МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДНІПРОВСЬКИЙ ДЕРЖАВНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
Дранишников Л.В.

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни

«Менеджмент проектів програмного забезпечення»

для здобувачів вищої освіти першого (бакалаврського) рівня
зі спеціальності 121 Інженерія програмного забезпечення
за освітньо-професійною програмою
«Інженерія програмного забезпечення»

ЗАТВЕРДЖЕНО:
редакційно-видавничою секцією
науково-методичної ради ДДТУ
« » _____ *20* р., протокол №

Кам'янське 2019

ЗМІСТ

ТЕМА 1 (4 години). **Введення в програмну інженерію**. Історія та основні поняття. Еволюція підходів до управління програмними проектами. Моделі процесу розробки ПЗ. Що треба робити для успіху програмного проекту

ТЕМА 2 (4 години). **Управління проектами**. Визначення і концепції. Проект — основа інновацій. Критерії успішності проекту. Проект і організаційна структура компанії. Організація проектної команди. Життєвий цикл проекту. Фази і продукти.

ТЕМА 3 (4 години). **Ініціація проекту**. Управління пріоритетами проектів. Концепція проекту. Цілі та результати проекту. Допущення і обмеження. Ключові учасники і зацікавлені сторони. Ресурси. Терміни. Ризики. Критерії приймання. Обґрунтування корисності.

ТЕМА 4 (4 години). **Планування проекту.** Уточнення змісту і складу робіт. Планування керування вмістом. Планування організаційної структури. Планування управління конфігураціями. Планування управління якістю. Базове розклад проекту.

ТЕМА 5 (4 години). **Управління ризиками проекту**. Основні поняття. Планування управління ризиками. Ідентифікація ризиків. Якісний аналіз ризиків. Кількісний аналіз ризиків. Планування реагування на ризики. Головні ризики програмних проектів та способи реагування. Управління проектом, спрямоване на зниження ризиків. Моніторинг і контроль ризиків.

ТЕМА 6 (4 години). **Оцінка трудомісткості і термінів розробки ПО**. Оцінка — імовірнісне твердження. Негативні наслідки «агресивної» розкладу. Прагматичний підхід. Метод PERT. Огляд методу функціональних точок. Основи методики COCOMO II

ТЕМА 7 (8 годин) **Формування команди**. Лідерство і управління. Правильні люди. Мотивація. Ефективна взаємодія.

ТЕМА 8 (4 години). **Реалізація проекту**. Робоче планування. Принципи кількісного управління. Завершення проекту.

Лекція 1-3. ТЕМА 1 ***Введення в програмну інженерію. Історія та основні поняття. Еволюція підходів до управління програмними проектами. Моделі процесу розробки ПЗ. Що треба робити для успіху програмного проекту***

Програмна інженерія є застосування певного систематичного вимірної підходу при розробці, експлуатації та підтримки програмного забезпечення [1]. Термін software (програмне забезпечення, ПЗ) ввів в 1958 році всесвітньо відомий статистик Джон Тьюкей (John Tukey). Термін software engineering (програмна інженерія) вперше з'явився у назві конференції НАТО, що відбулася в Німеччині в 1968 році, присвяченій так званої кризи програмного забезпечення. З 1990-го по 1995 рік велася робота над міжнародним стандартом, який повинен був дати єдине уявлення про процеси розробки програмного забезпечення. В результаті був випущений стандарт ISO/IEC 12207 [2]. У 2004 році в галузі був створений основоположна праця «Керівництво до зводу знань з програмної інженерії» (SWEBOK) [3], в якому були зібрані основні теоретичні і практичні знання, накопичені в цій галузі.
Щоб уникнути двозначностей, але не претендуючи на академічність, дозволю собі ввести робочі визначення ряду термінів, які я буду надалі активно використовувати.

Програмування — процес відображення певної множини цілей на безліч машинних команд і даних, інтерпретація яких на комп'ютері або обчислювальному комплексі забезпечує досягнення поставлених цілей.
Цілі можуть бути будь-які: відтворення звуку в динаміці ПК, розрахунок траєкторії польоту космічного апарата на Марс, друк річного балансового звіту і т. д. Важливо те, що вони мають бути визначені. Це звучить банально, але скільки б про це не говорили раніше, як і раніше, доводиться стикатися з програмними проектами, в яких відсутні які-небудь певні цілі.

Це відображення може бути дуже простим, наприклад, перфорування машинних команд і даних на перфокартах. А може бути багатоступеневим і дуже складним, коли спочатку цілі відображаються на вимоги до системи, вимоги — на високорівневу архітектуру і специфікації компонентів, специфікації — на дизайн компонентів, дизайн — на вихідний код. Далі вихідний код за допомогою компіляторів і збирачів відображається на код розгортання, код розгортання на виклики функцій З оточення (ОС, проміжне, бази даних), яке може розташовуватися на безліч комп'ютерів, об'єднаних в мережу, і тільки після цього — в машинні команди і дані.

Професійне програмування (синонім виробництво програм) — діяльність, спрямована на отримання доходів за допомогою програмування.

Принциповою відмінністю від просто програмування є те, що є або, принаймні, передбачається деякий споживач, який готовий платити за використання програмного продукту. Звідси випливає важливий висновок про те, що професійне виробництво програм це завжди колективна діяльність, в якій беруть участь мінімум дві людини: програміст і споживач.
Професійний програміст — людина, яка займається професійним програмуванням.

Професійного програміста слід відрізняти від професіонала (майстри в програмуванні). Розкид професійної майстерності в програмуванні досить широкий і далеко не кожен, хто заробляє на життя програмуванням, є майстром, але про це пізніше.

Рис. 1 Життєвий цикл програмного продукту

Програмний продукт — сукупність програм і супровідної документації по їх встановлення, настроювання, використання і доопрацювання.
Відповідно до стандарту [2] життєвий цикл програми, програмної системи, програмного продукту включає в себе розробку, розгортання, підтримку та супровід. Якщо програмний продукт не коробковий, а досить складний, то його розгортання у клієнтів, як правило, реалізується окремими самостійними проектами впровадження. Супровід включає в себе усунення критичних несправностей в системі і часто реалізується не як проект а, як процесна діяльність. Підтримка полягає в розробці нової функціональності, переробці вже існуючої функціональності, у зв'язку зі зміною вимог, і поліпшенням продукту, а також усунення некритичних зауважень до ПЗ, виявлених при його експлуатації (Рис. 1). Життєвий цикл програмного

продукту завершується виведенням продукту з експлуатації і зняттям його з підтримки і супроводу.

Процес розробки ПЗ — сукупність процесів, що забезпечують створення і розвиток програмного забезпечення.

Найпоширеніший процес розробки ПЗ, який довелося спостерігати за роки роботи в галузі, можна назвати «як вийде». Це не означає, що процесу як такого немає. Він є і, як правило, забезпечує розробку ПЗ при прийнятних витратах і якості, але цей процес не документований, є «знанням зграї», тримається на людях і передається з покоління в покоління. Цілеспрямована робота по оцінці ефективності та поліпшенню процесу не ведеться.

Модель процесу розробки ПЗ — формалізоване подання процесу розробки ПЗ. Часто при описі процесів замість слова модель вживається термін методологія, що призводить до невиправданого розширення даного поняття.
Згідно SWEBOK 2004, програмна інженерія включає в себе 10 основних та 7 додаткових галузей знань, на яких базуються процеси розробки ПЗ.

До основних галузей знань відносяться такі області:

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 у програмній інженерії.
8. Software engineering process — процеси програмної інженерії.
9. Software engineering tools and methods — інструменти і методи.
10. Software quality — якість програмного забезпечення.

Додаткові галузі знань включають в себе:

1. Computer engineering — розробка комп'ютерів.
2. Computer science — інформатика.
3. Management — загальний менеджмент.
4. Математика — математика.
5. Project management — управління проектами.
6. Quality management — управління якістю.
7. Systems engineering — системне проектування.

Все це необхідно знати і вміти застосовувати, для того щоб розробляти ПЗ. Як бачимо, управління проектами, про який ми будемо говорити далі, лише одна з 17 областей знань програмної інженерії, і то допоміжна. Однак

основною причиною більшості провалів програмних проектів є саме застосування неадекватних методів управління розробкою.

Відмінності програмної інженерії від інших галузей

Standish Group, проаналізувавши роботу сотень американських корпорацій і підсумки виконання кількох десятків тисяч проектів, пов'язаних з розробкою ПО, у своїй доповіді з промовистою назвою «Хаос» [4] прийшла до таких невтішних висновків (Рис. 2):

• Тільки 35 % проектів завершилися в строк, що не перевищили запланований бюджет і реалізували всі необхідні функції і можливості.

• 46 % проектів завершилися з запізненням, витрати перевищили запланований бюджет, необхідні функції не були реалізовані в повному обсязі. Середнє перевищення строків склало 120%, середнє перевищення витрат 100%, зазвичай виключалося значне число функцій.

• 19 % проектів повністю провалилися і було анульовано до завершення.

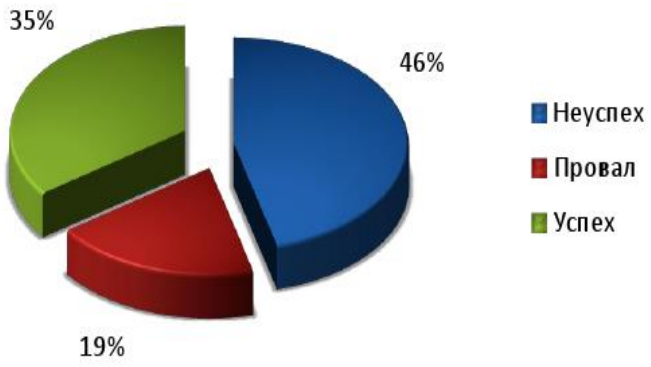


Рис. 2. Результати аналізу успішність програмних проектів за 2006 рік

Відразу дам відповіді на одвічні російські питання «Хто винен?» і «Що робити?».

Хто винен? Ніхто. Як ніхто не винен в тому, що на небі хмари, що йде дощ, що дме вітер. Оскільки «хаосу» не було, і немає, а є лише Богом дана (для атеїстів — об'єктивна) реальність, яка полягає в особливій специфіці виробництва програм, порівняно з будь-якою іншою виробничою діяльністю, тому що те, що роблять програмісти — нематеріальне, це колективні ментальні моделі, записані на мові програмування. І з цією специфікою ми зобов'язані рахуватися, якщо, звичайно, не хочемо «дути проти вітру».

Що робити? Керувати людьми. Успіх, як і провал, проектів по виробництву по лежать в області психології.

Те, що виробляють програмісти нематеріальне — це колективні думки та ідеї, виражені на мові програмування. Я не кажу, що виробництво ПО

надскладна інтелектуальна діяльність. Галузь ще тільки зароджується. Час входження в професію сильно менше, ніж у інших інженерних дисциплінах. Розробляти ЗА точно не складніше, ніж робити ракети. Просто в силу унікальності галузі досвід професіоналів, накопичений в матеріальному виробництві і викладений в стандарті PMI PMBOK [6], мало сприяє успіху в управлінні програмним проектом. Керувати розробкою ПЗ треба інакше.

Творчість — це інтелектуальна діяльність людини, закони якої нам невідомі. Якщо б ми знали закони творчості, і картини, і вірші, і музику, і програми вже давно б створювали комп'ютери. Творче начало це те, що ріднить програмування з наукою і мистецтвом.

Творчість в програмуванні починається з визначення цілей програми і закінчується тільки тоді, коли в її коді, написаному на якій-небудь мові програмування, поставлена остання крапка. Спроби поділяти програмістів на творчу еліту, архітекторів і проектувальників, і нетворчих програмістів-кодерів не мають під собою об'єктивних підстав. Навіть якщо алгоритм програми строго визначений математично, два різних програміста його закодують по-різному, і отримана програма буде мати різні споживчі якості.

Творчість нерозривно пов'язане з натхненням, а це річ примхлива і непередбачувана (пам'ятаєте знаменитий сон Д. І. Менделєєва, про Періодичну таблицю елементів його імені?). Знаю тільки, що без натхнення в програмуванні не обійтися. І чим складніше завдання, тим важче винести це натхнення з підсвідомості. Іноді для цього потрібні години, а іноді тижня.

Програмування це не мистецтво, в тому сенсі, що воно не є творчим відображенням і відтворенням дійсності в художніх образах. Про мистецтво в програмуванні можна і треба говорити тільки в сенсі вміння, майстерності, знання справи, як і в будь-якій іншій професії. І як у будь-якій іншій професії программистское майстерність може доставляти справжнє естетичне задоволення, але тільки для людей, причетних до цієї професії.

Програмування це не наука. Напрацювання математиків в області логіки, теорії інформації, чисельних методів, реляційної алгебри, теорії графів та деяких інших дисциплінах на частку відсотка не покривають складність програмістських завдань. У програмуванні немає системи знань про закономірності створення програм. Навіть видатні програмісти не візьмуть на себе сміливість стверджувати про архітектуру нової програмної системи те, що вона буде успішною. Хоча в програмуванні вже накопичено певний досвід провалів, який може дозволити досвідченому програмісту побачити в архітектурі нової системи антипаттерны — джерела майбутніх серйозних проблем. Але не більше того.

Існуючий стан програмної інженерії нагадує велику кулінарну книгу з численними описами рецептів одного разу успішно приготовлених страв з інгредієнтів, які в майбутньому вже не буде. Завтра в новій системі будуть інші обчислювальні машини, технології, мови програмування, інструменти та навколишнє ПО, нові проблеми взаємодії з якими обов'язково доведеться вирішувати.

Професійне творчість програміста принципово відрізняється від творчості в науці і мистецтві. Программистские завдання з кожним роком стають все складніше і об'ємніше, а терміни, за які потрібно вирішити ці завдання, навпаки, з кожним роком скорочуються. Тому сучасні програми створюються колективами від кількох до тисяч програмістів, у той час як творчі діячі науки і мистецтва працюють, як правило, поодинці.

Є ще щось, що відрізняє працю професійного програміста від вченого, художника, композитора і поета. Предметом діяльності вчених є спрощені моделі, в яких вони можуть абстрагуватися від більшості деталей реального світу, не істотних для їх цілей. Математик, доводячи нову теорему про тензорах, не дбає ні про що, крім системи постулатів, покладених в основу диференціальної геометрії. Фізик, описуючи динаміку рідини в трубі, абстрагується від того, як рухаються і стикаються молекули і від того, як рухаються планети навколо Сонця. Діячі мистецтва теж багато в чому оперують абстракціями. Поету, композитору, художнику достатньо лише зробити натяк, абрис об'єкта творчості, і на цьому його робота закінчена. Решту нехай додумає читач, слухач, глядач.

Програміст теж працює з абстракціями, але йому доводиться тримати в голові набагато більше абстракцій, ніж будь-якому вченому. Абстракції супроводжують програмісту на всіх рівнях розробки програми від опису її цілей до виконуваного машинного коду. І цих рівнів можуть бути десятки. І на кожному рівні абстракцій їх деталей стає все більше і більше.

Додатково до абстрактного мислення, програміст повинен володіти сильно вираженим системним мисленням, щоб утримувати численні взаємозв'язки, що існують на всіх рівнях програмістських абстракцій, а також взаємозв'язки між цими рівнями. Ще однією складністю є те, що всі ці абстракції і взаємозв'язку між ними змінюються в часі, і програміст повинен враховувати цю динаміку.

Крім того, програміст повинен володіти маніакальною посидючістю, зосередженістю і завзятістю для перебору всіх можливих варіантів поведінки своїх абстракцій і досконалого опрацювання всіх деталей. Опрацювання повинна бути абсолютно точною і не повинна містити жодної помилки, неправильного, зайвого або відсутнього символу вихідного коду (а це часом

мільйони рядків). Інструменти програмування: синтаксичні аналізатори, компілятори та ін., — лише трохи допомагають у цій роботі.

Ще одна особливість, яка притаманна программистскому творчості, це постійне оновлення інформаційних технологій, які програмісту необхідно знати і успішно застосовувати у своїй роботі. Тому професійний програміст повинен, як сказав один з наших колишніх вождів, «вчитися, вчитися і вчитися». Програміст повинен утримувати в голові, постійно поповнювати й активно застосовувати на практиці гігабайти професійної інформації. Це пристрій комп'ютерів, комп'ютерних мереж і мережеві протоколи. Це операційні системи і мови програмування. Це програмні інтерфейси проміжного та прикладних бібліотек з особливостями і багами їх реалізації в конкретних продуктах. Це технологічні стандарти, технології розробки та інструменти, які їх підтримують. Це архітектури програмних систем, патерни і антипаттерны проектування і багато-багато іншої інформації.

Ще на початку 70-х чудовий вчений академік А. П. сказав Єршов [8]: «Програміст повинен володіти здатністю першокласного математика до абстракції і логічного мислення в поєднанні з эдисоновским талантом споруджувати все, що завгодно, з нуля і одиниць. Він повинен поєднувати акуратність бухгалтера з проникливістю розвідника, фантазію автора детективних романів з тверезою практичністю економіста».

Програмування — не мистецтво і не наука — це ремесло. Сьогодні ми так само далекі від індустріальної розробки програм, як і 50 років тому.

