оформленні технічної документації на елементи конфігурації.

Контроль конфігурації ПЗ – це роботи з координації, затвердження або відкидання реалізованих змін в елементах конфігурації після ідентифікації, а також з аналізу вхідних компонентів конфігурації.

Облік статусу або стану конфігурації ПЗ – комплекс заходів для визначення ступеня зміни конфігурації, а також правильності внесених змін у систему при супроводі. Інформація і кількісні показники накопичуються у відповідній базі даних і використовуються при складанні звітності, оцінюванні якості й виконанні процесів життєвого циклу.

Аудит конфігурації – це діяльність, що виконується для оцінки відповідності продукту і процесів стандартам, інструкціям, планам і процедурам. Аудит визначає ступінь відповідності конфігурації функціональним і фізичним (апаратним) характеристикам системи.

Керування версіями ПЗ – це відстеження наявної версії компонентів конфігурації; складання компонентів; створення нових версій системи на основі існуючих шляхом внесення змін у конфігурацію; узгодження версії продукту з вимогами і проведеними змінами на етапах життєвого циклу; забезпечення оперативного доступу до інформації про елементи конфігурації і системи, до яких вони належать. Дане керування містить у собі такі основні поняття.

Базис ПЗ (baseline) – формально позначений набір елементів ПЗ, зафіксований на етапах життєвого циклу.

Бібліотека ПЗ – колекція об'єктів ПЗ і документації, призначена для полегшення процесу розроблення, використання і супроводження.

Складання ПЗ – об'єднання коректних елементів і конфігураційних даних у єдину виконувану програму.

7. Керування інженерією програмного забезпечення. Керування інженерією ПЗ (Software Engineering Management) – керування роботами команди розробників ПЗ у процесі виконання плану проекту, визначення критеріїв ефективності роботи цієї команди й оцінка процесів і продуктів проекту з використанням загальних методів планування і контролю робіт.

Як будь-яке керування, воно полягає у плануванні, координації, контролі, вимірі й обліку виконаних робіт у процесі розроблення програмного проекту. Координацію людських, фінансових і технічних ресурсів виконує менеджер проекту аналогічно до того, як це робиться в технічних проектах. У його обов'язки входить дотримання запланованих бюджетних і часових характеристик і обмежень, стандартів і сформульованих вимог.

Загальні питання керування проектом містяться в ядрі знань PMBOK, а також у стандарті ISO/IEC 12207 – Software life cycle processes, де керування проектом розглядається як організаційний процес життєвого циклу.

Область знань, «Керування інженерією ПЗ (Software Engineering Management)» складається з таких розділів:

- організаційне керування (Organizational Management),
- керування процесом/проектом (Process/Project Management),
- інженерія вимірювання ПЗ (Software Engineering Measurement).

Організаційне керування – це планування і складання графіка робіт, підбір і керування персоналом, контроль виконання й оцінка вартості робіт згідно з прийнятими стандартами і договорами із замовником. Головним об'єктом організаційного керування проектом є персонал (навчання, мотивація тощо), комунікації між співробітниками (зустрічі, презентації тощо), а також попередження й усунення ризику невиконання проекту.

Для керування проектом створюється спеціальна структура колективу. Фахівці розподіляються за видами робіт і розв'язують задачі проекту під керуванням менеджера з урахуванням заданої вартості та термінів розробки. Для реалізації завдань проекту підбираються необхідні програмні, інструментальні й апаратні засоби.

Керування проектом/процесом – це складання плану проекту, побудова графіків робіт (мережних або часових діаграм) з урахуванням наявних ресурсів, розподіл персоналу за видами робіт у проекті, виходячи із заданих термінів і вартості їх виконання. Для ефективного керування проектом проводиться аналіз фінансової, технічної, операційної і соціальної політики організації-розробника для вибору правильної стратегії виконання робіт і контролю плану, а також проміжних продуктів (проектних рішень, діаграм UML, алгоритмів тощо).

У задачі керування проектом входять також уточнення вимог, перевірка їх відповідності заданим специфікаціям характеристик якості, а також верифікація функцій проекту. Процес керування базується на планових термінах, що відображені мережними діаграмами: PERT (Program Evaluation and Review Technique), CPM (Critical Path Method) тощо. У них указуються роботи, зв'язки між ними і час виконання.

На сьогоднішній день найбільш поширена мережна діаграма PERT – граф, у вершинах якого розміщуються роботи, а дуги задають взаємні зв'язки між цими роботами. Інший тип мережної діаграми – CPM. У її вершинах указують події, а роботи задають лініями між двома вузлами-подіями. Очікуваний час виконання робіт за допомогою мережних діаграм оцінюється середнім ваговим значенням трьох оцінок: оптимістичної, песимістичної й очікуваної, тобто ймовірнісної. Ці оцінки надають експерти, що враховують обсяги виконаної роботи і відведений на неї час.

Коректно складений план забезпечує виконання вимог і цілей проекту. Контроль здійснюється при виконанні і внесенні змін у проект з урахуванням ризиків і прийнятих рішень щодо їх мінімізації.

Під ризиком розуміють імовірність виникнення несприятливих обставин, що можуть негативно вплинути на керування розробкою (наприклад, звільнення співробітника і відсутність заміни для продовження робіт). При складанні плану проекту проводиться ідентифікація й аналіз ризику, планування непередбачених ситуацій щодо ризиків. Запобігання ризику полягає у виконанні дій, що знімають ризик (наприклад, збільшення часу розробки). Причиною появи ризику може бути реорганізація проекту, баз даних або транзакцій, а також помилки при виконанні ПЗ.

Інженерія вимірювання ПЗ проводиться з метою визначення окремих характеристик продуктів і процесів (наприклад, кількість рядків у продукті, помилок у специфікаціях тощо). Попередньо проводяться роботи з вибору метрик процесів і продуктів з урахуванням обставин, що впливають на вимірювання характеристик програмного продукту.

Інженерія вимірювання – удосконалювання процесів керування проектом; оцінювання часових витрат і вартості ПЗ, їх регулювання; визначення категорій ризиків і відстеження чинників для регулярного розрахунку ймовірностей їх виникнення; перевірка заданих у вимогах показників якості окремих продуктів і проекту в цілому.

Проведення різного роду вимірювань – важливий принцип будь-якої інженерної діяльності. У програмному проекті результати вимірювань необхідні замовнику і споживачу для встановлення правильності реалізації проекту. Без вимірювання в інженерії ПЗ процес керування стає неефективним і перетворюється в самоціль.

8. Процес інженерії. Процес інженерії – метарівень, що визначає основні поняття, способи реалізації, оцінювання, вимірювання, дії з керування змінами й удосконалення самого процесу. Як уже згадувалася, для оцінювання й удосконалення процесу програмної інженерії застосовується модель зрілості CMM, яку розроблено Інститутом програмної інженерії SEI (Software Engineering Institute). Ця модель встановлює рівні готовності організації-розробника ПЗ створювати задовільно,

середньо, добре і дуже добре програмну продукцію. Поняття рівня готовності визначається наявністю в організації необхідних ресурсів (людських, програмних, технічних і фінансових), стандартів і методик, а також здатністю колективу створювати програмні продукти. Модель CMM має п'ять рівнів. Перший і другий рівні фіксують недостатню готовність виконувати розробку продукту. Третій, четвертий і п'ятий рівні характеризують певний ступінь готовності, зрілості і здатності фахівців (а отже, й організації) виготовляти, відповідно, середній, гарний і відмінний продукти. Чим вищий рівень зрілості, тим більше вимог ставиться до процесу програмної інженерії, придатного для виконання цілей і задач створення продукту, що задовольняє користувача.

Існують різновиди цієї моделі: CMM-SW (Software) для оцінки зрілості ПЗ, CMMI (CMM Integrated) – для обліку потреб великих державних структур у ПЗ, а також інші моделі, наприклад, Bootstrap – для оцінки зрілості малих і середніх комерційних компаній, стандарт ISO 15504 (Software Process Improvement and Capability) – для удосконалення процесу (наприклад, удосконалювати процес на другому рівні, щоб одержати сертифікат на третій рівень зрілості).

Концепція зрілості процесу програмної інженерії ґрунтується на процесі ПЗ (software process), широті його можливостей (software process capability), результативності (software process performance) і зрілості (software process maturity). Процес ПЗ у моделі CMM – це множина діяльностей (activities), методів (methods), практичних прийомів (practices), що використовують при розробці ПЗ шляхом планування робіт і оцінювання проміжних результатів, які приводять до кінцевого продукту високої якості.

Область знань «Процес програмної інженерії (Software Engineering Process)» складається з таких розділів:

- концепції процесу інженерії ПЗ (Software Engineering Process Concepts);
- інфраструктура процесу (Process Infrastructure);
- визначення процесу (Process Definition);

- оцінки процесу (Process Assessments);
- якісний аналіз процесу (Qualitative Process Analysis);
- виконання і змінювання процесу (Process Implementation and Change).

Концепції процесу інженерії ПЗ – задачі та дії, пов'язані з керуванням, реалізацією, оцінкою, змінами й удосконаленням процесу або ПЗ. Ціль керування процесом – це створення інфраструктури процесу, виділення необхідних ресурсів, планування реалізації та зміни процесу з метою впровадження його у практику і, нарешті, оцінка переваг від його впровадження у практику проектування ПЗ.

Інфраструктура процесу містить у собі ресурси (людські, технічні, інформаційні і програмні), стандарти, методики керування якістю, проектом і структуру колективу розробників ПЗ типу: команда, бригада, експериментальна фабрика (Experimental Factory), каркас виробництва на лінії продуктів (Framework for Product Line Practice) тощо. До основних задач інфраструктури належать керування і комунікації в колективі, інженерні методи виробництва програмного продукту й удосконалення процесу з накопиченим досвідом розробки ПЗ.

Визначення процесу ґрунтується на: типах процесів і моделей (каскадна, спіральна, ітераційна тощо); моделях життєвого циклу процесів і засобів, стандартах життєвого циклу ПЗ (ISO/IEC 12207, ISO/IEC 15504, IEEE std. 1074, IEEE std. 1219 тощо); методах і нотаціях подання процесів і автоматизованих засобів їх підтримки. Основною метою процесу є підвищення якості одержуваного продукту, поліпшення різних аспектів програмної інженерії, автоматизація й удосконалення процесів.

Оцінка процесу проводиться з використанням відповідних моделей і методів оцінки. Наприклад, оцінка потенційної здатності фахівця до розробки і виконання відповідного процесу, а також оцінювання зрілості процесу, згідно з яким проводиться розробка ПЗ.

Оцінки стосуються також технічних робіт у сфері програмної інженерії, керування персоналом і якості ПЗ. Для цього проводяться експериментальні дослідження середовища,

збирання інформації, моделювання, класифікація отриманих помилок і дефектів, а також статичний аналіз недоліків процесу порівняно з існуючими стандартами (наприклад, ISO/IEC 12207) і потенційних аспектів необхідності вдосконалювати процес.

Якісний аналіз процесу полягає в ідентифікації і пошуку «слабких місць» у процесі створення ПЗ на початку його функціонування і після експлуатації. Розглядається дві техніки аналізу: огляд даних і порівняння процесу з основними положеннями стандарту ISO/IEC 12207, збирання даних про якість процесів; аналіз головних причин відмов у функціонуванні ПЗ, повернення назад від точки виникнення відхилення до точки правильної роботи системи для з'ясування причин зміни процесу. На якість результатів проекту і процесу впливають застосовувані інструменти і досвід фахівців.

Існує ряд фундаментальних аспектів вимірювань у програмній інженерії, що покладені в основу детальних вимірювань процесу. Оцінка вдосконалення процесу проводиться шляхом встановлення кількісних характеристик процесу і продуктів. Після процесу розгортання ПЗ виконуються обчислення функцій і аналіз отриманих результатів, які можуть застосовуватися при оцінюванні якості, продуктивності, трудовитрат тощо. Якщо результати не задовольняють користувача ПЗ, проводять обговорення і приймають рішення щодо необхідності виправлення ситуації шляхом або внесення зміни у процес, або вдосконалення процесу, а також організаційну структуру і деякі інструменти керування змінами.

9. Методи та інструменти інженерії. Методи забезпечують проектування, реалізацію і виконання ПЗ. Вони накладають деякі обмеження на інженерію ПЗ у зв'язку з особливостями застосування їхніх процедур, а також дають можливість оцінити й перевірити процеси та продукти. Інструменти уможливлюють програмну підтримку окремих методів інженерії ПЗ для автоматизованого виконання завдань процесів життєвого циклу.

Область знань «Методи та інструменти інженерії ПЗ (Software Engineering Tools and Methods)» складається з розділів:

- інструменти інженерії ПЗ (Software Engineering Tools);

- методи інженерії ПЗ (Software Engineering Methods).

Методи інженерії ПЗ – це евристичні методи (heuristic methods), формальні методи (formal methods) і методи прототипування (prototyping methods).

Евристичні методи містять у собі: структурні методи, засновані на функціональній парадигмі; методи, орієнтовані на структури даних, якими маніпулює ПЗ; об'єктно-орієнтовані методи, що розглядають предметну область як колекцію об'єктів; методи, орієнтовані на конкретну область застосування, наприклад, на системи реального часу, безпеки тощо.

Формальні методи базуються на формальних специфікаціях, аналізі, доведенні та верифікації програм. Специфікація записується мовою, синтаксис і семантика якої визначені формально й ґрунтуються на математичних концепціях (алгебрі, теорії множин, логіці). Розрізняють такі категорії формальних методів:

- мови і нотації специфікації (specification languages and notations), орієнтовані на модель, властивості та поведінку;

- уточнення специфікації (refinement specification) шляхом трансформації в кінцевий результат, близький до кінцевого програмного продукту, що виконується;

- методи верифікації/доведення (verification/proving properties), що використовують твердження (теореми), передумови і постумови, формально описуються і застосовуються для встановлення правильності специфікації програм.

Методи доведення застосовувалися в основному в теоретичних експериментах. Наприкінці минулого століття їх застосування було обмежено через трудомісткість і економічну невигідність. У 2005 р. проблема верифікації знову набула актуальності в запропонованому новому міжнародному проекті «Цілісний автоматизований набір інструментів для перевірки коректності ПЗ», який поставив такі перспективні задачі:

- розробка єдиної теорії побудови й аналізу програм;

- побудова багатостороннього інтегрованого набору інструментів верифікації на всіх виробничих процесах – розроблення формальних специфікацій, їх доведення і перевірка

правильності, генерація програм і тестових прикладів, уточнення, аналіз і оцінка;

- створення депозитарію формальних специфікацій, верифікованих програмних об'єктів різних типів і видів.

Формальні методи верифікації будуть охоплювати всі аспекти створення і перевірки правильності програм. Це приведе до створення потужної верифікованої виробничої основи і сприятиме значному зменшенню помилок у ПЗ.

Методи прототипування (Prototyping Methods) засновані на використанні прототипу ПЗ для моделювання на ньому завдань нової системи і базуються на:

- стилях прототипування, що уособлюють тривалість використання прототипів, наприклад, стиль створення тимчасово використовуваних прототипів (throw away);

- моделях еволюційного прототипування – перетворення прототипу в кінцевий продукт і розроблення специфікацій, відповідно до яких він виконується;

- техніках оцінки/дослідження (evaluation) результатів прототипування.

Інструменти інженерії ПЗ забезпечують автоматизовану підтримку процесів розроблення ПЗ і містять у собі множину інструментів, що охоплюють усі процеси життєвого циклу.

Наприклад, інструменти роботи з вимогами (Software Requirements Tools) – це:

- інструменти розробки (Requirement Development) і керування вимогами (Requirement Management), орієнтовані на аналіз, збирання, специфікацію та перевірку вимог;

- інструменти трасування вимог (Requirement traceability tools), що є невід'ємною частиною роботи з вимогами, функціональний зміст яких залежить від складності проектів і рівня зрілості процесів.

Інструменти проектування (Software Design Tools) – це інструменти для створення ПЗ із застосуванням базових нотацій (структурної SADT/IDEF, моделювання UML і т.п.).

Інструменти конструювання ПЗ (Software Construction Tools) – це інструменти для трансляції і об'єднання програм. До них належать:

- редактори програм (program editors) і програми редагування загального призначення;
- компілятори і генератори коду (compilers and code generators) як самостійні засоби об'єднання програмних компонентів в інтегрованому середовищі для одержання вихідного продукту з використанням препроцесорів, складальників, завантажувачів тощо;
- інтерпретатори (interpreters), які забезпечують контрольоване виконання програм за їх описом. Намітилася тенденція злиття інтерпретаторів і компіляторів (наприклад, Java, Microsoft .NET);
- наладники (debuggers), призначені для перевірки правильності опису вихідних програм і усунення помилок;
- інтегроване середовище розробки (IDE – integrated development environment) та бібліотеки компонентів (libraries components), що утворюють середовище виконання процесу розробки ПЗ;
- програмні платформи (Java, J2EE, Microsoft .NET тощо) і платформи для розподілених обчислень (CORBA, Web Services тощо).

Інструменти тестування (Software Testing Tools) – це:

- генератори тестів (test generators), що допомагають у розробці сценаріїв тестування;
- засоби виконання тестів (test execution frameworks), які забезпечують виконання тестових сценаріїв та відстежують поведінку об'єктів тестування;
- інструменти оцінки тестів (test evaluation tools), які підтримують оцінювання результатів виконання тестів і ступеня відповідності поведінки тестованого об'єкта очікуваній поведінці;
- засоби керування тестами (test management tools), що забезпечують інженерне керування процесом тестування ПЗ;
- інструменти аналізу продуктивності (performance analysis tools), кількісної її оцінки та оцінки поводження програм у процесі виконання.

Інструменти супроводу (Software Maintenance Tools) містять у собі:

- інструменти полегшення розуміння (comprehension tools) програм, наприклад різні засоби візуалізації;

- інструменти реінженерії (reengineering tools), що підтримують діяльність з перетворення програм і зворотної інженерії (reverse engineering) для відновлення специфікації та архітектури застарілого ПЗ з метою подальшої генерації нового продукту.

Інструменти конфігураційного керування (Software Configuration Management Tools) – це:

- інструменти відстеження (tracking) дефектів;

- інструменти керування версіями;

- інструменти керування складанням, випуском версії (конфігурації) продукту та його інсталяції.

Інструменти керування інженерною діяльністю (Software Engineering Management Tools) підрозділяються на:

- інструменти планування і відстеження ходу проектів, кількісної оцінки зусиль і вартості робіт у проекті (наприклад, Microsoft Project);

- інструменти керування ризиками, які використовуються для ідентифікації, моніторингу ризиків і оцінки завданої шкоди;

- інструменти кількісної оцінки властивостей ПЗ шляхом вимірювань і розрахунків остаточного значення надійності та якості.

Інструменти підтримки процесів (Software Engineering Process Tools) розділені на:

- інструменти моделювання та опису моделей ПЗ (наприклад, UML і його інструменти);

- інструменти керування програмними проектами (наприклад, Microsoft Project);

- інструменти керування конфігурацією для підтримки версій і всіх артефактів проекту.

Інструменти забезпечення якості (Software Quality Tools) діляться на дві категорій:

- інструменти інспектування для підтримки перегляду (review) і аудиту;

- інструменти статичного аналізу артефактів, даних, потоків робіт і перевірки їх властивостей на відповідність показникам.

Додаткові аспекти інструментального забезпечення (Miscellaneous Tool Issues) стосуються:

- техніки інтеграції інструментів (платформ, зображень, процесів, даних) для їх природного сполучення в інтегрованому середовищі;
- метаінструментів для генерації інших інструментів для ПЗ;
- оцінки інструментів при їх еволюції.

10. Якість програмного забезпечення. Якість ПЗ – набір властивостей продукту (сервісу або програм), що характеризують його здатність задовольнити встановлені або передбачувані потреби замовника. Поняття якості має різні інтерпретації залежно від конкретної програмної системи і вимог до неї. Крім того, у різних джерелах таксономія (класифікація) характеристик у моделі якості відрізняється.

Моделі мають різну кількість рівнів і повністю або частково збігаються щодо набору характеристик якості. Наприклад, модель якості Мак Колла на найвищому рівні має три характеристики: функціональність, модифікованість і переносність, а на нижчих рівнях моделі – 11 підхарактеристик якості та 18 критеріїв (атрибутів) якості.

Стандарт ISO 9126:2001 регламентує зовнішні і внутрішні характеристики якості. Перші відображають вимоги до функціонування програмного продукту. Для кількісного встановлення критеріїв якості, за якими буде здійснюватися перевірка і підтвердження відповідності ПЗ заданим вимогам, визначаються відповідні зовнішні вимірювані властивості (зовнішні атрибути) ПЗ, метрики (наприклад, час виконання окремих компонентів), діапазони зміни значень і моделі їх оцінки. Метрики використовуються на стадії тестування або функціонування і називаються зовнішніми метриками. Вони являють собою моделі оцінки атрибутів.

Внутрішні характеристики якості і внутрішні атрибути ПЗ використовуються для складання плану досягнення необхідних зовнішніх характеристик якості продукту. Для квантифікації

внутрішніх характеристик якості застосовують внутрішні метрики як інструмент перевірки відповідності проміжних продуктів внутрішнім вимогам до якості, які формулюються на етапах, що передують тестуванню.

Зовнішні і внутрішні характеристики якості відображають властивості самого ПЗ (працюючого або непрацюючого), а також погляд замовника і розробника на таке ПЗ. Безпосереднього кінцевого користувача ПЗ цікавить експлуатаційна якість ПЗ – сукупний ефект від досягнення характеристик якості, що виміряється строком результату, а не властивістю самого ПЗ. Це поняття ширше, ніж будь-яка окрема характеристика (наприклад, зручність використання або надійність).

Остаточна оцінка якості проводиться відповідно до стандарту ISO/IEC 14598. Якість може підвищуватися за рахунок постійного поліпшення використовуваного продукту виявленням, усуненням дефектів у ПЗ та запобіганням їх появі.

Область знань «Якість ПЗ (Software Quality)» складається з таких розділів:

- концепція якості ПЗ (Software Quality Concepts);
- визначення і планування якості (Definition & Planning for Quality);
- техніки й види діяльності, що забезпечують гарантію якості, валідацію і верифікацію (Activities and Techniques for Software Quality Assurance, Validation & Verification – V&V);
- вимірювання при аналізі якості ПЗ (Measurement in Software Quality Analysis).

Концепція якості ПЗ – це зовнішні та внутрішні характеристики якості, їхні метрики, а також моделі якості, визначені на множині цих характеристик, що наведені в стандартах з якості. До характеристик якості належать:

- функціональність (functionality);
- надійність (reliability);
- зручність застосування (usability);
- ефективність (efficiency);
- супровід (maintainability);
- переносимість (portability).

Базова модель якості містить у собі ці характеристики і вони притаманні будь-якому типу програмних продуктів. При розробці вимог замовник формулює такі вимоги до якості, які найбільшою мірою підходять для програмного продукту, що замовляється.

Визначення і планування якості ПЗ ґрунтується на положеннях стандартів у цій області, складанні планів і графіків робіт, процедурах перевірки тощо. План забезпечення якості містить у собі набір дій для перевірки процесів забезпечення якості (верифікація, валідація тощо) і формування документа з керування якістю.

Планування якості призначено для підтримки керування процесами досягнення якості продуктів проекту (зокрема проміжних та робочих) і ресурсів – програмних, технічних, виконавчих тощо. Воно також передбачає керування вимогами до процесів і продуктів і полягає в такому:

- виготовлення продукту в межах часу, який забезпечує задані характеристики якості;
- планування процесів для гарантії одержання якості;
- вибір методів оцінки запланованих характеристик якості та встановлення відповідності продукту сформульованим вимогам.

У стандарті ISO/IEC 12207 визначені спеціальні процеси забезпечення якості – верифікація, валідація (атестація), спільний аналіз і аудит.

Види діяльності і техніки гарантії якості містять у собі, зокрема, інспекцію, верифікацію і валідацію ПЗ.

Інспекція ПЗ – аналіз і перевірка різних видів подання системи і ПЗ (специфікації, архітектурної схеми, діаграм, початкового коду тощо). Виконується на всіх етапах життєвого циклу розробки ПЗ.

Верифікація ПЗ – процес забезпечення правильної реалізації ПЗ відповідно до специфікацій, виконується протягом усього життєвого циклу. Верифікація дає відповідь на питання, чи правильно створюється система.

Валідація ПЗ – процес перевірки відповідності ПЗ функціональним і нефункціональним вимогам та очікуваним потребам замовника.

Верифікація і валідація (V&V) можуть виконуватися, починаючи з ранніх стадій життєвого циклу. Вони орієнтовані на отримання правильних функцій ПЗ, плануються і забезпечуються визначеними ресурсами з чітким розподілом ролей. Перевірка ґрунтується на використанні відповідних технік тестування для виявлення тих або інших дефектів і збирання статистики. Після зібрання даних оцінюється правильність реалізації вимог і роботи ПЗ у заданих умовах.

Вимірювання при аналізі якості ПЗ ґрунтується на метриках продукту і даних, зібраних у процесі створення продукту при заданих ресурсах (оцінок процесів, ПЗ і його моделей) і передбачає документування вимірів. Оцінювання якості продукту полягає у вимірюванні й оцінюванні якісних показників за допомогою даних про різні типи помилок і відмов під час тестування ПЗ і виконання коду на тестових даних. Ці дані аналізуються, перевіряються і використовуються при якісній і кількісній оцінці ПЗ.

Для імітації роботи системи в режимі тестування розробляються тести з реальними вхідними даними для перевірки правильності роботи ПЗ на різних фрагментах програми і шляхах проходження в них операторів. У процесі тестування ПЗ виявляються різного роду помилки, кількість яких може істотно вплинути на одержання правильного та якісного результату.

З урахуванням типів виявлених помилок можна встановити наявність (або відсутність) відповідності реалізованих і нереалізованих функцій, заданих у вимогах до системи, а також оцінити способи реалізації нефункціональних вимог (продуктивності, надійності тощо). Оцінюються процеси керування планами, інспекціями, прогонами тощо. За цими оцінками приймаються рішення про завершення розробки продукту проекту і передачі його замовнику в експлуатацію, під час якої можуть бути внесені зміни, як-от усунення помилок, визначення адекватності планів і вимог, оцінки ризиків переробки ПЗ тощо.

Мета інспекцій – виявлення різних аномальних станів у ПЗ незалежними фахівцями та командами експертів із залученням

авторів проміжного або кінцевого продукту. Експерти інспектують виконання вимог, інтерфейси, вхідні дані тощо, а потім документують виявлені відхилення в проекті.

Призначення аудиту – незалежна оцінка продуктів і процесів на відповідність регулюючим і регламентуючим документам (планам, стандартам тощо), формулювання звіту про випадки невідповідності та пропозицій для їх коригування.

Таким чином, розгляд розділів SWEBOK свідчить про те, що це ядро містить весь необхідний набір знань з програмної інженерії, який повинні мати фахівці різних профілів (аналітики, інженери, програмісти, оцінювачі, контролери тощо), що виробляють програмний продукт.

Зазначимо, що ядро знань SWEBOK не позбавлене недоліків тактичного характеру. Так, між областями знань у цьому ядрі існують перетини за методами, концепціями і стратегіями, а деякі важливі напрями програмної інженерії взагалі не відбиті у ньому наявними областями знань. Це стосується, наприклад, методів доведення правильності програм, еволюції програм, розподілених і неоднорідних середовищ, взаємодії систем, таких методів програмування, як аспектне, агентне, сервісне й інші, а також аспектів захисту, безпеки тощо.

Контрольні запитання та завдання

1. Обґрунтуйте необхідність акредитації освітніх програм у сфері інженерії програмного забезпечення.

2. Охарактеризуйте процес навчання програмної інженерії.

3. Опишіть історію виникнення та основні розділи «ядра знань» в області програмної інженерії (The Guide to the Software Engineering Body of Knowledge, SWEBOK).

4. Ознайомтеся з освітнім проектом у сфері програмної інженерії SWEnet (http://www.swenet.org/). Опишіть у вигляді короткого реферату історію та мету створення проекту. Ознайомтеся та наведіть у рефераті анотацію 2 – 3 навчальних модулів, розміщених для вільного доступу на сайті проекту.

РОЗДІЛ 5. КОДЕКС ЕТИКИ ТА ПРОФЕСІЙНОЇ ПРАКТИКИ ПРОГРАМНИХ ІНЖЕНЕРІВ

Однією з ознак зрілої професії є наявність кодексу етики або стандартів професійної поведінки [4]. З юридичної точки зору до професіоналів висуваються вищі вимоги, ніж до непрофесіоналів, задіяних у тій самій галузі. Наприклад, якщо знайомий сантехнік порадить вам прийняти пігулку від болю в шлунку, а у вас виявиться апендицит, то нічого неетичного сантехнік не вчинить. Але якщо цю ж пораду ви отримаєте від лікаря, що не оглянув вас, то його поведінка буде неетичною.

Кодекс етики встановлює стандарти поведінки в кожній професії. Так, дипломовані бухгалтери у США повинні складати тригодинний іспит з кодексу етики у своїй сфері діяльності. Адвокати повинні складати іспит з етики тривалістю півдня. У професіях, що сформувалися, можна втратити професійний статус або ліцензію в разі серйозних порушень кодексу етики.

5.1. Основні поняття інженерної етики

Як вступ до обговорення основ інженерної етики розглянемо історичний приклад [17 – 21]. Уранці 28 січня 1986 року велика кількість телеглядачів США стежили за стартом багаторазового транспортного космічного корабля «Челленджер». До цього часу був виконаний не один політ як у США, так і в СРСР. Ніл Армстронг зробив свої перші кроки на Місяці, за якими у прямому ефірі стежили мільйони людей на Землі. За стартом «Челленджера» також стежили мільйони і нічого не передвіщало біди. Але ряд технічних «промахів» виявився фатальним для космічного корабля та його екіпажу. Ще перед стартом, після запуску двигунів, одна прокладка, втративши еластичність, стала пропускати паливо. Через кілька секунд після старту від вібрації прокладка зрушилася ще більше, що дозволило полум'ю досягти основного паливного баку. «Челленджер» вибухнув на очах мільйонів телеглядачів!

Загальний шок перетворився на відчай і обурення, коли через кілька днів люди дізналися, що в ніч напередодні старту 14

інженерів з фірми, яка займалася розробкою ракетоносія, виступили проти старту. Вони попереджали, що температурні режими, які будуть мати місце при запуску, зменшать еластичність прокладки, що може призвести до аварії. Вони вже працювали над подоланням цієї проблеми, але поки не знайшли її розв'язку. Проте менеджер фірми приховав цей факт, повідомивши керівництво NASA, що для хвилювання немає причин, чим допустив старт, понадіявшись на те, що все буде гаразд (адже і раніше були старти, які закінчувалися вдало).

Цей і багато інших прикладів катастроф [17 – 21] (наприклад, вибух на Чорнобильській АЕС) демонструють необхідність виконання інженерами не лише їх безпосередніх професійних обов'язків, а ще й морально-етичних. Цих катастроф могло б і не бути, якби хтось не переступив через дозволений поріг ризику. Це не суто технічні проблеми, як може здатися на перший погляд, а у тому числі й етичні.

Досить часто перед інженерами постають етичні проблеми. При цьому вони повинні вибирати між двома або більшою кількістю варіантів дій. Одне з рішень змушує прийняти необхідність здійснення проекту, при обмежених можливостях (матеріальних ресурсів, часу тощо). Інше рішення змушує зважати на вимоги техніки безпеки, природоохоронні та інші аналогічні вимоги. Якщо всі ці вимоги врахувати, то ціна проекту настільки зросте, що ваша фірма не зможе витримати конкуренцію з іншими фірмами, де ці вимоги не враховуються. А у вас як інженера, що ухвалив це непопулярне рішення, можуть виникнути неприємності з керівництвом вашої фірми.

Кожного разу інженерам доводиться вирішувати, де ця межа (поріг дозволеного) між ризиком і надійністю. Чи завжди дозволено все, що не заборонено? Чи виправданий ризик? Яка ціна цьому відчуттю надійності та чи виправдана вона? Отже, у процесі інженерної діяльності людині досить часто необхідно приймати не тільки інженерно-технічні, а ще й морально-етичні рішення.

У кожної людини власний моральний або етичний кодекс, який визначає життєву філософію цієї людини, виходячи з якої

вона й вибирає цю межу (поріг дозволеного). Окрім цих внутрішніх кодексів, є ряд етичних кодексів, якими керуються цілі групи людей, наприклад етичний кодекс лікарів – загальновідома «клятва Гіппократа», сформульована ще у V ст. н.е., але досить актуальна по всьому світі й на теперішній час.

Щодо особливостей інженерної діяльності, то основні принципи інженерної етики вироблені, виходячи з різних (часто суперечливих) загальних етичних теорій. Ці принципи зведені до інженерних кодексів, яких по всьому світу складено дуже багато й на різних рівнях (державні, районні, вузькоспеціалізовані тощо).

Інженерні кодекси містять правила, згідно з якими слід оцінювати та управляти інженерною діяльністю. На додаток до спеціалізованих (орієнтованих на практикуючих інженерів) норм поведінки, ці кодекси включають іноді й загальні етичні норми поведінки.

Хоча різні інженерні етичні кодекси виражають конкретні обов'язки інженерів по-різному, їх загальна спрямованість і кінцеві висновки досить схожі. Усі інженерні етичні кодекси та інші нормативні документи професійних об'єднань інженерів підкреслюють зобов'язання інженерів перед людьми та перед самим собою. Це такі зобов'язання [4, 10]:

1. *Зобов'язання перед суспільством*: інженер зобов'язаний бути компетентним, істинним професіоналом своєї справи, а також поводитися й діяти етично, уникати завдання фізичного, фінансово-економічного і морального збитку іншим членам суспільства і суспільству в цілому, у випадку якщо вони залежать від діяльності інженера.

2. *Зобов'язання перед працедавцем*: інженер зобов'язаний чинити чесно і лояльно по відношенню до свого працедавця і берегти його комерційну таємницю. Інженер зобов'язаний інформувати працедавця щодо всіх суперечностей, у результаті яких інженер може одержувати вигоду, завдаючи шкоду працедавцю.

3. *Зобов'язання перед клієнтом*: зобов'язання такі самі, що і перед працедавцем. Відмінністю є те, що, в основному, має місце короткостроковий характер відносин.

4. *Зобов'язання перед колегами*: зобов'язання діяти професійно, тобто не допускати, щоб особисте і те, що роботи не стосується, заважало робочим взаєминам. Поведінка з колегами повинна бути пристойною, доброзичливою. Неетичною вважається недоброзичлива критика роботи колеги, особливо якщо ви не в курсі подій.

5. *Зобов'язання перед підлеглими*: зобов'язання поважати права інших, особливо, якщо вони працівники, які повинні працювати в підпорядкуванні інженера відповідно до укладеного договору.

6. *Зобов'язання перед званням інженера*: зобов'язання підтримувати престиж і гідність інженерного звання і не принижувати репутацію звання інженера скандальною та нечесною поведінкою.

7. *Зобов'язання перед самим собою*: інженер повинен піклуватися про те, щоб його права та обов'язки були в рівновазі. Зобов'язання вимагати зарплату, яка відповідає виконаній роботі, нормальних умов праці та інших прав. Зобов'язання зберігати свою компетентність у світі техніки та технологій, який швидко розвивається.

Оскільки в рамках курсу «професійна практика програмної інженерії» періодично використовуються поняття, які відносяться до так званої «загальної етики», то наведемо ряд визначень відносно терміну «етика».

Етика (від грецького *ethos* – звичай, вдача, звичка) – філософська наука, об'єктом вивчення якої є мораль або моральність як частина суспільної свідомості та важлива основа побудови життєвого устрою.

Етику можна визначити як вчення про правильну та неправильну поведінку. Якщо висловлюватися більш точно, то етика визначає, аналізує, оцінює та розв'язує проблеми моралі, а також розвиває моральні критерії з метою управління поведінкою

людини. Простіше кажучи, мета етики – допомогти розрізнити добро і зло.

Метою ж інженерної етики є не повторення положень загальної етики (вони і так діють для інженера як людини), а визначення високоморальної стратегії поведінки в деяких спеціалізованих, властивих лише інженерній діяльної напрямках.

Якщо метою загальної етики є регулювання відносин між людьми, їх поведінки і реакцій за різних обставин, а також визначення оцінок, то метою інженерної етики є регулювання відношень між інженером і продуктом його творчості, між продуктом його творчості та суспільством, а також визначення відповідних цінностей та пріоритетів.

Таким чином, метою інженерної етики є встановлення початкових положень і правил при розгляді та оцінці ситуацій, в яких можуть опинитися інженери.

Інженер, у загальному випадку, – це творча особа, і продукт його творчості тісно пов'язаний з його особистісними характеристиками. Створений інженером об'єкт або технологія часто суттєво впливає на оточення, тобто вступає в сильну взаємодію з рештою суспільства. Отже, ухвалені інженером рішення досить часто спричиняють серйозні наслідки. Тому при ухваленні цих рішень слід виходити з певних правил і положень.

Інженерна етика, з одного боку, дає інженерам положення для ухвалення правильних рішень, а з іншого – охороняє їх від можливої критики щодо ухвалених ними рішень.

Під час інженерної практики досить часто виникають етичні проблеми й дилеми. Наприклад, інженер повинен вибрати між ризиком для людей, що працюють над якимось проектом, і зупинкою цього проекту з метою встановлення додаткових засобів безпеки. Це, у свою чергу, веде до затримок у термінах виконання проекту і його дорожчання для працедавця або клієнта, що багатьом може здаватися непотрібною перестраховкою.

Як визначити межу, починаючи з якої величина ризику й потенційний збиток працівникам перевищують реальні витрати

працедавця або клієнта на техніку безпеки у випадку, якщо інженер припинить роботи над проектом?

При розв'язанні таких щоденних проблем було би корисно використовувати деякий методичний матеріал.

Для розвитку методів вирішення етичних питань необхідно проаналізувати, як розв'язуються складні технічні проблеми. Інженер, зіткнувшись з технічним питанням, пропонує кілька альтернативних рішень, аналізує їх, використовуючи відомі закони науки і техніки, аксіоми і теореми з математики, які є загальновідомими, перевіреними та правильними.

Зіткнувшись з етичною проблемою, можна використовувати аналогічний набір етичних теорій і законів, які створені людством досить давно, перевірені протягом тривалого часу і досить успішно працюють. Ці етичні теорії є основою при створенні етичних кодексів, якими інженери повинні керуватися при розв'язанні етичних проблем.

Для вирішення технічного питання, як правило, існує більш ніж один варіант дій, і метою інженера є вибір якнайкращого і найоптимальнішого рішення відповідно до деякого критерію. Аналогічно відбувається і з етичними проблемами – можливих рішень може бути кілька, і слід вибрати краще в етичному сенсі.

При цьому вибір оптимальних рішень не завжди є простим процесом, оскільки при застосуванні різних етичних теорій може виникнути й низка суперечностей. Альтернативні рішення іноді вимагають діаметрально протилежної поведінки, з яких жодна неприйнятна. Це призводить до моральної дилеми, яка вимагає ігнорування одного етичного закону заради дотримання іншого.

Можна навести цілий ряд прикладів подібних ситуацій з нашого повсякденного життя.Зокрема, загальновизнана етична норма – повідомляти лише правдиву інформацію. Проте, якщо є небезпека того, що правдива, але досить «гірка» інформація може нашкодити здоров'ю хворої людини, ми можемо її «пом'якшити» або просто «забути передати», вважаючи, що так ми завдамо менше шкоди, ніж коли говоритимемо правду. Аналогічні ситуації виникають під час спілкування з дітьми, коли правдива інформація іноді може травмувати дитячу психіку.

При розв'язуванні етичних проблем в інженерній практиці майже завжди необхідно оцінювати рішення, які пропонуються конкуруючими теоріями, а потім вибрати те, яке найкраще підходить для вирішення конкретної етичної дилеми.

Як уже зазначалося, розв'язання етичних проблем багато в чому нагадує вирішення інженером технічних питань. При розв'язанні обох видів проблем, якщо вони складні, використовують вироблений та перевірений часом алгоритм, який містить шість таких кроків:

1. Усвідомлення проблеми.
2. Збір інформації та визначення проблеми.

Зазвичай після виконання цих двох кроків проблема легко розв'язується. Якщо ж рішення немає, то йдемо далі.

3. Генерування альтернативних рішень (мозкова атака).
4. Оцінка альтернатив.
5. Оптимізація альтернатив та ухвалення рішень.

Якщо жодне з рішень після зазначених кроків не влаштовує, то слід повернутися, до другого кроку, і спробувати визначити проблему точніше.

6. Реалізація кращого рішення.

5.2. Етичні кодекси інженерних співтовариств

Останні сторіччя відзначилися цілим рядом наукових, технічних і технологічних досягнень, які мають суперечливий характер, оскільки вони принесли користь і одночасно породили нові проблеми. Успіхи в галузі фізики, хімії та космічної техніки дозволили оволодіти мікро- і макрокосмосом, що значно просунуло науково-технічний прогрес. Разом з тим вони надали людству засоби для самознищення в контексті неадекватних ядерних та космічних випробувань.

Сучасні дослідники розвитку науки і техніки стурбовані негативними техногенними й антропогенними навантаженнями на природу [17 – 21]. Почала усвідомлюватися необхідність включення принципу відповідальності в структуру науково-технологічної діяльності на правах найвищої норми. Отже, йдеться про зв'язок професійної і соціальної відповідальності. У

своєму змістовому наповненні професійна відповідальність передбачає компетентне й добросовісне виконання покладених професійних обов'язків. Якщо говорити про соціальну відповідальність науково-технічних працівників та інженерів, то, зі всією очевидністю, проблема відповідальності переводиться в соціальну площину: інженер повинен нести відповідальність за наслідки впровадження і використання результатів власної технічної діяльності в соціальній практиці.

Інженерна етика концентрується на поведінці індивіда – інженера і на виробленні етичних норм, які регулюють його професійну діяльність.

До норм інженерної етики ми можемо віднести такі, як необхідність сумлінно виконувати свою роботу; створювати пристрої, які приносили б людям користь і не заподіювали шкоди (особливий випадок у цьому відношенні – військова техніка); відповідальність за результати своєї професійної діяльності; певні форми відносин (звичаї і правила, які регулюють відносини) інженера з іншими учасниками процесу створення і використання техніки. Ряд таких норм фіксується в юридичних документах, наприклад, у законах, що стосуються питань безпеки, інтелектуальної власності, авторського права. Деякі норми професійної діяльності інженерів закріплені в адміністративних постановах, які регулюють діяльність тієї або іншої організації (підприємства, фірми, інституту тощо).

Етичні норми можуть існувати (і, звичайно, існують) у вигляді «неписаних правил», але можуть одержувати формулювання в етичних кодексах.

Загальним прикладом регулятора інженерної діяльності є розпорядження німецького мислителя Г. Йонаса, яке свідчить: «Поводься так, щоб наслідки твоєї діяльності не були руйнівними для майбутньої можливості життя на Землі». Цей принцип орієнтує на перспективне бачення життя, перш за все – на життя подальших поколінь на Землі.

Окремими прикладами регуляторів, що носять конкретніший характер, є кодекси інженерних співтовариств.

В усвідомленні етичних норм професійної діяльності відіграє свою роль об'єднання інженерів у професійні співтовариства (асоціації) з метою створення умов як для кращого задоволення своїх інтелектуальних потреб, так і для захисту матеріальних інтересів. Такого роду асоціації в країнах Заходу здійснюють функції підтримки інженерів, зокрема, в тих випадках, коли професійний обов'язок інженера відносно суспільства (наприклад, безпека того або іншого проекту для оточуючих) вступає в суперечність з безпосередніми інтересами фірми, де він працює.

На даний час етичні кодекси мають багато професійних об'єднань. Ці кодекси є зведеннями правил, дотримання яких є умовою членства в даному професійному суспільстві. Потрібно мати на увазі, що нерідко правила, що усвідомлюються спочатку як тільки етичні, згодом закріплюються в юридичних нормах.

Прийняті в багатьох державах світу такі кодекси (електротехніків, машинобудівельників, системотехніків, інженерів-атомників тощо) структурно включають три типи норм: норми-обов'язки, посадові норми і норми, що обмежують деякі види діяльності. На сьогодні в практиці інженерної діяльності фахівці стикаються з найбільш апробованими нормами-обов'язками. Ці норми передбачають можливість запобігання катастрофічним наслідкам (смерть, спричинення болю, обмеження тих або інших дій суб'єкта, завдання відчутного збитку природі, суспільству, окремим його членам тощо).

Відповідальність інженерів-техніків або вчених, які впроваджують свої відкриття в різноманітних практичних проектах, стає особливо актуальною в тих випадках, коли шкідливі наслідки можуть бути наперед виявлені та передбачені, наприклад, при реалізації проектів, безпосередньо орієнтованих на практичне застосування. У певних випадках можна говорити і про особисту відповідальність, але спільно з іншими учасниками (групами, підрозділами і т.п.). Розширена відповідальність спрямована, зокрема, й на майбутнє існування людства, прийдешніх поколінь людей і враховує їх моральне право на гідне людини життя в сприятливому оточуючому середовищі.

Дієвість етичного кодексу залежить значною мірою від того, як він застосовуватиметься в конкретних ситуаціях. Одна річ – сформулювати правила, інша – визначити, як відповідно до цих правил слід діяти в тому або іншому випадку. Питання про те, чи відповідає поведінка суб'єкта даному етичному кодексу, далеко не завжди може бути вирішене однозначно.

Ефективність етичних кодексів істотно залежить від зацікавленості інженера бути членом даної професійної асоціації (оскільки порушення кодексу може спричинити виключення з асоціації), а також від здатності співтовариства встановлювати в тому або іншому конкретному випадку сам факт порушення моральних кодексів. Зміни в інженерно-технологічній діяльності передбачають і відповідне корегування етичних норм.

Разом з тим ефективність етичних кодексів залишає бажати кращого навіть у країнах із сильними професійними інженерними суспільствами і асоціаціями.

У різного роду моральних кодексах інженерів (американських, німецьких, російських) разом із загальноморальними вимогами звичайно додаються характеристики, яким повинні відповідати представники інженерної професії. На їх основі професійно-моральна зовнішність інженера виглядає так: він – раціоналіст, володіє набором технічних навичок і умінь, має схильність до винахідницької діяльності, наполегливий, скрупульозний, працелюбний, пильний, відданий своїй справі, щирий, спирається у своїй діяльності на експеримент. Інженер не байдужий до долі людей, оскільки він сприяє досягненню ними високого рівня добробуту. Представники інженерних професій є по-справжньому моральними людьми.

Зазвичай етичні кодекси інженерних суспільств містять норми, що регулюють відносини «інженер – суспільство», «інженер – працедавець», «інженер – клієнт», «інженер – інші інженери».

Наприклад, «Кодекс етики» Національного товариства професійних інженерів США свідчить, що інженер повинен завжди усвідомлювати, що його щонайпершим обов'язком є захист безпеки, здоров'я і добробуту людей.

Етичні норми, що регулюють відносини «інженер – працедавець» та «інженер – клієнт», вимагають добросовісного виконання ділових зобов'язань: надавати клієнту або працедавцю те, що інженер обіцяв реалізувати; завершувати роботу у встановлений час і в рамках бюджету, а у випадку, якщо цього досягти неможливо, якомога раніше попередити клієнта або працедавця з тим, щоб могли бути зроблені корегуючі дії; не передавати іншим сторонам і не обнародувати інформацію, що стосується стану справ або технічних процесів свого колишнього або нинішнього клієнта чи працедавця, без їх згоди. Як записано в Кодексі національного товариства професійних інженерів, «інженер, використовуючи проекти, надані клієнтом, визнає, що ці проекти залишаються власністю клієнта і не можуть бути скопійовані іншими без дозволу». У тому ж кодексі серед положень, що стосуються відносин інженера з колегами, містяться такі: «Інженери не повинні шкодити, зловмисно або помилково, прямо або опосередковано, професійній репутації, планам, діяльності та службовому положенню інших інженерів, а також піддавати несправедливій критиці роботу інших інженерів». «Інженери, які вважають, що діяльність інших неетична або незаконна, повинні надати інформацію про це в розпорядження відповідних органів, щоб могли бути вжиті належні заходи». «Інженери перед початком роботи з людьми, з якими вони можуть працювати над удосконаленням планів, проектів або інших досягнень, що можуть бути засвідчені правом копіювання або запатентовані, повинні укласти угоду щодо власності». Залежно від специфіки інженерних співтовариств, етичні кодекси можуть містити різні вимоги до інженерів. Проте загальноморальні принципи залишаються незмінними.

Базові принципи професійної етики учених та інженерів всього світу виходять з того, що вільна, творча праця на благо людини, прагнення до новаторства – справа честі та професійної гідності членів громадських об'єднань інженерів, головний мотив наукової та інженерної діяльності.

Поважаючи досягнення колишніх поколінь, професійний інженер, учений націлений на їх вдосконалення і пошук

принципово нових рішень, відкриває або створює нове, сприяє його утвердженню та розповсюдженню. Відкриття, винаходи, раціоналізація, створення принципово нової техніки і технології, впровадження інновацій у життя суспільства для блага людини – основа його творчої діяльності.

Основними етичними принципами інженера як творчої особистості повинні стати:

- постійний пошук достовірних фактів, навіть якщо він пов'язаний з якими-небудь труднощами, для встановлення і захисту істини як основної мети пізнання;
- пошана до творчої праці своїх колег;
- критична оцінка власних результатів і досягнень, протидія будь-яким спробам привласнення результатів праці інших дослідників, фахівців;
- відсутність користолюбства та інтелектуальна чесність;
- здатність розглядати проблему або ситуацію в перспективі та з урахуванням всіх її соціальних, екологічних та інших наслідків для суспільства;
- уміння виділити цивільні та етичні аспекти проблем, пов'язаних з пошуком нових знань, інженерних рішень, які на перший погляд здаються виключно технічними;
- готовність до творчого спілкування з представниками суміжних професій;
- прагнення звести до мінімуму пов'язані із застосуванням техніки негативні дії на людину, суспільство і навколишнє середовище;
- заперечення консерватизму і застою у творчій діяльності;
- підвищення престижу вченого та інженера.

Людина може зробити більше, ніж вона має на те моральне право. У зв'язку з цим і виникає потреба в особливій етиці, орієнтованій на зміст технічної діяльності людини.

Жоден з аспектів техніки не є морально нейтральним. Неприпустимо робити людину придатком машини. Кожна технічна новація повинна пройти перевірку на предмет того, чи дійсно вона сприяє розвитку людини як творчої відповідальної

особистості. Інженери зобов'язані брати на себе відповідальність і ризик за здійснення розумно керованого технічного прогресу.

5.3. Аналіз етичного кодексу, розробленого асоціаціями ACM та IEEE-CS

Розробка програмного забезпечення продовжується вже багато років, обходячись без загальновизнаного кодексу етики. У кінці 90-х років XX століття спільний комітет організацій ACM і комп'ютерного товариства IEEE почав розробку кодексу етики в інженерії програмного забезпечення. Кодекс зазнав кілька редакцій і розглядався практикуючими розробниками програмного забезпечення по всьому світі. У 1998 р. Кодекс етики і професійної практики був прийнятий як асоціацією ACM, так і комп'ютерного товариства IEEE [2]. Повний текст кодексу наведено нижче. Оригінальний англомовний його варіант можна знайти на сайті комп'ютерного товариства IEEE за адресою: www.computer.org.

У преамбулі кодексу сформульовані його цілі, а далі у восьми пунктах викладені принципи.

Перша основоположна мета – «інженери програмного забезпечення повинні служити тому, щоб аналіз, специфікація, проектування, розробка, тестування і обслуговування програмного забезпечення стали вигідною і поважаною професією». Іншими словами, одне із завдань кодексу – сприяти розвитку самої професії інженера програмного забезпечення. Формулювання цієї мети неявно припускає, що інженерія програмного забезпечення ще не стала «вигідною і поважаною професією». В міру її становлення як зрілої професії формулювання може змінитися.

Друга основоположна мета полягає в тому, що інженери програмного забезпечення повинні бути «вірними цілям здоров'я, безпеки і блага суспільства». Це цілком відповідає уявленню про те, що інженери несуть відповідальність швидше перед суспільством у цілому, ніж перед окремими його представниками. Кодекси поведінки інших інженерів аналогічно підкреслюють

важливість захисту добробуту суспільства. Дві ці мети основні, а вісім принципів спрямовані на їх реалізацію.

Відіграючи важливу роль у розробці програмних систем, програмні інженери мають значні можливості творити добро або заподіювати зло, дозволяти іншим чи впливати на тих, хто творить добро або заподіює зло. Для забезпечення гарантії використання зусиль у благих цілях, наскільки це можливо, програмні інженери повинні неухильно перетворювати програмну інженерію на корисну і поважану професію. Для цього програмісти повинні дотримуватися кодексу професійної етики.

Вісім принципів, що містить кодекс, впливають на лінію поведінки і вибір рішення програмними інженерами, включаючи практиків, викладачів, менеджерів і вище керівництво, а також учнів і студентів. Принципи визначають етику відносин між окремими інженерами, групами і організаціями, а також пов'язані з цим зобов'язання. У кожен принцип включені ілюстрації деяких зобов'язань, що зумовлюються цими відносинами. Ці зобов'язання ґрунтуються на гуманності професії програмного інженера, особливій увазі до людей, на яких впливає діяльність програмних інженерів, і унікальності цієї діяльності. Кодекс проголошує ці зобов'язання для всіх, хто відносить себе до програмних інженерів або збирається ним стати.

Окремі частини кодексу не можуть бути використані ізольовано від інших для виправдання упущень і провини. Перелік принципів і положень не є вичерпним. Положення не можуть розглядатися як такі, що розділяють професійну поведінку на прийнятну і неприйнятну в усіх реальних ситуаціях. Кодекс не є простим етичним алгоритмом, що генерує етичні рішення. У деяких ситуаціях стандарти можуть суперечити один одному або іншим стандартам. Такі ситуації вимагають від програмного інженера дій відповідно до духу кодексу професійної етики залежно від конкретних обставин.

Слід вдумливо використовувати основні положення етики, а не покладатися всліпу на її докладні вказівки. Ці принципи спонукають програмних інженерів усвідомити, на кого чинить вплив виконувана ними робота; розібратися, чи ставляться вони

та їх колеги до оточуючих з належною пошаною; взяти до уваги те, як суспільство, будучи належно інформоване, поставилося б до їх рішень; нарешті, оцінити, чи відповідають їх професійні дії ідеалам програмної інженерії. У всіх цих оцінках турбота про благополуччя, безпеку і процвітання суспільства первинна; тобто «інтереси суспільства» є центральними в даному кодексі.

Динамічний і вимогливий контекст програмної інженерії вимагає кодексу, який можна пристосувати до нових ситуацій у міру їх появи. Проте навіть у такому загальному вигляді кодекс забезпечує підтримку програмним інженерам та їх керівникам, які потребують правильного вибору дій у специфічних умовах, шляхом документування професійних етичних настанов. Кодекс забезпечує етичну базу, до якої можуть звертатися як окремі члени команд, так і команди в цілому. Кодекс дозволяє визначити дії, які з етичних міркувань недоцільно вимагати від програмних інженерів або їх команд.

Даний кодекс призначений не тільки для оцінки суперечливих дій; він має також важливе освітнє значення. Оскільки в ньому висловлено загальну думку про етичну сторону професії, він є засобом, що дозволяє довести до відома як суспільства, так і професіоналів етичні зобов'язання всіх програмних інженерів.

Розглянемо принципи, відображені в кодексі етики і професійної практики інженерії програмного забезпечення [2].

Принцип 1: СУСПІЛЬСТВО

Програмні інженери мають діяти неухильно на користь суспільства. Зокрема, програмні інженери повинні:

1. Нести повну відповідальність за свою роботу.
2. Обмежувати інтереси програмних інженерів, працедавців, клієнтів і користувачів користю для суспільства в цілому.
3. Схвалювати програмне забезпечення лише у випадку, якщо вони твердо переконані в тому, що воно безпечно, відповідає специфікаціям, пройшло відповідне тестування і не загрожує якості життя, не порушує приватність і не шкодить навколишньому середовищу. Результат роботи повинен безумовно служити на благо суспільству.

4. Доводити до відома уповноважених осіб і організацій дійсну або потенційну небезпеку для користувачів, суспільства або навколишнього середовища, яке, на їх думку, пов'язане з використанням програмного забезпечення або супутньої йому документації.
5. Брати участь у розв'язанні проблем, що викликають занепокоєння в суспільстві, стосуються програмного забезпечення, його інсталяції, розвитку, підтримки або документування.
6. Бути чесними і не допускати брехні у всіх висловлюваннях, особливо публічних, відносно програмного забезпечення або пов'язаних з ним документації, методик та інструментів.
7. Звертати увагу на проблеми, пов'язані з фізичними недоліками, розподілом ресурсів, економічною відсталістю та іншими чинниками, здатними обмежити доступ до користування програмним забезпеченням.
8. Бути готовим добровільно використовувати свою професійну майстерність для загального блага і сприяти розповсюдженню знань про свою професію.

Принцип 2: КЛІЄНТ І ПРАЦЕДАВЕЦЬ

Програмні інженери повинні діяти згідно з інтересами клієнта і працедавця, якщо вони не суперечать інтересам суспільства. Зокрема, програмні інженери повинні:

1. Надавати послуги в межах своєї компетентності, бути чесними і не приховувати обмеженості своєї освіти і досвіду.
2. Не використовувати програмне забезпечення, одержане або свідомо нелегальним, або неетичним шляхом.
3. Користуватися власністю клієнта або працедавця тільки належним чином і з їх відома.
4. Переконатися, що всі використовувані ними документи, які повинні бути затверджені, дійсно затверджені уповноваженою особою.
5. Зберігати в таємниці будь-яку конфіденційну інформацію, одержану при виконанні професійних обов'язків, якщо це не суперечить інтересам суспільства і законодавству.

6. Ідентифікувати, документувати, збирати факти і негайно оповіщати клієнта або працедавця, якщо, на їх думку, проект близький до провалу, виявляється занадто дорогим, порушує закон про інтелектуальну власність або може спричинити інші проблеми.
7. Ідентифікувати, документувати і докладати працедавцю або клієнту про соціальні проблеми, пов'язані з програмною і супутньою документацією, про які їм стало відомо.
8. Не приймати пропозицій побічної роботи, яка може завдати збитку роботі, що виконується для основного працедавця.
9. Не діяти проти інтересів працедавця або клієнта, за винятком випадків, коли це суперечить вищим етичним міркуванням; у цьому випадку слід інформувати працедавця або іншу уповноважену особу щодо цих міркувань.

Принцип 3: ПРОДУКТ

Програмні інженери повинні забезпечувати відповідність якості своїх продуктів та їх модифікацій найвищим можливим професійним стандартам. Зокрема, програмні інженери повинні:

1. Прагнути до високої якості, прийнятної вартості і розумних термінів виконання проектів, доводячи істотні альтернативи до відома працедавця і клієнта, заручившись їх згодою із вибором, а також інформуючи про них користувачів і суспільство.
2. Забезпечувати адекватність і досяжність цілей та спрямованості для всіх проектів, над якими вони працюють або мають намір працювати.
3. Виявляти, визначати і вживати заходи відносно проблем, що пов'язані з проектом, над яким вони працюють, і мають відношення до етики, економіки, культури, законності й навколишнього середовища.
4. Гарантувати, що їх освіта, підготовка і досвід достатні для всіх проектів, над якими вони працюють або мають намір працювати.
5. Гарантувати, що в усіх проектах, над якими вони працюють або мають намір працювати, використовуються належні методики.

6. Працювати, дотримуючись найбільш відповідних професійних стандартів і відступаючи від них лише в тих випадках, коли це виправдано з етичних або технічних причин.
7. Прагнути до повного розуміння специфікацій програмного забезпечення, над яким вони працюють.
8. Гарантувати, що специфікації на програмне забезпечення, над яким вони працюють, добре документовані, відповідають вимогам користувачів і належно затверджені.
9. Гарантувати реалістичність кількісних оцінок вартості, термінів виконання, трудовитрат, якості і витрат за всіма проектами, над якими вони працюють або мають намір працювати, а також уникати невизначеності цих оцінок.
10. Гарантувати адекватність тестування, налагодження і ревізій програмного забезпечення і супутньої документації, над якими вони працюють.
11. Гарантувати адекватність документації, включаючи виявлені проблеми та їх схвалені рішення, для всіх проектів, над якими вони працюють.
12. Розробляти програмне забезпечення і супутню документацію, ставлячись із повагою до приватності тих, чиї інтереси зачіпає дане програмне забезпечення.
13. Використовувати тільки надійні дані, одержані прийнятними з погляду моралі та закону засобами, і використовувати їх тільки належним чином.
14. Підтримувати цілісність даних, схильних до старіння і втрати актуальності.
15. Ставитися до всіх видів підтримки програмного забезпечення з тим же професіоналізмом, що і до нових розробок.

Принцип 4: ОЦІНКИ

Програмні інженери мають підтримувати цілісність і незалежність своїх професійних оцінок. Зокрема, програмні інженери повинні:

1. Спрямовувати всі технічні думки на службу людським цінностям.

2. Рекомендувати лише ті документи, які або розроблені під їх контролем, або належать до їх компетентності та зі змістом яких вони згодні.
3. Дотримуватись професійної об'єктивності по відношенню до програмного забезпечення або супутньої документації, які їх попросили оцінити.
4. Не брати участі у фінансових махінаціях, таких як підкуп, подвійна оплата та інші незаконні фінансові дії.
5. Розкривати всім зацікавленим сторонам конфлікти інтересів, яких неможливо уникнути розумними засобами.
6. Відмовлятися від участі як член команди або радником у приватних, урядових або професійних заходах, пов'язаних з програмним забезпеченням, через які може бути завданий потенційний збиток їх власним інтересам, інтересам їх працедавців або клієнтів.

Принцип 5: МЕНЕДЖМЕНТ

Програмні інженери-менеджери та провідні співробітники повинні дотримуватися етичних підходів до управління розробкою і підтримкою програмного забезпечення і просувати ці підходи. Зокрема, керівники і провідні фахівці з програмної інженерії повинні:

1. Гарантувати якісне управління всіма проектами, над якими вони працюють, включаючи ефективні процедури підвищення якості та зменшення ризику.
2. Гарантувати, що їхні підлеглі ознайомлені зі стандартами, яких мають намір дотримуватися.
3. Гарантувати, що програмні інженери знають політику та процедури працедавця відносно захисту паролів, файлів і конфіденційної інформації, що стосується працедавця або інших осіб.
4. Розподіляти роботу тільки після з'ясування освіти і досвіду співробітника, враховуючи його бажання вдосконалювати свої освіту та досвід.
5. Гарантувати реалістичність кількісних оцінок вартості, термінів виконання, трудовитрат, якості та прибутку за всіма

проектами, над якими вони працюють або мають намір працювати, а також невизначеністю цих оцінок.
6. Долучати до роботи програмних інженерів тільки після того, як їм надано повний і точний опис умов роботи.
7. Пропонувати справедливу винагороду за працю.
8. Не перешкоджати безпричинно призначенню співробітника на посаду, для якої він має відповідну кваліфікацію.
9. Гарантувати справедливу угоду щодо прав власності на будь-яке програмне забезпечення, технологію, дослідження, рукописи та іншу інтелектуальну власність, в яку програмний інженер зробив свій внесок.
10. Належно повідомляти про відповідальність за порушення політики працедавця або даного кодексу.
11. Не вимагати від програмного інженера нічого, що суперечить даному кодексу.
12. Не карати нікого, хто виражає заклопотаність щодо порушення етичних норм, пов'язаних з проектом.

Принцип 6: ПРОФЕСІЯ

Програмні інженери повинні піднімати престиж і репутацію своєї професії на користь суспільства. Зокрема, програмні інженери повинні:

1. Сприяти створенню в організації атмосфери, що сприяє етичній поведінці.
2. Поширювати знання з програмної інженерії.
3. Розширювати знання у сфері програмної інженерії шляхом участі в професійних організаціях і зібраннях, а також своїми публікаціями.
4. Підтримувати інших колег, які прагнуть дотримуватися даного кодексу.
5. Не ставити власні інтереси вище за професійні інтереси, інтереси клієнта або працедавця.
6. Підкорятися всім законам, що регулюють їх роботу, за винятком особливих ситуацій, коли це суперечить інтересам суспільства.
7. Бути точними в оцінках програмного забезпечення, над яким вони працюють, уникаючи не тільки свідомо брехливих

обіцянок, але й обіцянок, які справедливо можуть бути сприйняті як спекулятивні, необґрунтовані, такі, що вводять в оману, що збивають з пантелику або сумнівні.

8. Брати на себе відповідальність за виявлення, виправлення і повідомлення про помилки в програмному забезпеченні та пов'язаній з ним документації, над якими вони працюють.
9. Повідомити клієнтів, працедавців і керівництво про те, що програмні інженери дотримуються даного кодексу етики, і про наслідки цього.
10. Уникати організацій, які знаходяться в конфлікті з даним кодексом.
11. Усвідомлювати, що порушення даного кодексу несумісні з належністю до професійних програмних інженерів.
12. Виражати свою заклопотаність у разі істотного порушення даного кодексу людям, причетним до цього, за винятком випадків, коли це неможливо, приводить до серйозних конфліктів або небезпечно.
13. Повідомляти про випадки істотного порушення даного кодексу у відповідні інстанції, якщо очевидно, що діалог з причетними до цього людьми неможливий, приводить до серйозних конфліктів або небезпечний.

Принцип 7: КОЛЕГИ

Програмні інженери повинні бути справедливі по відношенню до своїх колег, допомагати їм і підтримувати. Зокрема, програмні інженери повинні:

1. Закликати колег дотримуватися даного кодексу.
2. Допомагати колегам у професійному зростанні.
3. Поважати роботу інших, але утримуватися від необґрунтованої довіри до неї.
4. Оглядати роботу інших об'єктивно, неупереджено, належно документуючи.
5. Прислухатися до думки, заклопотаності або скарг колег.
6. Допомагати колегам в освоєнні поточних робочих стандартів, включаючи політики і процедури захисту паролів, файлів та іншої конфіденційної інформації, а також заходів безпеки в цілому.

7. Не втручатися без необхідності в робочі справи колег; проте, щира турбота про інтереси працедавця, клієнта або суспільства може змусити програмного інженера поставити під сумнів компетентність колеги.
8. У ситуаціях, що виходять за межі їх власної компетентності, дізнаватися думку інших професіоналів, компетентних у даній сфері.

Принцип 8: ОСОБИСТА ВІДПОВІДАЛЬНІСТЬ

Програмні інженери повинні постійно вчитися навичкам обраної професії і сприяти просуванню етичного підходу до своєї діяльності. Зокрема, програмні інженери повинні безперервно прагнути до такого:

1. Заглиблювати знання з аналізу, специфікації, проектування, розробки, підтримки і тестування програмного забезпечення і супутньої документації, а також управління процесом розробки.
2. Удосконалювати свої здібності для створення безпечного, надійного і функціонально якісного програмного забезпечення за розумною ціною і в розумні терміни.
3. Удосконалювати свою здатність до виробництва точної, інформативної, якісно написаної документації.
4. Удосконалювати знання програмного забезпечення і супутньої документації, над якою вони працюють, а також середовища, в якому вони використовуватимуться.
5. Удосконалювати знання відповідних стандартів і законів, що регулюють програмне забезпечення і супутню документацію, над якими вони працюють.
6. Удосконалювати знання даного кодексу, його інтерпретацію і використання у своїй роботі.
7. Не допускати несправедливого поводження з ким-небудь унаслідок упереджень, що не стосуються справи.
8. Не підбурювати інших до дій, що порушують даний кодекс.
9. Усвідомлювати, що особисте порушення даного кодексу несумісне з належністю до професійних програмних інженерів.

5.4. Особливості професійних та етичних вимог до програмних інженерів. Етичні дилеми професійної практики програмних інженерів

Подібно до будь-яких інших професіоналів, спеціалісти з програмного забезпечення повинні погодитися, що до них ставиться більш широке коло вимог, аніж проста необхідність мати той чи інший професійний рівень. Вони працюють у певному правовому та соціальному оточенні. Галузь інженерії програмного забезпечення, як і будь-яка інша сфера людської діяльності, має обмеження у вигляді місцевих, національних та міжнародних законів, норм та традицій. Тому спеціалісти з програмного забезпечення повинні взяти на себе певні юридичні, етичні та моральні обов'язки, щоб стати справжніми професіоналами [2, 4, 10].

Не потребує зайвих пояснень твердження, що спеціалісти мають бути чесними та порядними людьми. Вони не повинні використовувати свої професійні навички та можливості для діяльності, що дискредитує професію спеціаліста з програмного забезпечення. Разом з тим вимоги до спеціалістів не обмежуються лише моральними або юридичними нормами, до їх кола також входять значно більш тонкі професійні обов'язки, які відображені в кодексі етики та професійної практики програмних інженерів.

Етичний кодекс забезпечує широку підтримку професійної інженерії програмного забезпечення. Він дає працедавцям і замовникам упевненість у професійних стандартах і особистостях інженерів, які його дотримуються.

Кодекс забезпечує організаціям можливість виразити свою підтримку інженерії програмного забезпечення. Якщо компанія готова дотримуватися кодексу, то зобов'язана забезпечити робоче середовище, в якому етика поведінки є пріоритетом та інженери програмного забезпечення можуть дотримуватися їй, не піддаючи ризику свою кар'єру. Це вигідно як самій компанії, так і інженерам програмного забезпечення, які в ній працюють: компанія залучає інженерів з високими професійними стандартами, а вони отримують можливість самореалізації в обстановці, де подібні стандарти належно оцінюються.

Однією з найсуттєвіших переваг кодексу є загальний напрям, який він задає в сенсі етичної і професійної поведінки інженерів програмного забезпечення, що дотепер відсутнє абсолютно. Розглянемо кілька ситуацій [4]:

• *Проекти, що ведуть до безвихідних ситуацій*. Не маючи етичного кодексу, інженери програмного забезпечення, які вважають, що терміни реалізації проекту нереальні, сумніваються, перш ніж повідомляти про це замовнику або своєму керівнику. Спираючись на кодекс, в подібній ситуації інженери зобов'язані зібрати докази й документально підтвердити свої побоювання. Кодекс свідчить, що професійним обов'язком є негайне повідомлення керівництву або замовнику про свої побоювання.

• *Заниження вартості розробки з метою отримання контракту будь-якою ціною*. У галузі програмного забезпечення поширена практика комерційних пропозицій замовникам з нереалістично низькими розцінками. Розробникам програмного забезпечення, можливо, не дуже подобається така поведінка, але багато хто не готовий піти наперекір своїм босам і відмовитися занизити вартість розробки в заявці. Згідно з кодексом, інженери програмного забезпечення повинні забезпечити реалістичність кошторисів витрат і схвалювати документи, тільки якщо вони з ними згодні. Кодекс також закликає інженерів програмного забезпечення зробити свою професію гідною пошани, наприклад не брати участь в операціях із заниженою вартістю. Інженер програмного забезпечення, який дотримується етичних принципів, повинен відмовитися затверджувати такі комерційні пропозиції.

• *Розробка програмного забезпечення за принципом «напишемо і виправимо»*. Недостатньо інформовані замовники і керівники часто наполягають на застосуванні розробниками підходу «напишемо і виправимо». Останні усвідомлюють неефективність цього підходу, але після суперечок із замовниками і керівництвом багато хто капітулює: «Нехай ця контора отримує те, що заслужила». Проте дотримання даного принципу суперечить етичному обов'язку інженера програмного

забезпечення створювати високоякісні продукти за прийнятними цінами і в розумні терміни. Подальше використання зазначеного методу також компрометує інженерію програмного забезпечення як професію, тому, дотримуючись етичних принципів, розробники програмного забезпечення повинні відмовитися від нього.

• *Застій знань.* Для того, щоб залишатися в курсі всього нового в розробці програмного забезпечення, потрібно чимало часу, і багато розробників навіть не намагаються робити це. Один з видавців стверджував, що середній розробник програмного забезпечення читає в рік менше однієї книги зі сфери своєї професії і не виписує жодного професійного журналу. Можливо, проблема не пов'язана з етикою, але вже напевно пов'язана з *професійною поведінкою*. Неможливо працювати на рівні професіонала, не цікавлячись останніми досягненнями у своїй галузі. Не займаючись безперервною самоосвітою, можна продовжувати працювати в галузі програмного забезпечення на якомусь рівні, проте, згідно з кодексом етики і професійної поведінки, не можна при цьому бути фахівцем програмного забезпечення.

Поза дією кодексу інженерам програмного забезпечення доводиться покладатися на власні думки й оцінки, стикаючись з етичними дилемами. Інженери, що дотримуються кодексу, знатимуть, що їм не доведеться поодинці відстоювати свої позиції.

У деяких випадках ситуація не така однозначна, як у вищенаведених прикладах [10]. Інтереси клієнта-замовника суперечитимуть інтересам суспільного блага. Або інтереси працедавця можуть суперечити інтересам колег по цеху розробників програмного забезпечення. Кодекс не може передбачити всі етичні дилеми, але закликає інженерів програмного забезпечення ухвалювати рішення, виходячи з вищих етичних критеріїв, дотримуючись при цьому духу кодексу.

Звичайно, в одній і тій самій ситуації різні люди мають різні погляди та принципи вирішення етичних питань, які перед ними постають. Наприклад, як діяти, якщо ви не згодні з політикою,

яку провадить керівництво компанії? Очевидно, це залежить від конкретної людини та суті розбіжностей. Що при цьому краще – дистанціюватись від позиції компанії чи переглянути свої принципи? Якщо ви вважаєте, що ці розбіжності можуть породити проблеми у виконанні програмного проекту, чи будете ви відкрито відстоювати свою позицію перед керівництвом? Якщо ви розраховуєте працювати в цій компанії надалі, необхідно рано чи пізно розв'язати цю моральну проблему.

Такі етичні дилеми постають перед всіма спеціалістами в процесі їх професійної діяльності [10]. На щастя, в більшості випадках вони не мають істотного принципового підґрунтя та вирішуються порівняно просто. Якщо ж не вирішуються, то це означає, що спеціаліст зіштовхнувся, скоріше за все, із великою етичною проблемою. В принципі її завжди можна розв'язати, просто звільнившись з роботи, але в цьому випадку можливе виникнення інших проблем, наприклад із матеріальним забезпеченням родини.

Особливо складна ситуація для спеціаліста виникає тоді, коли працедавець поводить себе неетично. Скажімо, компанія розробляє програмну систему, критичну по відношенню до безпеки. Але внаслідок дефіциту часу були підроблені протоколи перевірки системи на захищеність. Чи повинен спеціаліст у цій ситуації підтримувати конфіденційність, тобто нерозголошення інформації відносно працедавця? Чи все ж таки слід попередити замовника програмного забезпечення або надати розголосу тому факту, що система може бути незахищеною?

Складність цієї проблеми полягає в тому, що не існує критеріїв абсолютної захищеності систем. Фактична захищеність системи може бути перевірена лише в процесі її тривалої експлуатації. Навіть якщо система задовольняє заздалегідь визначені критерії захищеності, це ще не означає, що вона позбавлена помилок і не може виникнути якихось збоїв у роботі.

Рання постановка розглянутої етичної проблеми може привести до напруги між працедавцем та підлеглими, а несприятливий результат її розв'язання може завдати збитки іншим співробітникам. Ви повинні мати власний погляд на

вирішення питання, яке виникло. Але ця точка зору обов'язково повинна враховувати можливі неприємності та збитки, заподіювані іншим людям. Якщо ситуація дуже серйозна, можна, наприклад, звернутися за підтримкою до представників відповідних професійних інженерних асоціацій. Але в будь-якому випадку необхідно намагатися розв'язати проблему без завдання збитку правам працедавця.

Інша етична проблема виникає, якщо ви берете участь у розробці військових або атомних систем. Велика кількість людей відмовляються працювати над будь-якими розробками, що мають хоча б якесь відношення до військової тематики. Деякі згодні працювати над військовою тематикою, не пов'язаною із виробництвом зброї, а хтось вважає, що національна безпека – достатнє обґрунтування для того, щоб не мати етичних проблем при роботі над системами озброєнь. Тут відповідна етична позиція повністю залежить від поглядів та світогляду конкретної людини.

У цій ситуації важливо, щоб як працедавець, так і співробітники завчасно дізнавалися про погляди один одного. Коли організація долучається до військових або атомних проектів, керівництво повинно повідомити колективу, що він повинен бути готовим виконати будь-яку роботу відповідного профілю. З іншого боку, якщо штатні співробітники не хочуть брати участь у розробці військових систем, керівництво не повинно на них тиснути, примушуючи до виконання таких робіт.

До питань професійних етичних норм та відношень інтерес зростає протягом останніх років. Їх можна розглядати, виходячи з філософських категорій, які описують базові принципи етики, та на основі кодексу етики та професійної практики інженерії програмного забезпечення, яка також посилається на ці принципи.

Кодекс інформує суспільство, включаючи клієнтів і керівну ланку, про те, чого слід чекати від фахівців програмного забезпечення. Зрозуміло, його існування має сенс, тільки якщо працедавці й замовники зможуть розраховувати на дотримання кодексу професіоналами інженерії програмного забезпечення. У

кожній професії потрібно мати дисциплінарні засоби, щоб закликати до порядку працівників, які не дотримуються професійних стандартів. Не маючи подібних засобів, можна підірвати довіру до професії в результаті поступового її руйнування недобросовісними працівниками. Сьогодні ані комп'ютерне товариство IEEE, ані ACM, ані будь-яка інша організація не мають повного авторитету, щоб змусити своїх працівників дотримуватися кодексу. Дотримання його добровільне. У довгостроковому плані, проте, інженерія програмного забезпечення буде розвиватися так само, як інші професії: повноцінний професійний статус і дотримання кодексу стануть її невід'ємними частинами. Кодекс буде обов'язковим для виконання, і це буде вигідно і фахівцям програмного забезпечення, і їх працедавцям, і замовникам, і суспільству в цілому.

Семюель Флорман зауважив, що «в міру дорослішання у нас росте бажання зробити щось на благо суспільства» [4]. Він говорив про осіб, але його слова можна віднести і до нашої теми. Кодекс етики і професійної поведінки, який звертає особливу увагу на відповідальність по відношенню до професії та внесок у суспільне благо в цілому, є однією з головних ознак початку дорослішання інженерії програмного забезпечення.

Контрольні запитання та завдання

1. Які основні поняття інженерної етики?

2. Обґрунтуйте необхідність виникнення етичних кодексів інженерних співтовариств.

3. Ознайомтесь із кодексом етики та професійної діяльності в області програмної інженерії (Software Engineering Code of Ethics and Professional Practice). Придумайте два гіпотетичних сценарії, які відображають ті чи інші положення етичного кодексу програмного інженера. При цьому ви можете порадитися зі знайомим професійним програмним інженером (можливо, під час його професійної практики йому доводилося вирішувати певні морально-етичні дилеми) або відвідати відповідні Internet-ресурси, присвячені проблемам професійної етики в галузі інформаційних технологій.

Приклади

Гіпотетичний сценарій 1

Під час навчання в університеті ви вивчали мову програмування Java. Вас недавно було прийнято на роботу компанією, яка традиційно використовувала мову програмування Java. Проте компанія вирішила перейти на іншу об'єктно-орієнтовану мову програмування під час реалізації нового проекту, до якого ви були задіяні. Ви ніколи не використовували нову мову. Проект має дуже інтенсивний графік і щільний бюджет. Ви хочете справити гарне враження на нового керівника і ваших колег.

У світлі морального кодексу ACM/IEEE-CS виберіть одну з таких можливих дій:

a) Попросіть переведення до іншого проекту, в якому задіяно мову програмування Java.

b) Спеціально для даного проекту самостійно вивчіть нову для вас мову програмування, але не повідомляйте нікому про відсутність у вас досвіду роботи з нею.

c) Обговоріть дану проблему з вашим керівником і попросіть організувати для вас додаткове навчання у вигляді професійних тренінгів.

d) Відправте анонімне повідомлення клієнту із застереженням відсутності досвіду в деяких членів компанії.

e) Робіть найкраще, що ви можете, з надією на те, що ніхто не зверне увагу на незнання вами нової мови програмування.

f) Інший варіант відповіді_____

Напишіть одне або два речення для пояснення вашої відповіді.

Гіпотетичний сценарій 2

Ви – старший технічний співробітник компанії, яка уклала контракт по створенню програмного забезпечення для системи автоматизації насосних станцій у пустелі Західної Австралії. На початковому етапі ви були впевнені, що ваша фірма володіла відповідними експертними висновками та оцінками, необхідними для розробки проекту. На середньому етапі виконання проекту замовник вирішує внести зміни у вимоги до системи й залучити нову технологію для полегшення телезв'язку між насосними станціями. Ваша фірма має обмежений досвід роботи із системами телезв'язків, і цей факт не був оговорений, коли клієнт уводив зміни у вимоги. Ви побоюєтесь, що зміни матимуть істотний вплив на якість системи, що розробляється, і можуть потенційно призвести до витоку необроблених відходів у річку Swan River. Бюджет проекту щільний і підписаний контракт передбачає жорсткі фінансові санкції щодо невчасного введення в експлуатацію системи. Перша стадія проекту вже була завершена, але ще не була апробована на практиці. Замовник проекту та відповідні державні установи, зокрема екологи, не підозрюють про ваші проблеми. Ви маєте проявити відповідальність та врахувати інтереси всіх зацікавлених сторін, зокрема акціонерів вашої компанії, ваших колег і громадян Західної Австралії.

Використовуючи етичний кодекс ACM/IEEE-CS, опишіть потенційні дії, які були б прийнятними в межах положень кодексу і враховували інтереси всіх зацікавлених сторін цього проекту.

РОЗДІЛ 6. ПРОФЕСІЙНА ДІЯЛЬНІСТЬ ПРОГРАМНИХ ІНЖЕНЕРІВ

Усі професіонали повинні пройти інтенсивний курс спеціалізованого навчання – початкову професійну освіту. Але перш ніж приступити до виконання професійних обов'язків, необхідно не тільки володіти знаннями, набутими на етапі початкової освіти. Досить важливим фактором є надбання та розвиток початкових навичок професійної діяльності. Тобто для того, щоб стати професіоналом, людина повинна розвивати навички в додаток до початкової освіти (через університетські програми кооперації з компаніями, навчання за місцем роботи, стажування або інші заходи).

Будь-яка професія передбачає наявність методів оцінки і гарантії адекватності освіти та компетентності окремих професіоналів. Загальні форми цих компонентів – акредитація професійних учбових програм і видача свідоцтв (сертифікатів або ліцензій) окремих професіоналів. Сертифікація і/або ліцензування гарантує компетентність людини під час професійної практики. Крім того, можливі періоди професійного розвитку, які приводять до необхідності повторної видачі свідоцтва або повторного ліцензування.

Професіоналізм передбачає також активний професійний розвиток, який являє собою додаткове навчання, що має місце під час професійної практики.

До професіонала, що здобув базову освіту, деякий досвід і, можливо, ліцензію, в більшості професій ставиться вимога безперервного навчання.

Безперервна освіта дозволяє забезпечити актуальність знань фахівців у відповідних галузях, що особливо важливо, наприклад, для медицини або інженерії програмного забезпечення, де знання постійно оновлюються. Якщо фахівець, здобувши базову освіту, припиняє вчитися, його професіоналізм з часом починає знижуватися.

Безперервність освіти стимулюється вимогою до фахівців бути в курсі важливих подій, нововведень в їх сфері діяльності.

6.1. Особливості розвитку професійних навичок майбутніх програмних інженерів

Після закінчення університетського навчання за фахом «Програмна інженерія» або за будь-якої іншої комп'ютерної спеціальності молодий фахівець має певний запас знань. Проте часто цих знань виявляється недостатньо, коли він приступає до роботи в якій-небудь організації, що займається розробкою програмного забезпечення. Чого ж не вистачає молодому фахівцю для успішної кар'єри за фахом? По-перше, не вистачає досвіду роботи над реальними крупними проектами програмістів; для його набуття необхідний час. По-друге, у молодого фахівця відсутнє вміння працювати в команді. Ці два види досвіду можна отримати тільки впродовж тривалого виробничого процесу, але багато проблем, які при цьому з'являються, можуть бути розв'язані й під час університетського навчання [3, 5].

Розглянемо два шляхи подолання проблеми розвитку професійних навичок – командний і проектний підхід у процесі підготовки студентів комп'ютерних спеціальностей – та подамо методики формування професійних навичок роботи у студентів комп'ютерних спеціальностей [3].

Аналіз науково-методичних джерел, експертних оцінок і практичного досвіду роботи вітчизняних та зарубіжних університетів свідчать про те, що проблема професійної підготовки фахівців у галузі комп'ютерних наук не має загальноприйнятих рішень [3 – 5, 8 – 12]. Багато дослідників цієї проблеми вивчають питання, пов'язані з формуванням пізнавальної самостійності студентів, і зазначають, що сучасна система вищої освіти повинна приділяти увагу підготовці фахівців, здатних самостійно вивчати відповідну їх професії інформацію та вміти застосовувати її в подальшій післяуніверситетській діяльності. Зокрема, активно вивчаються та впроваджуються у навчальний процес рекомендації фірмового керівництва Microsoft, де представлені підходи до роботи над крупними проектами, а також описані механізми управління роботою групи програмістів.

Розвиток ІТ-індустрії привів до того, що часи, коли програміст один працює над своєю програмою, вже пройшли. Для розробки конкурентоспроможного програмного продукту необхідна робота великої команди програмістів, а іноді й фахівців з різних галузей (наприклад, при розробці бухгалтерського програмного забезпечення не можна обійтися без економістів і бухгалтерів на етапі постановки завдання). Для того, щоб молодий фахівець міг успішно працювати в такій команді, недостатньо мати знання тільки з програмування, дуже важливо вміти взаємодіяти з рештою членів команди. Аналіз типових вимог працедавців у ІТ-індустрії дозволяє побачити, що на першому місці звичайно стоїть досвід успішної роботи в команді.