А оскільки це ремесло, то людина, навчився писати програми на C++, буде так само далекий від професіонала, як учень третього класу середньої школи, навчився писати по-російськи, від А. С. Пушкіна або Ф. М. Достоєвського. Шлях до майстерності в ремеслі лежить тільки через досвід. Не можна навчитися програмуванню, читаючи книги. Як можна за книгами навчитися писати романи, картини, вірші, музику. А ще програмістам потрібен постійний працю самовдосконалення і саморозвитку. Тому далеко не всі, хто пише програми, стають професіоналами.

Чому-то, якщо ми говоримо про поетів, художників, композиторів, то розкид творчої продуктивності нікого не дивує. «Творчий політ», «творчий застій» — це про діячів мистецтва. А коли говоримо про нерівномірність продуктивності програмістів, то багато менеджери починають з цим сперечатися, і намагаються «штовхати» програмістів, як ніби це змусить їх думати швидше. Не змусить. Але може змусити звільнитися або зайнятися імітацією роботи.

Правда, існують ще інженерні дисципліни такі, як будівництво, машинобудування, авіабудування і інші галузі матеріального виробництва, в

яких над створенням нових виробів працюють сотні тисяч людей. Дуже велика спокуса провести аналогію з цими галузями і говорити про індустріальному підході до розробки ПЗ. Не виходить.

Спрощено, шлях від ідеї до її реалізації в цих галузях виглядає наступним чином: НДР-ОКР-завод. У верхній частині цієї піраміди знаходяться галузеві НДІ, які виробляють ідеї і займаються проектуванням нових виробів. На другому поверсі піраміди працюють конструктори в конструкторських бюро, в завдання яких входить реалізація нового проекту в кресленнях деталей і технологій виготовлення та складання. На нижньому рівні знаходяться виробничі потужності — заводи, на яких інженери та робітники втілюють «в залізі» креслення і технології.

Якщо проводити аналогію, то програмісти працюють виключно на вершині описаної піраміди. Програмування — це проектування і тільки проектування. Роль конструкторського бюро для програмного проекту виконують компілятор і збирач програм. А програмістських аналогом заводу, який переводить конструкторську документацію продукт, доступний споживачу, служить обчислювальний комплекс, на якому розгортається і виконується створена програма.

А тепер давайте згадаємо, скільки НДР так і залишилися на папері, не дійшовши до ДКР, і скільки ще ОКР закінчилося закриттям тематики. Я думаю, що відсоток інновацій, що дійшли до виробництва від загального числа проектів, виконаних в галузевих НДІ, буде порівняємо з відсотком успішних програмних проектів. І давайте ще врахуємо, що вчені НДІ спираються на достатньо добре вивчені закони математики, фізики і хімії, а програмування, як ми відзначали вище, поки залишається лише ремісничим виробництвом.

Для колективного програміста творчості швидше доречна аналогія з створенням художнього кінофільму або театральної вистави. Кількість провальних проектів в цих областях нітрохи не менше, ніж у програмуванні. Дай Бог, якщо хоча б п'ята частина кінофільмів не «лягає на полиці» після першого показу. Про це ж пише авторитет в управлінні програмними проектами У. Ройс [7]: «Менеджери програмних проектів зможуть домогтися більшого, якщо будуть застосовувати методи управління, характерні для кіноіндустрії».

І ще одна аналогія програмних проектів з кінематографом. Наявність навіть самих зоряних акторів не забезпечує успіх фільму. Тільки талановитий режисер здатний організувати і надихнути акторів на створення шедевра, відкрити нові зірки. А талановитих режисерів, як, втім, і талановитих менеджерів програмних проектів, на жаль, не так багато, як хотілося б.

Еволюція підходів до управління програмними проектами

За 50 років розвитку програмної інженерії накопичилася велика кількість моделей розробки ПО. Цікаво провести аналогію між історією розвитку методів, що застосовуються в системах автоматичного управління літальними апаратами, і еволюцією підходів до управління програмними проектами.

«Як вийде». *Розімкнена система управління*. Повна довіра технічним лідерам. Представники бізнесу практично не бере участь у проекті. Планування, якщо воно і є, то неформальне і словесне. Час і бюджет, як правило, не контролюються. Аналогія: балістичний політ без зворотного зв'язку. Можна, але недалеко і неточно.

«*Водоспад*» або *каскадна модель*. Жорстке управління зі зворотним зв'язком. Розрахунок опорної траєкторії (план проекту), вимірювання відхилень, корекція і повернення на опорну траєкторію. Краще, але не ефективно.

«*Гнучке управління*». Розрахунок опорної траєкторії, вимірювання відхилень, розрахунок нової потрапляє траєкторії і корекція для виходу на неї. «Плани — ніщо, планування — все» (Ейзенхауер, Дуайт Девід)

«*Метод частих поставок*». Самонаведення. Розрахунок опорної траєкторії, вимірювання відхилень, уточнення мети, розрахунок нової потрапляє траєкторії і корекція для виходу на неї.

Класичні методи управління перестають працювати у випадках, коли структура і властивості керованого об'єкта нам не відомі і/або змінюються з часом. Ці підходи так само не допоможуть, якщо поточні властивості об'єкта не дозволяють йому рухатися з необхідними характеристиками. Наприклад, літальний апарат не може розвинути необхідну прискорення або руйнується при неприпустимою перевантаження. Аналогічно, якщо робоча група проекту не може забезпечити необхідну ефективність і тому постійно працює в режимі авралу, то це призводить не до зростання продуктивності, а до догляду професіоналів з проекту.

Коли структура і властивості керованого об'єкта нам не відомі, необхідно використовувати *адаптивне управління*, яке, додатково до прямих керуючих впливів, спрямована на вивчення і зміна властивостей керованого об'єкта. Продовжуючи аналогію з управлінням літальними апаратами — це розрахунок опорної траєкторії, вимірювання відхилень, уточнення мети, уточнення об'єкта управління, адаптація (необхідна зміна) об'єкта управління, розрахунок нової потрапляє траєкторії і корекція для виходу на неї.

Для того щоб зрозуміти структуру і властивості об'єкта і впливати на нього з метою їх приведення до бажаного стану, в проекті повинен бути додатковий контур зворотного зв'язку — контур адаптації.

Відомо, що продуктивність різних програмістів може відрізнятися в десятки разів. Стверджую, що продуктивність одного і того ж програміста може відрізнятися в десятки разів. Примусьте кращого в світі бігуна бігати в мішку, і він покаже в 10 разів гірший результат. Примусьте кращого програміста займатися «сізіфовою працею»: плодити документацію (яку, як правило, ніхто не читає) в догоду «Методології» (саме з великої букви), — і його продуктивність знизиться в 10 разів.

Тому, крім суто управлінських завдань керівник, якщо він прагне отримати найвищу продуктивність робочої групи, повинен спрямовувати постійні зусилля на вивчення і зміну об'єкту управління: людей та їх взаємодії.

Моделі процесу розробки ПЗ

Моделі (або, як ще люблять говорити, методології) процесів розробки ПЗ прийнято класифікувати за «вагою» — кількості формалізованих процесів (більшість процесів або тільки основні) і детальності їх регламентації. Чим більше процесів документовано, ніж більш детально вони описані, тим більше «вага» моделі.

Найпоширеніші сучасні моделі процесу розробки ПО представлені на рис. 3.

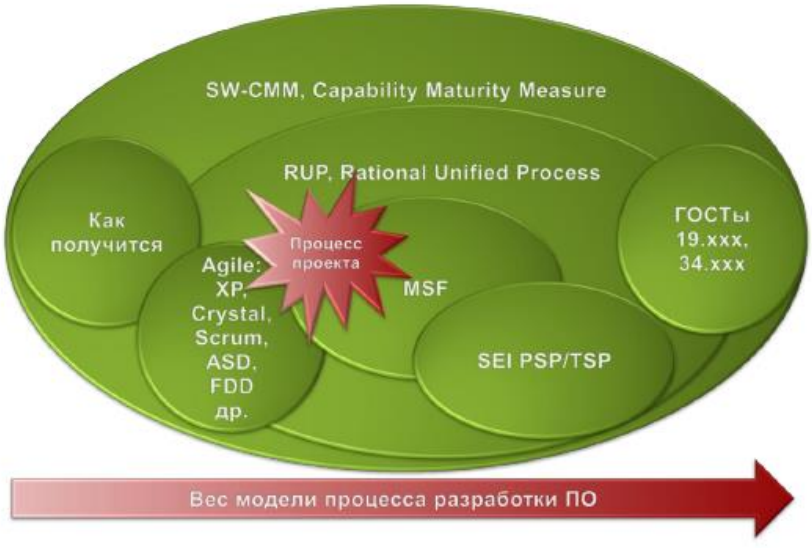


Рис. 3 Різні моделі процесу розробки ПЗ і їх розподіл за «вагою»

Гости

ГОСТ 19 «Єдина система програмної документації» і ГОСТ 34 «Стандарти на розробку та супровід автоматизованих систем» орієнтовані на послідовний підхід до розробки ПЗ. Розробка у відповідності з цими

стандартами проводиться за етапами, кожен з яких передбачає виконання певних робіт, і завершується випуском досить великого числа досить формалізованих і великих документів. Таким чином, суворе дотримання цих гостам не тільки призводить до водопадному підходу, але і вимагає дуже високого ступеня формализованности розробки.

SW-CMM

У середині 80-х років минулого століття Міністерство оборони США міцно задумався про те, як вибирати розробників ПЗ при реалізації великих програмних проектів. За замовленням військових Інститут програмної інженерії, що входить до складу Університету Карнегі-Меллона, розробив SW-CMM, Capability Maturity Model for Software [9] в якості еталонної моделі організації розробки програмного забезпечення.

Дана модель визначає п'ять рівнів зрілості процесу розробки ПЗ.

1. Початковий — процес розробки носить хаотичний характер. Визначено лише деякі з процесів, і успіх проектів залежить від конкретних виконавців.
2. Повторюваний — встановлені основні процеси управління проектами: відстеження витрат, термінів і функціональності. Упорядковані деякі процеси, необхідні для того, щоб повторити попередні досягнення на аналогічних проектах.
3. Певний — процеси розробки та управління проектами описано та впроваджено в єдину систему процесів компанії. У всіх проектах використовується стандартний для організації процес розробки та підтримки програмного забезпечення, адаптований під конкретний проект.
4. Керований — збираються детальні кількісні дані щодо функціонування процесів розробки та якості кінцевого продукту. Аналізується значення і динаміка цих даних.
5. Оптимізується — постійне поліпшення процесів ґрунтується на кількісних даних по процесах і на пробному впровадження нових ідей і технологій.

Документація з повним описом SW-CMM займає близько 500 сторінок і визначає набір з 312 вимог, яким повинна відповідати організація, якщо вона планує атестуватися за цим стандартом на 5-ий рівень зрілості.

RUP

Уніфікований процес (Rational Unified Process, RUP) [10] був розроблений Філіпом Крачтеном (Philippe Kruchten), Іваром Якобсоном (Ivar Jacobson) і іншими співробітниками компанії Rational Software" в якості доповнення до мови моделювання UML. Модель RUP описує загальний абстрактний процес, на основі якого організація або проектна команда повинна створити

конкретний спеціалізований процес, орієнтований на її потреби. Саме ця риса RUP викликає основну критику — оскільки він може бути чим завгодно, його не можна вважати нічим визначеним. В результаті такого загального побудови RUP можна використовувати і як основу для самого що ні на є традиційного водоспадні стилю розробки, так і в якості гнучкого процесу.

MSF

Microsoft Solutions Framework (MSF) [11] — це досить гнучка і легка модель, побудована на основі ітеративної розробки. Привабливою особливістю MSF є велика увага до створення ефективної і небюрократизированной проектної команди. Для досягнення цієї мети MSF пропонує досить нестандартні підходи до організаційної структури, розподілу відповідальності та принципами взаємодії всередині команди.

PSP/TSP

Одна з останніх розробок Інституту програмної інженерії Personal Software Process / Team Software Process [12,13]. Personal Software Process визначає вимоги до компетенцій розробника. Згідно цій моделі кожен програміст повинен вміти:

• враховувати час, витрачений на роботу над проектом;
• враховувати знайдені дефекти;
• класифікувати типи дефектів;
• оцінювати розмір задачі;
• здійснювати систематичний підхід до опису результатів тестування;
• планувати програмні завдання;
• розподіляти їх по часу і складати графік роботи.
• виконувати індивідуальну перевірку проекту та архітектури;
• здійснювати індивідуальну перевірку коду;
• виконувати регрессионное тестування.

Team Software Process робить ставку на самокеровані команди чисельністю 3-20 розробників. Команди повинні:

• встановити власні цілі;
• скласти свій процес і плани;
• відслідковувати роботу;
• підтримувати мотивацію і максимальну продуктивність.

Послідовне застосування моделі PSP/TSP дозволяє зробити нормою в організації п'ятий рівень CMM.

Agile

Основна ідея всіх гнучких моделей полягає в тому, що застосовуваний у розробці процес повинен бути адаптивним. Вони декларують своєю вищою цінністю орієнтованість на людей і їх взаємодію, а не на процеси і засоби. По суті, так звані, гнучкі методології це не методології, а набір практик, які можуть дозволити (а можуть і ні) домагатися ефективної розробки ПО, грунтуючись на итеративности, инкрементальности, самоврядності команди і адаптивності процесу.

Вибір моделі процесу

Важкі і легкі моделі виробничого процесу мають свої переваги і свої недоліки (Таблиця 1).

Таблиця 1. Плюси і мінуси важких і легких моделей процесів розробки ПЗ

Вага моделі	Плюси	Мінуси
Важкі	Процеси розраховані на середню кваліфікацію виконавців. Більша спеціалізація виконавців. Нижче вимоги до стабільності команди. Відсутні обмеження за обсягом і складності виконуваних проектів.	Вимагають істотної управлінської надбудови. Більш тривалі стадії аналізу і проектування. Більш формалізовані коммуникации.
Легкі	Менше непродуктивних витрат, пов'язаних з управлінням проектом, ризиками, змінами, конфігураціями. Спрощені стадії аналізу і проектування, основний упор на розробку функціональності, поєднання ролей. Неформальні комунікації.	Ефективність сильно залежить від індивідуальних здібностей, вимагають більш кваліфікованої, універсальної і стабільної команди. Обсяг і складність виконуваних проектів обмежені.

Один з авторів «Маніфесту гнучкої розробки ПО» [14] проаналізував дуже різні програмні проекти, які виконувалися по різним моделям від абсолютно полегшених і «гнучких» до важких (CMM-5) за останні 20 років [15, 16]. Він не виявив кореляції між успіхом чи провалом проектів і моделями процесу розробки, які застосовувалися в проектах. Звідси він зробив висновок про те, що ефективність розробки ПЗ не залежить від моделі процесу, а також про те, що:

• У кожного проекту повинна бути своя модель процесу розробки.
• У кожної моделі — свій час.

Це означає, що не існує єдиного правильного процесу розробки ПЗ, в кожному новому проекті процес повинен визначатися щоразу заново, в залежності від проекту, продукту та персоналу, у відповідності з «Законом 4-х П» (Малюнок 4). Абсолютно різні процеси повинні застосовуватися в проектах, в яких беруть участь 5 осіб, і в проектах, в яких беруть участь 500 чоловік. Якщо продуктом проекту є критичне, наприклад, система управління атомною електростанцією, то процес розробки повинен сильно відрізнятися від розробки, наприклад, сайту «відпочинь.ру». І, нарешті, по-різному слід організовувати процес розробки в команді вчорашніх студентів і в команді професіоналів.

Рис. 4. «Закон 4-х П». Процес у проекті повинен визначатися в залежності від проекту, продукту та персоналу

Команда, яка починала проект, не залишається незмінною, вона проходить певні стадії формування і, як правило, кількісно зростає в міру розвитку проекту. Тому процес повинен постійно адаптуватися до цих змін. Головний принцип: не люди повинні будуватися під обрану модель процесу, а модель процесу повинна підлаштовуватися під конкретну команду, щоб забезпечити її найвищу ефективність.

Що треба робити для успіху програмного проекту

Стів Макконнелл у своїй книзі [17] призводить тест програмного проекту на виживання. Цей чек-лист з 33-х пунктів, який я вважаю за необхідне процитувати з невеликими коригуваннями. Керівник програмного проекту

повинен його періодично використовувати для внутрішнього аудиту своїх процесів.
Щоб програмний проект став успішним, необхідно:
1. Чітко ставити цілі.
2. Визначати спосіб досягнення цілей.
3. Контролювати і керувати реалізацією.
4. Аналізувати загрози і протидіяти їм.
5. Створювати команду.

1. Ставимо цілі

1.1. Концепція визначає чіткі недвозначні мети.
1.2. Всі члени команди вважають концепцію реалістичною.
1.3. У проекту є обґрунтування економічної ефективності.
1.4. Розроблено прототип інтерфейсу.
1.5. Розроблена специфікація цільових функцій програмного продукту.
1.6. З кінцевими користувачами продукту налагоджена двосторонній зв'язок

2. Визначаємо спосіб досягнення цілей

2.1. Є детальний письмовий план розробки продукту.
2.2. У списку завдань проекту включені «другорядні» завдання (управління конфігураціями, конвертація даних, інтеграція з іншими системами).
2.3. Після кожної фази проекту оновлюється розклад і бюджет.
2.4. Архітектура і проектні рішення документовані.
2.5. Є план забезпечення якості, що визначає тестування та рецензування.
2.6. Визначено план багатоетапної поставки продукту.
2.7. У плані враховані навчання, вихідні, відпустки, лікарняні.
2.8. План проекту та розклад схвалений всіма учасниками команди.