Для того, щоб студенти набули досвіду командної роботи ще під час навчання, їм можна пропонувати працювати над сумісними проектами. Командна робота розвиває як професійні комунікативні, так і менеджерські навички [8, 9].

Одна з основних проблем, з якою доводиться стикатися на молодших курсах, полягає у великій різнорідності знань з програмування у студентів (серед студентів є ті, хто добре розбирається в програмуванні, знає кілька мов програмування, а є ті, хто, обираючи комп'ютерну спеціальність, не мав ніяких попередніх знань з програмування). Якщо починати пропонувати студентам працювати над сумісними проектами ще на молодших курсах, то більш знаючі студенти, як правило, підтягають до свого рівня слабкіших; на старших курсах схильність до такої взаємної підтримки слабшає, поступаючись місцем професійній конкуренції, але в перші роки навчання цей підхід сприяє вирівнюванню професійних знань у групі.

На старших курсах можна для організації роботи в групі використовувати класичну схему або модель рівноправних груп. Класичну схему можна організувати таким ролевим набором: менеджер проекту, провідний програміст, провідний тестувальник, провідний розробник документації, програмісти. Модель рівноправних груп передбачає відсутність менеджера проекту. За основу можна взяти Microsoft Solution Framework (MSF). Згідно з рекомендаціями MSF, робота груп ведеться

навколо шести ролевих кластерів: управління програмою, розробка, тестування, управління випуском, задоволення споживача, управління продуктом. На відміну від класичної схеми, ця модель легко налаштовується під проекти різних розмірів, тим самим її нескладно адаптувати до учбової роботи над проектом. Ці схеми допоможуть успішно імітувати роботу фірми, яка займається розробкою програмного забезпечення.

У процесі навчання студенти повинні залучатися до роботи над крупними реальними проектами. Фахівці індустрії розробки програмного забезпечення вважають, що однією з проблем діяльності IT-компаній є інтеграція нових молодих фахівців у процес розробки [22 – 25]. Коріння цієї проблеми в тому, що під час навчальних занять студенти не працюють над реальним проектом, а розв'язують модельні задачі. Звичайно комерційний продукт створюється командою розробників для клієнта. Програмний продукт повинен запускатися на різних комп'ютерах, мати зручний та інтуїтивно зрозумілий користувачу програми інтерфейс, мати інсталяційний пакет і поновлювач версій.

Весь цей набір вимог можна імітувати і в рамках учбових проектів. Проте важко імітувати жорсткість і безкомпромісність вимог замовника.

Професійні практичні навички майбутніх програмістів формуються в навчальних аудиторіях у рамках трьох технологій навчання: індивідуальні лабораторні заняття, учбові групові проекти і реальні комерційні проекти.

Зіставимо основні відмінності та очікувані результати лабораторних індивідуальних завдань, звичайних учбових групових проектів, реальних комерційних проектів (табл. 6.1) [23].

Таблиця 6.1

ЗІСТАВЛЕННЯ ТРЬОХ ТЕХНОЛОГІЙ НАВЧАННЯ

Лабораторні індивідуальні завдання	**Учбові групові проекти**	**Комерційні проекти**
Програма створюється однією людиною	Проект створюється групою програмістів	Проект створюється групою програмістів
Дизайн початкового коду, ділення на модулі довільні	Дизайн початкового коду та ділення на модулі регламентуються стандартами програмування, що використовуються командою розробників	Дизайн початкового коду та ділення на модулі регламентуються стандартами програмування, що використовуються сумісною командою розробників та замовників проекту
Програма створюється для одного поточного завдання, на одному комп'ютері	Проект створюється для одного поточного завдання, запускається на декількох комп'ютерах	Проект створюється для клієнта, використовується на різних комп'ютерах. Завдання, поставлене перед розробниками на початку, може істотно змінитися в процесі експлуатації і супроводу
Робота над програмою не має продовження і розвитку	Робота над проектом не має продовження і розвитку	Програмний продукт корегується і супроводжується

Продовження табл. 6.1

Початковий код може бути зрозумілий тільки автору	Початковий код і зовнішній інтерфейс модулів та класів, а також коментарі повинні бути зрозумілі та зручні для інших розробників проекту	Початковий код і зовнішній інтерфейс модулів та класів, а також коментарі повинні бути зрозумілі та зручні для інших розробників проекту і для користувачів
Програма не вимагає розвиненого інтерфейсу і спеціального оформлення	Проект не вимагає розвиненого інтерфейсу	Проект вимагає добре продуманого інтерфейсу користувача, спеціального оформлення, підготовки супроводжуючої документації
Програма не вимагає обробки виняткових випадків	Проект не вимагає обліку, аналізу й обробки виняткових випадків	Проект повинен бути захищений від некоректних дій користувача, повинен обробляти ситуації браку пам'яті, помилки роботи з файлами та інші некоректні події

Аналіз таблиці показує таке [23]:

1. Робота над лабораторними індивідуальними завданнями
- дає можливість студенту розвивати індивідуальні навички програмування;
- дозволяє вивчити практичне застосування теоретичних аспектів програмування.

2. Робота над учбовим груповим проектом, окрім вищепереліченого в пункті 1,

- дає студенту можливість набути досвіду роботи в різних ролях учасника проекту;
- знайомить студента із засобами управління початковим кодом;
- прищеплює студенту навички використання загальних нотацій та складання глосарію загальної термінології проекту;
- учить студентів писати самодокументований код;
- знайомить студента з методами екстремального програмування;
- дозволяє студенту одержати навички аналізу поточного стану проекту, побудови графіків виникнення та усунення помилок;
- дає студенту досвід складання технічного завдання, використання шаблонів проектування, універсальної мови проектування UML.

3. Робота над комерційним проектом, окрім вищепереліченого в пунктах 1 і 2,
- дає можливість роботи над сумісним проектом, при цьому не обов'язково знаходячись в одному приміщенні чи будівлі з іншими членами команди;
- знайомить із сучасними методиками розробки програмних продуктів (постановка технічного завдання, проектування, тестування, документування, управління кодом);
- дозволяє набути навички розподіленої розробки програмного продукту;
- дозволяє одержати навички роботи у реальному часі, в умовах реального (іноді жорсткого) тиску з боку керівництва та замовників.

Як можна формувати командні та проектні навички? На думку переважної більшості фахівців [1, 22 – 25], формування командних і проектних навичок неможливе тільки в рамках однієї учбової дисципліни, потрібен комплекс дисциплін, які пронизують весь інтервал навчання, з першого до останнього дня перебування студента в університеті. У перший рік навчання дуже важливі дисципліни «Вступ до спеціальності» та «Програмування та алгоритмічні мови». Дисципліна «Вступ до спеціальності» розвиває інтерес до вибраної спеціальності та формує мотивацію до навчання. Дисципліна «Програмування та алгоритмічні мови» стимулюючими завданнями та учбовими

груповими проектами сприяє розвитку алгоритмічного мислення, формує комунікативні та професійні здібності.

На другому році навчання дисципліна «Проектування та конструювання програмного забезпечення» сприяє розвитку у студентів навичок розробки та оформлення проектів, створення інсталяційних пакетів, підготовки документації тощо. Але найбільш важлива дисципліна, яка і формує командне проектне мислення, це «Проектний практикум програмної інженерії», що вивчається на старших курсах. У рамках цієї дисципліни студенти в команді працюють над проектами. Вважається, що після закінчення вивчення цієї дисципліни у студентів повинні з'явитися навички самоконтролю: уміння оцінювати свою роботу, свій прогрес, розуміння того, коли необхідна додаткова інформація, а коли потрібно перейти до наступного кроку для вдосконалення професійної майстерності.

Для роботи над реальними комерційними проектами в учбовому плані передбачена виробнича практика, під час якої студенти в команді працюють над реальними проектами на підприємствах, поряд з ними працюють досвідчені програмісти, розробники програмного забезпечення [1]. У деяких випадках ситуацію розробки програмного продукту в рамках реального проекту частково можна імітувати в рамках учбової технології Industrial Case Study, запрошуючи в як тимчасових викладачів персонал місцевих комп'ютерних фірм.

По завершенні проекту група студентів-розробників повинна робити презентацію свого проекту. Після презентації інші групи студентів мають ставити питання, що стосуються презентованого проекту, а потім висловлювати критичні зауваження й побажання. Частині студентів пропонується оцінити проект з позитивної точки зору, інші повинні його критикувати. Ця процедура розвиває навички професійного критичного мислення.

6.2. Особливості працевлаштування програмних інженерів

Незважаючи на те, що переважна більшість випускників вузів вважають, що вони добре підготовлені до самостійної роботи, перші дні, а іноді й місяці, не такі вже і прості. Цей процес зазвичай називають «входженням у реальний світ». Робота інженера сильно відрізняється від студентського навчання [8].

Багато хто відзначає, що розподіл часу стає критичнішим. Якщо в студентські роки все було жорстко регламентовано (час лекцій, лабораторних, контрольних, заліків тощо), то на роботі зазвичай фіксується лише термін завершення роботи. Усі проміжні етапи, їх терміни тощо інженер повинен встановити собі сам. Проект може тривати рік і навіть кілька років, і в деяких фірмах на проміжних етапах ніхто інженера не перевіряє. Багато хто з молодих інженерів вважають цей фактор достатньо складним та проблемним.

На деяких фірмах молодому інженеру призначається так звана «тренувальна програма», в процесі проходження якої до молодого спеціаліста прикріпляють керівника. На інших же фірмах йому можуть відразу дати самостійну роботу й перевірити, як він з нею справиться. У будь-якому випадку можна сформулювати загальні рекомендації молодому інженеру щодо його діяльності на початкових етапах [8]:

- Діяти як професіонал.
- Бути уважним і холоднокровним.
- Не намагатися здати незавершену роботу.
- Бути акуратним. Завжди перевіряти свою роботу.
- Не лякатися складних завдань.
- Вчитися доступно описувати і виражати свої дії.
- Брати приклад з успішних і компетентних інженерів.

Дослідження, проведені сімома крупними фірмами США над 200-ми інженерами-початківцями, яких керівництво вважало успішними, показали, що вони володіють такими здібностями [8]:

- вони завойовували прихильність інших своїх колег, поважаючи та враховуючи їх думку;
- вони вміли уважно слухати, ставлячи ділові запитання;

- вони не обурювалися, якщо з ними не були згодні, та не «задирали носа», якщо їх хвалили;
- вони збирали необхідну інформацію, перш ніж висловити свою думку;
- вони краще усвідомлювали, коли слід напружуватися, а коли ні, кого інформувати, а кого перевіряти.

Таким чином, молоді фахівці, що починають свою трудову діяльність, повинні зважати на період звикання. Вони повинні враховувати той факт, що товариське ставлення, чемна поведінка, поблажки по відношенню до підлеглих, які випливають з їх деяких рис вдачі, можуть часто принести користь загальній справі – робоча обстановка буде дружньою й колеги стануть соратниками та однодумцями.

Багато фірм мають спеціальні тренувальні програми, які повинні пройти молоді фахівці, перш ніж їм довірять самостійну роботу. Іноді ці програми містять деякі теоретичні курси, написання звітів і складання іспитів. Основна увага на цих курсах приділяється організації праці саме на певній фірмі (відносини підлеглий-начальник, рух документів тощо). Іноді даний процес триває до 2-х років. Звичайно, до тренувальних програм слід ставитися дуже відповідально, оскільки вони передбачають, перш за все, навчання на своєму робочому місці, тобто виконання своєї роботи та можливість консультацій зі своїм керівником у випадку виникнення деяких проблем.

Опитування представників інженерної спільноти показують, що саме цей вид стажування подобається молодим фахівцям більше, ніж теоретичні програми.

Щодо основних підходів, які широко використовують різноманітні фірми США і Канади відносно стратегій знайомства з робочим місцем, можна виділити такі [8]:

1. «*Потонути або плисти*» – молодому фахівцю відразу дають самостійний проект, мінімальну кількість настанов та оцінюють його діяльність за кінцевим результатом.

2. «*Негативний досвід*» – молодому фахівцю показують схему системи, яка за його уявленнями не може функціонувати, потім йому показують саму працюючу систему, яка за цією

схемою спроектована і яку фірма випускає вже досить давно. Непорозуміння, в яке потрапляє молодий фахівець, скоро проходить і він досить швидко усвідомлює той факт, що не все, що він вивчав на етапі початкової освіти, – завжди абсолютна істина і що до своїх знань слід підходити критично.

3. *Тренування роботою* – молодому фахівцю пропонують деяку невеличку самостійну роботу, що відповідає його досвіду та умінню, і ця робота проводиться під наглядом фахівця.

4. *Робота з тренуванням* – з працівником проводять теоретичне тренування, але одночасно йому довіряють маленькі реальні завдання, пов'язані з його майбутньою роботою. При цьому відбувається знайомство з різними ділянками роботи.

5. *Повномасштабне тренування* являє собою велику кількість формальних теоретичних завдань. Молодий фахівець спостерігає та навчається тому, чим займаються інші висококваліфіковані колеги, за яких умов та які рішення ухвалюються. При цьому його особиста участь мінімальна.

6. *Інтегруюча стратегія* – працівнику пропонується реальна робота й водночас він прослуховує теоретичні курси.

У випадку двох перших стратегій працівник досить швидко включається в трудовий процес. При цьому вважається, що перші завдання, які постають перед молодим інженером, повинні бути якомога відповідальнішими. Це повинно бути корисно як фірмі, так і молодому фахівцю, хоча тут, звичайно, має місце частка ризику. Незважаючи на це, подібні стратегії дозволяють уже на самому початку трудової діяльності використовувати максимальний потенціал молодого фахівця та стимулювати його активний розвиток.

З іншого боку, формальне тренування дозволяє молодому фахівцю знайти «відповідне місце» на фірмі. Часто це виявляється зовсім не та робота, до якої він себе спочатку готував. У ході такої програми молодий фахівець стикається з дуже різними посадами, з кожною приблизно по 6 місяців, обираючи собі найбільш відповідну.

У США проведено ряд опитувань серед молодих фахівців з питань основних труднощів, з якими вони зіштовхуються на

новій роботі. Наведемо тут проблеми, що найчастіше зустрічаються (послідовність залежить від частоти проблем) [8]:

1. Звикання до звичаїв, прийнятих на даній фірмі (іншими словами, до усталеної практики).
2. Повільне (або незрозуміле) кар'єрне зростання.
3. Необхідність займатися рутинною роботою.
4. Розуміння того, чого від тебе очікують.
5. Знаходження своєї ніші.
6. Нереалістичність реалізацій амбіцій.
7. Недостатність ініціативності.
8. Недостатність заохочень.
9. Недостатність спеціалізованих курсів.
10. Недостатність визнання.

Першу проблему можна зменшити або навіть усунути тренувальними програмами. Проблеми 4, 5, 8 зменшуються автоматично з часом, коли новий працівник звикає до фірми. Решту проблем розв'язати дещо складніше.

Молодих фахівців часто критикують за те, що в них немає достатнього рівня ініціативності (п. 7) і при цьому вони дуже амбітні (п. 6). Їх часто звинувачують у відсутності глибоких знань для виконання серйозних завдань, і тому їх навантажують рутинною роботою.

Ентузіазм – одна з найбільш цінних якостей у молодих фахівців, при цьому зайвий ентузіазм сприймається як агресивність і необґрунтована амбітність. Часта нереалізованість ентузіазму призводить до його зниження.

Якщо до рутинних завдань уміло підходити, роблячи їх швидко, то може залишатися багато часу на цікаву роботу. Таким чином, можна часто потрапляти в поле зору керівництва, що, в свою чергу, може призвести до кар'єрного зростання.

Після того, як інженер звик до особливостей роботи фірми, показавши свої вміння, старанність, ентузіазм, повинні послідувати кроки працедавця. Завдання інженера – робити добре свою роботу. Завдання керівництва фірми – всіляко в цьому допомагати інженеру й заохочувати його.

Те, що завжди допомагає справі та з чим завжди згодні обидві сторони – це гарні взаємини між інженерами і керівництвом фірми. Для цього інженери повинні мати інформацію щодо загальної стратегії та цілей фірми. Особливо важлива та інформація, яка могла б вплинути на проект, що розробляється інженером.

З іншого боку, багато керівників не поспішають забезпечувати молодих працівників цією стратегічною інформацією, оскільки побоюються, що працівник може на фірмі не затриматися і, таким чином, ця стратегічна інформація може потрапити до конкурентів. Тому повної відвертості тут бути не може, але завжди необхідно прагнути до цього в міру можливості в тих межах, які не зашкодять обом сторонам.

Інше коло питань пов'язане із заробітною платнею і кар'єрним зростанням. Звичайно всі погоджуються з тим, що зарплата повинна відображати внесок інженера в загальний бюджет фірми. Проте опитування показують, що це має місце далеко не завжди. Часто зарплата залежить від ринку праці та від того, наскільки той чи інший працівник на даний момент корисний і потрібний фірмі. Опитування показують, що лише у 23 % фірм США молодому фахівцю платять відповідно до його внеску.

Важлива в інженерній роботі також якість «інструментарію»: наявність високотехнологічного устаткування, комп'ютерів з найсучаснішим апаратним забезпеченням та інших пристроїв (ксерокси, факси, добре телефонне з'єднання, комп'ютерна мережа тощо), а також наявність допоміжного персоналу (секретарів, асистентів, працівників бібліотек тощо).

Результат опитування молодих фахівців у 1987 році в США виявив таку послідовність мотивацій роботи (вибрати слід було одне з двох у парі) [8]:

- цікава работа важливіша, ніж престиж і визнання (100%);
- завдання з розв'язання технічних проблем важливіші, ніж з управління (87 %);
- різностороння робота важливіша, ніж концентрація на одному вузькому проекті (80 %);

- робота над своєю ідеєю важливіша, ніж робота над тим, що від тебе очікують (76 %);
- відчуття самовираження важливіше, ніж хороші керівники та колеги (68 %);
- робота в групі важливіша, ніж самостійна робота (67 %);
- хороша заробітна плата важливіша, ніж визнання того, що робота добре виконана (64 %);
- кар'єрне зростання важливіше, ніж безпека і надійність роботи (61 %);

6.3. Групова динаміка та комунікації як основний стиль роботи програмних інженерів

Основна частина професійного програмного забезпечення розробляється командами програмістів (від двох до кількох сотень чоловік). Але, оскільки навряд чи хтось має змогу ефективно працювати над однією задачею в такій великій команді, ці команди діляться ще й на підгрупи. Кожна підгрупа відповідає за певну частину проекту та працює над однією підсистемою. При грамотному підборі група складається не більше ніж з восьми чоловік. У групах невеликого розміру легше знизити ризик виникнення проблем у взаємовідносинах між членами групи. Кожна група повинна бути забезпечена круглим столом для проведення зустрічей. Крім цього, члени групи мають можливість зустрічатися в офісах. Для таких груп немає необхідності застосовувати складні структури комунікації.

Організація команди, яка могла би ефективно працювати над розробкою програмного забезпечення, є достатньо складною задачею [9, 10, 23]. Необхідно, щоб у команді було рівне співвідношення технічних навичок, досвіду та вираження індивідуальності. Добре функціонуюча команда – це дещо більше, ніж простий набір людей з необхідним співвідношенням навичок. У добрій команді присутній товариський дух, який мотивує співробітників за рахунок успіхів усієї команди, включаючи й досягнення власних цілей. Тому керівники таких груп (менеджери) повинні стимулювати діяльність, спрямовану

безпосередньо на «розбудову команди», щоб сприяти формуванню почуття відданості її інтересам.

Перерахуємо чотири основних фактори, які тією або іншою мірою впливають на групову роботу [10].

1. *Склад команди*. Команда повинна мати правильне співвідношення навичок, досвіду та особистісних якостей.

2. *Згуртованість команди*. Члени робочої групи повинні сприймати себе як єдину команду, а не як просту сукупність індивідуумів, які працюють над однією проблемою.

3. *Спілкування в команді*. Між членами команди повинні бути дружні відносини.

4. *Організація команди*. Необхідно організувати команду так, щоб кожен відчував свою цінність та був задоволений своєю роллю.

На початку практичної діяльності для більшості спеціалістів з програмного забезпечення основним мотиваційним фактором служить робота. Тому до групи з розробки програмних продуктів часто залучені люди, які мають власні міркування на рахунок того, як повинні розв'язуватися задачі технічного характеру. Такий принцип підбору робочих груп викликаний постійними скаргами на проблеми, що виникають унаслідок ігнорування стандартів інтерфейсу, переробки системи після створення її програмного коду, непотрібних нововведень у систему тощо. Необхідною умовою для запобігання появі цих проблем є правильний підбір членів робочої групи.

Розглядаючи мотивацію з психологічної точки зору, можна виділити три типи професіоналів [10, 23].

1. *Люди з цільовою орієнтацією*, які отримують достатньо мотивації від роботи, яку виконують. До цього типу відносять «технарів», мотивація яких викликана інтелектуальними задачами з розробки програмного забезпечення.

2. *Люди із самоорієнтацією*, мотивація яких заснована на особистому успіху та визнанні. Вони зацікавлені в розробці програмного забезпечення, переслідуючи при цьому особисті інтереси.

3. *Люди із зовнішньою орієнтацією*, мотивація яких потребує присутності та діяльності співробітників. Оскільки в наш час створення програм стає чимраз більш орієнтованим на користувача, такі люди дедалі частіше залучаються до розробки програмного забезпечення.

Люди із зовнішньою орієнтацією вважають за краще працювати у колективі, тоді як співробітники із самоорієнтацією та цільовою орієнтацією прагнуть до роботи самостійно. Жінки належать до типу людей із зовнішньою орієнтацією частіше, ніж чоловіки. Вони також краще діють як розповсюджувачі інформації.

Мотивація кожного індивідуума містить у собі всі перераховані елементи, хоча в певний період часу переважає лише один тип мотивації. Ніхто не стверджує, що особисті властивості характеру будь-якого індивідуума постійні. Людині властиво змінюватися. Наприклад, у «технарів», які не задоволені оплатою своєї праці, може відбутися переоцінка цінностей, після чого особисті інтереси стануть для них вище зацікавленості у вирішенні технічних питань.

Водночас не слід опиратися виключно на особистісну мотивацію та забувати той факт, що люди відчувають себе частиною організації, професійної групи або частиною деякої корпоративної культури, яка склалася на робочому місці. Отже, у мотивації задіяні не лише особисті інтереси, але й інтереси цих груп. Членство у згуртованій групі є надзвичайно високою мотивацією для багатьох. Люди, які виконують більш прості завдання, часто полюбляють ходити на роботу лише тому, що їм подобаються співробітники, з якими вони працюють, і завдання, які вони виконують.

Група, в якій співробітники доповнюють один одного, може працювати набагато ефективніше за групу, відбір до якої проводився виключно на основі навичок програмування. Люди, які люблять свою роботу (цільова орієнтація), можуть стати чудовими професіоналами. Люди із самоорієнтацією на найкращий результат зможуть довести справу до кінця. Співробітники із зовнішньою орієнтацією успішно налагоджують

спілкування всередині групи. Такі співробітники – необхідна складова будь-якої робочої групи. Вони налаштовані на спілкування і тому можуть визначити та попередити виникнення якої-небудь напруги або конфлікту на ранній стадії. Саме такі люди допоможуть розв'язати особисті проблеми членів команди й усунути розбіжності між ними, перш ніж ті вплинуть на всю команду.

Іноді просто неможливо організувати команду так, щоб всі її члени доповнювали один одного за особистими якостями. Якщо таке трапляється, керівництво зобов'язане контролювати групу з тим, щоб не дозволити особистим цілям співробітників стати вище за цілі організації або групи. Досягти такого контролю набагато легше в тому випадку, якщо члени команди будуть брати участь у кожному етапі проекту. Прояв індивідуальної ініціативи буде більш імовірним, якщо з членами групи проводити інструктаж, не згадуючи тієї ролі, яку їх частина роботи відіграє у всьому проекті.

Наприклад, візьмемо інженера, який отримав проект програми для кодування і бачить, що в нього необхідно внести зміни, які роблять його більш досконалим. Якщо впроваджувати ці зміни без розуміння логіки початкового проекту, це може призвести до конфлікту з іншими частинами програмної системи. Якщо вся група залучена до розробки проекту із самого початку, працівник зможе оцінити наслідки внесення таких змін у систему, і це не спричинить конфлікту з тими спеціалістами, які її проектували.

Важливе місце в команді займає лідер. Він (або вона) відповідає за технічне керівництво та адміністративне керування. Лідери групи повинні бути в курсі повсякденної діяльності групи, гарантуючи ефективну роботу команди та тісне співробітництво з менеджерами проекту при плануванні діяльності щодо його реалізації.

Лідер – це, як правило, призначувана посада, він підзвітний головному менеджеру проекту. Призначуваний лідер може і не бути лідером команди у прямому сенсі цього слова, він веде групу лише в технічних питаннях. Члени групи можуть обрати іншого лідера команди. Він може краще за назначеного лідера

розбиратися в технічних питаннях або мотивувати членів групи до виконання роботи.

Іноді буває корисним узагалі розділити технічне лідерство та керування проектом. Часто трапляється, що компетентні в технічних питаннях працівники виявляються слабкими адміністраторами. Виконуючи роль адміністратора, такі лідери вже не можуть вносити повноцінний внесок у технічну роботу команди. В цій ситуації краще призначити адміністратора, який звільнить лідера від щоденних адміністративних проблем.

Якщо у групу призначається лідер, неприйнятний для членів команди, це може призвести до досить напруженої обстановки. Члени групи не будуть поважати нового лідера і можуть надавати перевагу своїм інтересам замість інтересів групи. Ця проблема досить суттєва для такої швидкозмінної галузі, як інженерія програмного забезпечення, де нові члени команди можуть володіти більш сучасними знаннями та досвідом, ніж лідери груп, які мають лише практичний досвід. Крім того, деяких досвідчених співробітників може образити призначення молодого лідера, навіть якщо він здатен внести у роботу нові ідеї.

Дуже важливим фактором є згуртованість команди. Члени згуртованої команди дбають про її інтереси більше ніж про власні. Люди у добре керованій, згуртованій команді віддані її інтересам. У них високо розвинений командний стиль та ототожнення цілей групи з особистими інтересами. Вони також сприймають групу як деяку єдину сутність і намагаються захистити її від зовнішнього впливу. Це укріплює групу, вона стає спроможною самостійно справлятися з проблемами та непередбачуваними ситуаціями. Члени групи підтримують один одного в процесі різноманітних змін, що допомагає їм долати труднощі.

Добре згуртована команда має ряд переваг [10, 23].

1. *Можливість становлення стандарту якості групи*. Оскільки цей стандарт визначається всією групою одноголосно, його контролювати легше, ніж чужі стандарти, які нав’язуються групі зовні.

2. *Члени команди підтримують тісні робочі контакти*. Працюючи в групі, люди навчаються один в одного. Скутість та затягування роботи, викликані незнанням або необізнаністю, зменшуються в міру того, як відбувається взаємне навчання.

3. *Члени команди ознайомлені з діяльністю один одного*. Цим досягається можливість продовження роботи навіть після відходу від роботи якогось зі співробітників.

4. *Можливе впровадження у практику групи безособистісного програмування*. Створена програма повинна бути власністю всієї команди, а не окремої людини.

Безособистісне програмування означає певний стиль роботи, при якому проекти, програми і документація вважаються власністю всієї групи, а не людини, яка займалася їх розробкою. Якщо розробники ставляться до своєї роботи саме так, вони охоче віддають її на перевірку іншим членам групи, легко сприймають критику, прагнуть працювати над удосконаленням системи. При цьому команда розробників стає більш згуртованою, оскільки кожен відчуває відповідальність за розробку всієї системи.

Крім того, що безособистісне програмування поліпшує якість системної архітектури, програм та документації, воно також удосконалює відносини всередині групи, стимулює невимушене обговорення задачі, незалежно від соціального положення, досвіду та статі. В ході виконання проекту члени групи більш активно підтримують робочі стосунки. Всі ці фактори сприяють згуртованості та тому, що кожен робітник сприймає себе як частину команди.

Багато факторів впливають на згуртованість групи, в тому числі організаційна культура та вираження особистісних якостей у групі. Менеджери можуть розвивати згуртованість кількома шляхами. Можна організувати соціальні заходи для робітників та їх родин. Можна прищепити групі відчуття самобутності, для чого необхідно дати їй назву, визначити сутність команди та сферу її діяльності. Менеджери повинні проводити заходи (наприклад, ігри або спорт), спрямовані на створення команди.

Однак найкращий спосіб виховати дух команди – дати можливість кожному відчути, що він несе певну долю

відповідальності та що йому довіряють, а також гарантувати доступ до проектної інформації для всіх членів групи. Іноді менеджерам здається, що вони не повинні розкривати певну інформацію. Однак така лінія поведінки буде постійно створювати в групі почуття недовіри. Простий обмін інформацією – найбільш дешевий та ефективний спосіб дати людям відчути себе частиною команди.

Однак у сильних та згуртованих групах також можуть виникати проблеми [10, 23].

1. *Неусвідомлений спротив заміні лідера*. Якщо лідера згуртованої групи необхідно замінити кимось зі сторони, члени групи можуть спільно виступити проти нового лідера. Робітники витратять багато часу, проявляючи спротив змінам та нововведенням, які пропонує новий лідер, при цьому поступово падає продуктивність праці. Тому по можливості слід призначати нового лідера зі складу групи.

2. *Групове мислення*. Це назва ситуації, в якій цінні робочі якості членів групи руйнуються в результаті відданості команді. Висування альтернативних пропозицій замінюється прихильністю нормам та рішенням групи. Будь-яка ідея сприймається, якщо більшості це вигідно, альтернативи при цьому не розглядаються.

Для запобігання проблемі групового мислення необхідно організовувати офіційні засідання, де члени команди будуть змушені так чи інакше висловлювати власні думки. Можна запросити незалежних експертів для аналізу рішення групи. Команда повинна складатися з людей, які можуть дискутувати, ставити питання та не звертати особливої уваги на усталене положення. Під час дискусій такі люди виступають палкими сперечальниками, постійно піддаючи сумнівам рішення групи, побуджаючи тим самим членів групи обдумувати та оцінювати свої дії.

Для групи з розробки програмних продуктів необхідний розвинений комунікаційний фактор, тобто спілкування та гарні засоби зв’язку між членами групи. Робітники повинні інформувати один одного про те, як просувається їх робота, щодо

рішень, які приймались відносно проекту, та тих необхідних змін, які вносились у попередні розпорядження. Постійне спілкування також сприяє згуртованості, оскільки робітники краще починають розуміти мотивацію своїх колег, їх слабкі та сильні сторони.

На ефективність спілкування можуть впливати нижчезазначені чинники [10, 23].

1. *Розмір групи*. Чим більша група, тим складніше забезпечити постійне спілкування між її членами. Різниця у соціальному положенні членів групи призводить до появи більшої кількості однобічних зв'язків. Члени групи з більш високим положенням у суспільстві частіше домінують під час спілкування зі своїми підлеглими колегами, які, у свою чергу, неохоче йдуть на контакт або висловлюють критичні міркування.

2. *Структура групи*. Робітники, які перебувають у групах з неформальною структурою, легше спілкуються між собою, аніж у групах, які мають певну офіційну ієрархію у відносинах. В останніх спілкування відбувається чітко в ієрархічній послідовності (у той чи інший бік). Співробітники на одній і тій самій ієрархічній сходинці можуть узагалі не спілкуватися між собою. Це в основному проблема широкомасштабних проектів, які включають кілька груп з розробки програмних продуктів. Якщо в такому проекті співробітники різних відділів спілкуються лише через своїх менеджерів, це може привести до запізнення у проведенні робіт та непорозуміння один одного.

3. *Склад групи*. Якщо у групі багато людей зі схожими особистісними характеристиками, вони можуть конфліктувати один з одним, унаслідок чого може значно знизитися рівень спілкування у групі. Краще за все люди спілкуються у змішаних різностатевих групах, аніж в однорідних за статтю. Серед жінок переважає зовнішня орієнтація, тому у групі вони служать свого роду регуляторами взаємовідносин, що полегшують спілкування.

4. *Робоче оточення*. Правильна організація робочого місця – основоположний фактор у розвитку або гальмуванні комунікаційних зв'язків у групі.

Невеликі команди програмістів найкраще організовувати в легкому неформальному дусі. Лідер бере участь у роботі над програмним продуктом нарівні з іншими членами групи. Технічним лідером може стати людина, яка якнайкраще буде керувати процесом розробки. Неформальна група завжди обговорює майбутню роботу зі всіма членами колективу, а завдання призначаються відповідно до можливостей і досвіду конкретного співробітника. Високорівневе проектування системи здійснюється старшими спеціалістами, проектування низького рівня пропонується тому співробітнику, який призначений на виконання конкретного завдання.

Робота неформальних груп може виявитися надзвичайно ефективною, особливо якщо більшість членів групи досвідчені та кваліфіковані спеціалісти. Група функціонує як демократичне утворення, де рішення приймаються більшістю голосів. У психологічному плані це породжує дух команди і, як наслідок, приводить до укріплення згуртованості та підвищення продуктивності праці. Неформальна організація групи може виявити медвежу послугу у випадку, якщо група складається з недосвідченого або некомпетентного персоналу. Виникає брак керівної ланки, яка може керувати роботою, що спричинює недостатню координованість робіт і, можливо, провал проекту.

Організація робочого місця має винятково важливе значення для продуктивності праці та отримання задоволення від роботи. Психологічні дослідження довели вплив на поведінку співробітників розміру кімнати, меблів, обладнання, температури, вологості, яскравості та якості світла, рівня шуму та можливості усамітнення. Поведінка членів групи розробників також залежить від архітектурних особливостей приміщення та можливостей доступних засобів зв’язку.

Команди з розробки програмних продуктів часто працюють у великих відкритих офісах, іноді з перегородками, і лише старшим за посадою співробітникам можуть надаватися окремі кабінети. У роботі ДеМарко «Peopleware: Productive Projects and Teams» наведено результати досліджень, які показали, що відкриті приміщення, що використовуються багатьма фірмами, абсолютно

непопулярні та непродуктивні. В цьому дослідженні визначені ключові фактори оформлення робочого простору [10].

1. *Усамітнення*. Програмістам необхідне певне місце, де вони можуть сконцентруватися та працювати без відволікання.

2. *Огляд зовнішнього світу*. Люди надають перевагу роботі при денному світлі, маючи перед собою привабливий вигляд з вікна.

3. *Індивідуалізація*. Люди засвоюють різноманітні звички в організації робочого процесу та мають різні думки відносно оформлення приміщення. Тут велике значення має можливість облаштовувати робоче місце так, щоб воно відповідало різним смакам і відповідало індивідуальним рисам.

Таким чином, людина за своєю природою вважає за краще працювати в окремому приміщенні, яке вона може оформити за своїм смаком та бажанням. Такі кабінети, на відміну від відкритих офісів, створюють більш сприятливу обстановку та знижують кількість пауз у роботі. У відкритому приміщенні співробітнику неможливо сконцентруватися настільки, як це можна зробити в тихій усамітненій робочій обстановці. Наслідком зниження концентрації стає падіння рівня продуктивності праці. Отже, окремі приміщення для програмістів істотно впливають на продуктивність.

З іншого боку, групи з розробки програмного забезпечення також потребують приміщень, де всі члени групи можуть збиратися разом та обговорювати проект у вигляді офіційного засідання або просто неформальної зустрічі. Кімнати для зустрічей повинні вміщувати всіх членів групи та забезпечувати їм необхідний рівень комунікації. Два види потреб, усамітнення та спілкування всередині групи, можуть здатися взаємовиключними. Виходом із ситуації може бути розміщення індивідуальних кабінетів навколо однієї великої кімнати для зустрічей.

6.4. Сертифікація та ліцензування програмних інженерів

Розглянемо ряд питань, які стосуються проблем сертифікації та ліцензування професійних програмістів [4]. Чи обов'язкова сертифікація? Чи потрібно одержувати ліцензію? Як це зачіпає фахівців? Цим питанням і присвячений даний розділ.

Сертифікація – це добровільна процедура, контрольована професійним співтовариством, мета якої – інформувати суспільство про те, що кваліфікація сертифікованого фахівця достатня для виконання певної роботи (або про відповідність товару, послуги необхідним стандартам). Для сертифікації зазвичай потрібні як освіта, так і практичний досвід. В більшості випадків компетентність претендента на сертифікацію визначається за наслідками письмового іспиту. Дія сертифікації, як правило, розповсюджується за межі обмеженої географічної зони, охоплюючи національний або міжнародний рівень. Найвідоміший приклад сертифікації в США – дипломований бухгалтер (Certified Public Accountant, CPA).

Деякі організації вже багато років пропонують сертифікацію фахівців у галузі програмного забезпечення. Інститут сертифікації професіоналів з використання комп'ютерів присвоює звання кандидата в професіонали і сертифікованого професіонала. Американське товариство контролю якості присвоює звання інженера з якості програмного забезпечення (хоча вживання терміна «інженер» піддає це товариство юридичним ризикам, оскільки його застосування регулюється законами в більшості штатів США і в Канаді).

Багато компаній мають програми сертифікації за власними конкретними технологіями. Microsoft пропонує диплом сертифікованого фахівця Microsoft, Novell видає диплом сертифікованого інженера мереж, у Oracle є сертифіковані фахівці Oracle, а Apple Computers присвоює звання професійного інженера серверів Apple [8]. Сфера такої сертифікації обмежена продуктами однієї компанії, що робить її швидше сертифікацією в галузі конкретної технології, аніж у сфері інженерії програмного забезпечення.

Сертифікація дає можливість працедавцям і клієнтам одержати інформацію про фахівців з програмного забезпечення, що мають принаймні якусь мінімальну кваліфікацію. І ринок уже відреагував на цей запит. З'являється велика кількість книг з питань різноманітних сертифікаційних іспитів у галузі програмного забезпечення і пов'язаних з комп'ютерами дисциплін. Майже всі ці іспити стосуються конкретних технологій.

У 2002 р. комп'ютерне товариство IEEE стало проводити сертифікацію з основ інженерії програмного забезпечення на звання сертифікованого фахівця розробки програм. Це перша загальна сертифікація в галузі інженерії програмного забезпечення, пропонована великою професійною організацією, і це істотний крок до системи атестації професіоналів розробки програм, що визнається у відповідній галузі.

Ліцензування – це обов'язкова дозвільна юридична процедура, що покликана захистити інтереси суспільства і зазвичай здійснюється територіальними органами юрисдикції (держав, штатів, провінцій тощо). У багатьох професіях національні організації надають територіальним органам допомогу в розробці ліцензійних вимог і формуванні програм іспитів.

Більшість професій ліцензуються (наприклад, лікарі, архітектори, юристи та інженери). У США жодна з професій, що так само суттєво впливають на життя суспільства, як програмне забезпечення, не залишилася неліцензованою.

В інженерній сфері США більшості фахівців не потрібно одержувати ліцензію. Інженерні фірми повинні мати деяку кількість ліцензованих інженерів, але не всі інженери зобов'язані мати ліцензії. Так, ліцензується половина інженерів-будівельників, а серед інженерів-хіміків – лише 8%. Відмінність полягає в ступені повторюваності проектованого об'єкта і його впливі на безпеку суспільства. Предмети, відтворювані у великих кількостях, можуть випробовуватися до їх запуску у виробництво для продажу населенню, що мінімізує ризик для суспільства і дозволяє зменшити кількість ліцензованих інженерів, потрібних

для виконання такої роботи. Інженери-електротехніки розробляють продукти, відтворювані у величезних кількостях: тостери, телевізори, телефони тощо. Тому лише невелика їх частина повинна мати ліцензію.

Інженери з будівництва цивільних споруд проектують велику кількість унікальних об'єктів, безпека яких критично важлива: шосе, мости, стадіони, злітно-посадочні смуги тощо. Тому інженерів з цивільного будівництва ліцензується значно більше, ніж інженерів-електротехніків.

Яка ситуація в цьому відношенні склалася в галузі програмного забезпечення? Розробники програм створюють багато унікальних продуктів, але також і велику кількість продуктів широкого користування: операційні системи, програми заповнення податкових декларацій, текстові процесори та інші програми, які відтворюються у мільйонах копій. Критично важливі для безпеки суспільства системи також розробляються, але значно більше систем, що менше впливають на неї, створюються для бізнесу.

Перебудовуючи будівлю, будівельник може самостійно ухвалювати багато проектних рішень. Якщо рішення може вплинути на міцність будівлі, то для оцінки такого рішення будівельнику потрібна допомога інженера. У галузі програмного забезпечення навіть додатки, що вимагають інженерного підходу, можуть переважно створюватися «будівельниками» – неліцензованими інженерами з програмного забезпечення і фахівцями з технологій. Тому більшість додатків програм не забезпечено інженерією зовсім, а там, де потрібна якась частина інженерії, додатки можуть створюватися персоналом, лише мала частка якого є ліцензованими інженерами з програмного забезпечення.

За оцінками Стіва Макконнелла [4], менш ніж 5% практикуючих сьогодні програмістів зрештою потрібно буде одержати ліцензію інженера з програмного забезпечення, і ця частка може легко наблизитися до одного відсотка.

Чи можливе взагалі адекватне ліцензування інженерів з програмного забезпечення? Деякі експерти з комп'ютерних наук

стверджують, що ліцензування буде неефективним або навіть контрпродуктивним. Ці заяви робляться дуже часто, щоб залишити їх без розгляду.

Суть деяких з них зводиться до того, що ліцензування в інженерії програмного забезпечення неможливе або практично безрозсудне, а інших – до того, що ідея ліцензування погана сама по собі.

Ось короткий перелік аргументів на користь непрактичності або неможливості ліцензування до фахівців з програмної інженерії [4]:

• Для інженерії програмного забезпечення не визначений загальноприйнятий обсяг знань.

• Знання в галузі інженерії програмного забезпечення міняються так швидко, що екзаменаційні матеріали застаріють до моменту іспиту.

• Важко втиснути скільки-небудь осмислений тест з інженерії програмного забезпечення у формат вибору відповіді з кількох варіантів. Фактично ніяка методика, заснована на іспитах, не зможе адекватно засвідчити компетентність програмного інженера.

• Спектр піддисциплін, що залучаються до розробки програм, настільки широкий, що спроби ліцензувати їх усіх не мають практичного сенсу.

• Іспит з основ інженерії для ліцензування працюючих інженерів не годиться для тих, хто здобув освіту за фахом «комп'ютерна наука».

Розглянемо кожний з цих аргументів.

Для інженерії програмного забезпечення не визначений загальноприйнятий обсяг знань.

Якщо років тридцять тому це твердження, можливо, і було схоже на правду, то сьогодні воно хибне. Як зазначалося в розділі 4, наочний обсяг знань у галузі інженерії програмного забезпечення добре визначений і достатньо стабільний.

Знання в галузі інженерії програмного забезпечення змінюються так швидко, що екзаменаційні матеріали застаріють до моменту іспиту.

Це твердження також засноване на застарілому розумінні наочного обсягу знань програмної інженерії, описаної в розділі 4.

Знання в деяких інших сферах (особливо в медицині) змінюються, принаймні, так само швидко, як і у сфері програмного забезпечення. Якщо можна ліцензувати лікарів, можна ліцензувати й програмних інженерів.

Важко втиснути скільки-небудь осмислений тест з інженерії програмного забезпечення у формат вибору відповіді з кількох варіантів. Фактично ніяка методика, заснована на іспитах, не зможе адекватно засвідчити компетентність програмного інженера.

Формування іспиту професійного рівня дійсно серйозна проблема. Проте наука і методика розробки іспитів на професійну придатність досить добре розвинені. На іспитах заснована процедура ліцензування в багатьох професіях: від медицини і права до теорії страхування й інших дисциплін. Розробка іспиту вимагає багато часу, сил, а також залучення справжніх знавців своєї справи. Розробка іспиту для сертифікованого фахівця з розробки програм – це серйозне завдання, але не більше ніж у будь-якій іншій професії.

Спектр піддисциплін, що залучаються до розробки програм, настільки широкий, що спроби ліцензувати їх усіх не мають практичного сенсу.

Різноманітність стилів розробки програм дійсно є серйозною трудністю для ліцензування програмних інженерів. На щастя, справа спрощується, оскільки, як і в інших дисциплінах, більшості інженерів немає потреби одержувати ліцензію. Ліцензувати треба лише тих, хто розробляє програмне забезпечення, що становить ризик для здоров'я або добробуту населення.

Питання про «широту піддисциплін» не унікальне. Лікарі складають окремі іспити з кардіології, радіології, онкології та інших спеціальностей. Інженери складають спеціалізовані іспити з цивільного будівництва, електротехніки, хімії тощо. Якщо в інших сферах цю перешкоду змогли подолати, то її буде подолано і в галузі програмного забезпечення.

Зважившись на всі ці міркування, можна зробити висновок, що цей аргумент швидше говорить не проти ліцензування, а на користь реалістичних очікувань результатів, яких можна досягти ліцензуванням. У інших інженерних дисциплінах діє кодекс поведінки, згідно з яким фахівці не повинні практикувати поза сферою своєї компетенції. Інженерія програмного забезпечення потребує аналогічного стандарту.

Екзамен з основ інженерії для ліцензування працюючих інженерів не годиться для тих, хто здобув освіту за фахом «комп'ютерна наука».

Екзамен з інженерних основ сьогодні зосереджений на основних положеннях інженерії, включаючи структуру і властивості інженерних матеріалів, динаміку та управління фізичними системами, електротехніку та електроніку тощо. Як мовилося в розділі 4, деякі навчальні програми з програмної інженерії справді вимагають від студентів вивчення цих традиційних інженерних предметів. Інші програми більше сконцентровані на програмному забезпеченні та менше на традиційних дисциплінах, тому студенти, випущені за такими програмами, не підготовлені для іспиту з інженерних основ.

Не викликає сумнівів, що належить виконати величезну роботу, перш ніж ліцензування з програмної інженерії стане реальністю. Але зараз ця робота багато в чому виконана, оскільки вона проводиться вже кілька років. На думку переважної більшості фахівців, усі суперечки щодо практичної користі ліцензування мають лише непряме значення; суть питання в тому, чи правильна сама ідея ліцензування.

Ось перелік аргументів на користь того, що ліцензування – це погана ідея:

• Ліцензії штучно скоротять кількість працівників, які могли б займатися інженерією програмного забезпечення, коли попит на програмних інженерів росте.

• Інженер одержує безстрокову ліцензію, що недоцільно в умовах швидкозмінного обсягу знань програмної інженерії.

• Ліцензування не зможе гарантувати компетентність кожного власника ліцензії. Ліцензування дасть суспільству помилкове відчуття захищеності.

Розберемо тепер кожний з цих аргументів.

Ліцензії штучно скоротять кількість працівників, які могли б займатися інженерією програмного забезпечення, коли попит на програмних інженерів росте.

Це твердження засноване на припущенні, що після впровадження ліцензування лише ті інженери, які мають ліцензію з розробки програмного забезпечення, зможуть його створювати. Як уже наголошувалося в даному розділі, в інших сферах ліцензування працює не так, не за цим правилом воно працюватиме і в галузі програмного забезпечення. Більшість працівників, які пишуть програми, будуть фахівцями з технологій або неліцензованими програмними інженерами, а не професійними інженерами. Більшість типів програм не містять ризику й можуть бути написані непрофесійними програмними інженерами.

Лише небагатьом програмним інженерам, які створюють певні типи систем програмного забезпечення, критично важливих для безпеки людей, узагалі потрібно буде мати ліцензію.

Інженер одержує безстрокову ліцензію, що недоцільно в умовах швидкозмінного обсягу знань програмної інженерії.

Це твердження засноване на неправильному розумінні двох аспектів ліцензування в програмній інженерії: по-перше, швидкості змін наочної ділянки знань і, по-друге, припущення, що ліцензія видається безстроково. Одна з умов ліцензування взагалі полягає в безперервності професійної освіти, що дає можливість залишатися в курсі всіх змін (у тому числі й для того, щоб зберегти ліцензію). Ліцензування стимулює актуальність знань професіоналів.

Ліцензування не зможе гарантувати компетентність кожного власника ліцензії. Ліцензування дасть суспільству помилкове відчуття захищеності.

Твердження про те, що хтось з некваліфікованих претендентів може дістати ліцензію, тоді як у ній буде відмовлено більш

кваліфікованим, частково справедливе. Ліцензування діє як фільтр, що відсіває гірших і підвищує рівень кваліфікації працівників, проте цей механізм не абсолютно бездоганний.

Без професійного ліцензування дорога буде відкрита не тільки надійним методикам розробки програм, але і непридатним, і навіть потенційно небезпечним. Щоб захистити інтереси суспільства, механізм ліцензування повинен працювати як фільтр, відсіваючи гірших і видаючи ліцензії тільки кращим розробникам програмного забезпечення.

Буде правильно припустити, що ліцензування не стане ідеальним фільтром. Всі ми чули про хороших і поганих лікарів, адвокатів і представників інших професій. Навіть у поєднанні з університетським дипломом, іспитами і практичним досвідом ліцензування у сфері програмної інженерії не буде виглядати краще, ніж в існуючих професіях. На початковому етапі, скоріш за все, процедура ліцензування буде навіть у гіршому стані, оскільки в інших професіях іспити та решта вимог ліцензування корегувалися й уточнювалися десятиліттями. Як відомо, навіть кращі сучасні підходи до ліцензування у сфері програмної інженерії можуть залишити лазівку для некваліфікованих працівників, дозволяючи їм проникати в галузь, і відсікти тих, кому дійсно треба було видати ліцензію.

Ідеальне ліцензування, напевно, недосяжне, але реальне ліцензування все ж таки має свою цінність. Гарантії добрі тоді, коли вони є. Коли їх немає, то краще більша впевненість, ніж менша.

Рух за ліцензування розробників програмного забезпечення по-справжньому розпочався у 1998 році, коли Рада професійних інженерів Техасу (США) прийняла програмну інженерію як окремо ліцензовану дисципліну з присвоєнням звання професійного інженера (або *P.E.* – *Professional Engineer*) фахівцям у галузі програмного забезпечення.

Замість загальновикористовуваного іспиту в Техасі почали ліцензувати інженерів-розробників програм згідно з «правилом допуску до іспиту». Щоб одержати ліцензію *P.E.*, перш ніж приступити до іспиту, претендент повинен був пред'явити:

документи, що підтверджують 16 років практичного інженерного досвіду або 12 років інженерної практики, і диплом бакалавра за атестованою університетською програмою. Крім цього кожен претендент повинен був представити принаймні 9 рекомендацій, не менше п'яти з яких повинні були дати ліцензовані інженери (необов'язково в галузі розробки програмного забезпечення).

Ті ж самі критерії допуску до іспиту застосовуються в Техасі й до інших інженерних спеціальностей.

Близько 50 практикуючих розробників програм змогли одержати ліцензію професійного програмного інженера згідно з встановленою процедурою ліцензування в Техасі в середині 2003 р., що не дуже багато. Природним було би бажання послабити строгість відбору на початковому етапі ліцензування, щоб більшість діючих інженерів автоматично підпадали під вимоги. Підсумком цього стала б девальвація звання «професійний програмний інженер» до рівня, що відповідає звичайному програмісту-початківцю. Посиливши правила, в Техасі максимально збільшили ймовірність того, що програмні інженери в штаті належатимуть до когорти кращих розробників програмного забезпечення, і захистили в такий спосіб їх репутацію.

Аналогічні події мали місце і в Канаді. Провінції Британська Колумбія й Онтаріо почали видавати ліцензії професійних програмних інженерів у 1999 р., і в Онтаріо ліцензії одержали близько 300 професіоналів. Як і в Техасі, програми ліцензування в обох провінціях передбачали успішне проходження деякої системи тестів. Але для допуску до процедури проходження потрібно заздалегідь надати свідоцтва свого практичного досвіду і професіоналізму.

Один з наслідків володіння ліцензією професійного інженера полягає в особистій відповідальності за роботи, що виконуються для компанії, тобто за авторство. Суди в США ухвалили, що тільки члени професійних співтовариств можуть визнаватися винними в злочинній халатності. Лікарі, адвокати і архітектори можуть бути визнані винними. Водії сміттєзбирачів, кухарі та комп'ютерні програмісти не можуть, тому що юридично не

вважаються професіоналами. Закріпивши за програмною інженерією статус професії, відкриваються можливості для судового переслідування програмних інженерів, які можуть бути обвинувачені в злочинній халатності так само, як і професіонали з інших галузей.

Окремий інженер не зобов'язаний одержувати ліцензію, але від деяких компаній потрібно буде мати в штаті ліцензованих інженерів. Найімовірніше, що це будуть компанії:

• які пропонують послуги програмної інженерії населенню;

• які виконують розробку програм для державних агентств;

• які виконують розробку програм, критично важливих для безпеки людей.

Інші компанії можуть за власним бажанням залучати професійних інженерів, щоб скористатися перевагою «блискучої упаковки», яка забезпечується репутацією кращих фахівців галузі, або для того, щоб зміцнити свій інженерний персонал. Наявність у штаті розробників програмного забезпечення, які одержали сертифікати, але не професійний статус в інженерії, може служити інтересам компаній також досить добре.

Професійні інженери в таких компаніях проводитимуть ревізію розробки і ставити «знак якості» на програмне забезпечення, яке випускається. Залучення ліцензованих інженерів стане життєво необхідним для компаній. Як і в інших інженерних галузях, компанія, що наймає професійного інженера, оплачує страховку його особистої відповідальності, що зводить до мінімуму витрати процесу отримання необхідної ліцензії.

Професійні інженери одержать й інші переваги. Ті з них, хто ставить свій підпис і репутацію під загрозу ризику заради своєї компанії, зрештою отримають від неї право вибору методик, підходів до проектування і рішень, що впливають на якість програмного забезпечення, за які їм доведеться відповідати. Не маючи статусу професіонала, програмному інженеру важко буде протистояти керівництву, яке може наполягти на ухваленні нереальних зобов'язань по термінах, зажадати закрити очі на вади проекту або пожертвувати якістю заради швидкого випуску програмного забезпечення. Якщо професія програмної інженерії

сформується зі всіма своїми атрибутами – освітою, кодексом професійної етики і ліцензуванням, то з'явиться можливість сказати: «Стандарти професії не дозволяють в даній ситуації нехтувати якістю. Можна втратити ліцензію або бути притягнутим до судової відповідальності». Створюються юридична і професійна основа для опору недалекоглядним менеджерам, маркетологам і клієнтам, яка абсолютно відсутня на сьогоднішній день.

На організаційному рівні можна буде спостерігати взаємну кореляцію між рейтингом компанії (рівнем зрілості) за SW-CMM, який обговорюється в розділі 7, та ліцензуванням програмних інженерів. Професійні інженери потенційно можуть бути притягненими до судової відповідальності за програмне забезпечення, написане під їх керівництвом. Вони не можуть особисто перевірити кожен рядок програми у великих проектах. Навіть якщо організації оплачують поліс страхування особистої відповідальності професійних інженерів, професійні інженери вважатимуть за краще працювати в тих організаціях, де їм буде забезпечена технічна і методична підтримка, тобто де найбільш розвинена організаційна інфраструктура розробки програмного забезпечення. Професійні інженери швидше за все зосередяться в організаціях з вищими рейтингами за SW-CMM. Це укріпить тенденцію, помічену Харланом Міллсом [13]: розробники програмного забезпечення, які мають професійний рівень вище середнього по галузі збираються в ефективних організаціях, розробники із рівнем нижче середнього – у неефективних.

Ключовим елементом будь-якого зрілого плану ліцензування є іспит на професійну придатність, що проводиться уповноваженим органом; проте його форма для програмного інженера поки не визначена. У інших інженерних галузях фахівці зазвичай складають восьмигодинний іспит, що включає рішення восьми задач, чотири з яких вимагають відповіді в описовій формі, а в чотирьох треба вибрати правильну відповідь з приблизно десяти варіантів. Конкретні особливості можуть змінюватися в різних територіальних утвореннях.

Самі по собі іспити не дають стовідсоткової гарантії, тому успішна здача іспиту програмного інженера недостатня для отримання ліцензії. Ліцензія професійного інженера традиційно вимагає як досвіду роботи, так і диплома зі ступенем атестованого учбового закладу для інженерів. У інженерії програмного забезпечення одержати подібний диплом проблематично, оскільки, наприклад, з десятка програм навчання програмній інженерії у США і Канаді, атестовані лише одиниці. Можна очікувати настання перехідного періоду (приблизно від 10 до 15 років) у ліцензуванні, коли диплом зі ступенем атестованого учбового закладу не буде обов'язковою вимогою, поки не сформується інфраструктура університетів, що забезпечує випуск достатньої кількості програмних інженерів.

Як уже мовилося у розділі 2, співтовариство розробників програм поки не дійшло єдиної думки про те, ким вважати фахівця з програмного забезпечення – інженером, який розробляє програми, або програмістом, який створює програми, застосовуючи інженерні підходи. На думку Стіва Макконнелла [4], відмінність між цими двома напрямами в розумінні програмної інженерії змушує припустити, що ліцензування зрештою розвиватиметься за одним з трьох можливих шляхів.

Перший шлях передбачає розробку спеціального іспиту з програмного забезпечення в рамках традиційного ліцензування інженерії. Інженери, що склали іспит з основ інженерії та володіють необхідним практичним досвідом, можуть одержати ліцензію *Professional Engineer* у галузі розробки програмних продуктів, склавши іспит за фахом «програмне забезпечення».

Другий шлях теж передбачає розробку іспиту за фахом програмного забезпечення, але вимагає ще і модифікації екзаменів з основ інженерії, які здають всі інженери (конкретний зміст іспитів змінюється залежно від територіальної юрисдикції). Сьогодні більшість інженерів активно користуються комп'ютерами, а багато з них створюють програми для себе або для своїх колег. Ці програми застосовуються для формування даних, необхідних при проектуванні мостів, будівель, нафтопереробних заводів і багатьох споруд, які потенційно

впливають на суспільний добробут. Інженери, що пишуть такі програми, повинні знати ефективні методики програмної інженерії. Можливо, потрібно буде просто переглянути зміст екзаменів з основ інженерії, збільшивши в них частку питань з інженерії програмного забезпечення. Іспити охоплюють досить широке коло проблем, і для їх складання, як правило, досить набрати близько 70% можливих балів, хоча висота бар'єра залежить від територіальної юрисдикції. Інженери, що пройшли підготовку з інших дисциплін, дадуть більше неправильних відповідей на питання з програмної інженерії, але правильно дадуть відповіді на більшу кількість питань з традиційної інженерії. Програмні інженери правильно дадуть відповіді на більшу кількість питань з програмного забезпечення, але дадуть більше хибних відповідей на традиційні питання з інженерії.

Третій шлях передбачає створення професійного статусу, який тісніше пов'язаний з програмним забезпеченням, аніж статус професійного інженера, наприклад «професійний програмний інженер – *Professional Software Engineer*» або що-небудь у цьому роді. Отримання цього звання було б орієнтоване виключно на програмне забезпечення й не вимагало б освіти у сфері термодинаміки або теплопровідності, знання матеріалознавства чи електротехніки або інших предметів традиційної інженерії. Один з аргументів на користь такого підходу полягає в тому, що фахівцям, що проектують системи бізнесу, фінансові додатки, освітні програми та інше програмне забезпечення, яке не використовується в інженерних цілях, не потрібні знання традиційних інженерних дисциплін, але корисно володіти інженерним підходом до створення програмного забезпечення.

На думку Стіва Макконнелла [4], з цих трьох можливих шляхів якнайкращий другий. Програмні інженери повинні вивчити достатній спектр суто інженерних предметів, щоб зрозуміти спосіб думки інженерів у традиційних галузях при проектуванні та розв'язанні задач, а третій шлях цього взагалі не вимагає. Але інженерам не треба вивчати всі інженерні дисципліни, щоб осягнути інженерний склад розуму, як цього вимагає перший шлях. Більше того, у такому разі дуже мало

програмних інженерів взагалі одержать ліцензії, а програма навчання інженерії програмного забезпечення повинна враховувати реалії: більшості дипломованих програмних інженерів не треба готуватися до іспиту з основ інженерії. Тих же програмних інженерів, які вирішать одержати ліцензію, другий шлях підготував би до складання іспиту з основ інженерії, не розбавляючи понадміру специфічний для програмної інженерії обсяг знань.

Чи буде корисним перегляд іспиту з основ інженерії для інших інженерних галузей? Скоріше за все, так. Зміна іспитів з основ інженерії не понизить статус інженерів, які у результаті одержать ліцензії. Вони, як і раніше, повинні будуть складати іспити з обраної інженерної спеціальності по закінченні свого практичного навчання: з цивільного будівництва, аерокосмічної, хімічної інженерії тощо. Етичний кодекс забороняє їм практикувати в інших інженерних галузях, де вони не є фахівцями. Крім того, програмне забезпечення настільки глибоко проникло в сучасну інженерію, що знайомство всіх інженерів з програмною інженерією може стати істотною перевагою: це дозволить їм розуміти проблеми, пов'язані зі створенням складних програмних систем, і усвідомлювати, коли необхідно удатися до експертної допомоги в проектах, що виходять за межі їх освіти і навичок.

У Канаді інженери, які одержують дипломи по закінченні навчання за атестованими інженерними програмами, під час випуску одержують сталеве кільце. З 1923 р. це кільце вручається на закритій церемонії, яку розробив Редьярд Кіплінг. Традиція свідчить, що ці колечка робляться зі сталі моста, який зруйнувався через інженерне недбальство та прорахунки, та інженери носять їх на робочій руці як нагадування про свою відповідальність перед суспільством. Були спроби пересадити цю традицію на американський ґрунт, але поки безуспішно.

Сталеве кільце важливе, оскільки, не позначаючи повного професійного звання, воно реально символізує належність до інженерії як професійного співтовариства. Сертифікація та ліцензування може зіграти схожу роль у галузі розробки

програмного забезпечення, символізуючи прихильність до вищих стандартів програмної інженерії.

Але водночас більшість програмних інженерів не стануть одержувати професійні ліцензії інженерів навіть після широкого розповсюдження ліцензування. Швидше за все, більшість не стане й сертифікуватися. Але у міру «дорослішання» програмної інженерії ліцензуванню і сертифікації надаватиметься дедалі більш важливе значення. Інженер програмного забезпечення, який хоче продемонструвати свою прихильність до професії, матиме можливість одержати ліцензію і / або сертифікат.

Контрольні запитання та завдання

1. Якими якостями та знаннями повинен володіти професійний програмний інженер?
2. Як можна розвинути навики програмного інженера?
3. Поясніть, як доступність інформації про хід розробки програмного проекту може підсилити згуртованість групи розробників.
4. Дайте визначення поняття «групова думка». Опишіть, які ускладнення можуть виникати в результаті цього явища і як їх можна уникнути.
5. Чому відкриті та спільні приміщення менш придатні для роботи команди програмістів, аніж індивідуальні кабінети? У яких випадках відкриті офіси виявляються більш підходящими?
6. Охарактеризуйте необхідність сертифікації та ліцензування професійних програмних інженерів. У чому відмінність між сертифікацією та ліцензуванням?
7. Обґрунтуйте необхідність безперервної освіти та підвищення кваліфікації програмних інженерів під час їх професійної практики.
8. Ознайомтеся із професійно-науковим журналом у галузі програмної інженерії (наприклад, «Advances in Software Engineering», «Open Software Engineering Journal», «International Journal of Computer Science» тощо). Опишіть у вигляді короткого реферату тематику публікацій обраного вами журналу. Наведіть анотації двох-трьох статей, які стосуються програмної інженерії.

РОЗДІЛ 7. ДІЯЛЬНІСТЬ КОМПАНІЙ, ПОВ'ЯЗАНИХ ІЗ РОЗРОБКОЮ ПРОГРАМНИХ ПРОДУКТІВ

Через два десятиліття, проведені в очікуванні зростання продуктивності та якості програмного забезпечення внаслідок застосування нових технологій і методик розробки, промислові та урядові організації почали усвідомлювати фундаментальну проблему, з якою вони зіткнулися: неможливість управління процесом розробки програмного забезпечення [9 – 15, 22 – 27]. Стало очевидним, що переваги, які виникли завдяки застосуванню якнайкращих інструментальних засобів і методів розробки, зводяться до нуля при роботі в рамках неорганізованого, хаотичного проекту. Багато організацій відзначають, що завершення проектів часто дуже запізнюється, а витрачений бюджет удвічі перевищує запланований. Як правило, подібні невдачі викликані тим, що організації не надають своїм групам розробників необхідної інфраструктури й підтримки.

Проте трапляється і так, що навіть у недисциплінованій організації окремі проекти дають чудові результати. Успішне завершення подібних проектів, як правило, вимагає героїчних зусиль з боку ентузіастів-розробників, на відміну від ітеративного повторення перевірених методів з боку організації, яка володіє зрілими виробничими процесами. За відсутності єдиного для всієї організації виробничого процесу повторення досягнутих результатів визначається виключно участю тих самих співробітників, які були задіяні в попередньому проекті. Отже, подібний успіх спричинений участю висококваліфікованих ентузіастів, а не наявністю в організації фундаменту, здатного забезпечити стійку, довготривалу продуктивність праці та безперервне поліпшення якості розробок. Досягнення ж останнього може відбутися тільки в результаті сфокусованих і безперервних зусиль, спрямованих на побудову інфраструктури процесів ефективної програмної інженерії та управління.

7.1. Особливості формування культури програмної інженерії в компаніях

Культуру складає множина цінностей, цілей і принципів, які керують діями, пріоритетами і рішеннями окремих осіб або групи, що працює у напрямі спільної мети [9, 10]. Культура групи розробників програмного продукту дуже сильно впливає на якість продукту, продуктивність розробників і робочу обстановку в групі.

Культура інженерії програмного забезпечення – це сукупність процесів, орієнтованих на якість у широкому сенсі, позицій розробників, людських відносин і технічних процесів. Кожна організація й особа має власну культуру, яка формується тільки шляхом копіткої роботи. Культура інженерії програмного забезпечення характеризується таким [10]:

- ясними організаційними цілями;
- зобов'язанням менеджменту вести організацію до досягнення постановленої мети;
- середовищем, яке дозволяє кожному розробнику вдосконалювати й ефективно застосовувати свої знання та навички;
- вимірюваннями, що уможливлюють відбир найбільш ефективних процесів.

Будь-яка «здорова» культура повинна включати три такі істотні компоненти:

- персональне зобов'язання кожного розробника створювати якісні продукти шляхом систематичного застосування передового досвіду інженерії програмного забезпечення;
- зобов'язання менеджерів усіх рівнів створювати середовище, в якому якість програмного забезпечення (у всіх його аспектах) є фундаментальною концепцією і кожен розробник може реалізовувати цю концепцію;
- зобов'язання всіх членів організації постійно вдосконалювати процеси, в яких вони беруть участь, і продукти, які вони створюють.

На рис. 7.1 показано, як культура інженерії програмного забезпечення пов'язана з метою, пріоритетами і технічною

практикою розробки програмного забезпечення. Культура інженерії програмного забезпечення визначає якісний рівень дій, процесів і технічних можливостей організації, задає проектні цілі й можливості організації. Рівень проектних цілей залежить від того технічного досвіду, який накопичений в організації. Чим багатший цей досвід, тим вище проектні цілі та тим більш відповідальне програмне забезпечення здатна створювати організація. Технічний досвід, зі свого боку, очевидно, впливає на загальну культуру організації, і чим він багатший, тим вище культура (А). Культура організації є, з одного боку, основою для діяльності кожного розробника, а з іншого – буде тим вищою, чим культурніший кожен розробник (В). Нарешті культура інженерії програмного забезпечення допомагає встановлювати пріоритети менеджменту. Очевидно, вищі пріоритети менеджменту визначають (підсилюють) загальну культуру організації (С).

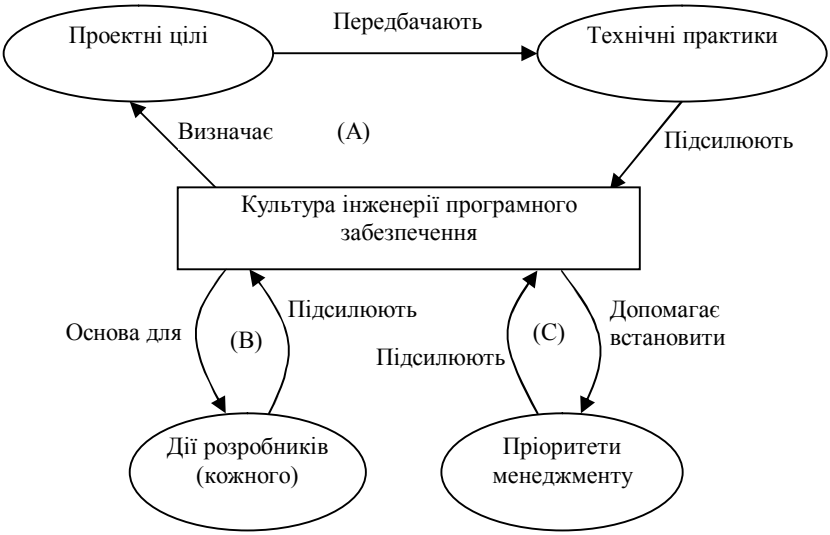

Рис. 7.1 Зв’язок культури з організацією [10]

Сьогодні відомі кілька моделей культур організацій. Найвідоміші з них такі [23]:

- Константіноса (Constantinos L.);
- Де Грака (DeGrace P.).

Модель Константіноса розглядає чотири організаційні парадигми:

- закрита – характеризується стабільністю, секретністю, незначною гнучкістю, низхідним процесом ухвалення рішень, неіноваційністю, авторитарністю;
- відкрита – характеризується інноваційністю, співпрацею, здатністю до переговорів, адаптивністю, колективним ухваленням рішень, розв'язанням проблем;
- синхронна – характеризується гармонійністю, рівністю, ефективністю, консерватизмом у змінах, координованістю, непрактичністю керівництва;
- випадкова – характеризується незалежністю ініціатив, творчістю і винахідливістю, індивідуалістичністю, нестабільністю, автономністю, неформальністю.

Модель Де Грака розглядає дві організаційні парадигми, які називаються римським і грецьким шляхами розвитку організацій (табл. 7.1). Де Грак указує, що римський шлях характерний для старих, великих і консервативних організацій, а грецький – для нових, середніх і маленьких мобільних організацій.

Таблиця 7.1

ОРГАНІЗАЦІЙНІ ПАРАДИГМИ ЗА МОДЕЛЛЮ ДЕ ГРАКА

Римський шлях	**Грецький шлях**
- структурні методи	- об'єктно-орієнтовані методи
- більше формалізмів	- менше формалізмів
- крупні програми	- менше програм
- CASE-інструменти	- окремі інструменти
- строго керовані проекти	- частково керовані проекти
- максимальна документація	- мінімальна документація
- управляти проектами	- писати програми

Р. Глас (R. Glass) додає «варварську» культуру для найменш цивілізованих маленьких організацій, що займаються розробкою невеличких програмних продуктів, порівнюючи її з грецькою і римською культурами в такий спосіб:

- греки організовують речі, римляни людей, варвари організовують «що-небудь»;
- у греків методології неформальні, у римлян – формальні, у варварів – відсутні;
- греки пишуть програми, римляни управляють проектами, варвари «стрибають» кодуючи;
- греки мотивуються проблемою «під рукою», римляни – груповими цілями, варвари – героями;
- греки мінімізують документацію, римляни максимізують її, варвари пишаються її відсутністю;
- греки працюють у маленьких групах, римляни – у великих, варвари – поодинці;
- греки використовують речі як інструменти, римляни – людей, варвари взагалі не використовують інструментів;
- греки демократи, римляни – імперіалісти, варвари – анархісти;
- греки – емпірично-індуктивні, римляни – аналітично-дедуктивні, варвари – не думають, емоційні;
- греки – інтуїтивні, римляни – логіки, варвари – імпульсні;
- греки приділяють увагу субстанції, римляни – формі, варвари – лініям коду;
- греки роблять речі, римляни планують речі, варвари руйнують речі.

Поняття «культура» на додаток до створення програм включає ряд певних рис і понять, що впливають на процес розробки програмного забезпечення. Культура формується перш за все принципами і діями керівної ланки організації. З часом культура росте або приходить у занепад залежно від минулих успіхів і невдач групи розробників програмного забезпечення, реакції на виникаючі проблеми і здатності виконувати поставлені завдання. Ці чинники врешті-решт визначають самооцінку групи. Вона може бути будь-якою: учасники групи можуть вважати себе

фахівцями вищого пілотажу, що створюють чудові продукти, або страждальниками, що ледве справляються зі своєю роботою.

Чому культура така важлива?

Як індивідууми, так команда розробників програмного забезпечення мають мати уявлення про призначення, важливість, рівень майстерності та здатність виконати поставлене перед нею завдання. Якщо самооцінка команди висока, її мотивація і працездатність достатні, щоб з високою продуктивністю праці створювати високоякісні продукти. І навпаки, якщо самооцінка низька, команда працюватиме нижче за свої здібності, навіть якщо вона складається з талановитих людей. У певному значенні культура може як підвищувати, так і знижувати загальну працездатність колективу.

Висока культура розробки програмного забезпечення, зокрема, має величезне значення при освоєнні нових можливостей або для виходу зі скрутної ситуації. Якщо в компанії розвинена культура створення якісних продуктів, є надія, що внутрішнє прагнення і прихильність до цієї практики не вичерпаються і в майбутньому.

З іншого боку, низька культура лише ускладнює і без того важку ситуацію. Припустимо, команда, в якій ніколи не виховувалася культура швидкої реакції на зовнішні події, опинилася в ситуації, що вимагає негайних дій. У цьому випадку успіх спроби швидко відреагувати на подію видається досить сумнівним. Такій групі, можливо, навіть не варто і намагатися зробити це.

Як виховати корпоративну культуру?

Кращий спосіб створити й виховати корпоративну культуру – почати з дотримання певних принципів і культивування деяких манер поведінки, після чого група якийсь час повинна працювати за новими правилами. Дії групи відповідно до цих принципів і будуть чинниками, що формують її культуру.

Виховуючи корпоративну культуру організації з розробки програмного забезпечення, можна вдатися до конкретних дій [5].

Чітко визначити, що собою являє організація і які цілі вона переслідує. Наприклад, співробітники компанії прагнуть створювати кращі у світі засоби розробки, зокрема інструменти,

які допомагають розробникам налагоджувати програми та усувати в них недоліки. Співробітники також знають, що будь-які методи досягнення цієї мети автоматично потрапляють в їх поле зору. Отже, вони повинні розібратися в цих методах, перейняти і застосовувати їх краще, ніж хто-небудь інший.

Культивувати елітарність. Строгий відбір при прийомі на роботу дозволяє багатьом вважати привілеєм належність до технічних фахівців компанії. Культивування оточення, побудованого на привілеях, дозволяє підтримувати високий бойовий дух і знижувати плинність кадрів. Елітарність дозволяє виховати такі риси культури, як прихильність до загальної справи, прагнення робити помітні речі та брати участь у розв'язанні нестандартних задач.

Відзначати успіхи і пам'ятати свою історію. З часом у будь-якій компанії формується історія випуску чудових продуктів, успішних як у технічному, так і в економічному відношеннях. «Батьки-засновники» таких продуктів можуть постійно наводити як приклад ці успіхи на зборах і неформальних зустрічах. Усі можуть стежити за тим, щоб кожен новачок знав історію і шлях розвитку компанії. Це стимулює формування культури прагнення до успіху, створення чудових продуктів і змушує бути експертом у вибраній компанією області. Ці принципи мають велике значення, бо історія компанії впливає на формування культури. Обов'язково слід відзначати успіхи й переказувати історію компанії новим працівникам.

Підтримувати дух суперництва. Ставлячи мету перед колективом, можна чітко визначати правила змагання і ставитися до поставлених цілей як до ворогів, після чого, «ущільнивши ряди», «атакувати їх». Мету змагань можна дуже чітко визначити під час корпоративних обговорень і роздумів. Команда розробників програмного забезпечення повинна зробити все можливе і неможливе, щоб добитися успіху, і сформувати, таким чином, чудову культуру суперництва.

Організувати свої способи відпочинку та веселого проведення часу. У працівників технічних компаній є два задоволення: можливість від душі повеселитися і різні способи відпочити від

повсякденних турбот. У компанію можна придбати купу всіляких дрібниць: футболок, рекламних матеріалів з виставок, безкоштовних обідів, крім того, можна організувати саморобні майданчики для гольфу і велосипедні доріжки прямо в приміщеннях компанії. Все це формує атмосферу винятковості та веселості. Зрозуміло, що це все дуже нетипово, але практика показує, що багатьом колективам IT-компаній це подобається.

Не відгороджувати співробітників від клієнтів. У багатьох успішних компаніях щиро переконані, що співробітників не слід «ховати» від клієнтів. Наприклад, можна попросити їх бути присутніми біля стенду компанії на якій-небудь виставці. Там вони зможуть не тільки відпочити від роботи, але і потраплять на передній край спілкування з клієнтами, зможуть відповісти на їх питання, одержати нові ідеї та почути про створені ними продукти як хороше, так і погане. Такі виставки піднімають бойовий дух розробників, оскільки кожний з них одержує відомості про потреби клієнтів з перших вуст. У такий спосіб вдається налагодити живий зв'язок між тими, хто створює програми, і тими, хто ними користується.

Отже, перший етап формування культури організації з розробки програмного забезпечення – визначення пріоритетних цінностей. Які цінності закладені в культурі компанії в даний момент? Які цінності повинні бути втілені в майбутньому? Чи можна сказати, що команда розробників:

- цінує виконану вчасно роботу;
- цінує технічну перевагу;
- цінує високу якість;
- цінує навіть мінімальний внесок;
- заохочує виняткову продуктивність праці;
- проявляє благородство;
- небайдужа до соціальних проблем;
- вважає, що кожен повинен брати участь у тестуванні продукту;
- оперативно реагує на зовнішні загрози;
- вважає, що тестування практичності програми має вирішальне значення;

• вважає наднормову роботу звичайною справою при відставанні від графіка?

Позитивні відповіді на всі ці питання – перша ознака здорової культури, яка сформувалася в організації.

Наступний етап – вибір дій, що формують корпоративну культуру. Які перешкоди необхідно подолати при цьому, які рішення необхідно ухвалити, яких проблем торкнутися і на які конфронтації піти заради виховання необхідної культури? Менеджмент організації повинен постійно думати про це і бути готовим послідовно втілювати ухвалені рішення.

Інший аспект культури полягає у використанні та освоєнні внутрішніх технологічних прийомів [5]. Характерною рисою деяких культур є наявність множини технологічних прийомів розробки, тоді як у інших їх майже немає. Молоді компанії часто неохоче освоюють нові технологічні прийоми, тоді як у більших організаціях без них не обходиться жодне завдання. У міру зростання групи слід подумати про те, як вона збираєтеся освоювати нові технологічні прийоми. Які типи прийомів необхідні, а без яких можна обійтися? Як не вдатися до крайнощів, пустивши всю роботу на самоплив або виснаживши себе різними правилами і положеннями?

Ось деякі правила, яких корисно дотримуватися при зростанні організації [5]:

• *Зважувати витрати на впровадження технологічних прийомів і користь від них*. Необхідно відразу розпізнавати ситуації, коли додавання нового прийому не принесе користі. Користь від кожного додаткового етапу або процесу повинна окупати витрати на його впровадження. Деколи це важко з'ясувати відразу, і в сумнівних випадках краще відмовитися від впровадження нововведення.

Якщо менеджмент компанії відчуває, що новий процес необхідний, важливо знати, як його впровадити. Нові технологічні прийоми приносять якнайкращі результати, коли вони чітко визначені, а їх впровадження обіцяє очевидну вигоду (забезпечуючи реальні та вимірювані результати) і не зустрічає

опору колективу. Якщо новий прийом відповідає цим критеріям, великий шанс, що його можна буде з успіхом використовувати.

З іншого боку, якщо прийом описаний туманно, не має очевидної цінності або команда не сприймає його, слід ще раз запитати: а чи потрібен він? Це не означає, що слід зовсім відмовитися від його впровадження, просто потрібно бути упевненим, що вигода від впровадження переважить витрати. Також необхідно проаналізувати проблеми, які можуть зустрітися на шляху впровадження нового прийому. Не слід поспішати з додаванням нових прийомів, але як тільки зникли останні сумніви в їх необхідності, слід без коливань приступати до впровадження.

• *Перш ніж упроваджувати нові прийоми, потрібно витягнути максимум з наявних*. Один із найбільш бентежачих аспектів розробки програмного забезпечення – впровадження нових прийомів, тоді як команда не повністю використовує всі можливості тих, які є у групи розробників. У команді зі здоровою культурою так бути не повинно. За правильне використання наявних прийомів відповідають менеджери і провідні фахівці. Можна втратити довіру учасників команди, санкціонуючи додавання нових технологічних прийомів і ігноруючи ті, що вже є. Якщо виявиться, що жоден з наявних документованих прийомів не використовується повністю, необхідно скликати збори групи та вирішити, чи становлять вони цінність. Якщо так – необхідно залишити їх та використовувати «на всі сто». Якщо немає – необхідно позбутися зайвих технологічних прийомів. Який сенс тримати прийоми, розвісивши їх по стінах, як картини? Вони тільки псують інтер'єр. Необхідно запровадити мінімальний набір технологічних прийомів, необхідний для виконання роботи, і стежити за тим, щоб група освоїла їх досконало. Не слід освоювати нові прийоми, якщо група розробників та менеджмент не впевнені, що вони будуть задіяні повною мірою.

• *Зважувати потреби окремого фахівця і всієї команди*. Зазвичай нові прийоми освоюються, щоб задовольнити потреби всієї команди. Проте деколи нові прийоми вводяться лише для

того, щоб полегшити життя кільком людям або навіть одній людині. У цих випадках треба ретельно проаналізувати всі витрати і вигоди. Якщо для впровадження цього прийому доведеться просити групу зробити щось понад заплановане з істотними витратами, потрібно бути упевненим, що користь для небагатьох учасників групи варта витрат, на які доведеться піти іншим. Якщо вони несумірні, від нового прийому слід відмовитися, інакше треба роз'яснити причини всім, кого торкнеться впровадження нового прийому і кому доведеться вкладати в нього свій час і сили, щоб забезпечити його стовідсоткове використання.

7.2. Організаційні структури та технології компаній, які займаються розробкою програмного забезпечення

Організаційні структури компаній, які займаються програмними проектами, – найбільш відповідна проекту тимчасова організаційна структура, що включає всіх його учасників і створена для успішного досягнення мети проекту [10].

Існують і використовуються різні організаційні структури. Нижче розглянуті найбільш популярні. Перш ніж описувати конкретні структури, слід визначити основні використовувані терміни. Таких термінів три: функція, роль, посада.

Функція – це вид діяльності, що виконується в ході проекту. Виконання кожної функції вимагає наявності певної специфічної кваліфікації і здібностей.

Наведемо приклади функцій:

• *Адміністрування*. Ведення договорів; розмови із замовником; складання зовнішніх формальних документів, доповіді начальству.

• *Проектування*. Складання паперових і/або електронних концепцій, моделей, специфікацій та планів.

• *Кодування*. Ручна і/або напівавтоматична генерація коду мовою програмування. Автономне (поблочне) налагодження коду. Малювання і тестування інтерфейсних елементів (форм).

• *Тестування*. Пробне використання додатка з метою зламати його, а не виконати завдання.

• *Супровід*. Розгортання, навчання користувачів, адміністрування розробленого або встановленого програмного забезпечення в процесі дослідної експлуатації.

Роль – це тимчасове призначення співробітнику набору функцій у рамках конкретного проекту.

Отримання ролі означає делегування повноважень для виконання певних функцій і ухвалення відповідальності за результати виконання цих функцій. Роль може вимагати виконання різних функцій; деякі функції можуть бути властиві кільком ролям. Визначальною ознакою ролі є не характер діяльності, а набір конкретних результатів, відповідальність за досягнення яких накладає роль.

У проектах різних типів використовуються різні набори ролей, наприклад:

• *Аналітик*. Функції: планування, адміністрування.

• *Адміністратор*. Функції: адміністрування, планування.

• *Програміст*. Функції: проектування, кодування, тестування, супровід.

• *Тестер*. Функції: тестування.

• *Експлуатаційник*. Функції: супровід, тестування.

Посада – це сертифікована здатність відігравати певні ролі та виконувати певні функції.

На відміну від ролі та функції, посада не прив'язана до конкретного проекту і носить постійний характер за часом.

У більшості організацій передбачається щонайменше три рівні ієрархії посад:

• начальник (відділу, підрозділу);

• керівник (групи, проекту, напряму);

• виконавець (інженер, технічний фахівець).

Розглянемо найбільш відомі моделі організаційних структур компаній, які займаються розробкою програмного забезпечення [9 – 15, 22 – 25].

Ієрархічна модель (або модель дерева субординації (див. рис. 7.2)) є найпоширенішою і найвідомішою організаційною моделлю.

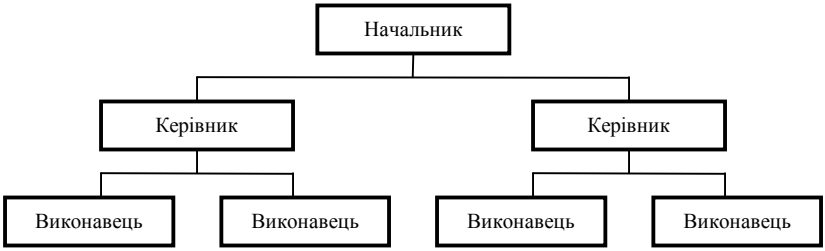

Рис. 7.2. Ієрархічна модель команди (дерево)

Ця модель має ряд переваг:

• *Єдиноначальність*. Принцип єдиноначальності забезпечує дуже високий ступінь надійності, стійкості та керованості команди. У критичних ситуаціях завжди використовується саме ця модель.