3. Контрольований і керований реалізацією

3.1. У проекту є куратор. Це такий топ-менеджер виконуючою компанії, який особисто зацікавлений в успіху даного проекту.
3.2. У проекту є менеджер, причому тільки один!
3.3. У плані проекту визначені «бінарні» контрольні точки.
3.4. Всі зацікавлені сторони можуть отримати необхідну інформацію про хід проекту.
3.5. Між керівництвом і розробниками встановлені довірчі відносини.

3.6. Встановлена процедура управління змінами в проекті.
3.7. Визначені особи, відповідальні за рішення про прийняття змін у проекті.
3.8. План, розклад і статусна інформація за проектом доступна кожному учаснику.
3.9. Код системи проходить автоматичне рецензування.
3.10. Застосовується система управління дефектами.

4. Аналізуємо загрози

4.1. Є список ризиків проекту. Здійснюється регулярний аналіз та оновлення.
4.2. Керівник проекту відстежує виникнення нових ризиків.
4.3. Для кожного підрядника визначено особа, відповідальна за роботу з ним.
5. Працюємо над створенням команди
5.1. Досвід команди достатній для виконання проекту.
5.2. У команди достатня компетенція в прикладній області.
5.3. У проекті є технічний лідер.
5.4. Чисельність персоналу достатня.
5.5. У команди є достатня згуртованість.
5.6. Всі учасники прихильні проекту.

Оцінка та інтерпретація тесту

Оцінка: сума балів, кожен пункт оцінюється від 0 до 3:
• 0 — навіть не чули про це;
• 1 — чули, але поки не застосовуємо;
• 2 — застосовується частково;
• 3 — застосовується в повній мірі.

Поправочні коефіцієнти:
• для малих проектів (до 5 осіб) — 1.5;
• для середніх (від 5 до 20 осіб) — 1.25.

Результат:
• <40 — завершення проекту сумнівно.
• 40-59 — середній результат. В ході проекту слід очікувати серйозні проблеми.
• 60-79 — хороший результат. Проект, швидше за все, буде успішним.
• 80-89 — відмінний результат. Ймовірність успіху висока.
• >90 — чудовий результат. 100% шансів на успіх.

Цей чек-лист перераховує, що треба робити для успіху програмного проекту, але не дає відповідь на питання, як це слід робити. Саме про це піде мова в інших лекціях.

Висновки

Те, що виробляють програмісти нематеріальне — це колективні думки та ідеї, виражені на мові програмування. У силу унікальності галузі досвід, накопичений в галузях матеріального виробництва, мало сприяє успіху в управлінні програмним проектом. Прямі аналогії з цими галузями не працюють. Керувати розробкою ПЗ треба інакше.

Не існує єдиного правильного процесу розробки ПЗ. Ефективний виробничий процес повинен ґрунтуватися на итеративности, инкрементальности, самоврядності команди і адаптивності. Головний принцип: не люди повинні будуватися під обрану модель процесу, а модель процесу повинна підлаштовуватися під конкретну команду, щоб забезпечити її найвищу продуктивність.

Щоб програмний проект став успішним, необхідно:

1. Чітко ставити цілі.
2. Визначати спосіб досягнення цілей.
3. Контролювати і керувати реалізацією.
4. Аналізувати загрози і протидіяти їм.
5. Створювати команду.

Лекція 4-5. ТЕМА 2 ***Управління проектами. Визначення і концепції. Проект — основа інновацій. Критерії успішності проекту. Проект і організаційна структура компанії. Організація проектної команди. Життєвий цикл проекту. Фази і продукти.***

Класичне управління проектами виділяє два види організації людської діяльності: операційна і проектна. Операційна діяльність застосовується, коли зовнішні умови добре відомі і стабільні, коли виробничі операції добре вивчені і неодноразово випробувані, а функції виконавців визначені і постійні. У цьому випадку основою ефективності служать вузька спеціалізація та підвищення компетенції. «Якщо водій трамвая почне шукати нові шляхи, чекай біди». Там, де розробляється новий продукт, зовнішні умови та вимоги до якого постійно змінюються, де застосовуються виробничі

технології використовуються вперше, де постійно потрібні пошук нових можливостей, інтелектуальні зусилля і творчість, там потрібні проекти.

Проект — це тимчасове підприємство, призначене для створення унікальних продуктів, послуг або результатів. У операційної та проектної діяльності є ряд загальних характеристик: виконуються людьми, обмежені доступністю ресурсів, плануються, виконуються й управляються. Операційна діяльність і проекти розрізняються, головним чином, тим, що операційна діяльність — це триваючий у часі і повторюваний процес, у той час як проекти є тимчасовими і унікальними. Обмеження по термінах означає, що у будь-якого проекту є чіткий початок і чітке завершення. Завершення наступає, коли досягнуто цілі проекту; або усвідомлено, що цілі проекту не будуть або не можуть бути досягнуті; або зникла необхідність у проекті, і він припиняється. Унікальність також важлива відмінність проектної діяльності від операційної. Якщо б результати проекту не носили унікальний характер, роботу по їх досягненню можна було б чітко регламентувати, встановити виробничі нормативи і реалізовувати у рамках операційної діяльності (конвеєр). Завдання проекту — досягнення конкретної бізнес-цілі. Завдання операційної діяльності — забезпечення нормального перебігу бізнесу.

Проект — це засіб стратегічного розвитку (Малюнок 5). Мета - опис того, що ми хочемо досягти. Стратегія — констатація того, яким чином ми збираємося ці цілі досягати. Проекти перетворення стратегії в дії, а цілі в реальність.

Рис. 5. Проект — засіб стратегічного розвитку

Таким чином, кожна робота, яку виконує конкретний співробітник, прив'язується до досягнення стратегічних цілей організації. Проекти об'єднуються в програми. Програма — ряд пов'язаних один з одним проектів, управління якими координується для досягнення переваг і ступеня керованості, недоступних при управлінні ними окремо. Проекти і програми об'єднуються в портфелі. Портфель — набір проектів або програм та інших робіт, об'єднаних разом з метою ефективного управління даними роботами

для досягнення стратегічних цілей. Проекти та управління ними існували завжди. В якості самостійної області знань управління проектами почало формуватися на початку XX століття. У цій дисципліні поки немає єдиних міжнародних стандартів. Найбільш відомі центри компетенції:

■ PMI, Project Management Institute, PMBOK — американський національний стандарт ANSI/PMI 99-001-2004.

■ IPMA, International Project Management Association. У Росії — СОВНЕТ.

Приблизно 50 років тому людство почало жити в новій суспільно-економічній формації, яка називається інформаційне або постіндустріальне суспільство. Ми живемо в епоху змін, глобалізації та інтелектуального капіталу.

Епоха змін. Все в світі стало безперервно і стрімко змінюватися. Достаток стало причиною гострої конкуренції. Інновації — невід'ємний атрибут нашого часу. Якщо у вас повільний доступ в Інтернет, ви можете назавжди відстати від розвитку інформаційних технологій. Практика повинна постійно перебудовуватися стосовно до нових умов. Приклад. Hewlett-Packard отримує більшу частку прибутку на товари, які рік тому навіть не існували.

Глобалізація. Загальна взаємозалежність і взаємопов'язаність. Транснаціональні компанії. Бізнес іде туди, де дешевша робоча сила. Інтернет. Конкуренція без кордонів. Приклад. Google. За ніч кожен з нас в принципі може створити багатомільйонну компанію у себе в гаражі. За допомогою Інтернету ви можете вийти на ринок, на якому більше 100 млн. споживачів.

Все вирішують таланти. Проста мобілізація коштів і зусиль вже не може забезпечити прогрес. Якщо проект не вкладається у терміни, то додавання робочої сили затримає його ще більше. Ідею багатства тепер пов'язують не з грошима, а з людьми, не з фінансовим капіталом, а з «людським». Ринок праці перетворюється в ринок незалежних фахівців та його учасникам все більше відомо про можливі варіанти вибору. Працівники інтелектуальної праці починають самостійно визначати собі ціну.

Людству відомі два виду діяльності. Репродуктивна діяльність (праця) є зліпком, копією з діяльності іншої людини або копією своєї власної діяльності, освоєної в попередньому досвіді. Така діяльність, як, наприклад, праця токаря в будь-якому механічному цеху, або рутинна повсякденна діяльність менеджера-управлінця на рівні раз і назавжди засвоєних технологій. Продуктивна діяльність (творчість) — діяльність, спрямована на отримання об'єктивно нового або суб'єктивно нового (для даного працівника) результату. Репродуктивна діяльність відходить у минуле. В постіндустріальному суспільстві інтелект — основна виробнича сила.

Сьогодні від 70 до 80% всього, що сьогодні робиться людьми, проводиться за допомогою їх інтелекту. В будь-якому товарі, зробленому в США, частка зарплати становить 70 відсотків. Але це в середньому по всіх товарах. Що стосується розробки ПЗ, то майже все, що в цій галузі виробляється, створюється за допомогою інтелекту.

Все менший обсяг людської діяльності може бути організований у вигляді повторюваних операцій. Приклад операційної діяльності - це робота бухгалтерії. Але життя так стрімко змінюється, що сьогодні, за твердженням обізнаних людей, підготовка та складання річного фінансового звіту щоразу реалізується як самостійний проект.

Проект-це основа інновацій. Зробити те, до чого інші компанії ще не додумалися, зробити це якомога швидше, інакше це зроблять інші. Запропонувати споживачеві більш якісний продукт або такий продукт, потреба в якому споживач навіть не може поки усвідомити.

Критерії успішності проекту

Завдання проекту - досягнення конкретної бізнес-цілі, при дотриманні обмежень «залізного трикутника» (Рис. 6). Це означає, що ні один з кутів трикутника не може бути змінений без впливу на інші. Наприклад, щоб зменшити час, потрібно збільшити вартість та/або скоротити зміст.

Рис. 6. «Залізний трикутник» обмежень проекту

Відповідно до поточної редакції стандарту PMBOK, проект вважається успішним, якщо задоволені всі вимоги замовника та учасників проекту. Тому у проекту розробки ПО сьогодні не три, а чотири фактори успіху:

1. Виконаний у відповідності зі специфікаціями.
2. Виконаний в строк.
3. Виконаний у межах бюджету.
4. Кожен учасник команди йшов з роботи о 18:00 з відчуттям успіху.

Цей четвертий фактор успіху має стати відтворюваним, якщо підприємство хоче бути ефективним. Для успішного проекту характерно постійне відчуття його учасниками почуття задоволення і гордості за результати своєї роботи, почуття оптимізму. ***Немає нічого більш згубного для проекту, ніж байдужість або зневіру його учасників.***

Ефективність-це відношення отриманого результату до виробленим затратам. Не можна розглядати ефективність, виходячи тільки з результативності: чим більше ти робиш, чим більше робиш, тим вище твоя ефективність. З таким підходом можна «зарізати на вечерю курку, що несе золоті яйця». Витрати не слід плутати з інвестиціями. Оплата оренди, електроенергії, комунальні платежі — витрати. Створення і закріплення ефективної команди — це стратегічне придбання компанії. Навчання учасників проекту — інвестиції. Вкладення в людей — це збільшення чисельника формули ефективності. Відхід з компанії всіх професіоналів після проекту, виконаного за принципом «будь-якою ціною», — витрати, причому дуже важко заповнюють. Наростаюча конкуренція вказує на абсолютно чіткий тренд у світовій економіці — персонал — це форма інвестицій, активів, які потрібно вміти нарощувати, управляти і зберігати. **Сьогодні люди — це капітал.**

Сучасне підприємство зобов'язане ставитися до своїх працівникам так само, як до своїм кращим клієнтам. Головний капітал сучасної компанії - це знання. Велика частина цих знань невіддільна від їх носія - людини. Ті підприємства, які цього не зрозуміли, не виживуть тому, що не зможуть бути ефективними. Сьогодні ефективне підприємство - це сервіс. Підприємство, з одного боку, надає послуги і продукти своїм клієнтам, а з іншого, — робочі місця для професійного персоналу. Принципи «Одне підприємство на все життя», «Працюй продуктивно, а підприємство про тебе подбає» - відходять у минуле. Подивіться на ринок робочої сили в IT - правила встановлюють професіонали.

Проект і організаційна структура компанії

Організаційна структура компанії відображає її внутрішній устрій, потоки керуючих впливів, розподіл праці і специфічні особливості виробництва. Функціональна та проектна організації — протилежні полюси, а матрична організація — проміжні стани. Немає однієї кращої організаційної структури. Немає сенсу протиставляти функціональні структури та проектні організації. Синонім функціональної структури - ієрархічна структура (Рисунок 7).

Рис. 7. Функціональна структура

Функціональна структура має такі особливості:

• *Зберігається принцип єдиноначальності*

• *Зрозумілі і стабільні умови роботи*

• *Добре пристосовані для операційної діяльності.*

• *Спеціалізація підрозділів дозволяє накопичувати експертизу.*

• *Сповільнене прийняття рішень і комунікації між виконавцями. Здійснюються лише через керівництво.*

• *Управління сконцентровано і тримається на компетенції вищого керівництва*

• *Як правило, неефективний контроль за ходом проекту (немає цілісної картини)*

Функціональна структура припускає багаторівневу ієрархію. Керівники функціональних підрозділів це начальники управлінь, начальники підпорядкованих їм служб, відділів, лабораторій, секторів, груп. А ще у кожного начальника є заступник і, часом, не один. Приклади: міністерства, відомства, наукові інститути і підприємства радянського періоду. На іншому краю спектра організаційних структур знаходиться проектна структура (Рисунок 8).

Рис. 8. Проектна структура

У чисто проектних організаціях:

• *Проект організовується як самостійне виробниче підрозділ.*
• *Персонал на проект набирається за тимчасовими контрактами.*
• *Після завершення проекту персонал звільняється.*
• *Повільний старт.*
• *Досвід не акумулюється.*
• *Команди не зберігаються.*

Проектні організації не найефективніші, а іноді єдино можливі для виконання проектів, які фізично віддалені від виконуючої організації, наприклад, будівництво нового нафтопроводу.

В розробці найбільш поширена матрична організація. Розрізняють три види матричної організаційної структури: слабка, збалансована і сильна (Рис. 9 – Рис.11). Причому, в компаніях, які займаються продуктовою розробкою ПО, функціональні підрозділи визначаються у відповідність з лінійкою продуктів. Наприклад, відділ розробки CRM-систем, відділ розробки фінансових систем, відділ розробки додаткових продуктів.

В компаніях, які орієнтовані в основному на власну розробку, функціональні підрозділи частіше об'єднуються у відповідність з використовуваними інформаційними технологіями. Наприклад, відділ розробки баз даних, відділ розробки J2EE-додатків, відділ веб-розробок, відділи тестування, документування і т. д.

Рис. 9. Слабка матриця

У слабкій матриці роль і повноваження працівника, який координує проект, сильно обмежені. Реальне керівництво проектом здійснює один із функціональних керівників. Координатор проекту, його ще часто називають «трекер», допомагає цьому керівникові збирати інформацію про статус виконуваних проектних робіт, враховує витрати, складає звіти.

Рис. 10. Збалансована матриця

Збалансована матриця характеризується тим, що з'являється менеджер проекту, який реально управляє виділеними на проект ресурсами. Він планує роботи, розподіляє завдання серед виконавців, контролює терміни та результати, несе повну відповідальність за досягнення цілей проекту, при дотриманні обмежень. У збалансованих матрицях найбільш яскраво проявляється проблема подвійного підпорядкування. Керівник функціонального підрозділу і менеджер проекту мають приблизно однаковий вплив на матеріальний і професійний ріст розробників.

Рис. 11 . Сильна матриця

У **сильній матриці** зізнається, що проектне управління є самостійною галуззю компетенції, в якій необхідно накопичувати експертизу і використовувати загальні ресурси. Тому в сильній матриці менеджери проектів об'єднуються у самостійний функціональний підрозділ — офіс управління проектами (ОУП). ОУП розробляє корпоративні політики та стандарти в області проектного управління, планує та здійснює професійний розвиток менеджерів.

Однією з особливостей матричних структур є те, що вони стають «плоскими», зникає багатоступенева ієрархія. Підприємство, як правило, ділиться на функціональні відділи, в яких працюють фахівці різних категорій, безпосередньо підпорядковуються начальнику відділу. Начальники лабораторій, секторів, груп скасовуються за непотрібністю. В матричних структурах роль начальника функціонального підрозділу у виробничому процесі помітно знижується, порівняно з функціональними структурами. У його компетенції залишаються питання стратегічного розвитку функціонального спрямування, планування і розвиток кар'єри співробітників, питання матеріально-технічного забезпечення робіт. Слід враховувати, що такий перерозподіл повноважень і відповідальності функціональних керівників до менеджерів проектів часто служить джерелом конфліктів у компаніях при їх переході від функціональної структури до матричної.