• *Популярність*. Ієрархічна модель звична і відома абсолютно всім. Її використання не вимагає додаткових заходів щодо впровадження і навчання персоналу.

• *Масштабованість*. Ієрархічна модель володіє високим ступенем масштабованості. Вона з успіхом застосовується в масштабах від десятків до десятків мільйонів виконавців.

Водночас ієрархічній моделі властиві деякі принципові недоліки:

• Ієрархічна модель екстенсивна. Нарощування функціональності забезпечується збільшенням складу.

• Ієрархічна модель жорстка, тобто практично не допускає перебудови «на ходу» (у процесі). Вона орієнтована на виконання строго визначених функцій. Зміна функцій вимагає хворобливої перебудови дерева субординації.

• Ієрархічна модель консервативна. При її використанні існує тенденція до жорсткого закріплення за кожним виконавцем його ролевої функції. Вона погано пристосована для швидкої зміни технологій і парадигм.

• Ієрархічна модель не стійка по відношенню до особистих якостей керівників. Негативні особисті якості керівників негативно впливають на ефективність команди, причому цей

вплив непропорційно збільшується зі зростанням рівня в дереві субординації.

Модель «Бригада головного програміста». Модель бригади головного програміста з'явилася під час першої технологічної революції в програмуванні на рубежі 1960–1970-х років. Довгий час модель головного програміста (модель «хірургічної бригади», або «модель зірки» (див. рис. 7.3)) була домінуючою моделлю при розробці програмного забезпечення.

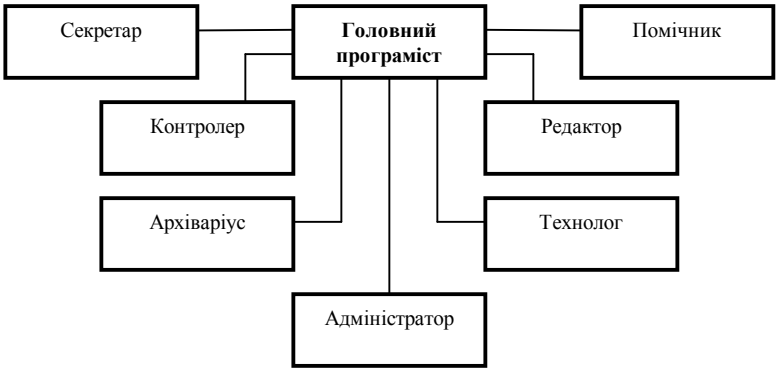

Рис. 7.3. Модель бригади головного програміста (зірка)

У цій моделі головний програміст виконує весь проект сам, а інші члени бригади асистують у рамках своїх ролей і функцій.

Модель головного програміста має такі переваги:

• *Передбачуваність*. Бригада головного програміста володіє високою передбачуваністю. Якщо головний програміст поганий, то це виявляється на ранніх стадіях проекту. Проект може бути припинений або реорганізований практично без збитків. Якщо головний програміст хороший, то вірогідність успішного завершення проекту висока навіть за наявності серйозних зовнішніх чинників ризику.

• *Автономність*. Бригада головного програміста володіє високою автономністю. Вона успішно функціонує навіть за умов змінного та несприятливого зовнішнього середовища.

• *Гнучкість*. Бригада головного програміста володіє достатньою функціональною гнучкістю. За рахунок зміни набору «променів у зірки» її легко можна орієнтувати на різні типи програмних проектів.

• *Єдиноначальність*. Бригада головного програміста успадковує всі переваги принципу єдиноначальності (оскільки головний програміст особисто ухвалює всі принципові рішення, які стосуються проекту).

Метод бригади головного програміста допускає різні модифікації при збереженні своєї суті.

Перша модифікація: зміна кількості та якості «променів у зірки». У практичних випадках кількість променів скорочують, наприклад, об'єднуючи секретаря, редактора та архіваріуса. Характеристичними ролями в бригаді є головний програміст, помічник головного програміста та адміністратор, решта ролей міняється у міру розвитку технологій програмування.

Друга модифікація: на роль головного програміста призначається не кодувальник, а наприклад, аналітик або менеджер проекту. В цьому випадку кодування веде помічник головного програміста, але всі принципові рішення ухвалює головний програміст. У сучасних умовах спостерігається тенденція до переміщення ухвалення ключових рішень зі стадії кодування на ранішні стадії, тому друга модифікація фактично стандартна.

Третя модифікація: «здвоєний центр». Ця модифікація дозволяє хоч би в деяких межах масштабувати бригаду. Є два головні програмісти, які ділять проект навпіл, всі рішення ухвалюють консенсусом і є помічниками один для одного.

При використанні більшої кількості головних програмістів модель перестає працювати, оскільки комунікації, досягнення консенсусу і взаємне дублювання роботи вимагає дуже великих накладних витрат.

Модель бригади головного програміста має певні недоліки.

• Бригада головного програміста не є масштабованим рішенням. Вона відмінно працює на проектах у складі 6–8 чоловік протягом 1–2 років. Якщо проект вимагає коротших

термінів або істотно більших обсягів, то використання бригади головного програміста утруднене.

Якщо формально розрізати значний проект на кілька частин і запустити кілька бригад паралельно, то результати їх роботи буде дуже важко синхронізувати та інтегрувати в один додаток. Річ у тому, що головний програміст оперує великою кількістю специфічної інформації, часто опускаючи етапи формального документування і специфікації. За рахунок цього підвищується продуктивність, але ускладнюється сумісність. Дуже короткий проект бригаді головного програміста важко провести тому, що головний програміст послідовно виконує всю основну роботу, і його особисті можливості обмежують продуктивність бригади.

• Бригада головного програміста не є розпаралелюваною структурою. Вона діє за принципом: один проект – одна команда. Практично неможливо виконувати бригадою одночасно різні фази різних проектів.

• Бригада головного програміста має вразливу центральну ланку. Дуже великий управлінський ризик миттєвого руйнування бригади, якщо щось трапляється з головним програмістом. Помічник програміста у бригаді головного програміста повинен ретельно відстежувати всі дії головного програміста. Це дещо знижує ризик руйнування.

Модель «Команда рівних». Модель команди рівних є складовою частиною Microsoft Solutions Framework (MSF) – методології розробки програмних проектів фірми Microsoft [8]. Це найбільш демократична модель, оскільки в ній немає явно виділеного центру. Схематично її прийнято зображати у вигляді циклу (рис. 7.4), де всі ролі рівноправні та пов’язані одна з одною.

Щоб підкреслити відсутність ієрархії в команді MSF, ролі в команді позначаються не назвами посад, а назвами функцій.

Переваги моделі команди MSF [8].

• Висока продуктивність, оскільки непродуктивні трудовитрати на підтримку формальних і субординаційних зв’язків зведені до мінімуму.

• Порівняно легка масштабованість. Кожен елемент у схемі команди може бути у свою чергу циклом.

• Сильна позитивна мотивація праці та рівномірно висока зацікавленість усіх учасників у кінцевому успіху.

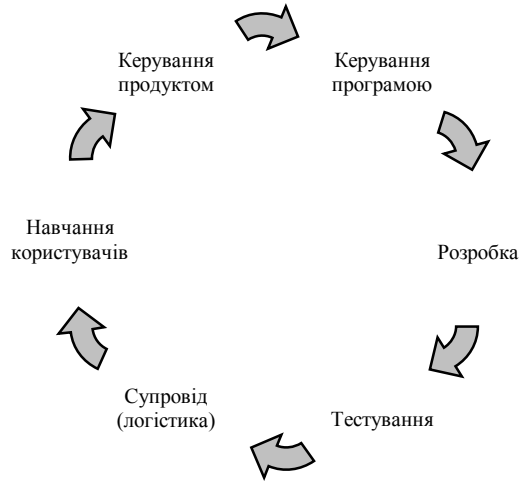

Рис. 7.4. Модель команди MSF (цикл)

Основні недоліки моделі MSF є продовженням її переваг.

• Для формування команди MSF потрібні рівні учасники (однаково кваліфіковані та однаково зацікавлені).

• Критичне значення має комунікабельність (велика частина комунікацій неформальні), уміння і готовність працювати в колективі (командний дух).

• Демократична модель команди MSF погано узгоджується з жорсткою ієрархічною моделлю підрозділу компанії.

Матриця відповідальності. Матриця розподілу відповідальності (Responsibility Assignment Matrix) – структура, яка ставить у відповідність організаційній структурі проекту структуру декомпозиції робіт для призначення відповідальних осіб за кожну роботу.

Наведене визначення повністю пояснює основне призначення матриці відповідальності. Для ефективної організації робіт за проектом необхідно точно визначити, хто за що відповідає і хто

що робить. Матриця розподілу відповідальності – це двовимірна таблиця, рядкам якої відповідають усі роботи проекту (узяті, наприклад, із структури декомпозиції робіт), а стовпцям – усі ролі, які використовуються у проекті. Якщо у проекті задіяні кілька співробітників, що відіграють одну й ту ж роль, то заводиться кілька стовпців. Іншими словами, в матриці розподілу відповідальності стільки рядків, скільки всього робіт у проекті, та стільки стовпців, скільки всього виконавців у проекті. Якщо якийсь співробітник, що відіграє певну роль, призначений відповідальним за виконання певної роботи, то на перетині відповідних рядка і стовпця робиться відмітка.

Матриця розподілу відповідальності є зручним інструментом управління. Вона дозволяє швидко і в наочній формі відповісти на важливі для менеджера питання.

• Хто відповідає за дану роботу?

• Чи є роботи, за які ніхто не відповідає (а отже, ніхто і не зробить)?

• Чи є роботи, за які відповідають кілька виконавців? (Це помилка планування).

• За що відповідає даний співробітник?

• Чи є співробітники, які ні за що не відповідають? (Їх можна вивести з проекту).

• Чи є співробітники, які відповідають за дуже велику кількість робіт? (Існує ризик, що вони не справляться зі всіма справами).

Матриця розподілу відповідальності є невід'ємним елементом плану проекту.

Розглянемо тепер основні принципи найбільш розповсюджених технологій розробки програмного забезпечення, які використовують компанії.

Для того, щоб зрозуміти, що таке технологія програмування, необхідно ознайомиться зі специфікою проблем, які виникають при розробці сучасного програмного забезпечення. Грубо кажучи, при розробці будь-якого програмного продукту необхідно повністю і у встановлені терміни виконати поставлене завдання, забезпечивши при цьому високий рівень якості. Дані вимоги можна поставити як до виконання лабораторної роботи з

програмування, так і до розробки сучасного програмного комплексу. Проте в останньому випадку розробник зіткнеться з розв'язанням задач, які, на перший погляд, не належать до дисципліни програмування (у вузькому сенсі). Такими завданнями є, наприклад, визначення і специфікація вимог, розробка архітектури системи, управління якістю продукту тощо. Саме для виконання подібних завдань і необхідна технологія програмування.

Технологія програмування не є набором абстрактних висновків і формальних методик. Використовувана технологія програмування пронизує всі фази життєвого циклу програмного продукту. Будь-яка дія в процесі розробки продукту виконується відповідно до використовуваної технології програмування.

Також важливо відзначити відмінність між методологією і технологією програмування. Методологія програмування є формою розпоряджень і норм, в яких фіксуються зміст і послідовність певних видів діяльності. Технологія програмування є сукупністю прийомів і способів виконання певних видів діяльності. Виходячи з вищенаведених визначень, ясно, що технологія програмування присутня при розробці будь-якого програмного продукту, навіть якщо це і не закріплено формальною методологією розробки.

Одним з ключових понять технології розробки програмного забезпечення, як і багатьох інших областей діяльності, є поняття проекту. Проект – унікальна тимчасова організована робота, спрямована на створення певного унікального продукту і/або послуги. Технологія управління проектом є сукупність знань, навичок, інструментів і методів для планування і реалізації дій, направлених на досягнення поставленої в рамках проекту мети.

Сучасні програмні системи розробляються у винятково складній обстановці. На шляху сучасного проекту з розробки програмного забезпечення постають численні організаційні та технічні перешкоди. Можна виділити такі змінні, які впливають на складність проекту з розробки програмного забезпечення [10]:

• Наявність висококваліфікованих фахівців на ринку праці. Причому чим новіша використовувана технологія, тим менша кількість фахівців, які їй володіють.

• Стабільність використовуваної технологічної платформи. Чим новіша використовувана платформа, тим менша її стабільність.

• Стабільність і функціональність використовуваних інструментів розробки. Чим новіший та потужніший використовуваний інструмент розробки, тим менше фахівців здатні з ним ефективно працювати і тим менш стабільна його функціональність.

• Ефективність використовуваних методів розробки, включаючи методи моделювання, проектування, тестування і управління версіями.

• Доступність фахівців, що володіють експертизою в прикладній області.

• Використовувана методологія та її відповідність даному проекту.

• Ситуація на ринку та її вплив на терміни проекту і заплановану функціональність продукту.

• Терміни і фінансування проекту.

• Множина інших організаційних і технічних змінних.

Можна припустити, що складність проекту є функцією від вищенаведених змінних. Важливо розуміти, що будь-яка з цих змінних може змінювати своє значення протягом життєвого циклу проекту. Наприклад, може істотно змінитися ситуація на ринку, що може привести до стиснення термінів і збільшення обсягу запланованих робіт.

Для управління проектами з високою складністю потрібні розвинені інструменти контролю й управління ризиками. Проте в управлінні проектами виділені такі проблеми [12]:

• Більшість процесів розробки некерована. Початкові дані та бажаний результат багатьох процесів розробки невідомі або визначені дуже нечітко. Більше того, процес досягнення бажаного результату не піддається формалізації. Прикладами некерованих

процесів розробки є розробка архітектури і вичерпне тестування продукту.

• Ідентифіковані процеси розробки супроводжуються невідомою кількістю неідентифікованих процесів розробки. Наприклад, у процесі моделювання й доказу адекватності моделі виникає множина процесів розробки, які насилу піддаються ідентифікації та формалізації.

• Вимоги до продукту часто змінюються протягом життєвого циклу проекту, що вимагає складної процедури зміни й узгодження вимог.

Спроби запропонувати формальну, деталізовану методологію розробки програмного забезпечення виявляються безуспішні, оскільки сам процес розробки не піддається деталізації та формалізації. Сліпе дотримання методологій, що передбачають керованість процесів розробки, призводить до неочікуваних результатів проекту.

Незважаючи на те, що процес розробки програмного забезпечення є погано визначеним і динамічним процесом, відомо кілька формальних і деталізованих методологій розробки програмного забезпечення. Розглянемо найбільш поширені традиційні методології розробки програмного забезпечення – водоспадну та спіральну, після чого проаналізуємо принципово інший підхід – так звані гнучкі технології розробки програмного забезпечення [22 – 25].

Водоспадна методологія є однією з перших запропонованих формальних методологій розробки програмного забезпечення. Схематично модель водоспаду може бути зображена, як показано на рис. 7.5 [24].

Хоча водоспадна методологія передбачає наявність невизначених процесів розробки, її лінійна природа не визначає способу реакції на непередбачені результати будь-якого з проміжних процесів розробки. Більше того, на кожному етапі проекту існує необхідність деякою мірою передбачати результати подальших етапів. Наприклад, на етапі планування необхідно передбачати результати всіх подальших етапів проекту, що часто виявляється практично нездійсненним через складність і

невизначеність процесів розробки програмного забезпечення. Це є основними проблемами водоспадної методології.

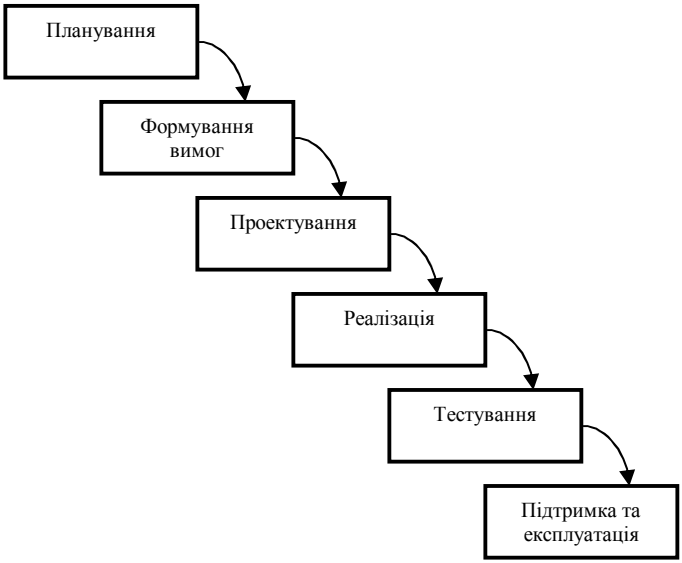

Рис. 7.5. Модель водоспаду [24]

Спіральна методологія розробки програмного забезпечення розв'язує основну проблему водоспадної методології. Кожна фаза водоспадного процесу розробки у спіральній методології завершується етапом прототипування та управління ризиками. Схематичне зображення спіральної моделі подано на рис. 7.6 [24].

Етап прототипування після кожної фази проекту дозволяє визначити, наскільки поточний стан проекту відповідає первинному плану. За підсумками прототипування виконується або перехід до наступної фази, або повернення на одну з попередніх фаз. Проте фази і послідовність фаз залишаються лінійними. Це, як і раніше, не цілком відповідає реальному стану речей, коли процеси розробки програмного забезпечення є невизначеними і непередбачуваними.

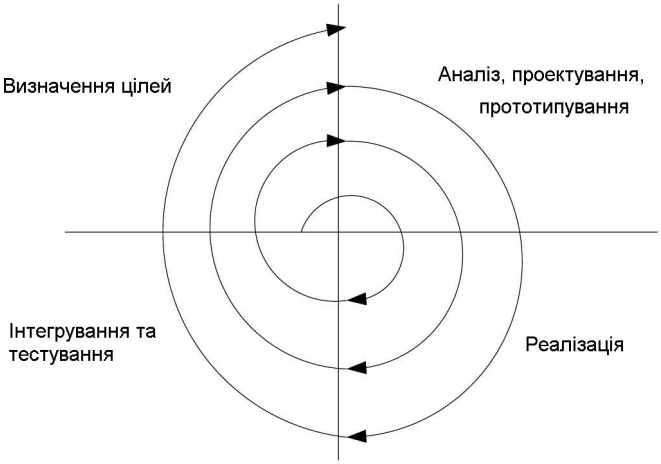

Рис. 7.6. Спіральна модель [24]

У ***гнучких технологіях*** розробки програмного забезпечення використовується принципово інший підхід до боротьби зі зростаючою складністю і непередбачуваністю проектів. Гнучкі технології передбачають максимальну гнучкість усіх процесів розробки за наявності необхідного рівня контролю над ходом проекту [22 – 25].

Як було показано вище, розробка програмного забезпечення стикається з високою складністю і непередбачуваністю процесів розробки. Гнучкі технології розробки програмного забезпечення мінімізують ризики завдяки розділенню процесу розробки на маленькі проміжки часу. Такий проміжок часу називається ітерацією і зазвичай займає від одного до чотирьох тижнів. Кожна ітерація може розглядатися як повноцінний проект з розробки програмного забезпечення. Так, ітерація може включати всі основні процеси розробки, такі як планування, аналіз вимог, проектування, реалізація, тестування й документування.

Зазвичай результатом ітерації не є продукт, готовий до виходу на ринок. Але метою кожної ітерації є отримання стабільної версії продукту. В кінці кожної ітерації відбувається переоцінка пріоритетів проекту, що значно скорочує ризики.

Маніфестом гнучких технологій розробки програмного забезпечення є такі основні принципи [23]:

• Особистості та взаємодії важливіші, ніж процеси та інструменти.

• Працююче програмне забезпечення важливіше, ніж всеосяжна документація.

• Співпраця із замовником важливіша, ніж переговори за контрактом.

• Адаптивність до змін важливіша, ніж дотримання плану.

Маніфест гнучких технологій заснований на таких основоположних міркуваннях [23]:

• Вищий пріоритет – це задоволення вимог замовника шляхом швидкого і безперервного постачання цінного та працездатного програмного забезпечення.

• Позитивно сприймаються вимоги, що змінюються, навіть на пізніх фазах розробки. Гнучкі технології розробки програмного забезпечення використовують зміни для підвищення конкурентоспроможності продукту.

• Працездатне програмне забезпечення поставляється якомога частіше, періодами від пари тижнів до пари місяців, з тяжінням до коротших інтервалів часу.

• Бізнесмени і розробники впродовж усього проекту щодня працюють спільно.

• Проекти будуються навколо мотивованих осіб. Цим особам висловлюється довіра і створюються всі необхідні умови роботи.

• Найбільш ефективним і дієвим способом передачі інформації (як усередині команди, так і зовні) є розмова «лицем до лиця».

• Основною мірою прогресу є працездатні програми.

• Гнучкі технології розробки програмного забезпечення встановлюють зручний режим ведення розробки. Спонсори, розробники і користувачі повинні бути здатні постійно дотримуватися такого режиму впродовж усього проекту.

• Безперервна увага до технічної досконалості та гарного дизайну підвищує гнучкість.

• Кращі архітектурні рішення, набори вимог і дизайни створюються самоорганізованими командами.

• Команда регулярно розглядає та упроваджує будь-які методи підвищення власної ефективності.

Гнучкі технології розробки програмного забезпечення представлені різноманітними методологіями розробки, як-от екстремальне програмування, Scrum, Feature Driven Development тощо. Проте всі гнучкі методології мають деякі загальні характеристики, такі як:

• ітеративна розробка;

• фокус на взаємодії та комунікації;

• повна або часткова відмова від створення дорогих проміжних артефактів проекту.

Застосовність гнучких технологій розробки може бути розглянута з кількох точок зору. З погляду створюваного продукту, гнучкі технології особливо добре застосовні у випадку, коли вимоги до продукту часто й несподівано змінюються. З іншого боку, існує думка, що гнучкі технології не застосовні для розробки систем, до яких висуваються високі вимоги надійності та безпеки.

З погляду організації, на застосовність гнучких технологій впливають такі чинники, як культура організації, склад колективу і сформованої комунікації. Основними чинниками, які забезпечують успіх впровадження гнучких технологій, є [23]:

• культура, що склалася в організації, повинна сприяти переговорам та ухваленню компромісних рішень;

• персоналу повинна виявлятися довіра з боку керівництва;

• висококваліфікований (але, можливо, менш численний) персонал;

• технічний персонал має право ухвалювати рішення, що роблять вплив на хід проектів;

• умови праці повинні сприяти інтенсивному спілкуванню членів команди.

З погляду розміру проекту, гнучкі технології добре застосовні на проектах невеликого і середнього розміру (до 40 чоловік). Використання гнучких технологій для реалізації великих проектів обмежене тим чинником, що у випадку великої команди достатньо складно організувати комунікацію «лицем до лиця».

Більшість гнучких методологій відображають загальний набір практик розробки, таких як: розробка через тестування, гра в планування тощо. Кожна конкретна гнучка методологія багато в чому спирається на різноманітні практики гнучкої розробки, без ефективного виконання яких неможливий успіх проекту [23].

Розробка через тестування є наріжним каменем гнучких технологій програмування. Використання більшості інших гнучких практик розробки програмного забезпечення (наприклад, рефакторингу або безперервної інтеграції) без використання розробки через тестування може виявитися невиправданим і навіть небезпечним. Основні принципи розробки через тестування:

• автоматичні тести пишуться для будь-якої частини програмного забезпечення, яка гіпотетично «може зламатися»;

• автоматичні тести пишуться безпосередньо перед написанням відповідного коду;

• уже існуючий код ніколи не змінюється без написання відповідних автоматичних тестів;

• виконується регулярний запуск усіх автоматичних тестів.

Стандарти кодування дозволяють упровадити такі практики, як парне програмування і колективне володіння кодом. Стандарти кодування, використовувані в гнучких технологіях розробки програмного забезпечення, задовольняють такі основні вимоги:

• стандарт кодування дозволяє спрощувати як написання, так і читання коду;

• стандарт кодування полегшує комунікацію між членами команди;

• стандарт кодування відносно добровільно сприймається всіма членами команди.

Парне програмування забезпечує високу якість усього коду, який створюється протягом гнучкого проекту. Під час парного програмування два програмісти працюють за одним комп'ютером. Грубо кажучи, один з програмістів зайнятий безпосереднім написанням коду, тоді як інший мислить стратегічно, обдумуючи такі питання, як працездатність

вибраного рішення, нетривіальні варіанти використання тощо. Обов'язки програмістів усередині пари і сам склад пар безперервно змінюються протягом проекту. Основні переваги парного програмування такі:

• збільшується дисципліна розробників.

• значно збільшується якість коду і ступінь його покриття автоматичними тестами;

• інформація про систему, яка розробляється, вільно розповсюджується всередині команди в усній формі.

Гра в планування дозволяє швидко сформувати приблизний план роботи і постійно оновлювати його у міру того, як умови завдання стають дедалі чіткішими. Основним артефактом гри в планування є набір паперових карток, на яких записані побажання замовника і приблизний план роботи з випуску наступної версії. Основним плюсом гри в планування є той факт, що замовник відповідає за ухвалення всіх рішень бізнесу, тоді як команда ухвалює всі технічні рішення. При цьому відповідальність розподіляється так.

Команда відповідає за:

• формування оцінки часу, необхідного для реалізації кожного з побажань;

• оцінку витрат, пов'язаних із використанням тієї або іншої технології;

• розподіл завдань між членами команди;

• оцінку ризиків, пов'язаних з реалізацією побажань;

• визначення порядку реалізації побажань в рамках поточної ітерації.

Замовник відповідає за:

• визначення набору побажань для чергової версії продукту;

• визначення дати завершення роботи над версією продукту;

• призначення пріоритетів для кожного з побажань.

Ці та інші практики гнучких технологій розробки програмного забезпечення формують складний і тісно взаємозв'язаний шаблон роботи, що дозволяє значно мінімізувати обсяг ризиків, пов'язаних з проектом з розробки програмного забезпечення.

7.3. Аналіз моделі зрілості можливостей та особливості сертифікації компаній за CMM

Процес створення програмного забезпечення – це фундаментальний компонент культури організації. Зрілість процесу створення програмного забезпечення визначається як ступінь, з яким можна стверджувати, що процес явно визначений, керований, вимірюваний, контрольований і ефективний.

На даний час відомо кілька моделей зрілості процесів (Capability Maturity Model, CMM), що відбуваються на підприємствах, які розробляють програмне забезпечення (табл. 7.2) [13, 14, 28].

Уперше модель зрілості для організацій з розробки програмного забезпечення, розроблена Інститутом програмної інженерії (SEI), була запропонована у 1987 р. і на даний момент реалізує найвідоміший і ефективніший підхід до систематичного вдосконалення організацій-розробників програм. У ній указується шлях, по якому зазвичай проходять компанії, щоб досягти успіху та бути конкурентоспроможними.

Таблиця 7.2

МОДЕЛІ ЗРІЛОСТІ ПРОЦЕСІВ

№ п/п	Модель	Опис
1	SW-CMM	CMM модель для програмного забезпечення
2	SA-CMM	CMM модель для придбання програмного забезпечення
3	P-CMM	CMM модель для оцінки персоналу
4	SE-CMM	CMM модель оцінки можливостей системної інженерії
5	SS-CMM	CMM модель захисту систем
6	IPD-CMM	CMM модель для розробки інтегрованих продуктів
7	CMMI	Інтегрована модель оцінки підприємств CMM

До найбільш уживаних моделей CMM належать чотири моделі: CMM – модель зрілості підприємств, SW-CMM (Software Capability Maturity Model) – модель зрілості програмного забезпечення, P-CMM (People-Capability Maturity Model) – модель зрілості персоналу, CMMI (CMM Integrated) – інтегрована модель зрілості підприємств.

Модель CMM забезпечує засоби для оцінки можливостей організації виготовляти якісне програмне забезпечення. Окрім цього, модель надає рекомендації, які повинні бути реалізовані в організації, щоб підвищити її можливості для виробництва якісного програмного забезпечення.

На рис. 7.7 показано процес сертифікації підприємств на рівень зрілості за CMM [13].

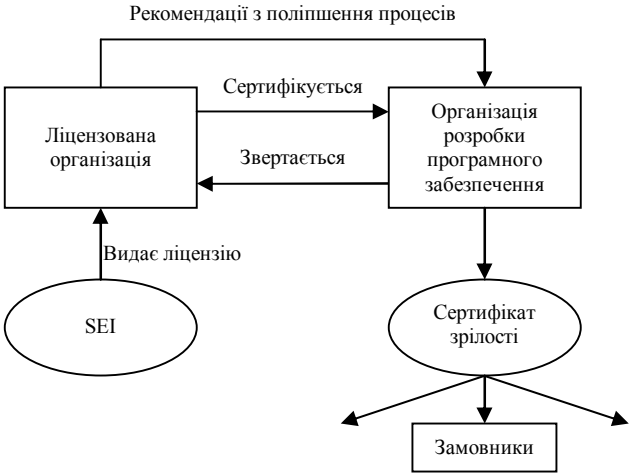

Рис. 7.7. Процес сертифікації [13]

Суть процесу сертифікації полягає в тому, що ліцензована організація одержує від інституту програмної інженерії (SEI) дозвіл (ліцензію) на проведення сертифікації за CMM. Організація-розробник програмного забезпечення звертається до ліцензованої організації для перевірки на зрілість на певному рівні CMM. Якщо організація-розробник програмного

забезпечення задовольняє вимоги рівня, то видається сертифікат зрілості. Інакше усуваються недоліки, вказані ліцензованою організацією.

На рис. 7.8 зображена схема моделі, яка визначає п’ять рівнів зрілості організації: початковий, повторюваний, сформований, керований, оптимізований [13].

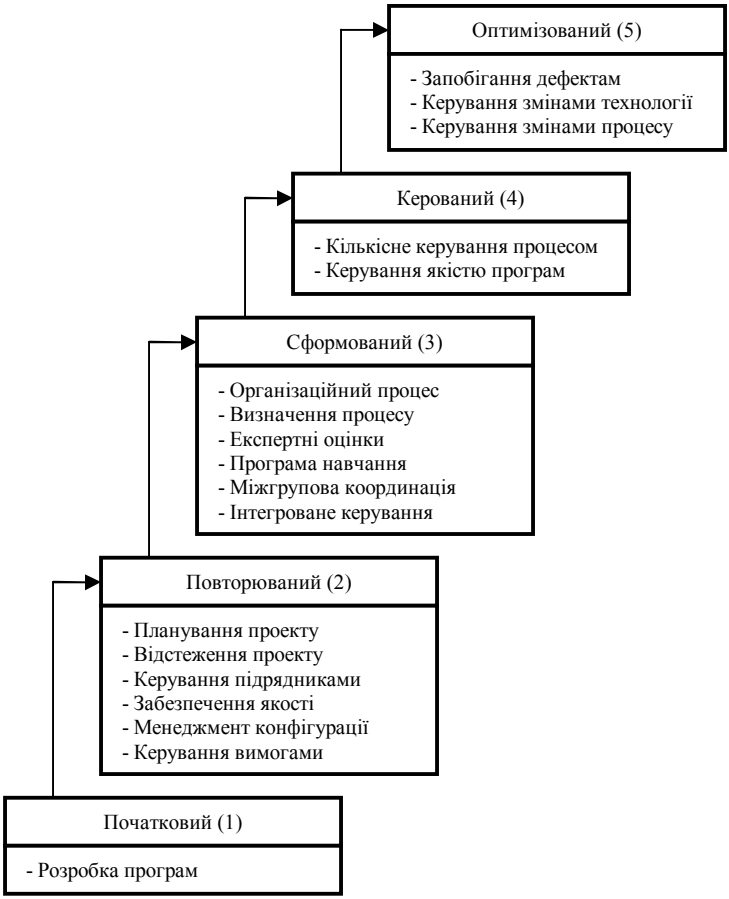

Рис. 7.8. Рівні та ключові області моделі CMM [13]

Кожен рівень має множину процесів (ключові області), що асоціюються з ним. Ці процеси визначають цілі, які повинні бути досягнуті організацією, щоб її зрілість відповідала даному рівню. Кожна мета досягається певними діями процесу. Організація може не виконувати всіх дій процесу, але вона повинна продемонструвати, що всі цілі ключових областей досягнуті на даному і всіх прилягаючих нижче рівнях. Пропуск рівнів у шкалі CMM не допускається, оскільки області нижніх рівнів забезпечують цілі областей вищих рівнів.

Відповідно до концепцій CMM, зрілість процесу в організації визначає її здібність до розробки і випуску високоякісних програмних продуктів.

Процес – це послідовність дій і завдань, які необхідно виконати для досягнення поставленої мети.

Здатність забезпечення якості складних програмних засобів – це діапазон результатів, які очікуються в результаті виконання даного технологічного процесу.

Взаємозв'язок рівнів зрілості, ключових областей процесу, ключових дій у межах цих областей і ключових індикаторів дій показаний на рис. 7.9 [13].

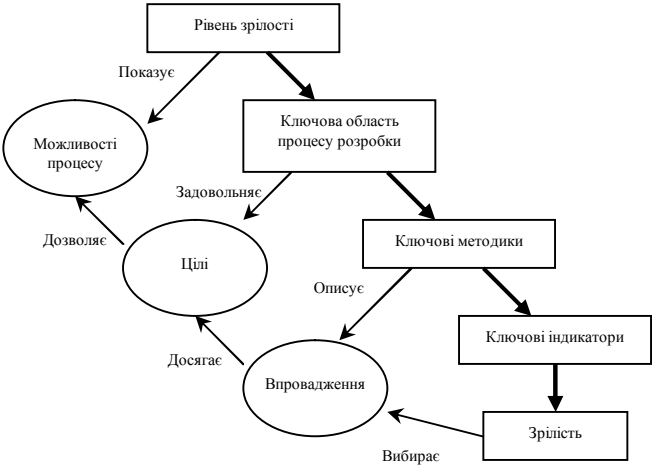

Рис. 7.9. Взаємозв'язки у моделі CMM [13]

Охарактеризуємо п’ять рівнів моделі CMM, за якими класифікуються організації-розробники програмного забезпечення [13, 14].

Рівень 1 – початковий. Процес розробки програмного забезпечення хаотичний. Проекти, як правило, реалізуються з перевищенням термінів і бюджету. Організаційними знаннями володіють лише окремі програмісти. З їх відходом зникають і відповідні знання. Успіх в основному залежить від внеску індивідуальних «майстрів». У таких фірмах процвітає метод «напишемо і виправимо». Розробники знаходяться на цьому рівні за умовчанням, якщо цілеспрямовано не упроваджують ефективніші підходи до розробки програмного забезпечення.

Рівень 2 – повторюваний. Базові методики практичного управління проектами визначаються за кожним окремим проектом, а організація забезпечує їх дотримання. Успіх проектів вже не залежить від окремих осіб. Сила організації, що знаходиться на цьому рівні, полягає у повторюваності досвіду, одержаного в аналогічних проектах. У таких організацій можливий збій при зустрічі з новими інструментами, методиками або типами програмного забезпечення.

Рівень 3 – сформований. На цьому рівні застосовуються стандартизовані технічні та управлінські процеси у всій організації, а індивідуальні проекти підлаштовують стандартні процеси під конкретні потреби. Спеціальна група відповідальна за роботи, що стосуються процесів розробки програмного забезпечення. Упроваджена програма навчання, що забезпечує необхідні знання та кваліфікацію менеджерів і технічного персоналу для роботи на даному рівні. Ці організації пішли далеко вперед від практики «напишемо і виправимо» і регулярно здають програмне забезпечення в строк, укладаючись при цьому в кошторис.

Рівень 4 – керований. Результати виконання проектів передбачені з високою точністю. Процес розробки достатньо сформувався, щоб виявляти причини відхилення і справлятися з ними. Організація накопичує інформацію за проектами в базі даних з метою оцінки ефективності різних процесів. У всіх

проектах дотримуються загальні для організації стандарти управління процесами, так що накопичувані дані можуть осмислено аналізуватися й порівнюватися.

Рівень 5 – оптимізований. Уся організація націлена на постійне реальне виявлення й розповсюдження методів удосконалення процесів. Процеси варіюються, результати змін оцінюються, і успішні зміни приймаються як нові стандарти. Основну увагу в частині забезпечення якості сфокусовано на попередженні дефектів за допомогою виявлення й усунення корінних причин їх виникнення.

Початковий принцип підходу CMM можна якоюсь мірою віднести до закону Конвея: Структура комп'ютерної програми відображає ієрархію організації, що створила її. У хаотичних організаціях народжується безладне програмне забезпечення. Організації, що вдаються до допомоги «програмістів-майстрів», що дають їм майже повну свободу заради створення чудес програмування, видають по черзі блискуче і недбале програмне забезпечення. Роздуті організації з неефективними процесами розробки видають неохайне й неврегульоване програмне забезпечення. Ну, а ефективні організації, які постійно оптимізують свої процеси, найбільш імовірно, створюють точно вивірене програмне забезпечення, яке задовольняє користувача.

З 1987 р. були оцінені можливості приблизно 2000 організацій, результати понад 10000 проектів були направлені в Інститут програмної інженерії SEI [14]. Три чверті організацій, що беруть участь у підвищенні рівня згідно із CMM, – це комерційні фірми-розробники або компактні групи розробників, що представляють різні галузі: фінанси, страхування, нерухомість, оптові продажі, будівництво, транспорт, зв'язок, ЖКГ, машинобудування, електронне устаткування, медичні інструменти та багато інших. Майже чверть усіх фірм виконували розробки за державними замовленнями. Близько 5 % представляли оборонні або урядові агентства. Розміри цих організацій змінювалися в широких межах. Майже половина мала в штаті менше 100 розробників. Персонал чверті організацій

перевищував 200 чоловік, і приблизно стільки ж організацій налічували менше 50 розробників.

Одне з найпоширеніших заперечень проти організаційного вдосконалення пов'язане з тим, що при цьому насаджується бюрократія, яка обмежує творчість. Це нагадує стародавнє заперечення, що мистецтво та інженерія несумісні. Звичайно, можна створити гнітючу обстановку, коли творчість і завдання бізнесу розходяться; так само можна побудувати й жахливі будівлі. Але також можливе створення гармонії між запитами розробника програмного забезпечення і завданнями бізнесу, що підтверджується галузевими даними. Серед тих, що взяли участь в опитуванні щодо впливу CMM на організаційні процеси, 84 % опитаних категорично не погодилися з твердженням, що вдосконалення в рамках моделі CMM зробило їх організації жорсткішими або бюрократичними [14].

Організації, націлені на організаційне вдосконалення, виявили, що ефективність практичних процесів стимулює творчість і високий моральний дух працівників. У опитуванні 50 організацій, проведеному у США у 1997 р., лише 20 % співробітників компаній рівня 1 охарактеризували моральний дух персоналу як «високий» або «відмінний». Наведені відповіді корелювали по групах менеджерів і розробників, що відповідають за організаційні удосконалення, і старшого технічного персоналу в цілому. У організаціях, яким був присвоєний рівень 2, частка співробітників, що оцінюють моральний дух своїх колег як «високий» або «відмінний», підскочила до 50 %. У організаціях рівня 3 вже 60 % опитаних оцінили так свій моральний дух [14].

Ці зведені статистичні дані підтверджуються глибшим аналізом організацій, що одержали вищі оцінки ефективності процесів розробки. Опитування, проведене в Центрі авіалогістики (м. Огден, США) – однієї з перших організацій, яка одержала рейтинг рівня 5 за CMM, – показав, що у співробітників сфери програмного забезпечення спостерігалося натхнення від змін, які виявилися результатом восьмирічних зусиль щодо вдосконалення організаційних процесів [14]. Респонденти дійсно вважали, що процеси рівня 5 звужують вибір методів виконання роботи, але

обмеження розглядалися як неминучий супутній елемент підвищення ефективності та не вважалися негативним чинником. Співробітники сфери програмного забезпечення визнали, що виконання робіт значно полегшилося за рахунок впровадження організаційних удосконалень. Переважна більшість співробітників визнали, що тепер вони роблять більший внесок у планування і контроль проектів. Кожен респондент стверджував, що ініціатива вдосконалення CMM зробила позитивний вплив.

Група розробників бортового програмного забезпечення для космічних кораблів «Шаттл» НАСА в Центрі космічних польотів імені Джонсона – ще одна організація, яка одержала рівень 5 за CMM. Там немає гір коробок з-під піци, пірамід банок з-під кока-коли, прямовисних стін для альпіністів, майданчиків для катання на роликових дошках або подібних елементів пейзажу «просунутих» організацій-розробників програмного забезпечення. Там не в іграшки грають, а прагнуть створювати бездоганний продукт. Робота цікава, але не всепоглинаюча. Група розробників програм працює зазвичай 5 днів на тиждень по 8 годин. У галузі з вираженим переважанням чоловічого персоналу майже половина групи розробників – жінки [14].

Люди, які йдуть з таких груп з високою продуктивністю, зазвичай здивовані неефективністю середніх організацій. В обстановці високого рівня зрілості співробітники не тільки не вважають її якось обмежуючою їх творчість, але досягають рівнів продуктивності та якості, просто неможливих для менш зрілих компаній.

Одна з не дуже привабливих особливостей організаційного вдосконалення полягає в тому, що його нелегко добитися. Джеймс Хербслеб у 1997 р. провів опитування керівників, які впровадили вдосконалення за CMM. 77 % опитаних заявили, що цей процес виявився тривалішим, ніж очікувалося, а 68 % сказали, що він коштував дорожче, ніж передбачалося [14].

Успіх удосконалення в CMM залежить від таких чинників:

• Самовіддача менеджерів вищої ланки, які повинні забезпечити керівництво і фінансування, присвоєння вищого пріоритету довгостроковим удосконаленням, активний

моніторинг динаміки вдосконалення процесу розробки програмного забезпечення.

• Утворення групи процесу програмної інженерії (Software Engineering Process Group, SEPG). У великій організації може потрібна не одна така група. До неї повинні увійти посадовці високого рівня, які усвідомлюють цілі вдосконалення процесів в організації та пов'язані з цим питання культури, а також і свою роль як внутрішніх консультантів.

• Необхідна підготовка і навчання керівників середньої ланки і технічного персоналу разом з винагородами за підвищення продуктивності, орієнтована на досягнення довгострокової мети CMM.

Цей список досить спрощений, і в кожній окремій організації знайдеться ще ряд особливих чинників, що впливають на її ініціативу організаційного вдосконалення. Деякі організації упроваджують CMM, ставлячись до цього як до «модного слівця». Спроби поставитися до CMM як до чергової панацеї навряд чи будуть успішні.