Організація проектної команди

Кожен проект розробки ПЗ має свою організаційну структуру, яка визначає розподіл відповідальності та повноважень серед учасників проекту, а також обов'язків та відносин звітності. Чим менше проект, тим більше

ролей доводиться поєднувати одному виконавцю. Ролі і відповідальності учасників типового проекту розробки ПЗ можна умовно розділити на п'ять груп:

1. Аналіз. Витяг, документування та супроводження вимог до продукту.
2. Управління. Визначення та управління виробничими процесами.
3. Виробництво. Проектування і розробка ПЗ.
4. Тестування. Тестування ПЗ.
5. Забезпечення. Виробництво додаткових продуктів і послуг.

Група аналізу включає в себе наступні ролі:

• Бізнес-аналітик. Побудова моделі предметної області (онтології).
• Бізнес-архітектор. Розробляє бізнес-концепцію системи. Визначає загальне бачення продукту, його інтерфейси, поведінку і обмеження.
• Системний аналітик. Відповідає за переведення вимог до продукту у функціональні вимоги до ПЗ.
• Фахівець за вимогами. Документування та супроводження вимог до продукту.
• Менеджер продукту (функціональний замовник). Представляє в проекті інтереси користувачів продукту.

Група управління складається з наступних ролей:

• Керівник проекту. Відповідає за досягнення цілей проекту при заданих обмеженнях (за термінами, бюджету і змістом), здійснює операційне управління проектом і виділеними ресурсами.
• Куратор проекту. Оцінка планів та виконання проекту. Виділення ресурсів.
• Системний архітектор. Розробка технічної концепції системи. Прийняття ключових проектних рішень щодо внутрішнього устрою програмної системи та її технічних інтерфейсів.
• Керівник групи тестування. Визначення цілей і стратегії тестування, управління тестуванням.
• Відповідальний за управління змінами, конфігураціями, за збірку і постачання програмного продукту.

У виробничу групу входять:

• *Проектувальник. Проектування компонентів і підсистем у відповідність із загальною архітектурою, розробка архітектурно значущих модулів.*
• *Проектувальник бази даних.*
• *Проектувальник інтерфейсу користувача.*
• *Розробник. Проектування, реалізація і налагодження окремих модулів системи.*

У великому проекті може бути кілька виробничих груп, відповідальних за окремі підсистеми. Як правило, проектувальник виконує роль лідера групи і управляє своїм подпроектом або пакетом робіт. Варто не забувати, що керівник проекту делегує повноваження, але не відповідальність.

Група тестування в проекті складається з наступних ролей:

• *Проектувальник тестів. Розробка тестових сценаріїв.*
• *Розробник автоматизованих тестів.*
• *Тестувальник. Тестування продукту. Аналіз та документування результатів.*

Учасники групи забезпечення, як правило, не входять у команду проекту. Вони виконують роботи в рамках своєї процесної діяльності. До групи забезпечення можна віднести наступні проектні ролі:

• *Технічний письменник.*
• *Перекладач.*
• *Дизайнер графічного інтерфейсу.*
• *Розробник навчальних курсів, тренер.*
• *Учасник рецензування.*
• *Продажі та маркетинг.*
• *Системний адміністратор.*
• *Технолог.*
• *Фахівець з інструментальним засобам.*
• *Інші.*

Залежно від масштабу проекту одну роль можуть виконувати кілька людей. Наприклад, розробники, тестувальники, технічні письменники. Деякі ролі завжди повинен виконувати тільки один чоловік. Наприклад, Керівник

проекту, Системний архітектор. Одна людина може виконувати кілька ролей. Можливі такі поєднання ролей:

• *Керівник проекту + системний аналітик (+ системний архітектор)*
• *Системний архітектор + розробник*
• *Системний аналітик + проектувальник тестів (+ технічний письменник)*
• *Системний аналітик + проектувальник інтерфейсу користувача*
• *Відповідальний за керування конфігураціями + відповідальний за складання і постачання (+ розробник)*

Вкрай небажано поєднувати такі ролі:

• *Розробник + керівник проекту*
• *Розробник + системний аналітик.*
• *Розробник + проектувальник інтерфейсів користувача.*
• *Розробник + тестувальник*

Програмісти люблять і вміють програмувати. Нехай вони цим і займаються. Не варто завантажувати програмістів невластивою для них роботою. В кожному проекті розробки програмного продукту багато інших робіт: бізнес-аналіз, проектування ергономіки, графічний дизайн, розробка користувальницької документації. Ці роботи з програмуванням не мають нічого спільного. Для них потрібні зовсім інша кваліфікація і інший склад мислення.

При кустарному виробництві програм ці завдання, як правило, доручаються програмістам, які це робити не вміють і не люблять. Виходить зазвичай погано, та ще й дорого. В силу своєї інтроверсії, що межує з аутизмом, програміст просто не в змозі побачити свою програму чужими очима — очима користувачів. Ніхто вже не хоче працювати з програмами з технологічної парадигмою навороченого користувальницького інтерфейсу — кустарним творінням програмістів — коли для того щоб працювати з системою, треба обов'язково знати, як вона влаштована. Це типове творіння програміста, яким набагато важливіше бачити, як працює його програма, ніж розбиратися в тому, що вона робить для користувача. Тому, необхідно залучати в проектну команду бізнес-аналітиків,ергономістів,художників-дизайнерів,документалістів. Поділ праці та спеціалізація — запорука переходу від кустарного виробництва до більш ефективного промислового виробництва. З професійних програмістів виходять відмінні тестувальники. Однак, поєднувати одночасно ролі програміста і тестувальника — погана

практика. Хороший програміст переконаний, що він пише програми правильно і йому психологічно важко припустити, що десь у його коді може бути помилка. А помилки є завжди!

Організаційна структура проекту обов'язково повинна включати в себе ефективну систему звітності, оцінки ходу виконання проекту і систему прийняття рішень. Можна рекомендувати щотижневі збори за статусом проекту, на яких аналізуються ризики, оцінюються результати, досягнуті на попередньому тижні, та уточнюються завдання на новий період.

У моделі Scrum рекомендуються щоденні наради щодо стану робіт — «Stand Up Meeting», але це стосується, швидше, для невеликих робочих груп від 3 до 5 розробників. Хоча в критичні періоди проекту, доводилося проводити щоденні наради. Організаційна структура проекту — «живий» організм. Вона починає складатися на стадії планування та може змінюватися в ході проекту. Нестабільність організаційної структури (часті заміни виконавців) - серйозна проблема в управлінні складними програмними проектами, оскільки існує час входження в контекст проекту, яке може вимірюватися місяцями.

Життєвий цикл проекту. Фази і продукти

Раніше вже зазначалося, що кожен програмний продукт має свій життєвий цикл, у який проект розробки чергового релізу входить як одна з фаз. Аналогічно, кожен проект розробки ПЗ має свій власний життєвий цикл, який складається з чотирьох фаз (Рис.12).

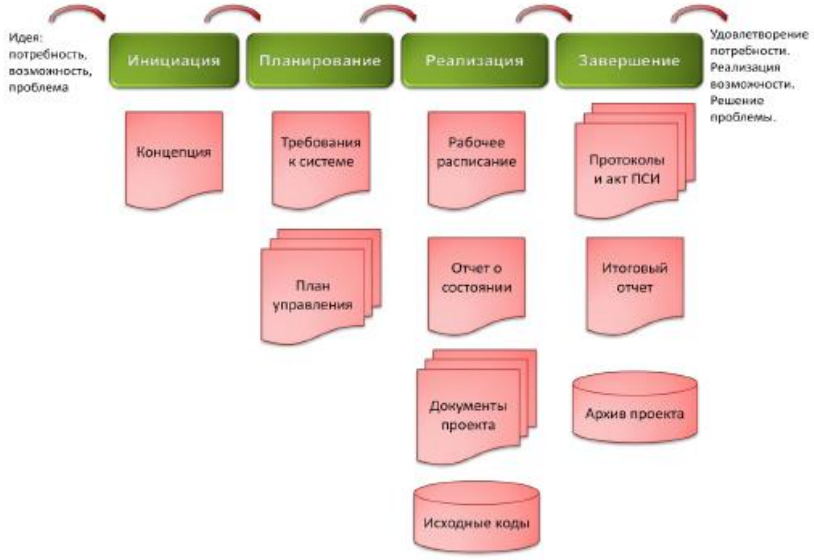


Рис. 12. Життєвий цикл та основні продукти програмного проекту

На фазі **ініціації** проекту необхідно зрозуміти, що і навіщо ми будемо робити - розробити концепцію проекту. Фаза **планування** визначає, як ми будемо це робити. На фазі **реалізації** відбувається матеріалізація наших ідей у вигляді документованого і протестованого програмного продукту. І, нарешті, на фазі **завершення** ми повинні підтвердити, що ми розробили саме той продукт, який задумали в концепції проекту, а також провести приймально-здавальні випробування (ПСІ) продукту на предмет відповідності його властивостей, визначеним раніше вимогам. Як правило, рідкісний проект виконується у відповідність з початковими планами, тому важливим елементом фази завершення є «зворотний зв'язок»: аналіз причин розбіжності і засвоєння уроків на майбутнє. Пам'ятаємо, що керуюча система без зворотного зв'язку не може бути стійкою. Завершуючи огляд управління проектами, необхідно згадати ще про одну особливість проекту порівняно з операційною діяльністю. Якщо в операційній діяльності ресурси витрачаються більш-менш рівномірно по часу, то в проектному управлінні витрачання ресурсів на одиницю часу має явно виражену колоколообразное розподіл (Малюнок 13)

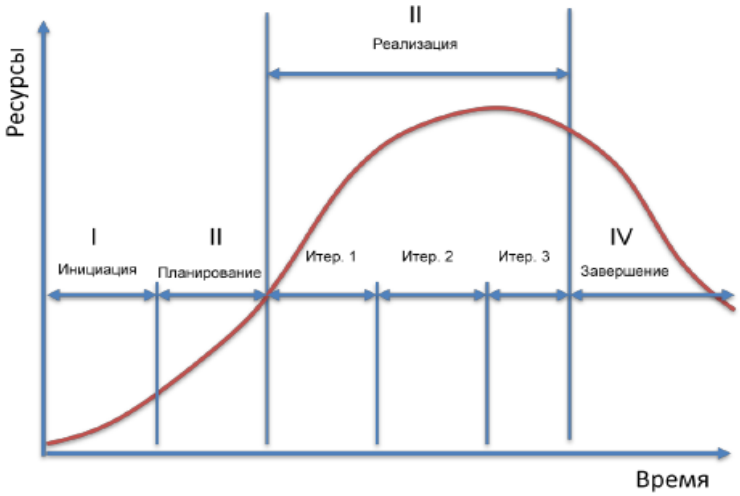


Рис. 13. Розподіл ресурсів по фазах проекту

Проект часто починається з ідеї, яка з'являється в однієї людини. Поступово, по мірі формулювання, аналізу і оцінки цієї ідеї, залучаються додаткові фахівці. Ще більше учасників потрібно на фазі планування проекту. Пік споживання ресурсів припадає на фазу реалізації. У сучасних моделях розробки реалізація здійснюється на основі поєднання **ітеративного** і **інкрементального** підходів. **Итеративность** передбачає, що вимоги до системи і її архітектура опрацьовуються не один раз, а поступово уточнюються від ітерації до ітерації. Це означає, що на кожній ітерації відбувається повний цикл процесів розробки: уточнення вимог,

проектування, кодування, тестування і документування. **Инкрементальность** полягає в тому, що результатом кожної ітерації є версія, яка реалізує частину функціональності майбутнього програмного продукту і може бути введено в тестову або дослідну експлуатацію, а також оцінено замовником і майбутніми користувачами. Це означає, що після кожної ітерації відбувається приріст необхідного функціоналу, а нереалізованих функцій майбутнього продукту залишається все менше. Поєднання итеративности і инкрементальности забезпечує ефективність розробки та істотне зниження ризиків по ходу проекту. Про це ми ще будемо говорити. На останній фазі відбувається поступове вивільнення учасників проектної команди. Слід пам'ятати, що проект повинен мати чітке закінчення часу, після якого всі роботи за проектом закриваються, і на проект перестають витрачатися ресурси. Не повинно залишатися «завислих» робіт.

Висновки

Проект — цей засіб стратегічного розвитку. Мета — опис того, що ми хочемо досягти. Стратегія — констатація того, яким чином ми збираємося ці цілі досягати. Проекти перетворення стратегії в дії, а цілі в реальність.
Учасників типового проекту розробки ПЗ можна умовно розділити на п'ять груп ролей:

1. Аналіз. Витяг, документування та супроводження вимог до продукту.
2. Управління. Визначення та управління виробничими процесами.
3. Виробництво. Проектування і розробка ПЗ.
4. Тестування. Тестування ПЗ.
5. Забезпечення. Виробництво додаткових продуктів і послуг.

У програмного проекту є чотири фактори, які визначають його успішність:

1. Виконаний у відповідності зі специфікаціями.
2. Виконаний в строк.
3. Виконаний у межах бюджету.
4. Кожен учасник команди йшов з роботи о 18:00 з відчуттям успіху.

Література

1. «PMBOK. Руководство к Своду знаний по управлению проектами», 3-е изд., PMI, 2004.

2. Стивен Р. Кови, «7 навыков высокоэффективных людей. Мощные инструменты развития личности», 2-е изд., М., Альпина Бизнес Букс, 2007
3. Брукс Фредерик, "Мифический человеко-месяц, или Как создаются программные комплексы", Пер. с англ., СПб., Символ-Плюс, 1999.
4. Кьелл А. Нордстрем, Йонас Риддерстрале, «Бизнес в стиле фанк. Капитал пляшет под дудку таланта», Стокгольмская школа экономики в Санкт-Петербурге, 2005.

Лекція 6-7. ТЕМА 3 *Ініціація проекту. Управління пріоритетами проектів. Концепція проекту. Цілі та результати проекту. Допущення і обмеження. Ключові учасники і зацікавлені сторони. Ресурси. Терміни. Ризики. Критерії приймання. Обґрунтування корисності.*

Ефективні процеси ініціації програмного проекту мінімум наполовину визначають його майбутню успішність. Недостатню увагу саме цій фазі проекту неминуче призводить до суттєвих проблем при плануванні, реалізації та завершення проекту. **Ініціація** складається з процесів, що сприяють формальної авторизації початку нового проекту або фази проекту. Процеси ініціації часто виконуються поза рамками проекту і пов'язані з організаційними, програмними або портфельними процесами. В ході процесу ініціації уточнюються первісне опис змісту і ресурси, які організація планує вкласти. На цьому етапі також вибирається менеджер проекту, якщо він ще не призначений, і документуються вихідні припущення та обмеження. Ця інформація заноситься в Статут проекту і, якщо він схвалюється, проект офіційно авторизується.

Статут проекту — документ, випущений ініціатором або спонсором проекту, який формально узаконює існування проекту і надає менеджеру проекту повноваження використовувати організаційні ресурси в операціях проекту. Концепція (від лат. conceptio — розуміння, система) - певний спосіб розуміння, трактування якого-небудь предмета, явища, процесу, основна точка зору на предмет і ін., керівна ідея для їх систематичного висвітлення. У компанії, яка приймає рішення про старт того чи іншого проекту розробки ПО, повинна існувати єдина система критеріїв для оцінки його значимості. Система критеріїв повинна дозволяти з безлічі можливих для реалізації проектів вибрати найбільш пріоритетні для компанії.

Пріоритет будь-якого проекту має визначатися на основі оцінки трьох його характеристик:

• *Фінансова цінність.*

• *Стратегічна цінність.*

• *Рівень ризиків.*

Шкала оцінки **фінансової цінності** проекту може виглядати наступним чином:

• ***Висока***. Очікувана окупність до 1 року. Очікувані доходи від проекту не менш ніж у 1.5 разів перевищують витрати. Всі допущення при проведенні цих оцінок чітко обґрунтовані.

• ***Вище середнього***. Очікувана окупність проекту від 1 року до 3 років. Очікувані доходи від проекту не менш ніж в 1.3 рази перевищують витрати. Більшість припущень при проведенні цих оцінок мають під собою певні підстави.

• ***Середня***. Проект дозволяє поліпшити ефективність виробництва в Компанії і потенційно може знизити витрати компанії не менш ніж на 30%. Проект може мати інформаційну цінність або допомогти краще контролювати бізнес.

• ***Низька.*** Проект трохи знижує витрати компанії не менш ніж на 10% і дає деякі поліпшення продуктивності виробництва.

Наприклад. Фінансова цінність проектів розробки, впровадження проектів або супроводу, які виконуються у відповідність з укладеними комерційними договорами, може бути оцінена як висока. Проект планового розвитку функціональності продуктів відповідно до вимогами ринку, зініційоване менеджером продукту на основі аналізу пропозицій відділів маркетингу, консалтингу, продажу та технічної підтримки, може отримати оцінку фінансової цінності вище середнього, а проекти зміни технологічних процесів або проекти внутрішньої автоматизації можуть мати середню фінансову цінність.

Однією фінансової цінності для визначення пріоритету проекту недостатньо. Важливим показником пріоритету проекту є його відповідність стратегічним цілям компанії.

Шкала оцінки **стратегічної цінності** проекту може мати такий вигляд:

• *Висока*. Забезпечує стратегічну перевагу, дає стійке збільшення ринку або дозволяє вийти на новий ринок. Вирішує значні проблеми, спільні для більшості важливих клієнтів. Повторення конкурентами утруднене або потребуватиме від 1 до 2 років.

• *Вище середнього*. Створює тимчасові конкурентні переваги. Виконання зобов'язань перед багатьма важливими клієнтами. Конкурентна перевага може бути утримано протягом 1 року.
• *Середня*. Підтримується довіру ринку до компанії. Підвищує думку клієнтів про якість надаваних послуг або сприяє виконанню зобов'язань перед кількома клієнтами. Конкуренти вже мають або здатні повторити нові можливості в межах року.
• *Низька*. Стратегічне вплив відсутній або незначно. Вплив на клієнтів несуттєво. Конкуренти можуть легко повторити результати проекту.

Третім обов'язковим показником пріоритету проекту повинна бути оцінка рівня його ризику. Ні один проект, який має найвищу оцінку фінансової вигідності, не буде запущений у виробництво, якщо досягнення цієї сверхвыгоды має мінімальні шанси.

Приблизна шкала оцінки рівня ризиків проекту може мати такий вигляд:

• *Низький*. Цілі проекту та вимоги добре зрозумілі і документовані. Масштаб і рамки проекту визначені чітко. Ресурси необхідної кваліфікації доступні в повному обсязі. Розроблювані системи не вимагатимуть нової технологічної платформи.
• *Середній*. Цілі проекту визначені більш-менш чітко. Гарне розуміння вимог до системи. Масштаб і рамки проекту визначені досить добре. Ресурси необхідної кваліфікації доступні в основному. Системи створюються на новій, але стабільною технологічній платформі.
• *Вище середнього*. Цілі проекту недостатньо чіткі. Завдання системи або бізнес-додатки зрозумілі недостатньо повно. Розуміння масштабу і рамок проекту недостатньо. Ресурси необхідної кваліфікації сильно обмежені. Системи створюються на новій технологічній платформі, сумніви в ринковій стабільності платформи.
• *Високий*. Цілі проекту нечіткі. Основні функціональні компоненти системи не визначені. Масштаб і рамки проекту незрозумілі. Ресурси необхідної кваліфікації практично відсутні. Системи створюються на новій технологічній платформі, щодо якої вкрай мало ясності. Технології мають непідтверджену стабільність.

Якщо компанія приділяє мало уваги управлінню пріоритетами своїх проектів, то це призводить до надлишку реалізованих проектів, перевантаженості виконавців, постійних авралів і надурочних робіт і, як наслідок, до низької ефективності виробничої діяльності. При старті нового проекту з високим пріоритетом, компанія повинна зупинити або закрити

менш значущі проекти, щоб забезпечити новий проект необхідними ресурсами, а не намагатися зробити все й одразу за рахунок інтенсифікації робіт, як правило, це не виходить.

Концепція проекту

У кожного проекту має бути концепція. Якщо проект невеликий, то для викладення концепції часто досить кілька абзаців. Однак, стартувати проект без концепції, це все одно, що відправляти судно в плавання, не визначивши для нього пункт призначення. Концепція проекту розробляється на основі аналізу потреб бізнесу. Головна функція документа — підтвердження і узгодження єдиного бачення цілей, завдань і результатів усіма учасниками проекту. Концепція визначає що і навіщо робиться у проекті. Концепція проекту-це ключовий документ, який використовується для прийняття рішень у ході всього проекту, а також на фазі приймання — для підтвердження результату. Вона містить, як правило, є наступні розділи:

• *Назва проекту*
• *Цілі проекту*
• *Результати проекту*
• *Допущення та обмеження*
• *Ключові учасники і зацікавлені сторони*
• *Ресурси проекту*
• *Терміни*
• *Ризики*
• *Критерії приймання*
• *Обґрунтування корисності проекту*

В якості прикладу, який дозволить ілюструвати теоретичний виклад основ управління проектами, візьмемо реальний проект розробки ПЗ для автоматизації одного з підрозділів великої виробничої компанії. Назвемо його «Автоматизована система продажу документації».

Коротка легенда проекту. Замовник ВАТ «XYZ» є одним з провідних виробників складних технічних виробів. Відділ «123», що входить у ВАТ «XYZ», відповідає за продаж додаткової документації для клієнтів ВАТ. Додаткова документація не входить у стандартну поставку, оскільки власник цього технічного виробу не завжди сам його експлуатує, а передає в експлуатацію іншої компанії, яка стає клієнтом «XYZ», і закуповує у неї експлуатаційну документацію. Ремонт і техобслуговування конкретного

виробу може виконувати третя компанія, якій вже знадобиться детальна технічна документація з ремонту та обслуговування. Вона також стає клієнтом «XYZ» і купує у неї необхідну продукцію. Основна функція відділу «123» — отримання і обробка замовлень на додаткову документацію, згідно щороку рассылаемому каталогу. У зв'язку з переїздом відділу «123» в нову будівлю, що була поставлена задача на розробку і постачання системи, що автоматизує основну діяльність відділу «123». Текст документа Концепція проекту, який буде приводитися в якості прикладу, будемо виділяти кольором фону.

Цілі та результати проекту

Цілі проекту повинні відповідати на питання, навіщо цей проект потрібен. Цілі проекту повинні описувати бізнес-потреби і завдання, які вирішуються в результаті виконання проекту. Цілями проекту можуть бути:

• *Зміни в Компанії. Наприклад, автоматизація ряду бізнес-процесів для підвищення ефективності основної виробничої діяльності*

• *Реалізація стратегічних планів. Наприклад, завоювання значної частки зростаючого ринку за рахунок виведення на нього нового продукту.*

• *Виконання контрактів. Наприклад, розробка програмного забезпечення на замовлення.*

• *Вирішення специфічних проблем. Наприклад, доопрацювання програмного продукту з метою приведення його у відповідність із змінами в законодавстві.*

Цілі повинні бути значущими (спрямованими на досягнення стратегічних цілей Компанії), конкретними (специфічними для даного проекту), вимірюваними (тобто мати перевіряються кількісні оцінки), реальними (досяжними). Чітке визначення бізнес-цілей важливо, оскільки істотно впливає на всі процеси і рішення в проекті. Проект повинен бути закритий, якщо визнається, що досягнення мети неможливо або стало недоцільним. Наприклад, якщо реальні витрати на проект будуть перевершувати майбутні доходи від його реалізації.

Результати проекту відповідають на питання, що має бути отримано після його завершення. Результати проекту повинні визначати:

• Які саме бізнес-вигоди отримає замовник у результаті проекту.

• Будь-який продукт або послуга. Що конкретно буде зроблено по закінченні проекту.

• Високорівневі вимоги. Короткий опис і при необхідності ключові властивості і/або характеристики продукту/послуги.

Слід пам'ятати, що результати проекту повинні бути вимірними. Це означає, що при оцінці результатів проекту має бути можливість зробити висновок досягнуті обумовлені в концепції результати чи ні. Відповідний розділ документа концепція проекту створення Автоматизованої системи продажу документації» буде виглядати наступним чином.

1. Цілі та результати проекту

1.1. Метою проекту є підвищення ефективності основної виробничої діяльності відділу «123».
1.2. Додатковими цілями проекту є:
1.2.1. Встановлення довгострокових відносин з важливим замовником ВАТ «XYZ».
1.2.2. Вихід на новий перспективний ринок сучасних B2C систем.

2. Результати проекту повинні забезпечити:

2.1. Зниження витрат на обробку заявок.
2.2. Зниження термінів обробки заявок.
2.3. Підвищення оперативності доступу до інформації про наявність продукції.
2.4. Підвищення оперативності доступу до інформації про проходження заявок.
2.5. Підвищення надійності та повноти збереження інформації про заявки, що надійшли, та результати їх обробки.

3. Продуктами проекту є:

3.1. Прикладне ПЗ і документація користувачів.
3.2. Базове ПЗ.
3.3. Обладнання ЛОМ, робочі станції, сервери та операційно-системне ПЗ.
3.4. Проведення пуско-налагоджувальних робіт та введення в дослідну експлуатацію.
3.5. Навчання користувачів та адміністраторів системи.
3.6. Супровід системи на етапі дослідної експлуатації.
3.7. Передача системи в промислову експлуатацію.

4. Система повинна автоматизувати такі функції:

4.1. Ідентифікація і аутентифікація користувачів.
4.2. Перегляд каталогу продуктів.
4.3. Пошук продуктів з каталогу.
4.4. Замовлення обраних вами продуктів.
4.5. Перегляд інформації про статус замовлення.
4.6. Інформування клієнта про зміну статусу замовлення.
4.7. Перегляд і обробка замовлень виконавцями із служби продажів.
4.8. Перегляд статистики надходження і обробки замовлень за період.
4.9. Підготовка та супровід каталогу продукції.

Допущення та обмеження

Даний розділ описує вихідні припущення та обмеження. Допущення, як правило, тісно пов'язані із управлінням ризиками, про який ми будемо говорити далі. В розробці часто доводиться формулювати ризики у вигляді припущень, тим самим передаючи його замовнику. Наприклад, оцінюючи проект розробки та впровадження за схемою з фіксованою ціною, ми повинні записати у допущення припущення про те, що вартість ліцензій на стороннє ПЗ не зміниться, до завершення проекту.

Обмеження, як правило, скорочують можливості проектної команди у виборі рішень. Зокрема вони можуть містити:

• *Специфічні нормативні вимоги. Наприклад, обов'язкова сертифікація продукту, послуги на відповідність певним стандартам.*
• *Специфічні технічні вимоги. Наприклад, розробка під задану програмно-апаратну платформу.*
• *Специфічні вимоги до захисту інформації.*

У цьому розділі також доречно сформулювати ті вимоги до системи, які можуть очікуватися замовником за замовчуванням, але не включаються в рамках даного проекту. Наприклад, в даний розділ може бути включений пункт про те, що розробка програмного інтерфейсу (API) для майбутньої інтеграції з іншими системами замовника не входить в задачі даного проекту. Зміст цього розділу для нашого проекту-приклад виглядає наступним чином.

5. Допущення та обмеження

5.1. Проектування прикладного ПЗ виконується з використанням UML1.
5.2. Засобом розробки ПЗ є Symantec Visual Cafe for Java2.
5.3. В якості проміжного ПО супроводу та підтримки каталогу використовується ОО БД «Poet»3.
5.4. Навантаження на систему не повинна бути більше 100 одночасно працюючих користувачів.
5.5. В рамках проекту не входять:
5.5.1. Захист системи від навмисного злому.
5.5.2. Розробка B2B API і інтеграція з іншими системами.

Ключові учасники і зацікавлені сторони

Одне із завдань фази ініціації проекту це виявити і описати всіх його учасників. Згідно [1] до учасників проекту відносяться всі зацікавлені сторони (stakeholders), особи і організації, наприклад, замовники, спонсори, виконуюча організація, які активно беруть участь у проекті, або чиї інтереси можуть бути порушені при виконанні або завершення проекту. Учасники також можуть впливати на проект і його результати поставки.

До ключовим учасникам програмного проекту, як правило, належать:

• *Спонсор проекту — особа або група осіб, що надає фінансові ресурси для проекту в будь-якому вигляді.*
• *Замовник проекту — особа або організація, які будуть використовувати продукт, послугу або результат проекту. Слід враховувати, що замовник і спонсор проекту не завжди збігаються.*
• *Користувачі результатів проекту.*
• *Куратор проекту — представник виконавця, уповноважений приймати рішення про виділення ресурсів і зміни в проекті.*
• *Керівник проекту — представник виконавця, відповідальний за реалізацію проекту в строк, у межах бюджету і з заданою якістю.*
• *Співвиконавці проекту. Субпідрядники та постачальники.*

Зміст цього розділу в концепції прикладі буде мати вигляд.

6. Ключові учасники і зацікавлені сторони

6.1. Спонсор проекту — директор Департаменту інформатизації ВАТ «XYZ» Ст. Васильєв.

6.2. Замовник — начальник Відділу «123» Ф. Федотов
6.3. Користувачі автоматизованої системи:
6.4. Клієнти ВАТ «XYZ» (пошук та замовлення документації).
6.5. Керівництво ВАТ «XYZ» (аналіз діяльності Відділу «123»).
6.6. Працівники виробничих департаментів ВАТ «XYZ» (супровід каталогу).
6.7. Співробітники Відділу «123» (обробка заявок та постачання документації).
6.8. Співробітники департаменту інформатизації ВАТ «XYZ» (адміністрування системи).
6.9. Куратор проекту — начальник відділу замовлених розробок В. Іванов.
6.10. Керівник проекту — провідний спеціаліст відділу замовлених розробок МП П. Петров.

7. Співвиконавці:

7.1. Постачальник обладнання та операційно-системного — ТОВ «Альфа».
7.2. Постачальник базового ПЗ — ТОВ «Бета».

Ресурси

Для того щоб зрозуміти, скільки буде коштувати реалізація програмного проекту, потрібно визначити та оцінити ресурси, необхідні для його виконання:
• Людські ресурси і вимоги до кваліфікації персоналу.
• Обладнання, послуги, витратні матеріали, ліцензії на ПЗ, критичні комп'ютерні ресурси.
• Бюджет проекту. План витрат і, при необхідності, передбачуваних доходів проекту з розбивкою за статтями і фазам/етапів проекту.
Специфіка програмного проекту полягає в тому, що людські ресурси вносять основний внесок у його вартість. Всі інші витрати, як правило, незначні, порівняно з цим витратами. Про те, як слід підходити до оцінок трудовитрат на реалізацію проекту розробки ПО, ми будемо докладно говорити в наступних лекціях. На фазі ініціації хорошою вважається оцінка трудовитрат з точністю від -50 до +100% [2].
Необхідно пам'ятати, що крім безпосередньо програмування в проекті розробки ПЗ є багато інших процесів, які вимагають ресурси відповідної кваліфікації, а саме програмування становить лише чверть усіх витрат. Розподіл роботи по основним виробничим процесам при сучасному процесі розробки ПО виглядає в середньому наступним чином:

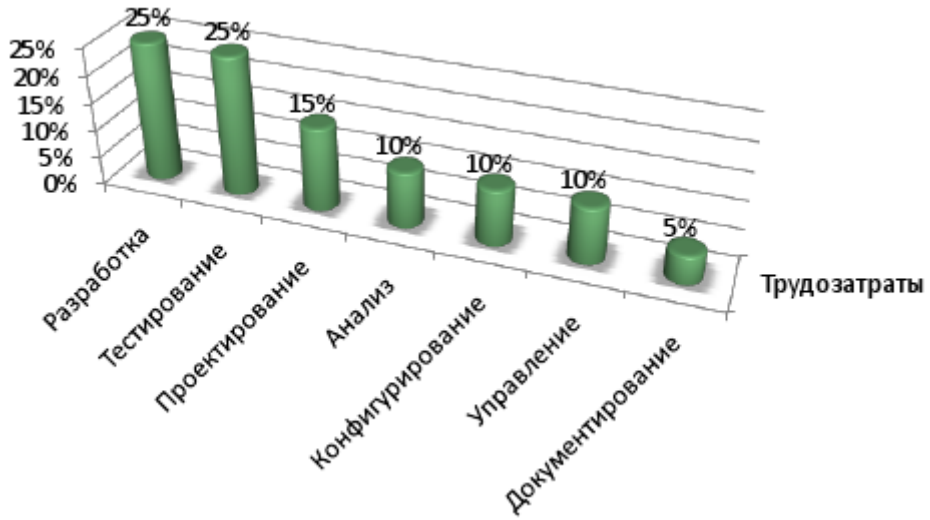


Рис. 14. Розподіл роботи по основних виробничих процесах при розробці ПЗ

Тому, якщо у вашої оцінки для реалізації необхідної функціональності в проекті необхідно написати 10 KSLOC (тисяч рядків вихідного програмного коду), а ваші програмісти пишуть в середньому по 100 SLOC в день, то загальні витрати на проект будуть не 100 чол.*днів, а не менш ніж 400 чол.*днів. Інші ресурси потрібні на аналіз і уточнення вимог, проектування, документування, тестування та інші проектні роботи.

Перш ніж визначати чисельність і склад проектної команди для нашого прикладу, нам необхідно зробити оцінку трудомісткості розробки ПО. В нашому випадку така експертна оцінка склала з урахуванням витрат на гарантійний супровід на етапі дослідної експлуатації 9000 чол*годину. Виходячи з емпіричної кривої Б. Боема (Малюнок 15), чисельність команди, близька до оптимальної, склала 10 осіб, з них

8. Ресурси проекту

8.1. Вимоги до персоналу

8.1.1. 1 — керівник проекту,

8.1.2. 1 — технічний лідер (архітектура, проектування),

8.1.3. 1 — системний аналітик (вимоги, тест-дизайн, документування),

8.1.4. 4 — програмісти (з урахуванням робіт з конфігураційного управління),

8.1.5. 3 — тестувальника.