CMM – це ефективна модель організаційного вдосконалення. Структуруючи організації за п'ятьма рівнями, ми одержуємо також ефективний спосіб їх оцінки. У інших, більш сталих професіях оцінка організацій є звичайною частиною програми підтримки високих практичних стандартів. У США бухгалтерські фірми кожні три роки повинні проходити атестацію. Атестація коледжів дійсна максимум три роки, після чого вона переглядається. Деякі індивідуальні програми коледжів атестуються окремо. Лікарні проходять атестацію в Об'єднаній комісії з атестації організацій охорони здоров'я (Joint Commission on Accreditation of Healthcare Organizations – JCAHO) і одержують її максимум на три роки.

JCAHO вказує такі причини, за якими лікарні прагнуть пройти атестацію [4]:

• це підвищує довіру пацієнтів;

• дає можливість звітувати за спеціальною формою перед громадськістю;

• забезпечує можливість об'єктивної оцінки роботи організації;

• стимулює зусилля з підвищення якості роботи організації;

• сприяє залученню професійного персоналу;

• служить освітнім засобом для персоналу;

• може використовуватися для задоволення деяких вимог програми страхової медицини Medicare;

• прискорює розрахунки з третіми сторонами;

• часто забезпечує дотримання ліцензійних вимог штатів;

• може сприятливо вплинути на розмір премій по страхових відшкодуваннях;

• позитивно впливає на ухвалення рішень за договорами контрольованого медичного обслуговування.

Атестація лікарень – справа добровільна. Проте більшість лікарень прагнуть її одержати через вищеназвані причини.

Паралель з організаціями, які займаються розробкою програмного забезпечення, очевидна. Оцінка за CMM забезпечує форму звіту, яку потенційні клієнти можуть використовувати при ухваленні рішення щодо умов контракту на розробку програм або придбання його пакета. Це об'єктивний загальновизнаний стандарт для порівняння. Цим стимулюються зусилля організацій з підвищення якості, заохочуючи їх підвищувати свій рейтинг. Можна припустити, що страхові компанії запропонують вигідніші ставки страхових полісів компаніям, що мають достатньо високий рівень зрілості.

Стара приказка стверджує, що успіх дорівнює плануванню, помноженому на виконання.

Якщо планування і виконання виразити у відсотках, то вийде величина успіху від 0 до 100 %. Якщо один з компонентів – планування або виконання – відсутній, то успіх дорівнюватиме нулю.

У міру досягнення галуззю програмного забезпечення вищого рівня професіоналізму доводиться враховувати чимраз більший обсяг досвіду, накопиченого у зв'язку з CMM, як позитивного, так і негативного.

У CMM важливий зміст, а не форма. Організації, які прагнуть до вдосконалення за CMM виключно заради присвоєння рівня 2 або 3, цілком імовірно, обмежаться напівсирим плануванням і не

дуже ретельним виконанням. Навряд чи їм вдасться одержати бажані рейтинги й добитися виграшу, як і продуктивності, до яких вони прагнуть.

Ті ж організації, які націлені на виграш від кінцевої якості та продуктивності в рамках вдосконалення за моделлю CMM, ймовірно, серйозніше поставляться до планування і краще виконуватимуть плани. Увага до процесу розробки дозволяє організаціям підвищувати продуктивність, створювати програмне забезпечення з меншою кількістю дефектів, піти на більший ризик, підвищувати точність прогнозування, укріплювати моральний дух і поліпшувати виконання значних проектів.

Досягнення вищих рівнів зрілості виробничого процесу поступове й вимагає від організації довгострокових зобов'язань з безперервного вдосконалення процесів. На побудову фундаменту для безперервного вдосконалення виробничого процесу і впровадження відповідної культури організації-розробники можуть витратити більше 10 років. Хоча подібна десятирічна програма вдосконалення процесу здається чужою для більшості американських компаній, саме такий обсяг робіт потрібен для побудови зрілих організацій-розробників. Цей термін узгоджується з досвідом в інших галузях, таких як американська автомобільна промисловість, яка добилася значних досягнень у області зрілості виробничого процесу.

Модель CMM не є панацеєю і не включає всіх питань, значимих для успішних проектів. Наприклад, на даний час CMM не розглядає досвід у конкретних наочних областях, не пропагує конкретних технологій розробки і не містить рекомендацій з вибору, найму, мотивації і збереження компетентних співробітників. Ці питання життєво важливі для успіху проекту. Проте вони не були інтегровані в CMM. Модель CMM була спеціально розроблена створити впорядковану, дисципліновану структуру, усередині якої можна розв'язувати проблеми управління розробкою та інженерією процесу.

7.4. Особливості реалізації компаніями програм професійного розвитку своїх співробітників

Яким чином компанія може підтримати формування справжніх професіоналів в області програмного забезпечення? У другому розділі вже мовилося, що загальногалузева підтримка розвитку кар'єри професіональних програмістів поступово формується, але вона ще слабка. Лише кілька десятків університетів пропонують початкову освіту в галузі програмної інженерії, а випускників за цією спеціальністю набагато менше, ніж вимагається в галузі. Ситуація швидко поліпшується, але все ж таки пройдуть роки, поки кількість випускників університетів стане достатньою [4, 8].

Приватні компанії теж не досить активно підтримують ефективний розвиток кар'єри програмістів. Замість просування по кар'єрних сходах більшість працівників галузі програмного забезпечення просто переходять від одного проекту до іншого без якого-небудь поліпшення своїх навичок. Зовсім мало IT-компаній роблять слабкі спроби запропонувати кар'єрне зростання своїм співробітникам, а галузь у цілому не має у своєму розпорядженні нічого, що хоч би віддалено нагадувало професійне зростання, таке як у лікаря, адвоката, юриста або бухгалтера.

Розглянемо приклад програми професійного розвитку компанії Construx Software, яку організував та активно розвиває Стів Макконнелл [4], для якого професіоналізм з програмного забезпечення викликає великий інтерес. У 1998 році він розпочав роботи із формування чіткого порядку кар'єрного просування і підтримки професійного зростання програмних інженерів у компанії Construx. Перш за все були визначені конкретні цілі програми розвитку програмних інженерів:

• *Розширення професійних навичок.* Підвищення майстерності та кваліфікації співробітників компанії.

• *Планування кар'єри.* Створення послідовного плану підвищення кваліфікації програмних інженерів під час роботи в Construx.

• *Підтримка звичайних спеціальностей програмної інженерії.* Підтримка всього контингенту посад, пов'язаних із розробкою

програмного забезпечення, включаючи розробників, інженерів, тестерів, аналітиків бізнесу, менеджерів проектів, архітекторів програмного забезпечення і працівників інших галузевих спеціальностей.

• *Узгодженість.* Наявність узгоджених засобів оцінки роботи персоналу і підвищення на посаді незалежно від технічної спеціалізації.

• *Узагальнення для інших компаній.* Можливість пропонувати програми іншим компаніям, щоб підтримати професійне зростання їх фахівців у галузі програмного забезпечення.

Стрижень програми професійного розвитку компанії Construx Software – «сходи професійного розвитку» (Professional Development Ladder, PDL), засновані на виділених у SWEBOK сферах знань, описаних у розділі 4. У Construx ці сфери знань одержали назву «сфери знань Construx» (Construx Knowledge Areas, CKA). Вони визначають обсяг знань, яким повинен володіти і який повинен уміти застосовувати технічний персонал. Як описано в розділі 4, є десять сфер знань:

• Вимоги до програмного забезпечення;
• Проектування програмного забезпечення;
• Створення програмного забезпечення;
• Тестування програмного забезпечення;
• Обслуговування програмного забезпечення;
• Управління конфігурацією програмного забезпечення;
• Якість програмного забезпечення;
• Управління інженерією програмного забезпечення;
• Інструментарій і методики інженерії програмного забезпечення;
• Інженерний процес розробки програмного забезпечення.

Кожна з цих сфер знань у SWEBOK визначена до нюансів, але Стів Макконнелл із колегами дійшли висновку, що необхідні також конкретні інтерпретації для умов компанії Construx Software (табл. 7.3) [4].

Таблиця 7.3

ОПИС СФЕР ЗНАНЬ CONSTRUX (СКА)

Сфера знань (СКА)	**Опис**
Вимоги	Ідентифікація, аналіз, моделювання і документування функцій, що реалізуються у програмному забезпеченні
Проектування	Визначення структури і динамічного стану системи на різних рівнях абстракції та за допомогою різних зображень системи
Випробо-вування/ тестування	Роботи, пов'язані з прогонами програм, призначені для виявлення дефектів та оцінки функціональних можливостей
Обслуговування	Роботи, пов'язані зі встановленням, впровадженням, супроводом та експлуатацією системи
Управління конфігурацією	Організація та зберігання артефактів системи, контроль та керування змінами цих артефактів, визначення виду здачі системи замовнику
Якість	Роботи, які виконуються на статичних артефактах, пов'язані із забезпеченням вимог якості даного елемента програми в даний час або в майбутньому
Управління інженерією	Усі аспекти керування – від керування бізнесом та персоналом до керування проектами
Інструменти і методики інженерії	Застосування інструментарію, технологій, методологій та способів інженерії програмного забезпечення
Процес	Роботи, пов'язані з вимірюванням та підвищенням якості розробки програм, дотриманням строків, ефективністю, продуктивністю та іншими характеристиками проекту та продукту

Сфери знань Construx забезпечують упорядкований спосіб організації знань інженерії програмного забезпечення, але цього недостатньо для визначення здібностей програмного інженера. Тому всередині кожної області компанія Construx виділяє 4 рівні здібностей: початковий, розвинутий, провідний і майстровий. Ці 4 рівні позначають просування в придбанні знань і досвіду на кожній ділянці знань. У кожній сфері описані типи діяльності (читання, групові семінари і досвід роботи), необхідної для досягнення певного рівня здібностей. Ці чотири рівні наведені в табл. 7.4 [4].

Таблиця 7.4

ПОРІВНЯННЯ РІВНІВ ЗДІБНОСТЕЙ

Рівень здібностей	**Опис**
Початковий	Основні роботи в даній області співробітник виконує під контролем та здійснює серйозні кроки з розвитку своїх знань і майстерності
Розвинутий	Співробітник ефективно і незалежно виконує роботи в даній області, є прикладом для менш кваліфікованих співробітників, іноді виступає в ролі наставника
Провідний	Співробітник виконує показові роботи в даній області. Регулярно виступає наставником і ведучим у проекті та, можливо, на рівні компанії. Розглядається як значний ресурс у даній сфері знань
Майстровий	Співробітник є еталоном робот у даній галузі та має багатий досвід участі у багатьох проектах. Веде заняття і семінари, є автором брошур або книг, які розширюють обсяг знань у даній сфері. Є лідером на рівні галузі, визнаний експертом у ній

Добре відомо, що здібності залежать від поєднання досвіду і знань. Неможливо досягти провідного рівня знань інженерної дисципліни, якщо останні не базуються на досвіді. Але досвід такого ж рівня неможливий без володіння найбільш передовими знаннями. Тому при розбіжності між досвідом і знаннями співробітника загальний рівень здібностей зазвичай ближче до нижчого.

Поєднання сфер знань і рівня здібностей дозволяє сформувати ступені «сходів» професійного зростання. Ступені забезпечують підвищення професіоналізму і просування по службі. Сходження сходами вимагає від інженера як розширення знань (охоплення більшої кількості сфер знань), так і їх глибини (підвищення рівня знань у даних галузях). Потрібно набувати і знання, і досвід.

З причин історичного характеру, у компанії Construx ступені сходів мають номери від 9 до 15. Випускники коледжів зазвичай починають зі ступеня 9, а досвідчені інженери можуть почати з 10-го або 11-го. Дванадцятий ступінь вважається в Construx повноцінним професійним статусом. Багато інженерів вирішують не рухатися вище дванадцятого ступеня, оскільки ступенів з 13-го по 15-тий можна досягти, тільки зробивши істотний інноваційний внесок як у Construx, так і в інженерію програмного забезпечення [4].

Розглянемо характеристику та вимоги до освоєння кожного ступеня сходів професійного розвитку компанії Construx Software.

9-й ступінь. Інженер ступеня 9, як правило, щойно здобув освіту й починає освоювати принципи інженерії програмного забезпечення. Працює під постійним контролем.

10-й ступінь. Інженер 10-го ступеня має базове уявлення про інженерію програмного забезпечення. Як правило, недавно здобув освіту або має 1–2 роки досвіду роботи. Може виконувати роботи при обмеженому контролі.

11-й ступінь. Має тверді знання у програмній інженерії, може працювати самостійно. Працював у кількох завершених проектах і має досвід участі в кожному етапі циклу розробки програмного забезпечення, необхідного для випуску продукту.

12-й ступінь. Інженер 12-го ступеня успішно бере участь у всіх аспектах невеликих і значних проектів, роблячи вагомий внесок в їх ефективність. Зарекомендував себе грамотними технічними рішеннями і компетентністю в питаннях проекту. Вносить інновації, забезпечує стійкий рівень робіт, перевищуючи обсяг поставлених завдань. Зазвичай здійснює технічне керівництво інших співробітників або нагляд за ними.

13-й ступінь. Інженер на ступені 13 – справжній керівник, здатний аналізувати зовнішні та внутрішні сторони проекту і забезпечувати грамотні та правильні рішення. Повністю володіє всіма аспектами свого проекту і робить унікальний внесок у роботу. Рішення цього інженера приносять істотний прибуток компанії та забезпечують загальне стійке положення.

14-й ступінь. Інженер ступеня 14 – це вагомий інженерний ресурс для компанії. Справляється із серйозними проблемами, ухвалює рішення по ключових завданнях і структурі компанії. Відомий багатьом інженерам програмного забезпечення всередині та поза компанією завдяки конкретним досягненням, які реально сприяли розвитку програмної інженерії. Обсяг знань охоплює всю галузь. Досягнення цього ступеня вимагає присвятити свою діяльність інженерії програмного забезпечення, зокрема за межами Construx.

15-й ступінь. Інженер, незамінний для успіху компанії. Постійно розробляє й реалізує визначні продукти світового класу. Практикуючими інженерами всередині та поза компанією визнаний провідним експертом в інженерії програмного забезпечення. На нього покладається основна відповідальність за розробку методик компанії. Здійснює постійний і різноманітний внесок у роботу всієї галузі. Робота на цьому ступені вимагає присвятити всю діяльність програмній інженерії (і не тільки в Construx) і зумовлює визнання досягнень, які виходять за межі освоєної сфери знань.

Сходи професійного розвитку передбачають кілька шляхів кар'єрного зростання, даючи можливість співробітникам вибрати галузі знань, на яких вони мають намір концентруватися. Цим забезпечується гнучкість і структура, оскільки співробітник

вибирає свій кар'єрний шлях, керуючись вимогами рівня галузей знань. Наприклад, розробник може зосередитися на знаннях інструментарію і методики інженерії, проектування і створення програмного забезпечення. А може й навпаки, зосередитися на вдосконаленні в управлінні інженерією, процесі розробки і вимогах до програмного забезпечення. Націлений на якість інженер міг би вибрати для вдосконалення аналогічну сферу знань, процес розробки і тестування. Сходи професійного розвитку Construx є універсальним засобом визначення кар'єрного просування незалежно від вузької спеціалізації та орієнтації інженера.

Сходи розвитку містять чотири рівні здібностей, але майстровий рівень, за визначенням, не може бути сформульований, тому зазвичай сходи описуються матрицею 10×3: десять стовпців – сфери знань і три рядки – рівні здібностей. У кожній ділянці матриці визначені елементи, які повинні відповідати її вимогам, такі як читання, семінари і досвід роботи. Всього сходи професійного розвитку на даний момент містять близько тисячі конкретних вимог [4].

Області управління інженерією в Construx є гарним прикладом широти і глибини вимог до всіх рівнів здібностей – початкового, розвинутого і провідного. Розглянемо, наприклад, що слід прочитати і який практичний досвід накопичити, щоб вийти на початковий рівень знань з управління інженерією програмного забезпечення [4].

Перш за все з літературних джерел слід проаналізувати праці Чарльза Фішмана «They Write the Right Stuff», Стіва Макконнелла «Software Project Survival Guide», а також «Кодекс етики і професійної практики програмної інженерії». Наступні матеріали необхідно вивчити інформативно: книги Алана Дейвіса «201 Principles of Software Development», Іана Соммервілля «Software Engineering» (частина 1 і розділи 22–23), Роджера Прессмана «Software Engineering: A Practitioner's Approach» (частина 4). З досвіду роботи необхідно: розібрати план проекту, знати методики оцінки, планувати й відстежувати роботу співробітників. Що стосується участі в різноманітних заходах

галузевої діяльності, семінарах, конференціях, а також наявності сертифікату, то це необов'язково.

Розглянемо вимоги до розвинутого рівня знань з управління інженерією програмного забезпечення.

Із літератури необхідно проаналізувати такі праці: книги Фредеріка Брукса «No Silver Bullets – Essence and Accidents of Software Engineering», Демарко і Лістера «Programmer Performance and the Effects of the Workplace», Стіва Макконнелла «Rapid Development». Наступні матеріали необхідно вивчити інформативно: працю Уєйта Гіббса «Software's Chronic Crisis» і результати досліджень NASA, «Manager's Handbook for Software Development, Revision 1».

З досвіду роботи необхідно: брати участь як експерт артефактів управління проектом; брати участь у створенні статуту проекту; брати участь у розробці плану проекту; брати участь у розробці оцінки проекту; володіти методами індивідуальної оцінки від низу до верху; очолити оцінювальну роботу; володіти методами індивідуальної звітності за станом справ проекту; формувати щотижневий звіт стану проекту; володіти плануванням робіт (включаючи способи розділення роботи на частини, оцінку й управління напрацьованою вартістю).

Необхідно також брати активну участь у семінарах: «Довідник з управління виживанням проектів» (2 дні); «Швидка розробка програмного забезпечення» (2 дні); «Оцінка програмного забезпечення» (2 дні).

Перехід з розвинутого на провідний рівень галузі знань не так жорстко прописаний, як перехід з початкового на розвинутий. Докладні вимоги виробляються спільно співробітником, його куратором і керівником (менеджером). На провідному рівні, крім читання, семінарів і досвіду роботи, від співробітника може бути потрібно отримання визнаного галузевого кваліфікаційного сертифікату і участь у галузевій роботі як викладача на семінарах, доповідача на конференціях тощо.

Розглянемо більш детально дії та масштаб роботи, необхідні для такого переходу.

Матеріали для читання підбираються індивідуально. Цільова сфера і вибір конкретних матеріалів визначаються спільно з куратором і указуються в плані професійного розвитку. Для переходу з розвинутого на провідний рівень у кожній галузі знань Construx орієнтовно потрібно прочитати тисячі сторінок.

Відносно досвіду роботи, то необхідно, по-перше: очолювати планування, оцінку й відстеження роботи за значним проектом; виробити статут проекту з техніко-економічним обґрунтуванням; скласти план проекту для великого і невеликого проектів; створити робочий план значущого проекту; визначати цикл існування проекту, набір характеристик і планування; володіти формальними методами управління ризиками проекту; володіти формальними методами управління проблемами проекту; володіти методами збору ретроспективних даних. По-друге необхідно: володіти груповими методами оцінок (наприклад, розширеним методом дельфійського оракула), методами оцінки аналогії та параметричної оцінки; створити оцінку проекту зверху вниз на стадії формулювання проекту. І по-третє: створити план-графік робіт (або етапів) значущого проекту; створити докладний план-графік етапу; володіти методами критичного шляху і критичного ланцюжка для створення розкладів; володіти методами звітності щодо стану проекту. Брати участь у консультаційній роботі/інструктажі з управління інженерією програмного забезпечення.

Необхідно взяти участь у таких семінарах: «Ефективне управління проектами програмного забезпечення» (3 дні); «Управління ризиками» (2 дні); «Залучення до роботи сторонніх організацій для виконання проекту» (2 дні).

Необхідно також одержати свідоцтво сертифікованого професійного розробника комп'ютерного товариства IEEE та сертифікат професіонала управління проектами PMI.

На провідному рівні очікується істотний внесок співробітників у роботу Construx і потенційно в галузь у цілому. Приклади участі в роботі галузі:

• Розробити вечірній, недільний або інститутський курси в цій галузі; вести вечірні, недільні або інститутські семінари;

• Викладати на семінарах у Construx або університеті;

• Брати участь у галузевому комітеті, дискусійній групі, раді зі стандартів і т.п.;

• Зробити доповідь на конференції;

• Опублікувати статтю в провідному виданні або огляді;

• Опублікувати статтю у виданні другого ешелону;

• Відрецензувати рукопис книги;

• Рецензувати статті для IEEE Software або аналогічного видання;

• Брати активну участь в інструктажі та кураторстві інших співробітників Construx з обраної провідної галузі.

Версія 1.0 сходів професійного розвитку була упроваджена в Construx в 1998 р. і відразу після цього опублікована. Її еволюція знайшла віддзеркалення у версії 2.0, що вийшла на початку 2002 року для внутрішнього використання. За цей час співробітниками компанії зроблений ряд важливих висновків щодо впровадження і підтримки функціонування сходів професійного розвитку [4].

Для забезпечення успішного професійного розвитку в організації програма сходів повинна бути вбудована в її організаційну культуру. Співробітники Construx визначили різноманітні заходи для гарантій ухвалення програми керівництвом компанії та досягнення бажаних результатів.

Включення конкретних заходів посилення, упроваджених у Construx, не обов'язкове для успішної реалізації сходів професійного розвитку, але важливе для визначення структурних та культурних стимулюючих елементів, які спрацюють у даній організації.

Конкретні заходи стимулювання, упроваджені в Construx, включають нижченаведені чинники [4].

План професійного зростання (Professional Development Plan, PDP). Це механізм планування, відстеження й документування просування співробітника по сходах професійного розвитку. Кожен PDP включає коротко- і довгострокові цілі співробітника і конкретні дії та чинники (читання, семінарські заняття, досвід роботи та іншу професійну діяльність), які виконуються (формуються) в проміжках між звітними датами.

До цілей PDP практично завжди входить підвищення категорії персоналу на період від одного до п'яти років. План визначає роботу, яку повинен виконати співробітник, щоб добитися підвищення. На період більше року ця робота не формулюється в остаточному вигляді, але складається загальний план. Коли до підвищення залишається менше року, робота розписується детально на кожен місяць для куратора, співробітника і його менеджера. План дає об'єктивні й узгоджені критерії підвищення по службі для всієї компанії.

Програма кураторів. Завдання куратора – керівництво і підтримка просування інженера по сходах. Весь інженерний склад Construx розробляє й обговорює свої плани з кураторами. Інститут кураторів дозволяє сформувати сходи професійного розвитку для кожного окремого співробітника з орієнтацією на нього планів професійного зростання. Для виконання вимог, що ставляться до знань співробітника, куратор і співробітник зустрічаються 6–8 разів на рік і частіше, коли до підвищення співробітника залишається менше півроку.

Активна взаємодія менеджера і куратора абсолютно необхідна для виконання всіх дій, передбачених планом зростання. З метою підтримки цього процесу керівник підрозділу і куратор повинні підписати план професійного зростання співробітника. На зустрічах з куратором відстежується виконання цього плану і, якщо необхідно, корегується термін підвищення.

Однією з цілей інституту кураторів є вироблення в інженерів смаку до самостійності в подальшому професійному розвитку. Інженери на 12-й сходинці та вище повинні проявляти самостійність, і куратори їм не потрібні, якщо тільки вони спеціально не просять про їх призначення або збираються добиватися нового підвищення категорії.

Свідоцтва професійного розвитку. Construx відзначає різноманітні етапи і події в процесі професійного розвитку співробітників: підвищення категорії, отримання професійних сертифікатів, перший проект у ролі провідного спеціаліста, перші проведені заняття, перші опубліковані праці та інші значні події. Кожен розробник одержує пам'ятну пластинку з вигравійованим

написом, що позначає етапні досягнення. Цим наголошуються істотні події і підкреслюється важливе місце, яке Construx відводить професійному розвитку.

Програма навчання. Крім звичайного, поточного навчання, яке незримо відбувається при розробці програмного забезпечення, Construx виділяє 10–12 днів на рік для цілеспрямованих занять. Для фахівців з невеликим досвідом вони зазвичай проводяться у формі семінарів і конференцій. На ступені 12 і вище навчання може полягати в підготовці виступів на конференціях, участі в комітетах зі стандартів, організації спеціальних груп з окремих питань та іншої професійної діяльності.

Структура заробітної плати. Було встановлено, що традиційна система винагород за працю повинна бути перебудована, для сприяння цілям професійного розвитку; інакше завдання проекту превалювали б над завданнями розвитку. Підвищення на посаді, зарплата і трудові показники, засновані на механізмі сходів розвитку, дають можливість міцно закріпити за ним особливе місце в структурі організації.

Кожному ступеню сходів професійного розвитку відповідає в точності один рівень оплати праці, при цьому в Construx ступінь кожного співробітника і зарплата на кожній ступені є відкритою інформацією. Тому у співробітників є очевидний стимул перейти на вищий ступінь, оскільки такий перехід означає новий розмір зарплати.

Групи з інженерії програмного забезпечення. Ці групи (Software Engineering Discussion Groups, SEDG) є місцем для обговорення й обміну досвідом з інженерії програмного забезпечення. У них під керівництвом провідного фахівця групи обговорюються й аналізуються книги, що входять у перелік для читання в рамках сходів професійного розвитку. Такі групи є на ступенях 9, 10 і 11. Інженерам на ступені 12 рекомендується брати участь у роботі груп з метою поділитися досвідом і знаннями з колегами, що знаходяться на нижчих ступенях.

Визнання ступеня 12. Досягнення ступеня 12 – повного професійного статусу в Construx – є значною віхою кар'єри

інженера програмного забезпечення в компанії. В ознаменування цього досягнення Construx преміює співробітників, які добилися цього, в сумі різниці річної зарплати на ступенях 11 і 12, влаштовує прийом на їх честь, публікує інформацію в місцевій газеті ділових кіл і вивішує портрет у фойє будівлі компанії. Цим підкреслюється важливість для компанії професійного розвитку.

У зв'язку із вагомістю професійного визнання співробітників постає ще одне питання: як вбудувати досвідчених інженерів у сходи професійного розвитку? Виявилось, що і нових співробітників, що практикували в інших компаніях, також потрібно якось помістити на сходи розвитку. Багато хто з претендентів на посаді має значний досвід роботи в галузі, але не відповідає іншим вимогам сходів розвитку. Якщо оформляти їх на зарплату ступеня 10 або 11, то компанія не зможе конкурувати з іншими працедавцями. У практичному плані не вдалося би привернути старший персонал. Тому важливо було розробити механізм переведення нового співробітника на ступінь 12, не руйнуючи цілісності сходів розвитку і не зменшуючи досягнення інших співробітників Construx, які пройшли шлях до ступеня 12 у компанії.

Щоб виконати це завдання, був створений «перехідний ступінь 12». Досвідчений інженер, що приходить на роботу, відразу знаходиться на ступені 12, але протягом року або менше зобов'язаний заповнити відсутні елементи цього ступеня сходів – в основному інтенсивним читанням. Протягом даного року співробітник щомісячно зустрічається зі своїм куратором для обговорення виконаних ним робіт і всіх питань просування по сходах. Коли невистачаючі елементи заповнені, співробітник одержує офіційне визнання інженера ступеня 12.

П'ять років роботи в рамках версії 1 і 2 сходів професійного розвитку компанії дозволили реалізувати множину переваг [4].

• *Прискорений професійний розвиток*. Компанія досягла головної мети – підвищення професіоналізму інженерного персоналу, розвиток якого відбувався надзвичайно швидко. Під час співбесід з претендентами на заняття вакантних посад менеджери з набору персоналу зазвичай бачили, що внутрішні

стандарти в Construx для інженера з двох-трирічним стажем порівнянні з вимогами, що ставляться до найдосвідченішого персоналу в багатьох інших компаніях.

• *Командний потенціал*. Стандартизувавши сфери знань, компанія змогла одержати загальну основу практичного досвіду і знань. Це налагодило взаєморозуміння між співробітниками й забезпечило високу ефективність роботи в проектах в особливій ролі лідерів.

• *Добре зрозумілі критерії підвищення на посаді*. Було встановлено, що технічний персонал компанії високо оцінює відвертість механізму підвищення. Співробітники бачать, що можуть контролювати своє кар'єрне зростання і що перед ними дійсно відкривається перспектива, а не просто можливість одержати робоче місце.

• *Залучення персоналу*. Досвід роботи в різних сферах знань усередині компанії дозволяє безпосередньо оцінювати кандидатів на інженерні посади. Можна оцінити здібності кандидатів у кожній галузі знань. Оцінки кандидатів у проектах різними співробітниками компанії виявляються досить узгодженими.

• *Відмова в прийомі на роботу*. Розроблений механізм сходів розвитку вимагає певної підзвітності, що не дуже подобається деяким кандидатам для прийому на роботу. Коли в процесі пошуку кандидатам прямо вказується на вимоги професійного розвитку, це дозволяє відсіяти багатьох з тих, чиї наміри у зв'язку з програмною інженерією не можна вважати серйозними.

• *Управління кваліфікаційними навичками*. Матриця 10×3 сфери знань є природним структурованим методом відстеження можливостей і здібностей персоналу компанії.

• *Моральний стан і утримання співробітників*. Хоча сходи професійного розвитку є лише частиною всієї картини, її розробники переконані, що вона робить величезний внесок у підвищення морального духу і стабільності колективу. З 2003 по 2006 р. фірма Construx кожен рік ставала фіналістом конкурсу, що проводився виданням Washington CEO Magazine, на звання «Малого підприємства, де краще всього працювати». Протягом

більше трьох років жоден співробітник не пішов з компанії за власним бажанням.

Сходи професійного розвитку використовуються в Construx як гнучкий і структурований механізм підтримки розвитку кар'єри. Визначаючи й розвиваючи здібності в різних галузях знань, Construx забезпечує широту і глибину кваліфікації персоналу відповідно до інженерного підходу до розробки програмного забезпечення. Гнучкість механізму дає можливість персоналу компанії вибирати просування по сходах, що відповідає їх власним інтересам.

Construx висуває більш жорсткі вимоги до здібностей, ніж більшість компаній відповідно до їхньої корпоративної місії – «розвивати науку і мистецтво інженерії програмного забезпечення». Більшість інженерів компанії, що знаходяться на високих посадах, займаються консультуванням, викладанням, очолюють проекти, а це означає, що більша кількість співробітників компанії здатні вийти на провідний рівень у порівнянні з іншими компаніями.

Розробники сходів професійного розвитку Construx під керівництвом Стіва Макконнелла [4] дійшли висновку, що опора на галузі знань, сформульована в SWEBOK, дала їм істотні вигоди. Вибудовування сфер знань у вигляді матриці 10×3 означає, що сходи професійного розвитку можна легко перебудувати, додаючи або прибираючи рівні здібностей, області знань і спеціальні вимоги для задоволення потреб конкретної організації. Використовуючи структуру матриці 10×3, можна на основі сходів формувати особливі шляхи кар'єрного зростання, наприклад, менеджера проектів, бізнес-аналітика, розробника, тестувальника програм, забезпечуючи структуру і напрям розвитку всередині організації, яка не схильна роз'яснювати персоналу внутрішній механізм функціонування сходів.

Можливі деякі відмінності при впровадженні сходів в інших компаніях, але її структура і основні поняття можуть використовуватися в будь-якій організації. У кожному окремому випадку фундаментальна основа сходів забезпечує високий ступінь єдності та цілісності підходу.

Контрольні запитання та завдання

1. Дайте визначення культурі інженерії програмного забезпечення та наведіть її компоненти.

2. Наведіть характеристику культури інженерії програмного забезпечення і вкажіть зв'язок культури з організацією.

3. Які організаційні структури компаній, що займаються розробкою програмного забезпечення, ви знаєте? Наведіть їх порівняльну характеристику.

4. Які технології розробки програмного забезпечення вам відомі? Наведіть їх порівняльну характеристику.

5. Наведіть приклади та дайте характеристику моделям зрілості процесів, що відбуваються на підприємстві, яке розробляє програмне забезпечення.

6. Викладіть суть моделі CMM (Capability Maturity Model).

7. Обґрунтуйте необхідність сертифікації компаній, що займаються розробкою програмного забезпечення.

8. Опишіть процес сертифікації компаній на рівень зрілості за моделлю CMM.

9. Обґрунтуйте необхідність реалізації компаніями програм професійного розвитку своїх співробітників.

10. Опишіть у вигляді короткого реферату діяльність компанії, що спеціалізується у сфері розробки та впровадження програмного забезпечення (наприклад, Sun Microsistems, IBM, Oracle, Hewlett-Packard, Sybase, Microsoft, Novell тощо). Особливу увагу приділіть питанням професійного розвитку персоналу в компаніях, сформованої корпоративної культури, організації робочого процесу, тренінгів, семінарів тощо.

РОЗДІЛ 8. ЯКІСТЬ ПРОГРАМНИХ ПРОДУКТІВ ЯК РЕЗУЛЬТАТ ВІДПОВІДАЛЬНОЇ ПРОФЕСІЙНОЇ ДІЯЛЬНОСТІ ПРОГРАМНИХ ІНЖЕНЕРІВ

Унаслідок зростання сфер застосування і відповідальності функцій, що виконуються програмами, різко зросла необхідність гарантування високої якості програмних продуктів, регламентації і коректного формування вимог до характеристик реальних комплексів програм та їх достовірного визначення. В результаті фахівці в галузі теорії і методів, що визначають якість продукції, змушені освоювати розвиток і застосування нового, специфічного продукту – програмних засобів і систем у цілому та їх якість при використанні. Складність аналізованих об'єктів – комплексів програм і психологічна самовпевненість ряду програмних інженерів у власній «непогрішності» часто призводять до того, що реальні характеристики якості функціонування програмних продуктів залишаються невідомими не тільки замовникам і користувачам, а й самим розробникам. Відсутність чіткого декларування в документах понять і необхідних значень характеристик якості програмного забезпечення викликає конфлікти між замовниками-користувачами і розробниками-постачальниками через різне трактування одних і тих самих характеристик [26].

Різноманіття класів і видів складних комплексів програм, зумовлене різними функціями і сферами застосування систем, визначає формальні труднощі, пов'язані з методами і процедурами доказу відповідності створюваних програмних продуктів умовам контрактів, вимогам замовників і споживачів. У міру розширення застосування і збільшення складності систем виділилися сфери, в яких дефекти, недостатня якість комплексів програм або даних можуть завдавати катастрофічного збитку, що значно перевищує позитивний ефект від їх використання. У таких критичних системах (наприклад, управління атомними електростанціями, великими банками або системами озброєння) недопустимі прояви катастрофічних ризиків функціонування програмних продуктів при будь-яких спотвореннях початкових

даних, збоях, часткових відмовах апаратури, помилках користувачів та інших нештатних ситуаціях. Подібні ризики комплексів програм можуть визначати безпеку функціонування об'єктів, підприємств і навіть країн. Унаслідок цього різко підвищилася відповідальність фахівців за якість результатів їх праці і створюваних програмних продуктів. Це вимагає безперервного вдосконалення, навчання і підвищення кваліфікації замовників, розробників і користувачів у галузі програмної інженерії, освоєння ними сучасних методів, процесів і міжнародних стандартів, а також високої корпоративної культури колективів фахівців, що забезпечують життєвий цикл критичних програмних продуктів [16].

Згода, досягнута щодо вимог до якості, нарівні з чітким доведенням до інженерів того, що складає якість одержуваного продукту, вимагають обговорення і формального визначення багатьох аспектів якості.

Інженери повинні розуміти сенс, що вкладається в концепцію якості, характеристики і значення якості відносно програмного забезпечення, що розробляється або супроводжується.

Важливою ідеєю є те, що програмні вимоги визначають необхідні характеристики якості програмного забезпечення, а також впливають на методи кількісної оцінки і сформульовані для оцінки цих характеристик відповідні критерії приймання.

Очікується, що програмні інженери сприймають питання якості програмного забезпечення як частину своєї професійної культури. Етичні аспекти можуть відігравати значну роль у забезпеченні якості програмного забезпечення, культурі та ставленні інженерів до своєї роботи.

Поняття «якість», насправді, не таке очевидне і просте, як це може здаватися на перший погляд. Для будь-якого інженерного продукту існує велика кількість інтерпретацій якості, залежно від конкретної «системи координат». Множину цих точок зору необхідно обговорити і визначити на етапі вироблення вимог до програмного продукту. Характеристики якості можуть бути потрібні тією чи іншою мірою, бути відсутніми або задавати певні вимоги, – усе це є результатом певного компромісу.

8.1. Надійність і безпека програмних продуктів та ризики, пов'язані з їх використанням

На даний час дедалі більша кількість складних і відповідальних процесів автоматизуються, а деколи й повністю виконуються на комп'ютерах, а людині залишається тільки спостерігати за коректністю їх роботи. Програми та програмні комплекси проходять багатостадійні тестування, що використовують різні методики: функціональне тестування, стрес-тестування, верифікацію тощо. Помилки, які виявляються при цьому, виправляються. Проте часто помилки виникають не під час розробки, а в період використання програмного продукту. У деяких галузях такі помилки можуть завдати великого економічного збитку або навіть призвести до загибелі людей. Прикладами таких ситуацій можуть бути [17 – 21]:

• *Падіння ракети «Аріан-5»*. Перший випробувальний політ цієї ракети відбувся 4 червня 1996 року і був невдалим. На 39-й секунді польоту ракета потерпіла крах унаслідок несправності в керуючому програмному забезпеченні, яка вважається найдорожчою комп'ютерною помилкою в історії. Частина програмного забезпечення ракети була запозичена з її попередньої версії – «Аріан-4», але при перенесенні цієї системи для використання на новій ракеті розробниками не були враховані всі особливості. Через іншу траєкторію виведення ракети через 30 секунд після запуску значення горизонтальної швидкості перевищило встановлені в програмі обмеження і викликало збій у роботі комп'ютера. Це призвело до видачі помилкової команди на відхилення сопел прискорювачів, а пізніше й основного двигуна. В результаті на 39-й секунді польоту ракета зруйнувалася під дією аеродинамічних сил. На ракеті «Аріан-4» збою не відбувалося через відмінності в характеристиках траєкторії польоту.

• *Медичний прискорювач Therac-25*. Доза лікувального опромінювання, призначеного для пацієнтів, була перевищена в 100 разів. Несправність у програмному забезпеченні приладу була зафіксована тільки через півроку; за цей час було зафіксовано шість смертельних випадків унаслідок

переопромінення. Основна помилка у власній багатозадачній операційній системі реального часу, що встановлювалася на контролері прискорювача, полягала у відсутності синхронізації та можливого через це так званого стану змагань (помилка програмування багатозадачної системи, при якій робота системи залежить від того, в якому порядку виконуються частини коду).

• *Збій у телефонній мережі AT&T 15 січня 1990 р*. Помилка в новій версії прошивки міжміських комутаторів призвела до того, що комутатор перезавантажувався, якщо одержував специфічний сигнал від сусіднього комутатора. Проблема полягала в тому, що цей сигнал генерувався в той момент, коли комутатор відновлював свою роботу після збою. Одного прекрасного дня один з комутаторів у Нью-Йорку перезавантажився і подав той самий нещасливий сигнал. Незабаром 114 сусідніх комутаторів безперервно перезавантажувалися кожні шість секунд, а 60 тисяч користувачів залишилися без міжміського зв'язку на дев'ять годин, поки інженери не встановили на комутатори попередню версію прошивки.

Це лише три приклади з величезної кількості випадків прояву помилок у програмному забезпеченні. Уже досить давно відомо, що комп'ютерні системи мають недоліки, тобто без явних причин іноді виходять із ладу, і не завжди зрозуміло, що необхідно для відновлення їх працездатності [17 – 21].

Функціональну надійність комп'ютерних систем можна визначити ступенем довіри до них, тобто упевненістю, що система буде працювати так, як передбачається, і що збоїв не буде. Цю властивість не можна оцінити кількісно. Для цього використовуються такі відносні терміни, як «ненадійні», «дуже надійні» або «понаднадійні», що відображають різний ступінь довіри до системи.

Існує чотири основні складові функціональної надійності програмних систем, неформальні визначення яких такі [10, 53]:

1. *Працездатність* – властивість системи виконувати свої функції в будь-який час експлуатації.

2. *Безвідмовність* – властивість системи коректно (так, як очікує користувач) працювати весь заданий період експлуатації.

3. *Безпека* – властивість системи, яка гарантує, що вона безпечна для людей і навколишнього середовища.

4. *Захищеність* – властивість системи протистояти випадковим або навмисним вторгненням у неї.

Працездатність і безвідмовність систем носять імовірнісний характер та можуть бути виражені кількісно. Безпека й захищеність рідко виражаються у вигляді числових показників, але їх можна порівнювати за відносною шкалою рівнів. Наприклад, безпека рівня 1 менша від безпеки рівня 2, яка, у свою чергу, менша за безпеку рівня 3, тощо.

Додаткові заходи, які підвищують функціональну надійність системи, можуть різко збільшувати вартість її розробки. Експоненційний характер залежності «вартість – надійність» не дозволяє говорити про можливість створення систем зі стовідсотковою надійністю, оскільки вартість їх створення була б дуже великою.

Високі рівні функціональної надійності можуть бути досягнуті лише за рахунок зменшення ефективності роботи системи. Наприклад, надійне програмне забезпечення передбачає додаткові, часто надлишкові, коди для перевірки позаштатних станів системи. Це ускладнює систему та збільшує обсяг пам'яті, необхідний для її ефективної роботи. Але у ряді випадків надійність більш важлива, ніж ефективність системи [10].

1. *Ненадійні системи часто залишаються непотрібними*. Якщо до системи немає довіри користувачів, то вона не буде використовуватися. Більше того, користувачі можуть відмовитися від інших програмних продуктів тієї ж компанії-розробника, оскільки будуть вважати їх ненадійними.

2. *Вартість відмови системи може бути величезна*. Для деяких застосувань, таких як системи керування реакторами або системи навігації, вартість наслідків відмови може перевищувати вартість самої системи.

3. *Складно модернізувати ненадійну систему для підвищення її надійності*. Зазвичай є можливість поліпшити неефективну систему, оскільки в цьому випадку основні зусилля будуть витрачені на модернізацію окремих програмних модулів.

Систему, до якої немає довіри, складно поліпшити, оскільки ненадійність «розподілена» по всій системі.

4. *Існують можливості компенсувати недостатню ефективність системи*. Якщо програмна система працює неефективно, то це постійний фактор, до якого користувач може підлаштуватися, побудувавши свою роботу з урахуванням цього фактору. Ненадійність системи, як правило, проявляється неочікувано. Ненадійне програмне забезпечення може порушити роботу всієї системи, до якої воно інтегровано, і зруйнувати дані користувача без попередження, що може мати серйозні наслідки.

5. *Ненадійні системи можуть бути причиною втрати інформації*. Збирання та зберігання даних – дороговартісна процедура, часто дані вартують більше, ніж комп'ютерна система, на якій вони обробляються. Дублювання даних для попередження їх втрати внаслідок ненадійності системи вимагає значних зусиль і фінансових засобів.