8.2. Матеріальні та інші ресурси

8.2.1. Сервер управління конфігураціями і підтримки системи контролю версій

8.2.2. 2 серверних комплексу (для розробки і тестування):

8.2.3. Сервер додатків з встановленим BEA Weblogic AS
8.2.4. Сервер БД оперативної з встановленою Oracle RDBMS
8.2.5. Сервер каталогу з встановленою OODB "Poet"
8.3. Ліцензії на засоби розробки і тестування:
8.3.1. Oracle Designer — 1 ліцензія
8.3.2. Symantec Visual Cafe for Java — 5 ліцензій.
8.3.3. IBM Rational Test Robot (1 ліцензія розробника + необмежена ліцензія на клієнт).
8.4. Видаткова частина бюджету проекта4
8.4.1. Розробка та супроводження прикладного:
*8.4.1.1. 9000 чол*годину. * $40 = $360 000*
8.4.2. Поставка обладнання та операційно-системного ПЗ:
*8.4.2.1. 3 сервера * $10 000 = $30 000*
8.4.3. Поставка базового ПЗ:
8.4.3.1. BEA Weblogic AS $20 000
8.4.3.2. Oracle RDBMS $20 000
Разом: $430 000

1. Проект стартував у 2000 році, тема UML тоді була на слуху і навіть залишалися ті, хто вірив, що з моделі на UML можна буде генерувати код.
2. Така була вимога Замовника, оскільки цей інструмент використовували його програмісти, яким передбачалося передавати систему на супровід.
3. Ще один приклад гарячої теми і не виправдалися надій — це об'єктно-орієнтовані бази даних. У замовника проекту вже були закуплені ліцензії на цю базу даних і він дуже хотів отримати прибутки від цих інвестицій. Тому її використання в проекті стало однією з вимог. На щастя, нам вдалося бути достатньо переконливими та обґрунтувати необхідність додатково використовувати RDBMS Oracle для вирішення транзакційних завдань.
4. В даному розділі оцінюємо собівартість проекту. Визначення продажної ціни проекту не входить в рамки даного курсу.

Терміни

Брукс [3] призводить виключно корисну, але чомусь рідко застосовується, емпіричну формулу оцінки терміну проекту на його трудомісткості. Формула була виведена Барії Боэмом (Barry Boehm) на основі аналізу результатів 63 проектів розробки ПО, в основному в аерокосмічній галузі. Згідно цій формулі, для проекту, загальна

трудомісткість якого становить N ч.*м. (людино-місяців), пожно стверджувати що:

• Існує оптимальне, з точки зору витрат, час виконання графіка для першої поставки:

$$T = 2{,}5\ (N\ \text{ч.*м.})1/3.$$

Тобто оптимальний час в місяцях пропорційно кореню кубічному передбачуваного обсягу робіт у людино-місяцях. Наслідком є крива, що дає оптимальну чисельність проектної команди (Рис.15).

• Крива вартості повільно зростає, якщо запланований графік довше оптимального. Робота займає весь відведений для неї час.

• Крива вартості різко зростає, якщо запланований графік коротше оптимального. Практично жоден проект неможливо завершити швидше, ніж за 3/4 розрахункового оптимального графіка незалежно від кількості зайнятих в ньому! (Рис. 16)

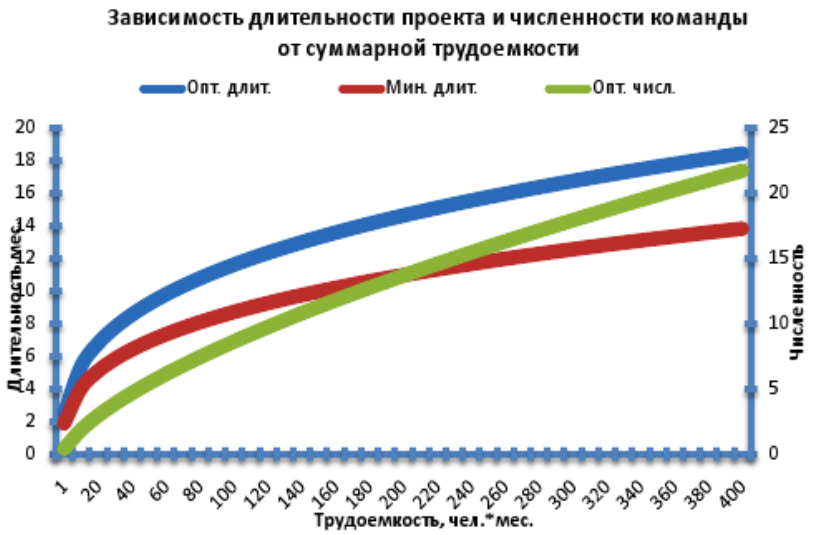


Рис. 15. Закон Б. Боема

Цей примітний результат дає менеджеру програмного проекту солідне підкріплення, коли вище керівництво вимагає прийняття неможливого графіка. Для скільки-небудь серйозного програмного проекту недостатньо визначити тільки термін його завершення. Необхідно ще визначити його етапи — контрольні точки, в яких буде відбуватися переоцінка проекту на основі реально досягнутих показників.

Контрольна точка — важливий момент або подія в розкладі проекту, відзначає досягнення заданого результату і/або початок / завершення певного обсягу роботи. Кожна контрольна точка характеризується датою та об'єктивними критеріями її досягнення. Як ми говорили раніше, сучасний проект розробки ПЗ повинен реалізовуватися з застосуванням

інкрементального процесу. У цьому випадку контрольні точки повинні відповідати випуску кожної проміжної версії, в якій буде реалізована і протестована певна частина кінцевої функціональності програмного продукту. В залежності від складності та масштабу проекту тривалість однієї ітерації може становити від 2 до 8 тижнів.

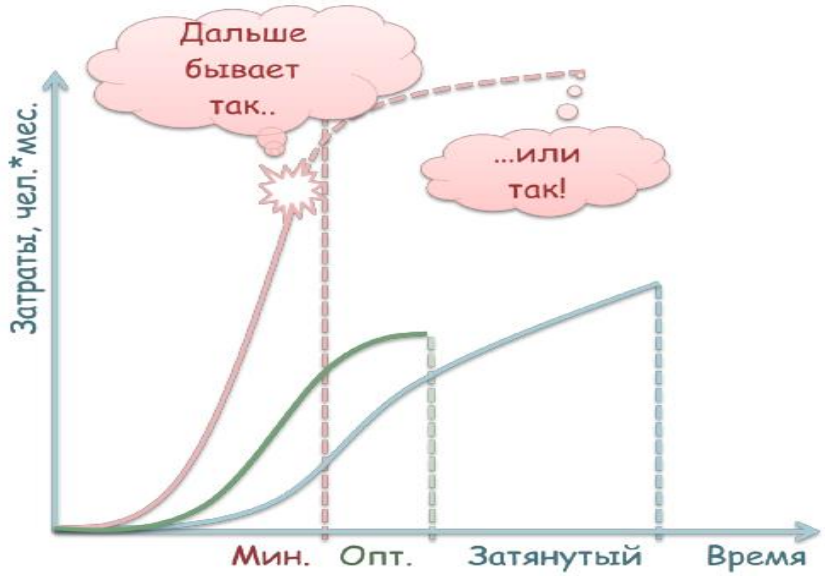


Рис. 16. Наслідки закону Б. Боема

Відповідний розділ концепції нашого проекту-приклад буде мати наступний вигляд.

9. Терміни проекту

9.1. 03.03 старт
9.2. 28.11 завершення
9.3. Контрольні точки:
9.3.1. 15.04 ТЗ затверджено
9.3.2. 30.04 1-я ітерація завершена. Підсистема замовлення документації передана в тестову експлуатацію (на серверах розробника).
9.3.3. 15.05 Монтаж обладнання у замовника завершений .
9.3.4. 30.05 Базове ПЗ встановлено у замовника.
9.3.5. 15.06 2-я ітерація завершена. Підсистема обробки замовлень передана в тестову експлуатацію на обладнанні Замовника
9.3.6. 02.09 3-я ітерація завершена. Акт передачі системи в дослідну експлуатацію затверджено
9.3.7. 28.11 Система передана в промислову експлуатацію.

Ризики

Ризик — невизначений подія або умова, наступ якого негативно або позитивно позначається на цілях проекту [1]. Як правило, у випадку виникнення негативного ризику, майже завжди вартість проекту збільшується і відбувається затримка у виконанні заходів, передбачених розкладом проекту. На етапі ініціації, коли немає необхідних даних для проведення детального аналізу, часто доводиться обмежуватися якісною оцінкою загального рівня ризиків: низький, середній, високий. У разі нашого проекту-прикладу розділ «ризики» буде виглядати наступним чином.

10. *Ризики проекту*

10.1. Завдання системи зрозумілі недостатньо повно. Розуміння масштабу і рамок проекту недостатньо. Системи створюються на новій технологічній платформі, сумніви в ринковій стабільності платформи. Сумарний рівень ризиків слід оцінити вище середнього.

Критерії приймання

Критерії приймання повинні визначати числові значення характеристик системи, які повинні бути продемонстровані за результатами приймально-здавальних випробувань чи пробної експлуатації і однозначно свідчити про досягнення цілей проекту.

У розглянутому прикладі розділ «Критерії приймання» буде виглядати наступним чином:

11. Критерії приймання. За підсумками дослідної експлуатації система повинна продемонструвати наступні показники:
11.1. Середні витрати співробітників Відділу «123» на регламентну обробку одного замовлення не перевищують 4 чол. годину.
11.2. Термін регламентної обробки 1-го замовлення не більше 2 тижнів.
11.3. Час пошуку та надання інформації про наявність додаткової документації не більше 1 хв.
11.4. Час надання інформації про зроблених замовлень і історії їх обробки не більше 1 хв.
11.5. Система зберігає всю інформацію про зроблені замовлення та історії їх обробки.

11.6. Показник доступності системи 98%.

Обґрунтування корисності проекту

Цей розділ концепції повинен містити короткий техніко-економічне обґрунтування проекту:

• Для кого призначені результати проекту.
• Опис поточної ситуації «As Is». Які у потенційного замовника існують проблеми.
• Яким чином результати проекту вирішують ці проблеми («To Be»).
• Наскільки значимо для клієнта рішення даних проблем (оцінка економічного ефекту).
• Які переваги в результаті цього може отримати компанія-виконавець проекту.

Відповідний розділ в концепції проекту-приклад буде мати наступний вигляд.

12. Обґрунтування корисності проекту

12.1. Для Замовника:
12.1.1. Підвищення продуктивності обробки замовлень в 2 рази.
12.1.1.1. "As Is": 2500 замовлень/рік по 8 чол. годину.
12.1.1.2. "To Be": 2500 замовлень/рік по 4 чол. годину.
*12.1.1.3. Економія: 2500 * 4 * $50 = $500 000 у рік.*
12.1.2. Підвищення оперативності контролю
12.1.2.1. "As Is": Щомісячна звітність.
12.1.2.2. "To Be": Звітність on-line.
12.1.3. Підвищення задоволеності клієнтів:
12.1.3.1. Скорочення терміну обробки замовлення в 2 рази.
12.1.3.2. Скорочення часу на пошук необхідної документації у 10 разів
12.1.3.3. Підвищення оперативності оновлення каталогу 10 разів. 12.2. Для компанії-виконавця:
12.2.1. Висока стратегічна цінність. Дає стійке збільшення ринку і завоювання нового ринку.
12.2.2. Фінансова цінність вище середнього. Очікувані доходи від проекту не менш ніж в 1.3 рази перевищують витрати.

Висновки

Ефективні процеси ініціації програмного проекту багато в чому визначають його майбутню успішність. Недостатня увага цій фазі проекту неминуче призводить до суттєвих проблем при плануванні, реалізації та завершення. Концепція проекту-це ключовий документ, який використовується для прийняття рішень у ході всього проекту, а також на фазі приймання — для підтвердження результату.
Пріоритет проекту визначається на основі оцінки трьох показників:
• Фінансова цінність.
• Стратегічна цінність.
• Рівень ризиків.

Література

1. «PMBOK. Руководство к Своду знаний по управлению проектами», 3-е изд., PMI, 2004.
2. С. Макконнелл, «Сколько стоит программный проект», «Питер», 2007.
3. Брукс Фредерик, «Мифический человеко-месяц, или Как создаются программные комплексы», Пер. с англ., СПб., Символ-Плюс, 1999.

Лекція 8-9. ТЕМА 4 ***Планування проекту. Уточнення змісту і складу робіт. Планування керування вмістом. Планування організаційної структури. Планування управління конфігураціями. Планування управління якістю. Базове розклад проекту.***

Людство поки що не придумало нічого більш ефективного для рішення складної задачі, ніж аналіз і її декомпозиція на більш прості підзадачі, які, в свою чергу, можуть бути розділені на ще більш прості підзадачі і так далі. Виходить деяка ієрархічна структура, дерево, корені якого знаходиться проект, а на листках елементарні завдання чи роботи, які треба виконати, щоб завершити проект в умовах заданих обмежень. Згідно [1]:
Ієрархічна структура робіт (ICP) (Work /Breakdown Structure, WBS) — орієнтована на результат ієрархічна декомпозиція робіт, виконуваних командою проекту для досягнення цілей проекту і необхідних результатів. З її допомогою структурується і визначається зміст проекту. Кожен наступний рівень ієрархії відображає більш детальне визначення елементів проекту. Основою для розробки ICP служить концепція проекту, що визначає

продукти проекту та їх основні характеристики. ІСР забезпечує виявлення всіх робіт, необхідних для досягнення цілей проекту. Багато проекти провалюються не від того, що у них немає плану, а від того що в цьому плані забуті важливі роботи, наприклад тестування і виправлення помилок, і продукти проекту, наприклад користувацька документація. Тому, якщо ІСР складена коректно, то будь-яка робота, яка в неї не увійшла не може вважатися роботою за проектом. ІСР поділяє проект на підпроекти, пакети робіт, подпакеты. Кожен наступний рівень декомпозиції забезпечує послідовну деталізацію змісту проекту, що дозволяє проводити оцінку строків і обсягів робіт. ІСР повинна включати всі проміжні та кінцеві продукти.

Виконувати декомпозицію робіт проекту можна по-різному. Наприклад, ГОСТ 19.102-77 передбачає каскадний підхід і визначає наступні стадії розробки програмної системи:

1. Технічне завдання
2. Ескізний проект
3. Технічний проект
4. Робочий проект
5. Запровадження

Якщо слідувати цим стандартом, то на першому рівні ІСР повинні знаходитися саме ці проектні продукти. Якщо б довелося розробляти АСУ для управління ядерним реактором або пілотованим космічним апаратом, то саме так і слід чинити. Однак у комерційній розробці, такий підхід не ефективний. Сучасний процес розробки комерційного ПЗ повинен бути инкрементальным. Це означає, що на верхньому рівні декомпозиції нашого проекту повинні перебувати продукти проекту, а на наступному рівні - компоненти, з яких ці продукти складаються. Компоненти далі можуть бути декомпозованих на «фічі» — функції, які вони повинні реалізовувати. Виділення компонентів, що складають програмний продукт, це елемент високорівневого проектування, яке ми повинні виконати на фазі планування проекту, не чекаючи опрацювання всіх функціональних вимог до розроблюваного ПЗ. Компонентами можуть бути як прикладні підсистеми, так і інфраструктурні або ядерні, наприклад, підсистема логування, безпеки, бібліотека візуальних компонентів GUI. При складанні базового плану робіт не варто прагнути максимально деталізувати всі роботи. ІСР не повинна містити надто багато рівнів, достатньо 3-5. Наприклад, ІСР нашого проекту-приклад розробки Автоматизованої системи продажу документації» може виглядати наступним чином (Рис.17).

1. Проект розробки Автоматизованої системи продажу документації»

1.1. Підготовка технічного завдання на автоматизацію
1.1.1.1. Проведення аналітичного обстеження
1.1.1.2. Розробка функціональних вимог
1.1.1.3. Розробка вимог базового ПЗ
1.1.1.4. Розробка вимог до обладнання та до операційно-системному ПО
1.1.1.5. Узгодження і затвердження ТЗ
1.1.1.6. ТЗ затверджено
1.2. Поставка та монтаж обладнання

1.2.1.Розробка специфікації на обладнання 1.2.2.Закупівля і поставка обладнання
1.2.3. Монтаж обладнання
1.2.4. Установка і настройка операційно-системного
1.2.5. Монтаж обладнання завершено

1.3. Поставка та встановлення базового ПЗ

1.3.1 .Розробка специфікацій на базове ЗА 1.3.2.Закупівля базового ПЗ
1.3.3. Розгортання і налаштування базового ПЗ
1.3.4. Базове ПЗ встановлено у замовника

1.4. Розробка і тестування прикладного ПЗ

1.4.1. Розробка специфікацій на прикладне
1.4.2. Установка і налаштування робочого середовища
1.4.3. Проектування і розробка ПО
1.4.3.1. Ідентифікація і аутентифікація користувачів.
1.4.3.2. Розробка підсистеми замовлення документації
1.4.3.2.1. Перегляд каталогу продуктів.
1.4.3.2.2. Пошук продуктів з каталогу.
1.4.3.2.3. Замовлення обраних вами продуктів.
1.4.3.2.4. Перегляд інформації про статус замовлення.
1.4.3.2.5. Інформування клієнта про зміну статусу замовлення.
1.4.3.2.6. Підсистема замовлення документації передана в тестову експлуатацію (на серверах розробника).
1.4.3.3. Розробка підсистеми обробки замовлень
1.4.3.3.1. Перегляд і обробка замовлень виконавцями із служби продажів.

1.4.3.3.2. Перегляд статистики надходження і обробки замовлень за період.
1.4.3.3.3. Підсистема обробки замовлень передана в тестову експлуатацію на обладнанні Замовника
1.4.3.4. Розробка підсистеми супроводу каталогу 1.4.3.4.1. Підготовка та супровід каталогу продукції.
1.4.3.5. Виправлення помилок
1.4.4. Тестування З
1.4.4.1. Раунд 1
1.4.4.2. Раунд 2
1.4.4.3. Раунд 3
1.4.4.4. Вихідне тестування
1.4.5. Документування прикладного ПЗ
1.5. Навчання користувачів
1.5.1 .Підготовка навчальних курсів
1.5.2.Навчання співробітників Відділу 123
1.5.3.Навчання керівництва ВАТ XYZ 1.5.4.Навчання адміністраторів системи
1.6. Введення в дослідну експлуатацію
1.6.1. Розгортання і налаштування прикладного ПЗ
1.6.2. Проведення приймально-здавальних випробувань
1.6.3. Акт передачі системи в дослідну експлуатацію затверджено
1.7. Супровід системи в період дослідної експлуатації
1.8. Система передана в промислову експлуатацію

Рис. 17. Ієрархічна структура робіт проекту з розробки Автоматизованої системи продажу документації» (курсивом виділені контрольні точки проекту)

Повинна бути встановлена персональна відповідальність за всі частини проекту (підпроекти і пакети робіт). Для кожного пакету робіт повинен бути чітко визначений результат на виході. Роботи та оцінки проекту повинні бути узгоджені з ключовими учасниками команди, керівництвом компанії-виконавця та, при необхідності, із замовником. В результаті узгодження члени команди беруть на себе зобов'язання щодо реалізації проекту, а керівництво приймає на себе зобов'язання по забезпеченню проекту необхідними ресурсами. ІСР є одним з основних інструментів (засобів) у механізмі управління проектом, з допомогою якого вимірюється ступінь досягнення результатів проекту. Найважливіша її функція це забезпечити консистентное подання всіх у приватників проекту щодо того, як буде

робитися проект. В подальшому базовий план буде служити орієнтиром для порівняння з поточним виконанням проекту і виявлення відхилень для цілей управління.

Планування управління вмістом

Одна з поширених «хвороб» програмних проектів називається «повзучий фичеризм». Це, коли до початку спроектованої будці для улюбленої собаки спочатку прилаштовують сарайчик для зберігання садового інвентарю, а потім і будиночок в кілька поверхів для її господаря. І все це намагаються побудувати на одному і тому ж фундаменті і з тих же самих матеріалів. Ця хвороба стала причиною летального результату багатьох проектів розробки ПО. Тому відразу, як тільки вдалося стабілізувати і узгодити ICP, необхідно розробити план управління змістом проекту. Для цього слід:

• *Визначити джерела запитів на зміну.*
• *Встановити порядок аналізу, оцінки і затвердження/відхилення зміни змісту.*
• *Визначити порядок документування змін вмісту.*
• *Визначити порядок інформування про зміну змісту.*

Перша задача, яку необхідно вирішити при аналізі запиту на зміни - виявити об'єкти змін: вимоги, архітектура, структури даних, вихідні коди, сценарії тестування, користувацька документація, тощо. **Потім** потрібно спроектувати і детально описати зміни у всіх виявлених об'єктах. І *нарешті*, слід оцінити витрати на внесення змін, тестування змін і регрессионное тестування продукту і їх вплив на терміни проекту.

Ця робота, яка вимагає витрат робочого часу різних фахівців: аналітиків, проектувальників, виробників, тестувальників, нарешті, менеджера проекту. Тому ця робота обов'язково повинна бути врахована в плані.

Планування організаційної структури

Організаційна структура це узгоджене та затверджене розподіл ролей, обов'язків і цілей діяльності ключових учасників проекту. Вона в обов'язковому порядку повинна включати в себе систему робочих взаємин між робочими групами проекту, систему звітності, оцінки ходу виконання проекту і систему прийняття рішень. Слід пам'ятати, що організаційна структура проекту — «живий» організм. Вона починає складатися на стадії

планування і повинна змінюватися по ходу проекту. Нестабільність організаційної структури - часта зміна виконавців — може стати серйозною проблемою в управлінні проектом, оскільки, існує ціна заміни, яка визначається часом входження нового учасника в контекст проекту.

Планування управління конфігураціями

Конфігураційне управління один з важливих процесів виробництва програмного забезпечення. Ми будемо говорити про те, що ця робота повинна бути спланована. План проекту повинен включати в себе роботи по забезпеченню єдиного сховища всієї проектної документації та розроблюваного програмного коду, забезпечення схоронності і відновлення проектної інформації після збою. Роботи з налаштування робочих станцій і серверів, використовуваних учасниками проектної команди, теж повинні увійти в план. Крім цього в плані повинні міститися роботи, необхідні для організації складання проміжних випусків системи, а також її кінцевого варіанту. Ці роботи, як правило, виконує одна людина — інженер по конфігурації. Якщо проект невеликий, то ця роль може бути додатковою для одного з програмістів. Я якось бачив, що цю роль виконував менеджер проекту. «Розмазувати» цю роботу на всіх учасників проекту, по-перше, неефективно. Встановлення і налаштування середовища розробки, наприклад, баз даних і серверів додатків, вимагає певних компетенцій і знань особливостей конкретних версій продуктів. Якщо ці навички доведеться освоювати всім розробникам, то на це піде дуже багато робочого часу. По-друге, «розмазування» робіт з управління конфігураціями може призвести до колективної безвідповідальності, коли ніхто не знає, від чого не збирається проект і як відкотитися до консистентним версії. Управління конфігураціями може багаторазово ускладнитися, якщо проектної команди паралельно з розробкою нової функціональності продукту доводиться підтримувати декілька релізів цього продукту, які були встановлені раніше у різних клієнтів. Всі ці роботи повинні бути враховані в плані проекту.

Планування управління якістю

Забезпечення якості ще одна з базових областей знань програмної інженерії. Щодо того, що таке якість і як його ефективно забезпечувати, можна міркувати дуже і дуже довго. У нашому курсі ми обмежимося твердженням про те, що забезпечення якості це важлива робота, яка повинна бути спланована заздалегідь і виконуватися по ходу всього програмного

проекту, а не тільки під час приймально-здавальних випробувань. При плануванні цієї роботи необхідно розуміти, що продукт проекту не повинен володіти найвищим можливим якістю, яке недосяжне за кінцевий час. Необхідну якість продукту визначається вимогами до нього. І ще. Основне завдання забезпечення якості це не пошук помилок в готовому продукті (вихідний контроль) а їх попередження в процесі виробництва. Для прикладу, гладкість обробки деталі на токарному верстаті тільки випадково може виявитися відповідній необхідній якості в 1 мікрон, якщо шпиндель, на якому кріпиться деталь, погано центрован.

План управління якістю повинен включати в себе наступні роботи:

• *Об'єктивну перевірку відповідності програмних продуктів і технологічних операцій застосовуваним стандартам, процедур і вимог.*
• *Визначення відхилень за якістю, виявлення їх причин, застосування заходів щодо їх усунення, а також контроль виконання прийнятих заходів та їх ефективності.*
• *Представлення вищого керівництва незалежної інформації про невідповідності, не усуваються на рівні проекту.*
Крім перерахованих розділів план проекту повинен включати:
• *План управління ризиками*
• *Оцінку трудомісткості і термінів робіт*

Базове розклад проекту

Після визначення трудомісткості робіт необхідно визначити графік їх виконання і загальні терміни реалізації проекту — скласти розклад робіт за проектом. Базове розклад — затверджений план-графік із зазначеними тимчасовими фазами проекту, контрольними точками і елементами ієрархічної структури робіт. Базове розклад може бути найбільш наочно представлено діаграмою Ганта. У цій діаграмі планові операції або елементи ієрархічної структури робіт перераховані з лівої сторони, дати відображаються зверху, а тривалість операцій показана горизонтальними смужками від дати початку до дати завершення.

Базове розклад це, як правило, елемент контракту з замовником. Контрольні точки (віхи) повинні служити точками аналізу стану проекту та прийняття рішення «GO/NOT GO», тому вони повинні зримо демонструвати статус проекту. Контрольна точка «Проектування завершено» — погано.

Найбільш ефективний підхід — метод послідовних поставок: контрольна точка «Завершено тестування вимог 1, 3, 5, 7»

Якщо роботи не пов'язані між собою, то будь-яку з них ми можемо починати і завершувати, коли нам зручно. Всі роботи можна робити паралельно і в цьому випадку мінімальна тривалість проекту дорівнює тривалості самої довгої роботи. Однак, на практиці між роботами існують залежності, які можуть бути «жорсткими», наприклад, аналіз — проектування — кодування — тестування і документування конкретної функції; або «нежесткими», які можуть переглядатися або пом'якшуватися. Наприклад, послідовне виконання завдань конкретним виконавцем (можна перепланувати на іншого виконавця) або розробка базового ПЗ, яка повинна передувати розробці прикладного ПО. У цьому випадку можна створювати «заглушки» эмулирующие роботу базового ПЗ. Таким чином, діаграма Ганта для розкладу проекту виглядає як гамак, складений з безлічі ланцюжків взаємопов'язаних робіт з єдиної точки початку і завершення.

Критичний шлях проекту (Critical path) — найдовша ланцюжок робіт у проекті. Збільшення тривалості будь-якої роботи в цій ланцюжка призводить до збільшення тривалості всього проекту. У проекті завжди існує хоча б один критичний шлях, але їх може бути кілька. Критичний шлях може змінюватися під час виконання проекту. При виконанні проекту керівник повинен звертати увагу на виконання завдань на критичному шляху в першу чергу і стежити за появою інших критичних шляхів. Практична рекомендація: на критичному шляху, мають стояти роботи з нежесткими зв'язками, які завжди можна перепланувати, якщо виникає загроза зриву термінів. Щоб проілюструвати поняття критичного шляху розглянемо приклад «суперпроекту»5. Концепція проекту виглядає наступним чином.

Мета проекту. Зробити сніданок у постіль

Результати проекту. Сніданок в ліжку з вареного яйця, тости і апельсинового соку.

Ресурси. Є один оператор і звичайне кухонне обладнання.

Терміни. Проект починається на кухні в 8:00 і завершується в спальні.

Критерій приймання. Використовуються мінімальні трудові ресурси і термін. Кінцевий продукт має високу якість: яйце свіжозварене, тост теплий, холодний сік.

Обґрунтування корисності. Проект служить досягненню стратегічних цілей.

Ієрархічна структура робіт, орієнтована на кінцевий продукт, з оцінкою їх тривалості представлена на рисунку 18.

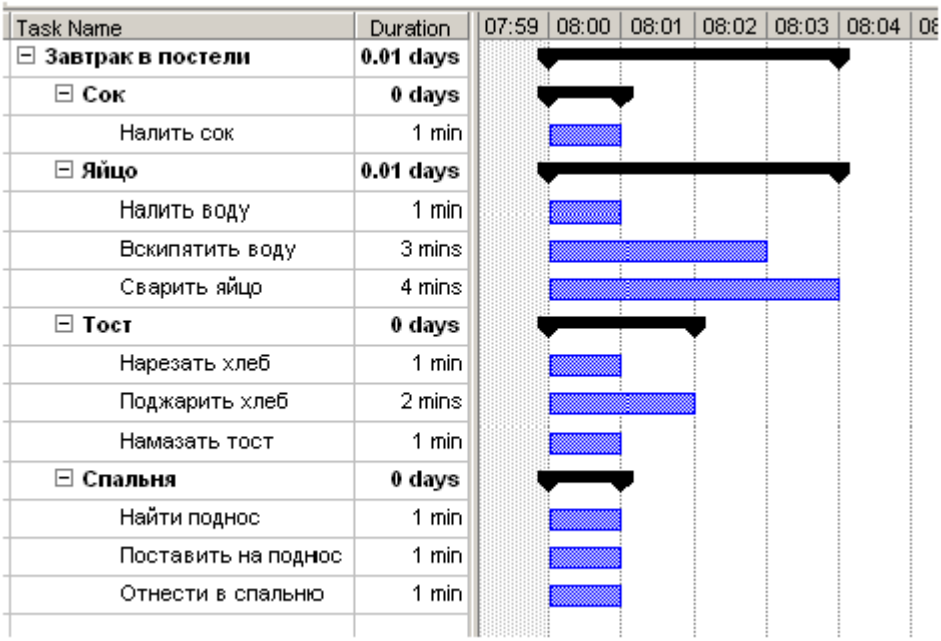


Рис. 18. Ієрархічна структура робіт «суперпроекту»

На наступному кроки ми повинні врахувати залежності між роботами, наприклад, не можна смажити хліб, поки ми його не нарізали.
З урахуванням залежностей ми отримаємо наступну діаграму розкладу нашого проекту (рис.19)

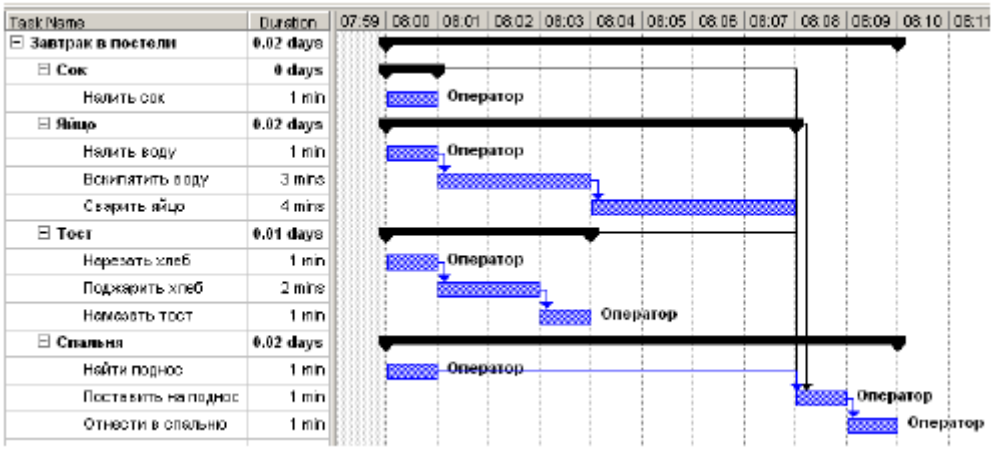


Рис.19. Діаграма розкладу «суперпроекту» з урахуванням залежностей між роботами.

В результаті ми визначили, що мінімальний термін реалізації нашого проекту становить 10 хвилин. Однак ми не можемо на цьому зупинитися, оскільки повинні ще врахувати обмеження по ресурсах. У нас тільки один оператор. Якщо ми подивимося на графік завантаженості ресурсів (рис. 20),

то побачимо, що наш критичний ресурс завантажений на першій хвилині на 400%. що неприпустимо.

Рис.20. Діаграма завантаженості ресурсів «суперпроекте»

Отже, ми повинні виконати вирівнювання ресурсів. Оскільки одним з критеріїв успіху проекту є його мінімальна тривалість, то якщо ми не хочемо її збільшувати, ми повинні виявити критичний шлях у проекті (Рис. 21) і не зрушувати роботи, які на ній знаходяться.

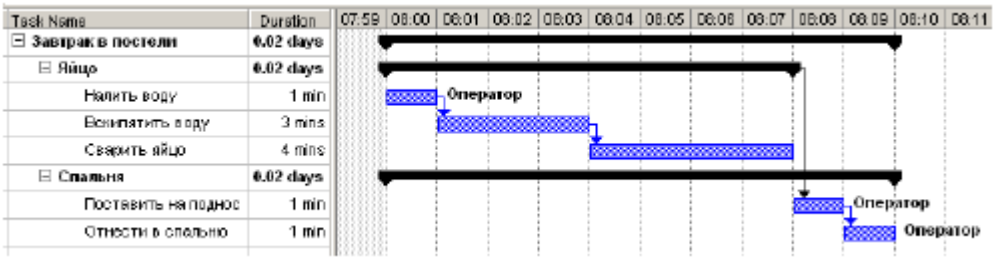


Рис. 21. Критичний шлях у «суперпроекте»

Тому, після вирівнювання ресурсів, розклад нашого проекту буде виглядати наступним чином (Рис. 22).