Надійність системи залежить від багатьох факторів [53], і, зокрема, від технології розробки програмного забезпечення. Багатократні тестування з метою виключення помилок сприяє розробці надійних систем. Однак не існує простого зв'язку між якістю процесу створення та якістю готової системи.

Зазвичай відмова систем, керованих за допомогою програмного забезпечення, викликає незручності, але вони не призводять до тривалих наслідків. Однак, як було продемонстровано вище, є системи, відмови яких можуть приводити до значних економічних втрат, фізичних пошкоджень або створювати загрозу людському життю. Такі системи зазвичай називають *критичними*. Функціональна надійність – необхідна вимога до критичних систем, і всі її складові (працездатність, безвідмовність, безпека та захищеність) дуже важливі. Не менш важливий для критичних систем і високий рівень надійності.

Існує три основні типи критичних систем [10]:

1. *Системи, критичні до забезпечення безпеки*. Системи, відмова яких призводить до руйнувань, створює загрозу життю людини або завдає шкоду оточуючому середовищу. Як приклад

можна навести систему керування виробництвом на хімічному заводі.

2. *Системи, критичні для цільового призначення*. Системи, відмова яких може призвести до помилок у діях, спрямованих на забезпечення визначеної мети. Прикладом може служити навігаційна система космічного корабля.

3. *Системи, критичні для бізнесу*. Відмова таких систем може завдати шкоду справі, в якій вони використовуються. Прикладом є система, яка обслуговує рахунки клієнтів у банку.

Ціна помилки критичної системи часто дуже велика. Вона включає витрати, пов'язані із внесенням змін у систему або її заміну, опосередковані витрати, наприклад судові, і витрати, пов'язані із втратами у бізнесі. Через високу можливу ціну відмови системи якість методів розробки і сам процес створення програмного забезпечення зазвичай більш важливі, ніж вартість використання цих методів.

Тому при створенні критичних систем зазвичай використовуються випробувані методи розробки, а не нові, які ще не мали значного практичного застосування. Лише порівняно недавно такі відносно нові методи, як, наприклад, об'єктно-орієнтовані, почали використовуватися для розробки критичних систем. Водночас досі при розробці багатьох критичних систем усе ще застосовуються функціонально-орієнтовані методи.

З іншого боку, методи розробки програмного забезпечення, які зазвичай нерентабельні, можуть використовуватись для розробки критичних систем (наприклад, метод формальних специфікацій і формалізованої перевірки програм на відповідність таким специфікаціям). Однією з причин використання цих методів є зменшення кількості необхідного тестування. Для критичних систем вартість перевірки та атестації зазвичай дуже висока і може складати більше 50 % загальної вартості системи.

Функціональна надійність – загальносистемне поняття. Розглядаючи надійність критичних систем, можна виділити три типи системних «компонентів», які схильні до відмов [53]:

1. Апаратні засоби системи, які можуть відмовити або через помилки конструювання, або через помилки виготовлення, або через повну зношеність.

2. Програмне забезпечення системи, яке може відмовити через помилки або в технічних вимогах до системи, або в архітектурі системи, або у програмному коді.

3. Людський фактор, який своїми діями порушує правильну роботу системи.

Отже, якщо мета полягає в тому, щоб підвищити надійність системи, необхідно розглядати всі ці аспекти у взаємозв'язку.

Як показує досвід, найбільш важливими складовими функціональної надійності є безвідмовність та працездатність. Якщо ж система ненадійна, то важко гарантувати її безпеку та захищеність. Ненадійність системи зумовлює великі матеріальні втрати, такі системи набувають репутації неякісних та в подальшому втрачають довіру споживачів.

Безвідмовність системи визначається відсутністю збоїв. Відмови системи можуть відбуватися через погане або неправильне її обслуговування, можуть бути наслідком помилок в алгоритмі, а можуть бути викликані несправностями систем зв'язку. Однак у багатьох випадках причиною помилкової поведінки системи є дефекти в самій системі. При розгляді безвідмовності корисно розуміти різницю в термінах *збій*, *помилка* та *відмова* [10]:

Відмова системи – припинення функціонування системи.

Системні помилки – неадекватна поведінка системи, яка не відповідає її специфікації.

Збій системи – неправильна поведінка системи, непередбачена її розробниками.

Помилка оператора – хибні дії користувача, які викликали збій у роботі системи.

Збої не обов'язково призводять до відмов системи, оскільки вони можуть бути короткочасними і система може прийти до нормального функціонування раніше, ніж відбудеться відмова. Системні помилки також не обов'язково спричинюють відмови

системи, оскільки системи зазвичай мають захист, який гарантує, що помилковий режим буде виявлений і виправлений.

Можна розглянути три взаємодоповнюючих підходи, які використовуються для підвищення безвідмовності систем [10].

1. *Попередження збоїв*. Підхід до розробки програмного забезпечення, який мінімізує можливість появи помилок і/або виявляє помилки перш ніж вони призведуть до збоїв системи. Приклад такого підходу – виключення певних конструкцій мов програмування, які легко підвергаються помилкам (наприклад, вказівники), та постійний аналіз програм для виявлення різноманітних аномалій програмного коду.

2. *Виявлення помилок та їх усунення*. Використання різноманітних методів перевірки системи в різноманітних режимах дозволяє виявити помилки та усунути їх до введення системи в експлуатацію. Регулярне тестування системи та її налагодження – приклад даного підходу.

3. *Стійкість до збоїв*. Використання спеціальних методів, які гарантують, що помилки в системі не призведуть до збоїв і що збої не спричинять відмов системи. Приклад такого підходу – застосування засобів самовідновлення системи з використанням дублювання модулів.

Розглянемо ще одну складову функціональної надійності систем – безпеку.

Безпека системи – це властивість, яка відображає здатність системи функціонувати, не погрожуючи людям або оточуючому середовищу. Там, де безпека є необхідним атрибутом системи, говорять про систему, критичну щодо забезпечення безпеки. Прикладами можуть бути контролюючі та керуючі системи в авіації, системи керування процесами на хімічних та фармацевтичних заводах і системи керування автомобілями.

Керування системами, критичними щодо забезпечення безпеки, найкраще організувати за допомогою апаратних засобів, ніж програмно-керованих. Однак на сьогоднішній день будуються системи такої складності, що керування не може здійснюватися лише апаратними засобами. Через необхідність керувати великою кількістю сенсорів та виконавчих механізмів зі

складними законами керування необхідне керуюче програмне забезпечення. Як приклад можна навести системи керування військовими безпілотними літаками. Вони потребують постійного програмного керування літаком, яке гарантує безпеку польоту.

Безвідмовність та безпека системи – взаємопов'язані, але, очевидно, різні складові функціональної надійності. Звичайно, система, критична щодо забезпечення безпеки, повинна відповідати своєму призначенню та функціонувати без відмов. Для забезпечення неперервного функціонування навіть у випадку помилок вона повинна мати захист від збоїв. Однак відмовостійкість не гарантує безпеки системи. Програмне забезпечення може лише один раз спрацювати неправильно, і це призведе до нещасного випадку.

Вважається, що система безпечна, якщо її експлуатація виключає аварії (нещасні випадки) або їх наслідки незначні. Цього можна досягти трьома доповнюючими один одного способами [10].

1. *Попередження небезпеки*. Система розробляється так, щоб попередити небезпечні ситуації. Наприклад, щоб під час експлуатації машини попередити потрапляння рук оператора під лезо, в системі розкрою передбачається обов'язкове одночасне натискання двох окремих кнопок керування.

2. *Виявлення та усунення небезпеки*. Система розробляється таким чином, щоб можливі небезпечні ситуації були виявлені та усунені до того, як вони призведуть до аварії. Наприклад, система, яка керує хімічним підприємством, для попередження вибуху від високого тиску, повинна вчасно виявити надлишковий тиск та відкрити запобіжний клапан, щоб зменшити цей тиск.

3. *Обмеження наслідків*. Система може містити способи захисту, які мінімізують пошкодження, що виникають у результаті аварії. Наприклад, до системи керування двигунами літака зазвичай додається автоматична система пожежогасіння. У випадку спалахування така система дозволяє попередити пожежу та не ставить під загрозу життя пасажирів та екіпажу.

Аварії та нещасні випадки зазвичай є результатом кількох подій, які відбуваються одночасно з непередбачуваними

наслідками. Аналізуючи серйозні аварії, Перроу К. (Perrow C.) показав, що майже всі вони відбулися через комбінацію системних збоїв, а не внаслідок окремих збоїв. Непередбачувана комбінація збоїв призводила до відмов систем. Цей дослідник стверджує, що неможливо попередити всі комбінації збоїв системи, а отже, ці аварії – невідворотний наслідок використання складних систем. Програмне забезпечення має тенденцію розростатися та ускладнюватися, а складність програмно-керованих систем збільшує ймовірність аварій і нещасних випадків.

Це, звичайно, не означає, що програмне керування обов'язково збільшує ризик, пов'язаний із системою. Програмне керування та поточний контроль можуть підвищити безпеку систем. Крім того, програмно-керовані системи можуть контролювати ширший діапазон умов у порівнянні, наприклад, з електромеханічними системами. Вони також доволі легко налаштовуються, передбачають використання комп'ютерних засобів, яким властива висока надійність та які відносно компактні. Програмно-керовані складні системи можуть блокувати небезпеку. Вони можуть підтримувати керування у шкідливих умовах, зменшуючи кількість необхідного обслуговуючого персоналу.

8.2. Основні поняття та визначальні чинники якості програмних продуктів

У сучасному світі розробка програмного забезпечення перетворилася на одну з найдорожчих індустрій і будь-які вузькі, місця в технологічному процесі його створення можуть привести до небажаних результатів [17 – 21]. Подовження термінів розробки програмного забезпечення зумовлює подорожчання кінцевого продукту, а невиявлені в ході тестування помилки спричинюють як мінімум зниження його продуктивності. Примітивні помилки, невиразні повідомлення і неохайний інтерфейс дратують користувачів, які у результаті вибирають якісніший продукт конкурента, а фірма ризикує втратити не тільки клієнтів, але і свою частку ринку. Отже, якість

програмного забезпечення набуває першорядного значення [29 – 54].

Є велика кількість визначень фундаментальної категорії «*якість*», які, по суті, зводяться до сукупності технічних, технологічних і експлуатаційних характеристик продукції або процесів, завдяки яким вони здатні відповідати вимогам споживача і задовольняти його при застосуванні. Відповідно до стандартів, поняття «*забезпечення якості*» – це сукупність планованих та систематичних заходів, необхідних для упевненості в тому, що продукція або процеси задовольняють певні вимоги споживачів до якості. Для реалізації цього положення призначені *системи якості*, кожна з яких включає: сукупність організаційної структури, відповідальності, процедур, процесів і ресурсів, що забезпечує здійснення керівництва якістю продукції або процесів.

Вивченням і реалізацією методів та засобів кількісного оцінювання якості продукції займається наукова дисципліна – кваліметрія [29 – 31]. Ефективне управління якістю можливе лише за наявності достатньо точних і об'єктивних методів вимірювання або оцінювання якості продукції або процесів. Створення і розвиток кваліметрії підготувало обґрунтоване застосування: числових, кількісних методів у розв'язанні задач при оцінці якості технологічних процесів і готової продукції, методів вибору переваг при аналізі альтернативних груп продуктів, методів розрахунку інтегральної якості, визначенні достовірності вибірок при статистичних оцінках якості та ряд інших завдань управління якістю. У основі кваліметрії лежать три базові положення:

- практична необхідність методів кількісної оцінки характеристик якості продукції для виконання завдань їх планування і контролю на різних рівнях управління створенням і застосуванням;
- підхід до якості як до єдиного динамічного поєднання ряду окремих властивостей, кожне з яких через свій характер і взаємозв'язки з іншими властивостями (з урахуванням їх

вагомості та пріоритету) впливає на формування ієрархічної структури узагальненої якості продукції;

- наявність принципової можливості вимірювання у кількісній формі, як окремих властивостей, так і їх поєднань, зокрема інтегральної якості.

Теоретична кваліметрія абстрагується від конкретних об'єктів і вивчає загальні закономірності та математичні моделі, пов'язані з оцінюванням якості. Її змістом є загальні методологічні проблеми кількісної оцінки якості, а також методи, спрямовані на подолання загальних труднощів, характерних для багатьох конкретних методик, призначених для кількісної оцінки якості конкретних об'єктів різного призначення.

Якість об'єкта залежить від того, з якою метою, для якого споживача і за яких умов робиться його оцінка. Один і той же об'єкт може мати кілька різних оцінок якості, вироблених для різних цілей і різних умов визначення. При кваліметричних вимірюваннях і оцінках якість розглядається як ієрархічна сукупність властивостей, розташованих на різних рівнях. Кожна з властивостей на одному рівні залежить від ряду інших властивостей, які знаходяться на нижчих рівнях. Кількість рівнів властивостей у міру поглиблення знань щодо конкретної продукції може зростати. Вивчення взаємозв'язку між властивостями, що входять до складу узагальненої якості, повинно теоретично обґрунтувати правомірність його розкладання для цілей з'єднання оцінок окремих властивостей у комплексні оцінки.

Практичним завданням кваліметрії є розробка і розвиток усіх комплексних і диференціальних методів оцінки якості. Диференціальні й економічні оцінки є основою комплексної оцінки й визначення інтегральних показників якості продукції, заснованих на узагальненні та зіставленні її окремих корисних властивостей і витрат ресурсів. Для отримання комплексної оцінки використовується експертне визначення вагомості кожної властивості. Насамперед повинен враховуватися вплив цієї властивості на ефективність використання даного виду продукції.

Значну роль у кваліметрії відіграють експертні методи. При експертних методах, оцінки, що даються окремими експертами, є суб'єктивними, залежать від цілого ряду їх індивідуальних особливостей: професії та кваліфікації експерта, знання ним умов застосування продукції, змістовності та кількості інформації, якою він користується. Математична обробка сукупностей суб'єктивних оцінок дозволяє одержувати об'єктивнішу оцінку якості. Величина похибки й надійність такої оцінки значною мірою залежать від точності оцінок окремих експертів, їх кількості, методів узагальнення та обробки результатів.

Велике місце в кваліметрії займають статистичні методи дослідження. Багато показників якості продукції визначаються за допомогою статистичних методів за дослідними даними або за матеріалами експлуатаційної статистики. Такі узагальнені кваліметричні оцінки якості часто отримуються шляхом вимірювання і порівняння фізичних, економічних, естетичних та інших характеристик з кращими зразками, які формально такими еталонами не є.

Різноманітність сфер застосування комп'ютерів стає дедалі ширшою та їх коректна робота часто є визначальною для якісного управління об'єктами, успіху підприємств або безпеки людини [29 – 54]. Тому ретельна специфікація та оцінювання характеристик якості програмного продукту – ключовий чинник забезпечення їх адекватного застосування. Це може бути досягнуто на основі виділення і визначення відповідних характеристик з урахуванням цілей використання і функціональних завдань програмного забезпечення. Важливо, щоб програми оцінювалися по кожній застосовній характеристиці якості з використанням стандартизованої або формалізованої метрики.

Стосовно програмних засобів «*система забезпечення якості*» – це сукупність методів і засобів організації керуючих та виконавчих підрозділів підприємства, що беруть участь у проектуванні, розробці та супроводі комплексів програм з метою додання їм властивостей, що забезпечують задоволення потреб замовників і споживачів при мінімальному або допустимому

витрачанні ресурсів. Для складних програмних комплексів з високими вимогами до якості проектування розвиток і застосування таких систем забезпечення якості повинні супроводжувати весь життєвий цикл основної продукції. Відмінності фактичних і необхідних показників якості об'єктів або процесів кваліфікуються як дефекти або помилки і є первинними стимулами для ухвалення і реалізації рішень по зміні визначальних значень якості. Для цього необхідні економічні та моральні причини, а також воля керівників, організація виконавців, методи і технологія для управління якістю і корегування програм [29 – 54].

Споживача-замовника, перш за все, цікавлять функції та якість готового кінцевого продукту – програмного засобу і зазвичай не дуже турбує, як вони досягнуті. Необхідну якість при розробці проектів програмного забезпечення, як і будь-якої продукції, можна забезпечити двома методами [37]:

- шляхом використання тільки завершального контролю та випробувань готових об'єктів і виключення з постачання або відхилення на доопрацювання продуктів, які не відповідають необхідній якості;
- за допомогою застосування регламентованих технологій і систем забезпечення якості процесів проектування і розробки, які запобігають дефектам і гарантують високу якість продукції під час її створення і модифікації.

Перший метод може спричинювати значні економічні втрати за рахунок витрат на створення частини не придатного до використання браку, що може дорого коштувати для складних систем. Досягнення необхідної якості за рахунок тільки вихідного контролю, за відсутності адекватної технології та системи забезпечення якості в процесі розробки, може призводити до тривалого ітераційного процесу масових доопрацювань і повторних випробувань продукції.

Другий метод забезпечує високу якість виконання всього процесу проектування та розробки і тим самим мінімум економічних втрат від браку, що рентабельніше при створенні складних систем. При цьому скорочується, але не виключається

вихідний контроль якості продукції. Для створення сучасних прикладних високоякісних інформаційних систем необхідні обидва методи, з акцентом на застосування регламентованих технологій. Таким чином, забезпечення й посвідчення якості складного програмного забезпечення мають базуватися на перевірках і випробуваннях [42]:

- технологій забезпечення життєвого циклу програмних засобів, підтриманих регламентованими системами якості;
- готового програмного продукту з повним комплектом адекватної експлуатаційної документації.

Глибокий взаємозв'язок якості розроблених програм з якістю технології їх створення і з витратами на розробку стає особливо істотним при необхідності отримання кінцевого продукту з гранично високими значеннями показників якості. Встановлено, що витрати на розробку різко зростають, коли показник якості наближається до межі, досяжної при даній технології і рівні автоматизації процесу розробки. Це привело до істотної зміни останніми роками об'єктів, методології та культури щодо створення і вдосконалення програмного забезпечення. Безперервне зростання вимог до якості програм стимулювало створення й активне застосування міжнародних стандартів і регламентованих технологій, що автоматизують основні процеси їх життєвого циклу, починаючи з ініціалізації проекту.

Основою для формування вимог до програмного забезпечення є аналіз властивостей, що характеризують якість його функціонування з урахуванням технологічних і ресурсних можливостей розробника. При цьому під *якістю функціонування* розуміється сукупність властивостей, що зумовлюють придатність програмних продуктів забезпечувати надійне і своєчасне подання необхідної інформації споживачу для її подальшого використання за призначенням. Адекватний набір показників якості програм залежить від функціонального призначення і властивостей кожного програмного продукту. Відповідно до принципових особливостей програмного продукту, при проектуванні повинні вибиратися номенклатура і значення показників якості, необхідних для його ефективного застосування

користувачами, які згодом відображаються в технічній документації і в специфікації вимог на кінцевий продукт [42].

Кожен критерій якості може використовуватися, якщо визначена його метрика і вказаний спосіб її оцінювання і зіставлення з потрібним еталонним значенням. Для конкретних видів програмного забезпечення домінуючі критерії якості виділяються на етапі визначення його функціонального призначення та відповідності вимогам технічного завдання. Програми для ЕОМ як об'єкти проектування, розробки, випробувань і оцінки якості характеризуються такими узагальненими показниками [36]:

- проблемно-орієнтованою областю застосування, технічним і соціальним призначенням програмного комплексу;
- конкретним типом розв'язуваних функціональних задач з достатньо визначеною сферою застосування відповідними користувачами;
- об'ємом і складністю сукупності програм і бази даних, які виконують єдине цільове завдання даного типу;
- необхідними складом і необхідними значеннями характеристик якості функціонування програм і величиною допустимого ризику (збитку) зумовлену недостатньою якістю;
- ступенем зв'язку розв'язуваних задач з реальним масштабом часу або допустимою тривалістю очікування результатів розв'язування задачі;
- прогнозованими значеннями тривалості експлуатації та перспективою створення множини версій комплексу програм;
- передбачуваним тиражем виробництва й застосування комплексу програм;
- ступенем необхідної документованості програм.

Якість у використанні – це основна якість програмного забезпечення, яка сприймається користувачами [48]. Вона вимірюється швидше в термінах результату функціонування і застосування програм, аніж внутрішніми властивостями самого програмного забезпечення. Мета такого оцінювання – визначення, чи має продукт необхідний ефект у специфічному контексті використання. Якість програмного забезпечення в

середовищі користувачів може відрізнятися від якості в середовищі розробників, оскільки деякі функції можуть бути невидимі користувачу або не використовуватися ним. Користувач оцінює тільки ті атрибути програмного забезпечення, які видимі та корисні йому в процесі реального застосування. Тому до дефектів комплексів програм слід відносити не тільки прямі втрати при їх застосуванні користувачами, але і надмірні властивості, які не потрібні користувачам і потребують додаткових витрат при розробці. Іноді атрибути програмного забезпечення, специфіковані користувачем на етапі аналізу вимог, згодом не виправдовують його надії при застосуванні продукту внаслідок зміни поглядів і понять, а також складності специфікації неявних потреб на початку проектування.

Якість змінюється протягом життєвого циклу програмного забезпечення, тобто його необхідне і реальне значення на початку розробки майже завжди відрізняються від фактично досягнутого по завершенні проекту і якості версії продукту, що поставляється користувачам. На практиці важливо оцінювати якість програм не тільки в завершеному вигляді, але і в процесі їх проектування, розробки і супроводу. Крім того, оцінки показників якості можуть бути суб'єктивними і відображати різні точки зору і потреби різних фахівців. Щоб ефективно управляти якістю на кожному етапі життєвого циклу, необхідно вміти визначати і застосовувати різні показники необхідної якості та її зміни. Характеристики цього процесу значною мірою визначаються сукупними витратами, необхідними для досягнення заданої якості кінцевого продукту – версії програмного засобу.

Необхідні характеристики якості програмного забезпечення з різних позицій відображають їх властивості й особливості, та, у свою чергу, залежать від ряду чинників і обмежень. При системному аналізі та проектуванні програмних засобів необхідно визначати і враховувати зв'язки, вплив і взаємодію таких основних чинників, які відбиваються на їх якості [48]:

- призначення, зміст і опис функціональних характеристик і атрибутів, що визначають специфічні особливості цілей, завдань,

властивостей і сфери застосування конкретного програмного засобу – його функціональну придатність;

- конструктивні характеристики якості, що сприяють поліпшенню і вдосконаленню призначення, функцій і можливостей застосування програмного забезпечення;
- метрики, міри і шкали вибраних та придатних для вимірювання і оцінювання конкретних характеристик і атрибутів якості програмного забезпечення з урахуванням певної достовірності;
- рівні можливої деталізації при описі та оцінюванні певних характеристик і атрибутів якості програмного забезпечення;
- цілі та особливості споживачів результатів оцінювання характеристик якості програмного забезпечення;
- зовнішні та внутрішні негативні чинники, що впливають на якість створення і застосування програмного забезпечення;
- доступні ресурси, що обмежують можливі величини реальних характеристик якості програмного забезпечення;
- конкурентоспроможність, виражена відношенням ефективності застосування до вартості придбання й експлуатації програмного забезпечення.

Вплив перерахованих чинників на якість програмного забезпечення залежить, перш за все, від його призначення і вимог до функцій. Як наголошувалося вище, множину характеристик якості програмних засобів можна розділити на дві групи, що принципово розрізняються [26]:

- функціональні характеристики, які визначають призначення, властивості та завдання, що виконуються комплексом програм для основних користувачів. Вони відрізняються дуже широким спектром і різноманітністю, склад і специфіку яких важко уніфікувати та можна класифікувати лише за великою кількістю класів і властивостей програмного забезпечення;
- конструктивні характеристики якості, номенклатура яких може бути уніфікована, адаптована й використана для опису інших, внутрішніх і зовнішніх, стандартизованих характеристик якості, що підтримують і поліпшують реалізацію основних

функціональних вимог до якості об'єктів і процесів життєвого циклу програмних засобів.

Визначення і порівняння *функціональної якості програм* доцільно розглядати в межах обмежених класів програмного забезпечення, що виконують подібні функції [26]. Такі класи функцій можуть виділятися в межах крупних проблемно-орієнтованих сфер застосування (адміністративні, банківські, медичні, авіаційні тощо) і для виконання дрібніших, спеціальних, функціональних завдань у цих сферах. Кожне з таких завдань може бути описане рядом специфічних властивостей, характеристик і атрибутів, повна номенклатура яких містить багато тисяч назв, мір та шкал, які важко або неможливо уніфікувати. Функціональні характеристики та їх параметри можуть піддаватися значним модифікаціям протягом усього життєвого циклу програмного забезпечення і є зазвичай найбільш динамічними компонентами зі всіх характеристик якості.

Функціональна придатність безпосередньо визначає основне призначення і функції програмного забезпечення для користувачів. У контракті та технічному завданні для кожного проекту, вона повинна бути виділена й формалізована для однозначного розуміння й оцінювання всіма партнерами на кожному етапі життєвого циклу і при значних модифікаціях завдань програмного забезпечення. Функціональна придатність є основною метою і головною характеристикою для всієї множини типів програмного забезпечення.

Друга група характеристик – конструктивних – відіграє другорядну роль і повинна завперш підтримувати і забезпечувати високу якість реалізації функцій програмного забезпечення та його застосування за основним призначенням. Номенклатура цих характеристик відносно невелика і стандартами рекомендується в складі: коректності, здатності до взаємодії, захищеності, надійності, ресурсної ефективності, практичності, супроводжуваності та мобільності. Їх вибір і значення визначаються вимогами до функціональної придатності програмного забезпечення. Початкова номенклатура цієї групи характеристик та їх атрибутів практично інваріантна до функцій

програмного забезпечення і стандартизована. Для кожного конкретного проекту програмного забезпечення з них може бути виділена представницька група найбільш важливих і найбільш впливових для розв'язання певних функціональних задач.

Розглянемо тепер деякі особливості впливу ресурсів створення програмного забезпечення на його якість [40 – 44].

Загальне поняття доступних ресурсів створення програм включає реальні фінансові, часові, кадрові та апаратурні обмеження, в умовах яких відбувається створення і вдосконалення комплексів програм. Залежно від характеристик об'єкта розробки на її виконання виділяються ресурси різних видів і обсягу. Ці чинники проявляються як додаткові характеристики програмних продуктів та їх рентабельності, які слід враховувати й оптимізовувати, починаючи з системного аналізу життєвого циклу програмного забезпечення. В результаті доступні ресурси стають непрямими критеріями, або чинниками, що впливають на вибір методів розробки, які визначають якість створюваного програмного забезпечення. Багато проектів інформаційних систем зазнавали й зазнають невдач через відсутність у розробників і замовників при підготовці контракту чіткого уявлення про реальні фінансові, трудові, часові та інші ресурси, необхідні для їх реалізації. Тому одним з основних завдань при системному проектуванні програмного забезпечення є економічний аналіз і визначення необхідних ресурсів для створення програмного забезпечення відповідно до вимог контракту і технічного завдання.

Найбільш загальним видом ресурсів, використовуваних у життєвому циклі програмного забезпечення, є *допустимі фінансово-економічні витрати* або еквівалентні їм величини трудомісткості відповідних робіт. При розробці, тестуванні та аналізі якості цей показник може застосовуватися або як вид ресурсних обмежень, або як оптимізований критерій, що визначає доцільну функціональну придатність програмного забезпечення. При цьому необхідно також враховувати витрати на розробку, закупівлю й експлуатацію системи якості, на технологію і комплекс автоматизації проектування програм і баз даних, які

можуть складати істотну частину сукупної вартості та трудомісткості розробки і всього життєвого циклу програмного забезпечення.

Час або допустима тривалість розробки певних версій програмного забезпечення є непоповнюваним обмеженим ресурсом реальних проектів. Цей ресурс чимраз більше визначає досяжну якість комплексів програм у процесі їх розробки і супроводу. Високі вимоги замовників до стислих термінів реалізації проектів, природно, обмежують розробників і випробувачів щодо тривалості та обсягу можливого системного аналізу і проектування, розробки і, особливо, тестування програм. Збільшення кількості фахівців, що залучаються для цього, при дослідній експлуатації або тестуванні, лише в деяких межах дозволяє прискорювати розробку і збільшувати сукупну кількість тестів при перевірках, для підвищення якості програм.

Кадри фахівців можна оцінювати чисельністю, а також тематичною і технологічною кваліфікацією, які завжди обмежені. У створенні великомасштабних програмних систем беруть участь системні аналітики й керівники різних рангів, програмісти і допоміжний обслуговуючий персонал у певному раціональному поєднанні. Визначальними є сукупна чисельність і структура колективу, а також його підготовленість до колективної розробки конкретного типу програмного забезпечення і до застосування ним системи забезпечення якості функціонування.

Доступні розробникам програмного забезпечення обчислювальні ресурси об'єктних і технологічних ЕОМ є одним з найважливіших чинників, що визначає досяжну якість складних програмних систем. У процесі проектування доцільно виділяти певні ресурси ЕОМ на оперативне забезпечення якості, підвищення захищеності та надійності функціонування. Допустима величина і раціональний розподіл ресурсів ЕОМ на окремі методи поліпшення конструктивних характеристик якості програмного забезпечення істотно впливають на їх значення.

Узагальненими ресурсами проекту програмного забезпечення є доступні вартість або сукупні трудові, часові та матеріальні витрати, необхідні для придбання, створення, модифікації та

експлуатації компонентів і всього комплексу програм. Ці характеристики безпосередньо впливають практично на всі показники якості та визначають рентабельність покупки або створення наново конкретного програмного продукту. Це означає, що якість є відносним поняттям, яке залежить від ресурсів і суб'єктів, що здійснюють його оцінку з позиції ефективності використання, а також від стану ринку відповідної продукції, її виробників і технологій. Орієнтація на споживачів передбачає аналіз їх потреб і визначення можливостей ринку задовольнити ці потреби. При цьому слід враховувати ринкову конкуренцію двох видів: між постачальниками готових до застосування програмних засобів з фіксованою якістю і між розробниками, здатними забезпечити життєвий цикл програмного забезпечення або його істотну частину, з характеристиками якості, потрібними конкретному замовнику. В останньому випадку на необхідну якість можуть впливати не тільки замовник і безпосередні користувачі, але і різні посередники, організаційні та торгові структури, а також виконавці проекту.

На початку проектування програмного забезпечення завжди виникає завдання оцінювання ресурсів, які необхідні та доступні для створення і забезпечення всього життєвого циклу програмного забезпечення, а також можливої економічної ефективності подальшого застосування комплексу програм за призначенням за умови реалізації необхідних характеристик якості. Економічна ефективність і витрати мають самостійне значення і методологію при аналізі життєвого циклу програмного забезпечення. При плануванні проектів програмних засобів ініціатором розробки часто є розробник-постачальник, який самостійно ухвалює всі рішення щодо проектування за рахунок власних ресурсів і припускає відшкодувати витрати при реалізації програмного забезпечення на ринку. У інших випадках є певний замовник-споживач, здатний задати основні цілі, характеристики якості та забезпечити ресурси для реалізації проекту. Таким чином, при економічному аналізі проектів програмного забезпечення можливі два сценарії [40 – 44]:

- створення і весь життєвий цикл комплексу програм орієнтується розробником на масове тиражування і розповсюдження на ринку, для наперед не відомих покупців-користувачів у різних сферах застосування, при цьому відсутній пріоритетний зовнішній споживач-замовник, який визначає і диктує основні вимоги, а також фінансує проект;
- розробка проекту програмного забезпечення передбачається постачальником-розробником для конкретного споживача-замовника, який його фінансує, з визначеним, необхідним йому тиражем і відомою, обмеженою сферою застосування результатів розробки.

Перший сценарій базується на маркетингових дослідженнях ринку програмних продуктів і на прагненні постачальника зайняти на ринку достатньо вигідне місце, що забезпечує йому необхідний прибуток. Найважливішим чинником конкурентоспроможності програмного забезпечення є співвідношення між цінністю наявного або передбачуваного продукту з позиції його використання споживачем, і вартістю його при створенні або придбанні в умовах реального ринку. Для цього йому потрібно визначити наявність на ринку спектра близьких за призначенням програмних продуктів, оцінити їх економічну ефективність, вартість і вживаність, а також можливу конкурентоспроможність передбачуваного програмного продукту для потенційних користувачів та їх можливу кількість. Крім того, слід оцінити рентабельність витрат на забезпечення всього життєвого циклу нового програмного забезпечення й виявити функціональні та конструктивні характеристики якості, здатні привернути достатню кількість покупців і виправдати витрати на майбутню розробку.

Другий сценарій передбачає наявність певного замовника-споживача проекту програмного забезпечення, який визначає основні технічні й економічні вимоги та характеристики. Він вибирає конкурентоспроможного постачальника-розробника, якого оцінює на можливість реалізувати проект з необхідною якістю з урахуванням обмеження термінів, бюджету та інших ресурсів. Цьому допомагає досвід і економічні характеристики

раніше виконаних постачальниками проектів, але деякі проекти можуть не мати явних прецедентів, і тоді доводиться використовувати наявну статистику в цій галузі. Передбачається, що результати розробки не обов'язково підлягають широкому тиражуванню, можуть не надходити на відкритий ринок, унаслідок чого маркетингові дослідження для таких проектів не є домінуючими і зазвичай заздалегідь можуть не проводитися.

Проте замовнику й розробнику перед укладенням контракту необхідний достатньо достовірний системний аналіз, прогнозування та техніко-економічне обґрунтування необхідних ресурсів за трудомісткістю, вартістю, термінами та іншим характеристикам. Неузгодженість економічних інтересів постачальника і споживача при оцінюванні вартості та необхідних ресурсів проекту вимагає пошуку компромісу, при якому розробник не продешевить, а замовник не переплатить за конкретні виконані роботи і весь проект. Тому обидва партнери зацікавлені в достовірному економічному прогнозуванні витрат ресурсів на проект програмного забезпечення.

Життєвий цикл програмного забезпечення можна розділити на дві частини, що істотно різняться економічними особливостями процесів, характеристиками та іншими чинниками, які на них впливають [40 – 44].

У першій частині життєвого циклу проводяться системний аналіз, проектування, розробка, тестування і випробування базової версії програмного забезпечення. Номенклатура робіт, їх трудомісткість, тривалість та інші економічні характеристики на цих етапах життєвого циклу істотно залежать від характеристик об'єкта, технології та інструментального середовища розробки. Особливо важливо враховувати можливе зростання сумарних витрат при завищенні вимог до якості програмного продукту. Як і для інших видів промислової продукції, поліпшення якості комплексів програм зазвичай досягається не пропорційним, а значнішим зростанням потрібних для цього ресурсів. Скорочення цієї потреби в ресурсах часто можливо тільки за рахунок принципової зміни й удосконалення технології проектування та розробки [40 – 44].

Багато молодих програмістів відважно стверджують, що вони можуть «писати» програми зі швидкістю тисячі рядків на тиждень. Проте такі програми здатні розв'язувати відносно невеликі, автономні задачі з невідомою і, як правило, низькою якістю, що не підтримуються повноцінною документацією і не відповідають вимогам стандартів на програмний продукт. Дослідження останніх десятиліть показали, що при розробці великомасштабних комплексів програм реальна середня продуктивність розробників майже на два порядки нижча за такі оцінки й вимірюється лише кількома десятками рядків на тиждень. При такому аналізі, природно, повинен враховуватися весь цикл розробки, від підготовки технічного завдання до завершення успішних кваліфікаційних випробувань, а також весь колектив учасників проекту, включаючи керівників, системних аналітиків і допоміжний персонал. Безпосереднім «написанням» текстів програм у колективі займається тільки третина або чверть розробників і майже стільки ж – їх автономним і кваліфікаційним тестуванням.

Друга частина життєвого циклу, що відображає експлуатацію, супровід, модифікацію і перенесення програмного забезпечення на інші платформи, меншою мірою пов'язана з функціональними характеристиками об'єкта й середовища розробки. Номенклатура робіт на цих етапах більш-менш визначена, але їх трудомісткість і тривалість можуть сильно варіювати, залежно від масовості та інших зовнішніх чинників розповсюдження і застосування версій програм. Успіх програмних продуктів у користувачів і на ринку, а також майбутній процес розвитку версій важко передбачити, і він не пов'язаний безпосередньо з економічними параметрами процесів розробки програмного забезпечення. Визначальними стають споживчі характеристики програмного забезпечення, а їх економічні особливості з позиції розробників і витрати на чергову версію відходять на другий план.

Унаслідок цього в широких межах можуть змінюватися трудомісткість і кількість фахівців, необхідна для підтримки цих етапів життєвого циклу. Це ускладнює статистичне узагальнення техніко-економічних показників різних проектів і прогнозування

на їх основі аналогічних характеристик нової розробки. Тому плани на цих етапах мають характер загальних взаємозв'язків змісту робіт, які вимагають розподілу в часі, індивідуально для кожного проекту. В результаті планування трудомісткості та тривалості етапів доводиться проводити ітераційно на базі накопичення досвіду й аналізу розвитку конкретних версій програмного забезпечення, а також їх успіху на ринку.

8.3. Огляд стандартів, які забезпечують якість програмних продуктів

Характеристики якості та керівництво з їх застосування є основою формальної регламентації характеристик якості програмного забезпечення. Розвиток міжнародних стандартів проводиться у напрямі уточнення, деталізації та розширення, описів характеристик якості комплексів програм [26, 45 – 48].

На сьогоднішній день розроблено та активно впроваджується міжнародний стандарт ISO 9126, який складається з чотирьох частин під загальним заголовком: «Інформаційна технологія – Якість програмних засобів» [26]:

Частина 1: Модель якості.

Частина 2: Зовнішні метрики якості.

Частина 3: Внутрішні метрики якості.

Частина 4: Метрики якості у використанні.

Частина перша стандарту ISO 9126-1 (переглянута і розширена редакція ISO 9126:1991) зберегла практично ту ж номенклатуру нормативних характеристик якості програмних засобів. У ній наводиться схема взаємозв'язку частин стандарту ISO 9126 і частин стандарту ISO 14598, а також сфера застосування, нормативні посилання, терміни і визначення. Модель характеристик якості програмного забезпечення складається з шести груп базових показників, кожна з яких деталізована кількома нормативними субхарактеристиками:

Функціональна придатність деталізується:

- придатністю для застосування;
- коректністю (правильністю, точністю);
- здатністю до взаємодії;

- захищеністю.

Надійність характеризується:
- рівнем завершеності (відсутність помилок);
- стійкістю до дефектів;
- відновлюваністю;
- доступністю та готовністю.

Ефективність рекомендується відображати:
- часовою ефективністю;
- використанням ресурсів.

Застосовність (практичність) пропонується описувати:
- зрозумілістю;
- простотою використання;
- легкістю вивчення;
- привабливістю.

Супроводжуваність представляється:
- зручністю для аналізу;
- змінністю;
- стабільністю;
- тестопридатністю.

Переносимість (мобільність) пропонується відображати:
- адаптованістю;
- простотою установки (інсталяції);
- співіснуванням та відповідністю;
- заміщуваністю.

Додатково кожна характеристика супроводжується субхарактеристикою «*узгодженість*», яка повинна відображати відсутність суперечностей з іншими стандартами і нормативними документами, а також з іншими показниками в даному стандарті. Характеристики і субхарактеристики в цій частині стандарту визначені дуже стисло (2 – 3 рядки), без коментарів і докладних рекомендацій з їх застосування до конкретних систем і проектів. Матеріали мають концептуальний характер і не містять рекомендацій з вибору і впорядкування пріоритетів необхідного мінімуму критеріїв залежно від особливостей об’єкта середовища розробки й застосування. Крім того, відсутні методики вимірювання характеристик і зіставлення з вимогами

специфікацій, а також рекомендації, на яких етапах життєвого циклу програмного забезпечення їх доцільно застосовувати.

Описи показників якості орієнтовані на висококваліфікованих системних аналітиків і замовників програмного забезпечення, яким надається можливість вибирати необхідну номенклатуру та спосіб оцінювання характеристик відповідно до призначення, сфери застосування і конкретних особливостей створюваних об'єктів.

Загальне уявлення про якість програмного забезпечення міжнародним стандартом ISO 9126 рекомендується відображати трьома взаємодіючими і взаємозалежними метриками характеристик якості, що відображають [26]:

- внутрішню якість, що проявляється в процесі розробки та інших проміжних етапів життєвого циклу програмного забезпечення;
- зовнішню якість, що задано вимогами замовника в специфікаціях і відображається характеристиками кінцевого продукту;
- якість при використанні в процесі нормальної експлуатації та результативністю досягнення потреб користувачів з урахуванням витрат ресурсів.

Ці типи метрик застосовні при визначенні цілей проекту і вимог до якості програмного забезпечення, включаючи проміжні компоненти і продукти. Відповідні внутрішні атрибути якості програмного забезпечення є передумовою досягнення в життєвому циклі необхідної зовнішньої поведінки, а прийнятна зовнішня поведінка – передумова досягнення якості у використанні.

Внутрішні метрики, відповідно до стандартів, можуть застосовуватися в ході проектування і програмування до невиконуваних компонентів програмного забезпечення, таких, як специфікація або початковий програмний текст. При розробці програмного забезпечення проміжні компоненти слід оцінювати з використанням внутрішніх метрик, які відображають деякі функціональні та конструктивні властивості програм. Основна мета застосування внутрішніх метрик – забезпечення необхідної

зовнішньої якості. Рекомендується використовувати внутрішні метрики, які мають найбільш сильні зв'язки з пріоритетними зовнішніми метриками, щоб вони могли допомагати при прогнозуванні їх досяжних значень.

Внутрішні метрики дають можливість розробникам, випробувачам і замовникам, починаючи із системного проектування, прогнозувати якість життєвого циклу програм і займатися питаннями технологічного забезпечення якості до того, як програмне забезпечення стає готовим до використання продуктом. Вимірювання внутрішніх метрик використовують властивості, категорії, числа або характеристики елементів зі складу програмного забезпечення, які, наприклад, є в процедурах початкового програмного тексту, в графі потоку управління, в потоці даних і в описах зміни станів пам'яті. Якість документації також може оцінюватися з використанням внутрішніх метрик.

Зовнішні метрики використовують міри програмного забезпечення, виведені зі спостережень за поведінкою системи, частиною яких вони є, шляхом випробувань, експлуатації та спостереження виконуваних програм або функціонування інформаційної системи. Перед придбанням або використанням програмного забезпечення його слід оцінити з використанням метрик, заснованих на реалізації ділових і професійних цілей, зв'язаних із застосуванням програмного продукту в певному організаційному і технічному середовищі. Зовнішні метрики забезпечують замовникам, користувачам і розробникам можливість простежувати й аналізувати якість програмного забезпечення в ході випробувань або дослідної експлуатації. Відповідні зовнішні метрики специфікуються для отримання числових значень або категорій і властивостей внутрішніх характеристик якості, щоб їх можна було використовувати для перевірки того, що проміжні продукти в процесі розробки задовольняють внутрішнім специфікаціям якості.

Метрики якості у використанні відображають, якою мірою продукт задовольняє потреби конкретних користувачів для досягнення заданої мети. Ця метрика не відображена шістьма базовими характеристиками програмного забезпечення, що

регламентуються стандартом ISO 9126-1 унаслідок її спільності, проте рекомендується для інтегральної оцінки результатів функціонування і застосування комплексів програм у стандарті ISO 9126-4. Якість у використанні – це об'єднаний ефект функціональних і конструктивних характеристик якості програмного забезпечення для користувачів. Зв'язок якості у використанні з іншими характеристиками програмного забезпечення залежить від *завдань і функцій споживачів* [26]:

- замовнику потрібна повна відповідність характеристик програмного продукту умовам контракту, технічного завдання і специфікацій вимог;
- для кінцевого оперативного користувача програмного забезпечення за основним призначенням якість у використанні зумовлюють, в основному, характеристики функціональних можливостей, надійності, практичності та ефективності;
- для персоналу супроводу програмного забезпечення якість у використанні визначається переважно супроводжуваністю;
- для персоналу, що виконує перенесення програмне забезпечення на інші платформи, а також інсталяцію і адаптацію до середовища застосування, якість у використанні визначається, перш за все, мобільністю.

Практично неможливо виміряти всі внутрішні або зовнішні субхарактеристики та їх атрибути для всіх компонентів великомасштабних програмних систем.

Аналогічно, зазвичай не практикується формалізація вимог та оцінювання якості у використанні для всіх можливих сценаріїв завдань користувачів. Тому їх необхідно ранжувати й виділяти пріоритетні процеси й об'єкти для оцінювання характеристик з різною достовірністю.

Для вибору характеристик якості програмного забезпечення і достовірного порівняння їх з вимогами, а також для зіставлення їх значень між різними програмними продуктами необхідні оцінки, вимірювання та використання певних заходів і шкал. Стандартами рекомендується, щоб було передбачене вимірювання кожної характеристики якості програмного забезпечення (субхарактеристики або її атрибуту) з точністю і

визначеністю, достатньою для порівнянь з вимогами технічних завдань і специфікацій, а також щоб вимірювання були об'єктивними та відтворюваними. Слід передбачати норми допустимих помилок вимірювання, викликаних інструментами і/або помилками людини-експерта. Щоб вимірювання були об'єктивними, повинна бути документована й узгоджена процедура для присвоєння числового значення, властивості або категорії кожному атрибуту програмного продукту. Процедури вимірювань повинні давати в результаті однакові заходи з прийнятною стійкістю, що отримуються різними суб'єктами при виконанні одних і тих самих вимірювань характеристик програмного забезпечення в різних випадках.

Характеристики, субхарактеристики й атрибути якості програмного забезпечення з позиції можливості та точності їх вимірювання можна розділити на три рівні деталізації показників, особливості яких слід уточнювати при їх виборі [26]:

- *категорійно-описові*, які відображають набір властивостей і загальні характеристики об'єкта – його функції, категорії відповідальності, захищеності та важливості, які можуть бути представлені номінальною шкалою категорій-властивостей;
- *кількісні*, які представляються множиною впорядкованих числових точок, що відображають безперервні або дискретні закономірності та описуються інтервальною або відносною шкалою, які можна об'єктивно виміряти й чисельно зіставити з вимогами;
- *якісні*, які містять кілька впорядкованих або окремих властивостей – категорій, які характеризуються порядковою або точковою шкалою набору категорій (є – немає, добре – погано), встановлюються, вибираються та оцінюються значною мірою суб'єктивно й експертно.

До першого рівня належать показники якості, які характеризуються найбільшою різноманітністю значень-властивостей програм і наборів даних і охоплюють весь спектр класів, призначень і функцій сучасних програмних продуктів. Ці властивості можна порівнювати тільки в межах однотипних продуктів і важко впорядковувати за принципом переваги. До

стандартизованих показників якості для цієї групи, перш за все, відносять *функціональну придатність*, що є найважливішою і домінуючою характеристикою будь-яких програмних продуктів. Номенклатура і значення решти показників якості безпосередньо визначається необхідними функціями програмного засобу й тією чи іншою мірою впливають на виконання цих функцій. Тому вибір функціональної придатності програмного забезпечення, докладний і коректний опис його властивостей є основними початковими даними для встановлення при системному проектуванні необхідних значень всіх стандартизованих показників якості.

До другого рівня показників якості належать достатньо достовірно й об'єктивно вимірювані числові характеристики програмного забезпечення. Значення цих характеристик зазвичай найбільшою мірою впливають на функціональну придатність і метрики у використанні програмного забезпечення. Тому вибір і обґрунтування їх необхідних значень повинно проводитися найбільш акуратно й достовірно вже при проектуванні програмного забезпечення. Їх субхарактеристики можуть бути описані впорядкованими шкалами об'єктивно вимірюваних значень, необхідні числові величини яких встановлюються та вибираються замовниками або користувачами програмного забезпечення. Такими характеристиками є надійність і ефективність комплексів програм. Надійність може відображатися часом напрацювання на відмову, середнім часом відновлення, а також коефіцієнтом готовності – ймовірністю застати програмний продукт у працездатному стані при нормальній експлуатації. Ці величини можуть вибиратися й фіксуватися в технічному завданні або специфікації вимог і супроводжуватися методикою об'єктивних, числових вимірювань при кваліфікаційних випробуваннях для зіставлення з вимогами.

Атрибути часової ефективності тісно пов'язані між собою і також значно впливають на функціональну придатність програмного забезпечення. Тривалість розв'язання основних задач, пропускна здатність певної кількості рішень за деякий інтервал часу, тривалість очікування результатів (відгуку) і деякі

інші характеристики динаміки функціонування програмного забезпечення можуть бути вибрані та встановлені кількісно у специфікаціях вимог замовником. Ця субхарактеристика не завжди може бути вибрана і достатньо точно зафіксована у вимогах на початкових етапах розробки, але вона може кількісно вимірюватися й послідовно уточнюватися в життєвому циклі програмного забезпечення.

Третій рівень стандартизованих показників якості програмного забезпечення важко повністю описати вимірюваними кількісними значеннями, та їх деякі субхарактеристики й атрибути мають описовий, якісний вигляд. Залежно від функціонального призначення програмного забезпечення за узгодженням із замовником можна визначати експертно ступінь необхідності (пріоритет) цих властивостей і бальні значення рівня реалізації їх атрибутів у життєвому циклі конкретного програмного забезпечення. Наприклад, не завжди може бути потрібна мобільність програм на інші операційні та апаратні платформи. У інших випадках мобільність можна оцінювати категоріями: відмінна, добра, задовільна або незадовільна. Такі оцінки проводяться експертно на основі аналізу можливої трудомісткості та тривалості, реалізації процесів перенесення комплексу програм на нову платформу.

Практичність тісно пов'язана з функціональною придатністю. Узагальнено цей показник можна відобразити трудомісткістю і тривалістю, які необхідні для вивчення й повного освоєння функцій і технології застосування відповідного програмного забезпечення. Кожна з субхарактеристик практичності має ряд якісних атрибутів, які можуть вибиратися й оцінюватися експертно з урахуванням функціонального призначення програмного забезпечення, а також надійності та ресурсної ефективності комплексу програм. Деякі з цих атрибутів можна кваліфікувати кількісно.

Супроводжуваність може мати обмежений характер повної заміни програм на знов розроблені версії і тим самим зливатися з процесами розробки або здійснюватися як безперервна підтримка множини користувачів консультаціями, адаптаціями і

корегуваннями програм. Залежно від цього розрізняються функції та трудомісткість процесів супроводу, яка може використовуватися як узагальнена якісна характеристика при виборі вимог до цього показника якості. Відповідно при проектуванні якісно можуть бути встановлені субхарактеристики супроводжуваності та описані необхідні їх властивості.

При будь-якому виді діяльності людям властиво неумисно помилятися, наслідки чого проявляються в процесі створення або застосування виробів або систем. У загальному випадку під помилкою слід розуміти дефект, похибку або ненавмисне спотворення об'єкта або процесу. При цьому передбачається, що відомий його правильний, еталонний стан, по відношенню до якого може бути визначено наявність відхилення – *дефекту або помилки*. Для систематичної, координованої боротьби з ними необхідні дослідження чинників, що впливають на якість програмного забезпечення зі сторони різних, існуючих і потенційно можливих дефектів у конкретних програмах. Це дозволить цілеспрямовано розробляти комплекси методів і засобів забезпечення якості складних програмних систем різного призначення при реально досяжному зниженні рівня дефектів проектування і розробки.

Відмінності між очікуваними й одержаними результатами функціонування програм можуть бути наслідком помилок не тільки у створених програмах і даних, але і *системних помилок в первинних вимогах специфікацій*, що є початковою базою при створенні програмного забезпечення [42 – 44]. Тим самим проявляється об'єктивна реальність, що полягає в неможливості абсолютної коректності початкових специфікацій складних програмних систем після проектування. На практиці в процесі розробки програмного забезпечення початкові вимоги уточнюються й деталізуються за погодженням між замовником і розробником. Базою таких уточнень є неформалізовані уявлення та знання фахівців, а також результати проміжних етапів життєвого циклу. Проте встановити помилковість початкових даних і специфікацій ще важче, ніж виявити помилки у створених програмах, оскільки принципово відсутні

формалізовані дані, які можна використовувати як еталонні, і їх замінюють на неформалізовані уявлення замовників і розробників.

Дефекти функціонування програмних засобів, що не мають зловмисних джерел або наслідків фізичних руйнувань апаратних компонентів, виявляються зовні як випадкові, мають різну природу і наслідки. Зокрема, вони можуть призводити до порушень функціональної працездатності та до відмов при використанні програмного забезпечення. У життєвому циклі на програми впливають різні негативні дестабілізуючі чинники, які можна розділити на внутрішні, властиві самим об'єктам уразливості, і зовнішні, зумовлені середовищем, в якому ці об'єкти функціонують.

Уведення строгих кількісних метрик у програмування має сприяти реалізації ряду практичних завдань [43]:

- передбачати ймовірну кількість помилок у системі із самого початку проектування;
- на основі аналізу фази проектування системи передбачати рівень складності подальшого супроводу;
- на основі аналізу початкового коду програм прогнозувати рівень складності процесів тестування і відсоток помилок, що залишаються;
- за оцінками складності фази проектування системи визначати кінцевий розмір коду;
- визначати кореляцію окремих характеристик програмного коду з якістю готової системи;
- контролювати стадії розвитку проекту;
- аналізувати явні і приховані дефекти;
- на основі експериментального порівняння виявляти кращі методи і технології.

У міру зростання актуальності програмних метрик на ринку стали з'являтися різні «вимірювальні» програми. Одні з них досліджували характеристики проектів і програмного забезпечення комплексно, інші орієнтувалися на цілком конкретні цілі: аналіз початкового коду, розмірів і структури окремих модулів.

8.4. Посвідчення якості та сертифікація програмних продуктів

Основною метою сертифікації програмних засобів і систем якості, що забезпечують їх життєвий цикл, є контроль і посвідчення якості технологій і продукції, гарантування їх високих споживчих властивостей [27]. Завдання полягає у підвищенні ефективності витрат у сфері створення й застосування кінцевого програмного продукту, а також у поліпшенні об'єктивності оцінок його характеристик і конкурентоспроможності. Формальною метою сертифікації є підготовка й ухвалення рішення щодо доцільності видачі заявнику сертифіката відповідності з урахуванням таких чинників:

- повноти, точності та достовірності початкового технічного завдання і специфікацій вимог, представлених у документації на програмне забезпечення, а також на технологію підтримки його життєвого циклу;
- достовірності та точності вимірювання й узагальнення результатів сертифікаційних випробувань і отримання адекватних показників якості кінцевих програмних продуктів і/або технологічних процесів їх створення;
- методології та якості інтерпретації даних про об'єкт випробувань і/або технології з урахуванням достовірності оцінок, кваліфікації та об'єктивності випробувачів, замовників і користувачів.

У міжнародних стандартах сертифікація відповідності визначена як дія третьої незалежної сторони, яка доводить, що забезпечується необхідна упевненість у тому, що належно ідентифікована продукція, процес або послуга відповідає конкретним стандартам і/або іншим нормативним документам. У поняття нормативні документи включені документи, що містять правила, загальні принципи або характеристики, що стосуються різних видів діяльності або їх результатів, стандарти, технічні умови, інструкції та регламенти із застосування конкретної продукції або технології.

Результатом позитивних випробувань є сертифікат відповідності – документ, виданий за правилами системи сертифікації, який засвідчує, що забезпечується необхідна упевненість у тому, що в належний спосіб ідентифікована продукція, процес або послуга відповідає конкретним стандартам і/або іншим нормативним документам. Термін дії сертифіката зазвичай обмежений або за часом (наприклад, 3 роки), або до проведення досить значної модифікації продукту або процесу. Сертифікат вступає в дію з моменту його реєстрації в державному реєстрі.

Фахівці третьої сторони мають право на розширення умов випробувань у межах вимог нормативної документації, при яких повинні забезпечуватися задана якість і безпека результатів застосування програмного забезпечення. При цьому як перша сторона в процесі сертифікації виступають розробники або постачальники програмного забезпечення та його компонентів, а другою стороною є замовники, споживачі або користувачі. Одна з цих двох сторін може виступати ініціатором – заявником на сертифікаційні випробування [27, 52].

Для посвідчення якості кінцевого продукту – програмних засобів і їх компонентів, слід сертифікувати технологічні процеси, що забезпечують їх життєвий цикл. Тому далі розглядаються спільно завдання сертифікації кінцевих об’єктів – програмних продуктів, а також технологій і систем якості, що забезпечують їх створення і вдосконалення. У ряді випадків сертифікат на технологію і систему якості підприємства може задовольнити споживача і замінити в контракті його вимоги наявності сертифікату на продукцію. При аналізі та організації процесів сертифікаційних випробувань технологій і/або об’єктів програмного забезпечення слід враховувати ряд базових компонентів методології сертифікації, що підлягають розгляду і затвердженню перед випробуваннями конкретного проекту [49 – 54]:

- цілі сертифікації – правові, економічні, формальні;
- початкові дані та документи, необхідні для проведення сертифікації, – стандарти, нормативні та експлуатаційні документи, їх структура і зміст;
- характеристики і класифікація об'єктів і/або процесів випробувань і сертифікації, а також необхідні значення характеристик і атрибутів якості;
- ресурси, необхідні для проведення випробувань – фінансові, кадри фахівців, апаратурна оснащеність, нормативні та програмно-інструментальні засоби.

Залежно від сфери застосування системи, від призначення і класу програмного забезпечення, його сертифікація може бути обов'язковою або добровільною. Первинні витрати на її проведення повинні нести ініціатори випробувань: або замовник і конкретні споживачі системи і програмного продукту, або її розробники і постачальники. Відповідно змінюються економічні та юридичні механізми їх взаємодії, розподіли відповідальності за дефекти і додатковий прибуток за підвищення якості сертифікованої продукції або технології. Розподіл відповідальності за збиток у користувачів при використанні дефектної продукції, що має сертифікат, рекомендується встановлювати в договорах на її постачання і на сертифікаційні випробування.

У початкових нормативних документах і вимогах повинні бути зосереджені всі функціональні та експлуатаційні характеристики, що забезпечують замовнику і користувачам можливість коректного застосування сертифікованого об'єкта і/або технологічного процесу в усьому різноманітті його функцій і характеристик якості. Для особливо важливої продукції (наприклад, програмних продуктів за державними замовленнями для оборонної техніки) результати позитивної сертифікації системи якості можуть використовуватися замовником як підстава для видачі ліцензії на виробництво й постачання цієї продукції. Така ліцензія дає перевагу відповідному постачальнику програмного забезпечення у конкурсах на

виробництво певної продукції і на укладення контракту на її постачання.

Сертифікація систем якості підприємства або проекту проводиться для оцінки переваг потенційного постачальника, за наявності пропозиції від нього щодо встановлення договірних відносин із замовником, на проектування або виробництво програмного забезпечення [27, 49 – 54]. Крім того, в рамках договірних відносин вона проводиться, щоб встановити, що система якості постачальника відповідає встановленим вимогам і застосовується повністю, а також для внутрішньої оцінки постачальником власної системи якості підприємства по відношенню до стандартів. Випробування для сертифікації здійснюються в проблемно-орієнтованих, технічно компетентних, випробувальних центрах або лабораторіях, акредитованих на право проведення випробувань, передбачених в її нормативних документах. Такі перевірки можуть проводитися за графіком або в разі важливих змін системи якості підприємства, процесів життєвого циклу і якості продукції, а також після проведення корегуючих дій програмного забезпечення. Проведення сертифікації систем якості підприємств зазвичай планується і здійснюється для цілей [27]:

- визначення відповідності або невідповідності технології і елементів системи якості встановленим вимогам стандартів;
- визначення ефективності вживаної системи якості підприємства з погляду відповідності поставленим цілям із забезпечення якості продукції;
- виявлення слабких місць у технології і системі якості підприємства, що найбільшою мірою негативно впливає на якість продукції;
- забезпечення можливості підприємству, що перевіряється, поліпшити свою систему якості;
- запобігання і скорочення рекламацій за недостатню якість і/або дефектну продукцію.

Сертифікаційні випробування є найбільш формалізованим і регламентованим етапом тестування як об’єктів – програмних продуктів, так і процесів їх створення, які підтримуються

значною кількістю стандартів і документів. При сертифікації зазвичай керуються такими основними початковими документами [27]:

- діючими міжнародними, державними й відомчими стандартами на проектування та випробування комплексів програм, на життєвий цикл програмного забезпечення, системи забезпечення і характеристики їх якості, а також на технологічну документацію;
- затвердженим замовником і узгодженим з розробником технічним завданням і/або специфікацією вимог, затвердженим комплектом експлуатаційної документації на програмне забезпечення і його компонентах, а також на систему забезпечення їх якості;
- програмою сертифікаційних випробувань на всі вимоги технічного завдання і положенням експлуатаційної документації;
- методиками випробувань по кожному розділу вимог технічного завдання і документації.

Підготовка регламентованої документації і такі випробування виправдані, коли необхідний тривалий розвиток і модифікація значних комплексів програм з гарантією малої ймовірності прояву дефектів і помилок. Таким чином, звичайний процес життєвого циклу програмного забезпечення доповнюється відповідною системою послідовних офіційних перевірок. При змінах програм необхідне підтвердження наявного сертифіката і проведення деякого мінімуму перевірок, що засвідчують коректність виконаних модифікацій. При цьому використовується система офіційних повідомлень користувачів щодо проведених змін, сповіщень про зміни і додаткові контрольні випробування, що підтверджують їх коректність.

Ресурси для сертифікації програмних засобів і систем якості підприємства повинні виділятися залежно від характеристик випробовуваного об’єкта або процесу [27]. Визначальними ресурсами сертифікації зазвичай є: можлива трудомісткість і тривалість випробувань, сукупна чисельність і структура колективу спеціалістів-сертифікаторів, а також їх кваліфікація і підготовленість до колективної перевірки конкретного типу

програмного забезпечення і його компонентів або системи якості підприємства. Апаратурна оснащеність випробувачів конкретного програмного продукту визначається, перш за все, ресурсами та іншими характеристиками ЕОМ, доступними для використання колективу фахівців при сертифікації.

Сертифікація складається з ряду організаційних процесів, що становлять *систему сертифікації* і підтримуються регламентованими процедурами і документами та повинні виконуватися кваліфікованими, атестованими експертами – інспекторами [27]. Для сертифікації підприємства-розробника і результатів його діяльності – програмних продуктів, моделями CMMI або профілю стандартів ISO рекомендується певна дисципліна, яка повинна бути адаптована до конкретних характеристик об'єктів і зовнішнього середовища життєвого циклу програмного забезпечення [14].

Роботи із сертифікації починаються з акредитації органу або випробувальної лабораторії, формування і представлення в Центральний орган із сертифікації заявки і комплекту документів для ухвалення рішення про доцільність акредитації. При позитивних результатах перевірки оформляється й видається атестат акредитації.

Положення про орган сертифікації або лабораторії є основним документом, що встановлює тематичну сферу акредитації, юридичний статус, функції, структуру, права і обов'язки, методи, засоби й організацію випробувань. Паспорт сертифікаційної лабораторії (центру) повинен містити відомості про оснащеність засобами обчислювальної техніки, необхідними для проведення випробувань, про персонал і кадровий склад, оснащеність інструментальними засобами проведення випробувань, про забезпечення нормативними, технічними і методичними документами, а також іншими ресурсами, необхідними для випробувань.

Керівництво з якості містить виклад принципів, опис методів і процедур, пов'язаних з виконанням основних функцій і завдань органу із сертифікації або лабораторії, які забезпечують якість проведених випробувань і довіру до результатів оцінок,

випробувань і експертиз. Керівництво з якості, як правило, включає розділи [48]:

- політика у сфері забезпечення якості проведення випробувань і експертиз;
- оснащення центру актуальними методологічними матеріалами і програмно-інструментальними засобами випробувань;
- формалізація вимог до об'єктів випробувань;
- політика в галузі технічної оснащеності центру і підвищення кваліфікації персоналу;
- архівація і контроль збереження документації результатів сертифікації.

Заявник для оцінювання продукції або процесу, що підлягає сертифікації, направляє в орган із сертифікації заявку за формою, прийнятою у системі сертифікації. Орган із сертифікації проводить роботу з підготовки і організації сертифікації продукції за заявкою. Ця робота включає [48]:

- вибір схеми сертифікації з урахуванням специфіки продукції (об'єм, технологія, вимоги нормативних документів тощо) і пропозицій розробника;
- визначення кількості та порядку відбору зразків і компонентів, що підлягають випробуванням, якщо це не вказано у стандартах;
- вибір і визначення акредитованої випробувальної лабораторії, яка повинна проводити випробування;
- підготовку проекту договору на виконання робіт.

Підготовча частина роботи із сертифікації закінчується випуском рішення за формою, прийнятою в системі сертифікації. Рішення разом з проектами договору на виконання робіт надається заявнику. При організації сертифікаційних випробувань здійснюється підбір і вивчення діючих нормативних документів на продукцію, заявлену до сертифікації, методів її випробувань і оцінки результатів.

Заявник ухвалює остаточні рішення про те, які елементи системи якості, ділянки і види організаційної і технічної діяльності підлягають перевірці при сертифікації в заданий

інтервал часу. Заявник повинен створити умови й подати документи для забезпечення процесів перевірок. Він має подати в орган із сертифікації протоколи випробувань, проведених при розробці та поставці продукції на виробництво, документи про випробування, виконані сторонніми випробувальними лабораторіями, та інші документи, що свідчать про відповідність технології або продукції встановленим вимогам. На основі аналізу поданих із заявкою документально підтверджених доказів відповідності його продукції встановленим вимогам, орган із сертифікації може ухвалити рішення про скорочення обсягу випробувань або про видачу сертифіката.

Випробування проводяться випробувальними лабораторіями, акредитованими на проведення тільки тих випробувань, які передбачені в їх нормативних, акредитаційних документах. При неможливості проведення випробувань на випробувальній базі акредитованої випробувальної лабораторії, випробування можуть проводитися персоналом акредитованої випробувальної лабораторії у виробника або споживача даної продукції з використанням власних засобів випробувальної лабораторії або засобів випробувань, що є в постачальника.

Процес сертифікації програмних продуктів і систем якості підприємства включає [27]:

- аналіз і вибір розробником або замовником (заявником), компетентних у даній галузі органу й атестованої лабораторії для виконання сертифікаційних випробувань;
- подачу заявником заявки на випробування в орган сертифікації та ухвалення сертифікаторами рішення щодо заявки, вибір схеми сертифікації, укладення договору на сертифікацію;
- ідентифікацію вимог до системи якості підприємства і/або до версії програмного продукту, що підлягають випробуванням;
- виконання сертифікаційних випробувань системи якості підприємства або версії програмного продукту сертифікаційною лабораторією;
- аналіз отриманих результатів і ухвалення рішення лабораторією і/або органом сертифікації про можливість видачі заявнику сертифіката відповідності;

- видачу органом сертифікації заявнику сертифіката й ліцензії на застосування знака відповідності та на випуск сертифікованої продукції – версій програмного продукту;
- здійснення інспекційного контролю органом сертифікації сертифікованої системи якості підприємства і/або продукції;
- проведення заявником корегуючих заходів при порушенні відповідності процесів системи якості і/або продукції встановленим вимогам і при неправильному застосуванні знака відповідності.

При перевірці відповідальності керівництва розробника за якість продукції повинно бути визначено наявність у підприємства або проекту документально оформлених політики, цілей і зобов'язань щодо якості, а також ступінь розуміння цієї політики, її практичне здійснення і підтримки в робочому стані на всіх рівнях організації. Має бути встановлено наявність на підприємстві представника керівництва, який незалежно від інших обов'язків має повноваження і несе відповідальність за постійне виконання вимог стандартів і нормативних документів системи якості. Слід перевіряти наявність вимог, процедур, засобів і навченого персоналу для практичної реалізації процесів системи якості, а також актуальність і систематичність оформлення документації на всі компоненти, вимоги і положення системи якості, що є інтегрованим процесом, що має місце впродовж всього життєвого циклу програмного забезпечення. Перевірки системи якості повинні включати визначення [27]:

- наявності та повноти технологічної документації і дотримання її вимог на практиці;
- стану засобів технологічного оснащення і наявності системи їх технічного обслуговування;
- наявності та ефективності системи контролю і випробувань;
- стану засобів вимірювань і випробувань;
- наявності системи виявлення й усунення виявлених недоліків продукції або технології.

На підставі випробувань оцінюються отримані результати та обґрунтовуються висновки щодо відповідності або невідповідності продукції або процесів вимогам нормативних

документів. Протоколи випробувань подаються в орган із сертифікації, а також заявнику на його вимогу. Протоколи випробувань підлягають зберіганню протягом термінів, встановлених у правилах систем сертифікації продукції і в документах випробувальної лабораторії, але не менше трьох років.

Після отримання і перевірки комплектності та якості документації фахівцями випробувальної лабораторії слід провести експертизу ступеня реального застосування системи якості на підприємстві. Випробування починаються зі складання *програми перевірки системи якості*, яка повинна служити робочим планом проведення подальших робіт. Програма є внутрішнім робочим документом випробувальної лабораторії і повинна містити перелік робіт, що деталізується відповідно до специфіки підприємства-розробника, і включає аналіз повноти та якості поданих початкових документів і ступеня їх практичного застосування при проектуванні, розробці і постачанні програмного забезпечення. Експертиза застосування процедур системи якості здійснюється випробувальною лабораторією на робочих місцях підприємства, що забезпечує життєвий цикл програмного забезпечення. Перевірки проводяться за наявності на робочих місцях фахівців-розробників відповідних документів і щодо повноти використання їх положень і рекомендацій. Аналізи стану проекту і внутрішні перевірки системи якості, процесів і/або продукції повинні проводитися персоналом, незалежним від осіб, безпосередньо відповідальних за виконання цих робіт.

Методики перевірок якості розробки повинні бути забезпечені необхідними ресурсами для виконання програми випробувань, доповнені методиками планування і розробки приватних процедур перевірок. Методики повинні містити: об’єкти і цілі випробувань; оцінювані показники якості; умови і порядок випробувань; методи обробки, аналізу і оцінки результатів випробувань; технічне забезпечення випробувань і звітність. Слід указувати технічні та програмні засоби, використовувані під час проведення випробувань, і порядок проведення випробувань, а також очікувані результати

перевірок. Повинні бути розроблені методики контролю за корегуваннями, діями із виправлення дефектів, якщо в службу управління перевірок надійде такий запит. Служба управління програмами випробувань повинна розробити методики збереження конфіденційності будь-якої інформації про випробування, а також дані, наявні в експертів.

Протоколи випробувань подаються заявнику і в орган із сертифікації. Заявник може подати в орган із сертифікації протоколи випробувань з урахуванням термінів їх дії, проведених при розробці та поставці продукції на виробництво, або документи про випробування, виконані вітчизняними або зарубіжними випробувальними лабораторіями, акредитованими або визнаними в системі сертифікації. На підставі протоколів сертифікаційних випробувань оцінюються отримані результати і обґрунтовуються зроблені висновки про відповідність або невідповідність продукції вимогам нормативних документів.

Висновок за наслідками сертифікаційних випробувань розробляється сертифікаторами й містить узагальнені відомості про результати випробувань і обґрунтування доцільності видачі сертифіката. У разі отримання негативних результатів сертифікаційних випробувань ухвалюється рішення про відмову у видачі сертифіката відповідності. Після доопрацювання продукції, що сертифікується, або системи якості випробування можуть бути повторені. Результати аналізу стану технології або якості продукції оформляються актом, в якому даються оцінки по всіх позиціях програми випробувань і містяться висновки, що включають загальну оцінку стану виробництва і продукції, необхідність корегуючих заходів. Акт використовується органом із сертифікації разом з протоколами випробувань, заявкою для видачі та визначення терміну дії сертифіката на програмний продукт, періодичності інспекційного контролю, а також для складання корегуючих заходів.

За наслідками сертифікаційних випробувань і експертизи документації ухвалюється рішення про видачу сертифіката. Крім того, підприємству-заявнику можуть бути направлені пропозиції щодо усунення передбачуваних причин негативних результатів

випробувань, які після доопрацювання продукції випробування, що сертифікується, можуть бути повторені.

Орган із сертифікації після аналізу протоколів випробувань, оцінки виробництва, сертифікації системи якості, аналізу документації, вказаної в рішенні по заявці, здійснює оцінку відповідності продукції встановленим вимогам, оформляє сертифікат на підставі висновку експертів і реєструє його. При внесенні змін у конструкторську або експлуатаційну документацію, які можуть вплинути на якість системи або програмний продукт, що засвідчується при сертифікації, заявник повинен сповістити про це орган із сертифікації для ухвалення рішення про необхідність проведення додаткових випробувань. Після реєстрації сертифікат набуває чинності та надсилається підприємству-заявнику. Одночасно з видачею сертифіката підприємству-заявнику може видаватися ліцензія на право застосування знака відповідності.

За сертифікованими програмними продуктами в процесі їх експлуатації протягом усього терміну дії сертифіката відповідності повинен здійснюватися інспекційний контроль. Інспекційний контроль проводиться у формі періодичних і позапланових перевірок дотримання вимог до якості технології та сертифікованої продукції. Об’єктами контролю, залежно від схеми сертифікації, є сертифікована продукція, система якості або стабільність виробництва підприємства-розробника. При визначенні періодичності й обсягу інспекційної перевірки враховуються такі чинники: ступінь потенційної небезпеки програмного продукту, стабільність виробництва, обсяг випуску, наявність і застосування системи якості при розробці, інформація про результати випробувань продукту і його виробництва, проведених виробником, органами державного контролю і нагляду.

Результати інспекційного контролю оформляються актом, в якому дається оцінка результатів випробувань зразків та інших перевірок, робиться загальний висновок про стан виробництва сертифікованої продукції і можливості збереження дії виданого сертифіката. Акт зберігається в органі із сертифікації, а його копії

надаються розробнику та організаціям, які брали участь в інспекційному контролі. За наслідками інспекційного контролю орган із сертифікації може припинити або відмінити дію сертифіката й анулювати ліцензію на право застосування знака відповідності в разі невідповідності продукції вимогам нормативних документів, контрольованих при сертифікації, а також у випадках [27]:

- принципових змін моделі зрілості, профілю стандартів, нормативних документів на продукцію або методу випробувань;
- зміни конструкції (складу), комплектності продукції;
- зміни організації або технології розробки і виробництва;
- невиконання вимог технології, методів контролю і випробувань, системи якості, якщо перераховані зміни можуть викликати невідповідність продукції вимогам, контрольованим при сертифікації.

Рішення про припинення дії сертифіката і ліцензії на право застосування знака відповідності не ухвалюється в тому випадку, якщо шляхом корегуючих заходів, узгоджених з органом із сертифікації, який його видав, заявник може усунути виявлені причини невідповідності та підтвердити без повторних випробувань в акредитованій лабораторії відповідність продукту або процесів нормативним документам. Якщо цього зробити не можна, то дія сертифіката відміняється й ліцензія на право застосування знака відповідності анулюється. Інформація про припинення або відміну дії сертифіката доводиться органом із сертифікації, який його видав, до відома заявника, споживачів та інших зацікавлених організацій. Дія сертифіката і право маркування продукції знаком відповідності можуть бути відновлені при виконанні підприємством-розробником таких умов [27]:

- виявлення причин невідповідності та їх усунення;
- подання до органу із сертифікації звіту про виконану роботу з поліпшення і забезпечення якості продукції;
- проведення за методиками та під контролем органу із сертифікації додаткових випробувань продукції і отримання позитивних результатів.

Склад і зміст документації для сертифікації системи якості підприємства залежать від характеристик проектування, розробки і модифікації програмних засобів, а також від вимог до їх якості та особливостей технологічного середовища.

Контрольні запитання та завдання

1. На сайті за адресою http://www5.in.tum.de/~huckle/bugse.html є велика колекція інформації щодо програмних дефектів та помилок, які призвели свого часу до значних збитків. Опишіть у вигляді короткого реферату причини та наслідки конкретного прикладу збою комп'ютерної системи (згідно з обраним варіантом). Як можна було цього уникнути?
2. Чи можна створити абсолютно працездатне та безвідмовне програмне забезпечення?
3. Опишіть основні поняття якості програмних продуктів.
4. Охарактеризуйте визначальні чинники, що впливають на якість програмних продуктів.
5. Які стандарти щодо забезпечення якості програмних продуктів ви знаєте? Охарактеризуйте їх.
6. Обґрунтуйте необхідність сертифікації якості програмних продуктів.
7. Опишіть процедуру надання сертифіката якості програмним продуктам.

Список літератури

1. ACM/IEEE Computing Curriculum – Software Engineering. Final Report (May 21, 2004).
2. ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices, Software Engineering Code of Ethics and Professional Practice, Version 5.2 [Electronic resource]. – Access mode : http://www.acm.org/serving/se/code.htm
3. Гвоздёва В. А. Введение в специальность программиста : учебное пособие / В. А. Гвоздёва – М. : ИД «ФОРУМ», 2007. – 208 с.
4. Макконнелл С. Профессиональная разработка программного обеспечения / С. Макконнелл – СПб. : Символ-Плюс, 2006. – 240 с.
5. Макконнелл С. Совершенный код. Мастер-класс / С. Макконнелл – М. : Издательство «Русская редакция», 2013 – 896 с.
6. Ford G. A Mature Profession of Software Engineering / G. Ford, N. Gibbs – Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1996.
7. Guide to the Software Engineering Body of Knowledge (SWEBOK) [Electronic resource]. – Access mode : www.swebok.org
8. Лакман Макдауэлл Г. Карьера программиста. Как устроиться на работу в Google, Microsoft или другую ведущую IT-компанию / Г. Лакман Макдауэлл – СПб. : Питер, 2012. – 416 с.
9. Брукс Ф. Мифический человеко-месяц или как создаются программные системы / Ф. Брукс – СПб. : Символ-Плюс, 2001. – 171 с.
10. Соммервилл И. Инженерия программного обеспечения / И. Соммервилл – М. : Вильямс, 2002. – 623 с.
11. Липаев В. В. Программная инженерия. Методологические основы : учебное пособие / В. В. Липаев – М. : ТЕИС, 2006. – 608 с.

12. Архипенков С. Лекции по управлению программными проектами / С. Архипенков – М. : Диалог МИФИ, 2009. – 128 с.
13. Агапова А. С. Оценка и аттестация зрелости процессов создания и сопровождения программных средств и информационных систем (ISO/IEC TR 15504 – CMM) / А. С. Агапова – М. : Книга и бизнес, 2001. – 348 с.
14. Chrissis M. CMM: Guidelines for Process Integration and Product Improvement / M. Chrissis, M. Konrad, S. Shrum. – Addison-Wesley Professional, 2006. – 421 p.
15. Фланнес С. Навыки работы с людьми для менеджеров проектов / С. Фланнес, Дж. Левин – М. : Вильямс, 2004. – 380 с.
16. Мищенко В. О. CASE-оценка критических программных систем. В 3-х томах. Том 1. Качество / В. О. Мищенко, О. В. Поморова, Т. А. Говорущенко – Харьков: Нац. аэрокосмический университет «ХАИ», 2012. – 201 с.
17. Цена программной ошибки [Електронний ресурс]. – Режим доступу : http://www.cusoft.ru/error.php
18. Software goes wrong, we all know that, but just how wrong can it go? [Electronic resource]. – Access mode : http://www.datareservoir.co.uk/bugs/
19. 20 Famous Software Disasters [Electronic resource]. – Access mode : http://sandipsandilya.wordpress.com/2011/01/17/20-famous-software-disasters/
20. Explaining Settlement Fails [Electronic resource]. – Access mode : http://www.ny.frb.org/research/current_issues/ci11-9/ci11-9.html
21. Cost of a bug within a software lifecycle [Electronic resource]. – Access mode : http://www.testically.org/2012/02/09/cost-of-a-bug-within-a-software-lifecycle/
22. Чернев Д. А. Технология разработки програмного обеспечения / Д. А. Чернев – Ташкент, 2004. – 224 с.
23. Фатрелл Р. Т. Управление программными проектами: достижение оптимального качества при минимуме затрат / Р. Т. Фатрелл, Д. Ф. Шафер, Л. И. Шафер – М. : Издательский дом «Вильямс», 2003. – 1136 с.

24. Брауде Э. Дж. Технология разработки программного обеспечения / Э. Дж. Брауде. – СПб. : Питер, 2004. – 655 с.
25. Зыль С. Проектирование, разработка и анализ програмного обеспечения систем реального времени / С. Зыль – СПб. : БХВ-Петербург, 2010. – 336 с.
26. Липаев В. В. Выбор и оценивание характеристик качества программных средств: методы и стандарты / В. В. Липаев – М. : Синтег, 2001. – 224 с.
27. Липаев В. В. Сертификация программных средств. Учебник / В. В. Липаев – М.: СИНТЕГ, 2010. – 348 с.
28. CMM Capability maturity model for Software // From Wikipedia, the free encyclopedia [Electronic resource]. – Access mode : http://en.wikipedia.org/wiki/Capability_Maturity_Model
29. Стандарты качества и моделирование качества программного обеспечения [Електронний ресурс]. – Режим доступу : http://softwarequality.narod.ru/qualitymodelstandards.html
30. Современные модели качества программного обеспечения [Електронний ресурс]. – Режим доступу : http://www.interface.ru/fset.asp?Url=/misc/qs.htm
31. Стандарты качества и сложности программного обеспечения [Електронний ресурс]. – Режим доступу : http://www.rol.ru/news/it/press/cwm/25_96/teh.htm
32. Software Process Improvement and Capability Determination (SPICE) [Electronic resource]. – Access mode : http://www.sqi.gu.edu.au/spice
33. Software Process Improvement in Regions of Europe [Electronic resource]. – Access mode : http://www.cse.dcu.ie/spire
34. Коган Б. И. Автоматизация оценивания качества программного обеспечения [Електронний ресурс]. – Режим доступу: http://www.febras.ru/~conf/seminar/kogan.html
35. Software Design – IEEE Guide to the Software Engineering Body of Knowledge – SWEBOK. Software Quality [Electronic resource]. – Access mode : http://www.computer.org/portal/web/swebok/html/ch11

36. Документирование программного обеспечения и эффективный процесс разработки [Електронний ресурс] – Режим доступу : http://creograf.ru/?messPress_ShowR_161=1
37. Скляр В. В. Оценка качества и экспертиза программного обеспечения. Лекционный материал / В. В. Скляр – Харьков : НАУ «ХАИ», 2008. – 204 с.
38. Харченко В. С. Верификация программного обеспечения / В. С. Харченко, В. В. Скляр, А. А. Гордеев – Харьков : НАУ «ХАИ», 2006. – 132 с.
39. Крайер Э. Успешная сертификация на соответствие нормам ИСО серии 9000 / Э. Крайер – М. : ИЗДАТ, 1999. – 211 с.
40. Вендров А. М. Современные технологии создания программного обеспечения [Електронний ресурс] – Режим доступу : http://citforum.ru/programming/application/program/
41. Якунин Ю. Ю. Технологии разработки программного обеспечения / Ю. Ю. Якунин – Красноярск: ИПК СФУ, 2008. – 156 с.
42. Вигерс К. И. Разработка требований к программному обеспечению / К. И. Вигерс – М. : Издательство «Русская редакция», 2004. – 296 с.
43. Кобёрн А. Современные методы описания функциональных требований к системам / А. Кобёрн – М. : Лори, 2002. – 321 с.
44. Леффингуелл Д. Принципы работы с требованиями к программному обеспечению / Д. Леффингуелл, Д. Уидриг – М. : Вильямс, 2002. – 251 с.
45. IEEE Standard 610.12-90. Standard Glossary of Software Engineering Terminology [Electronic resource]. – Access mode : http://web.ecs.baylor.edu/faculty/grabow/Fall2011/csi3374/secure/Standards/IEEE610.12.pdf
46. IEEE Standard 1219-98. IEEE Standard for Software Maintenance [Electronic resource]. – Access mode : http://www.cs.uah.edu/~rcoleman/CS499/CourseTopics/IEEE_Std_1219-1998.pdf
47. IEEE Standard 14764-2006 (ISO/IEC 14764). Standard for Software Engineering – Software Maintenance [Electronic

resource]. – Access mode : http://webstore.iec.ch/preview/info_isoiec14764%7Bed2.0%7Den.pdf
48. ISO 9001: 2008-12. Quality management systems – Requirements [Electronic resource]. – Access mode : http://nads.gov.ua/sub/data/upload/publication/cherkaska/ua/6331/iso-9001_2008_dvuyazychnyj.pdf?s398224032=63e9aac82dc234cb077145d99b22b9aa
49. Мацяшек Б. Анализ и проектирование систем. Разработка инфорамционных систем с использованием UML / Б. Мацяшек, А. Лешек – М. : Издательский дом «Вильямс», 2005. – 316 с.
50. Вендров А. М. Проектирование программного обеспечения экономических информационных систем / А. М. Вендров – М. : Финансы и статистика, 2002. – 148 с.
51. Бахтизин В. В., Глухова В. А. Применение метрик сложности при разработке программных средств [Електронний ресурс]. – Режим доступу : http://www.giac.unibel.by/docs/pdf/1-2005/s10-1-2005.pdf
52. Марков А. С. Модели оценки и планирования испытаний программных средств по требованиям безопасности информации / Вестник МГТУ им. Н.Э. Баумана. Сер. «Приборостроение» Специальный выпуск «Технические средства и системы защиты информации», 2011. – с. 90–103.
53. Тейер Т. Надежность программного обеспечения. / Т. Тейер, М. Липов, Э. Нельсон – М. : Мир, 1981. – 326 с.
54. Эмпирические модели надежности программного обеспечения // [Електронний ресурс]. – Режим доступу : http://info-tehnologii.ru/kac_sr/Mod_nad/Emp_mod/index.html

Навчальне видання

ПРОФЕСІЙНА ПРАКТИКА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Навчальний посібник

Укладач ***Жихаревич Володимир Вікторович***

Відповідальний за випуск *С.Е. Остапов*
Літературний редактор *О.В. Колодій*

Друкарня видавництва “Рута”

384