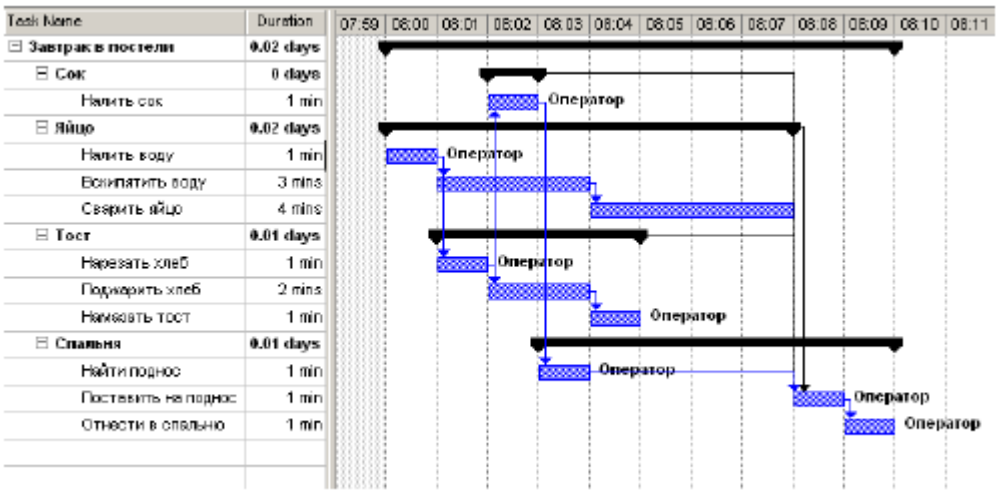


Рис. 22. Розклад «суперпроекту» після вирівнювання ресурсів

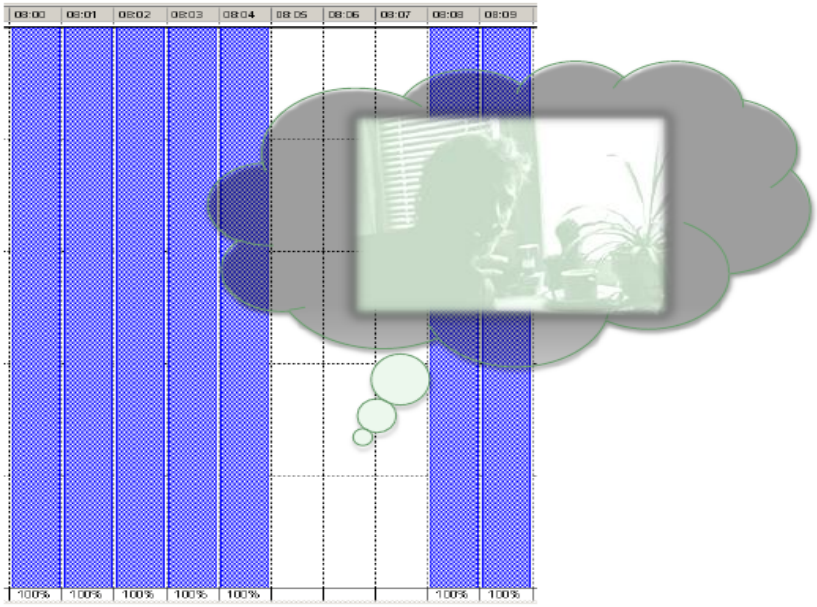


Рис. 23. Діаграма завантаженості ресурсів після вирівнювання

Тепер діаграма завантаженості ресурсів (Рис.23) виглядає прийнятно і в оператора навіть з'явилося три хвилини вільного часу на перекур. При цьому загальна тривалість реалізації проекту, як і раніше становить 10 хвилин.

Висновки

На верхньому рівні ІСР повинні знаходитися не процеси, а продукти проекту, на наступному рівні — компоненти з яких ці продукти складаються. Виділення компонентів, що складають програмний продукт, це елемент високорівневого проектування, яке ми повинні виконати на фазі планування

проекту, не чекаючи опрацювання всіх функціональних вимог до розроблюваного ПЗ.
Крім робіт, безпосередньо спрямованих на створення програмного забезпечення, в плані проекту повинні бути передбачені необхідні ресурси для забезпечення робіт з наступних процесів:

• *управління вмістом;*
• *управління конфігураціями,*
• *управління якістю,*
• *управління ризиками,*
• *управління проектом.*

У проекті завжди існує хоча б один критичний шлях, але їх може бути кілька. Критичний шлях може змінюватися під час виконання проекту. При виконанні проекту керівник повинен звертати увагу на виконання завдань на критичному шляху в першу чергу і стежити за появою інших критичних шляхів.

Додаткова література

1. «PMBOK. Руководство к Своду знаний по управлению проектами», 3-е изд., PMI, 2004.

Лекція 10-12. ТЕМА 5 ***Управління ризиками проекту. Основні поняття. Планування управління ризиками. Ідентифікація ризиків. Якісний аналіз ризиків. Кількісний аналіз ризиків. Планування реагування на ризики. Головні ризики програмних проектів та способи реагування. Управління проектом, спрямоване на зниження ризиків. Моніторинг і контроль ризиків.***

В силу специфіки галузі, програмна інженерія залишається і, в найближчому майбутньому, буде залишатися виробництвом з високим рівнем ризиків [1]. Якщо задуматися, то все, що ми робимо, керуючи проектом розробки ПО, спрямоване на боротьбу з ризиками, не вкластися в термін, перевитрачати ресурси, розробити не той продукт, який потрібен. Визначення ризику було дано в попередній лекції.

Ризик характеризується наступними характеристиками [2] (рис. 24):

• Причина або джерело. Явище, обставина обумовлює настання ризику.
• Симптоми ризику, вказівка на те, що подія ризику сталося або ось-ось відбудеться. Першопричина нам може бути не наблюдаема, наприклад, заразилися грипом. Ми спостерігаємо деякі симптоми - піднялася температура.
• Наслідки ризику. Проблема або можливість, яка може реалізуватися в проекті в результаті події ризику.
• Вплив ризику. Вплив реализовавшегося ризику на можливість досягнення цілей проекту. Вплив зазвичай стосується вартості, графіка і технічних характеристик розроблюваного продукту. Багато ризики відбуваються частково і надають відповідне негативне або позитивне вплив на проект.
Ризик це завжди імовірність і наслідки. Наприклад, завжди є ймовірність того, що метеорит впаде на офіс центру програмних розробок, і це матиме катастрофічні наслідки для проекту. Проте ймовірність настання цієї події настільки мала, що ми в більшості проектів приймаємо це ризик і не намагаємося їм управляти.

Майк Ньюелл, віце-президент компанії PSM Consulting, розповідав, як він пояснює аудиторії на своїх лекціях, що таке ризик. Він пропонує зіграти в кості на таких умовах, якщо на кубику випадає шістка, то він виграє. Якщо будь — яке інше число, то виграє слухач. Ставка по 1 долару. Зазвичай, більша частина аудиторії погоджується зіграти на таких умовах. Майк піднімає ставки: $10, $100, $1000. Поступово кількість бажаючих пограти стає все менше і менше. При ставці $1000, як правило, бажають ризикувати не залишається.

Прийнято [3] виділяти дві категорії ризиків:

• «Відомі - невідомі». Це ті ризики, які можна ідентифікувати і піддати аналізу. Щодо таких ризиків можна спланувати відповідні дії.
• «Невідомі - невідомі». Ризики, які неможливо ідентифікувати і, отже, спланувати дії.

Рис.24. Приклад характеристик ризику

Невідомі ризики це непередбачені обставини. Єдине, що ми можемо в цьому випадку зробити, це створити управлінський резерв бюджету проекту на випадок незапланованих, але потенційно можливих змін. На витрачання цього резерву менеджер проекту, як правило, зобов'язаний отримувати схвалення вищого керівництва. Управлінські резерви на непередбачені обставини не входять в базовий план по вартості проекту, але включаються в бюджет проекту. Вони не розподіляються за проектом, як бюджет, і тому не враховуються при розрахунку освоєного обсягу.

Девіз розробників ПЗ Microsoft [2]: «Ми не боремося з ризиками — ми ними керуємо». Цілі управління ризиками проекту — зниження ймовірності виникнення та/або значущості впливу несприятливих для проекту подій. Адекватне управління ризиками в компанії — ознака зрілості виробничих процесів. Тому Демарко пише [1]: «Розглядати тільки сприятливі сценарії і вбудовувати їх у план проекту — справжнє дитинство. І все ж ми постійно так чинимо. ...Якщо тих, хто говорить про можливі проблеми до відкриття проекту, називають troublemakers, а тих, хто здає проект через 2 місяці після обіцяного терміну, працюючи при цьому за 6080 годин на тиждень, — героями, то у вас погана команда».

Планування управління ризиками

Управління ризиками-це певна діяльність, яка виконується в проекті від його початку до завершення. Як і будь-яка інша робота в проекті управління ризиками вимагає часу і витрат ресурсів. Тому ця робота обов'язково

повинна плануватися. Планування управління ризиками — це процес визначення підходів і планування операцій з управління ризиками проекту. Ретельне і детальне планування управління ризиками дозволяє:

• виділити достатню кількість часу і ресурсів для виконання операцій з управління ризиками,

• визначити загальні підстави для оцінки ризиків,

• підвищити ймовірність успішного досягнення результатів проекту.

Планування управління ризиками має бути завершено на ранній стадії планування проекту, оскільки воно украй важливо для успішного виконання інших процесів.

Згідно з [3] вихідними даними для планування управління ризиками є:

• Ставлення до ризику та толерантність до ризику організацій та осіб, які беруть участь у проекті, впливає на план управління проектом. Воно повинно бути зафіксовано у викладі основних принципів і підходів до управління ризиками.

• Стандарти організації. Організації можуть мати заздалегідь розроблені підходи до управління ризиками, наприклад категорії ризиків, загальні визначення понять і термінів, стандартні шаблони, схеми розподілу ролей і відповідальності, а також визначені рівні повноважень для прийняття рішень.

• Опис змісту проекту докладно описує результати постачання проекту і роботи, необхідні для створення цих результатів поставки.

• План управління проектом, формальний документ, в якому зазначено, як буде виконуватися проект і як відбуватиметься моніторинг і управління проектом.

План управління ризиками зазвичай включає в себе наступні елементи:

• Визначення підходів, інструментів і джерел даних, які можуть використовуватися для управління ризиками в даному проекті.

• Розподіл ролей і відповідальності. Список позицій виконання, підтримки та управління ризиками для кожного виду операцій, включених у план управління ризиками, призначення співробітників на ці позиції і роз'яснення їх відповідальності.

• Виділення ресурсів і оцінка вартості заходів, необхідних для управління ризиками. Ці дані включаються у базовий план по вартості проекту.

• Визначення строків та частоти виконання процесу управління ризиками на протязі всього життєвого циклу проекту, а також визначення операцій з управління ризиками, які необхідно включити в розклад проекту.

• Категорії ризиків. Структура, на підставі якої проводиться систематична і всебічна ідентифікація ризиків з потрібним ступенем

деталізації. Таку структуру можна розробити за допомогою складання ієрархічної структури ризиків (Малюнок 25).

• *Загальні підходи для визначення рівнів ймовірності, шкали впливу і близькості ризиків на проект.*

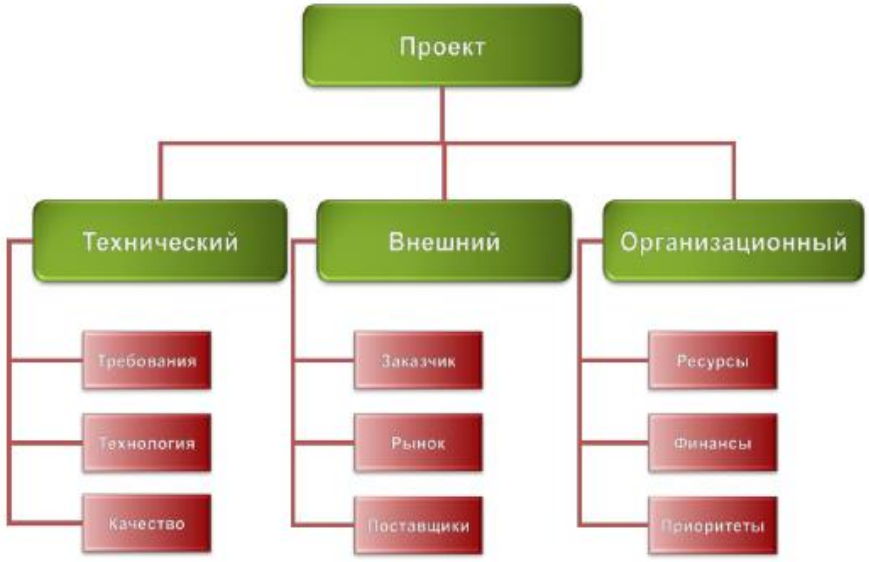


Рис.25. Приклад ієрархічної структури ризиків проекту

Шкала оцінки впливу відображає значимість ризику (Таблиця 2) у разі його виникнення. Шкала оцінки впливу може відрізнятися в залежності від потенційно порушеної ризиком мети, типу і розміру проекту, прийнятими в організації стратегіями і його фінансовим станом, а також від чутливості організації до конкретного виду впливів.

Вага	**Значення**	**Критерій**
3	Катастрофічні	Втрати понад $100K
2	Критичні	Втрати від $10K до $100K
1	Помірні	Втрати менше $10K

Таблиця 2. Приклад шкали оцінки впливу ризиків

Хоча ризик може впливати і на терміни проекту, і на якість одержуваного продукту, але всі ці відхилення можуть бути оцінені в грошовому еквіваленті. Наприклад, наслідки затримка за термінами для розробки на замовлення може бути виражена у сумі грошових санкцій, визначених у контракті. Схожа шкала може бути застосована для оцінки ймовірності настання ризику (Таблиця 3).

Вага	Значення	Критерій
3	Дуже ймовірно	Шанси настання дуже великі
2	Можливо	Шанси рівні
1	Мало ймовірно	Настання події вельми сумнівно

Таблиця 3. Приклад шкали оцінки ймовірності здійснення ризику

Ще однією важливою характеристикою ризику є близькість його настання. Природно, що при інших рівних умовах ризиків, які можуть відбутися вже завтра, сьогодні слід приділяти більше уваги, ніж тим, які можуть відбутися не раніше, ніж через півроку. Для шкали оцінки близькості ризику може бути застосована, наприклад, наступна градація: дуже скоро, не дуже скоро, дуже скоро.

Ідентифікація ризиків

Ідентифікація ризиків — це виявлення ризиків, здатних вплинути на проект, та документальне оформлення їх характеристик. Це ітеративний процес, який періодично повторюється протягом проекту, оскільки в межах його життєвого циклу можуть виявлятися нові ризики. Вихідні дані для виявлення і опису характеристик ризиків можуть братися з різних джерел. В першу чергу це база знань організації. Інформація про виконання попередніх проектів може бути доступна в архівах попередніх проектів. Слід пам'ятати, що проблеми завершених і виконуваних проектів, це, як правило, ризики в нових проектах. Іншим джерелом даних про ризики проекту може бути різноманітна інформація з відкритих джерел, наукових праць, маркетингова аналітика та інші дослідницькі роботи в даній області. Нарешті, багато форумів з програмування можуть дати безцінну інформацію про виниклі раніше проблеми у схожих проектах.

Кожен проект замислюється і розробляється на підставі ряду гіпотез, сценаріїв і припущень. Як правило, в описі змісту проекту перелічуються прийняті допущення — це чинники, які для цілей планування вважаються вірними, реальними або певними без залучення доказів. Невизначеність у припущеннях проекту слід також обов'язково розглядати в якості потенційного джерела виникнення ризиків проекту. Аналіз допущення дозволяє ідентифікувати ризики проекту, що походять від неточності, несумісності або неповноти припущень.

Для збору інформації про ризики можуть застосовуватися різні підходи. Серед цих підходів найбільш поширені:

• Опитування експертів
• Мозковий штурм
• Метод Дельфі
• Картки Кроуфорда

Мета опитування експертів - ідентифікувати і оцінити ризики шляхом інтерв'ю відповідних кваліфікованих фахівців. Спеціалісти висловлюють свою думку про ризики і дають їм оцінку, виходячи зі своїх знань, досвіду та наявної інформації. Цей метод може допомогти уникнути повторного наступу на одні і ті ж граблі. Перед опитуванням експерт повинен отримати всю необхідну базову інформацію. Діяльність експертів необхідно направляти, задаючи питання. Під час опитування уся інформація, яка видається експертом, повинна записуватися і зберігатися. При роботі з декількома експертами вихідна інформація узагальнюється і доводиться до відома всіх задіяних експертів.

До участі у мозковому штурмі залучаються кваліфіковані фахівці, яким дають домашнє завдання - підготувати свої судження щодо певної категорії ризиків. Потім проводиться загальні збори, на якому фахівці по черзі висловлюють свої думки про ризики. Важливо: суперечки та зауваження не допускаються. Всі ризики записуються, групуються за типами і характеристиками, кожному ризику дається визначення. Мета - скласти первинний перелік можливих ризиків для подальшого відбору і аналізу.

Метод Дельфі багато в чому схожий на метод мозкового штурму. Однак є важливі відмінності. По-перше, при застосуванні цього методу експерти беруть участь в опитуванні анонімно. Тому результат характеризується меншою суб'єктивністю, меншою упередженістю і меншим впливом окремих експертів. По-друге, опитування експертів проводиться в кілька етапів. На кожному етапі модератор розсилає анкети, збирає і обробляє відповіді. Результати опитування розсилаються експертам знову для уточнення їх думок і оцінок. Такий підхід дозволяє досягти якоїсь спільної думки фахівців про ризики.

Для швидкого виявлення ризиків можна скористатися ще однією з методик соціометрії є відомою як "Картки Кроуфорда" [5] . Суть цієї методики в наступному. Збирається група експертів 7-10 чоловік. Кожному учаснику міні-дослідження лунає по десять карток (для цього цілком підійде звичайний папір для нотаток). Ведучий задає питання: "Який ризик є найбільш важливим у цьому проекті?" Всі респонденти повинні записати найбільш, на їхню думку, важливий ризик в даному проекті. При цьому ніякого обміну думками не повинно бути. Ведучий робить невелику паузу, після чого питання повторюється. Учасник не може повторювати відповіді

один і той самий ризик. Після того як питання прозвучить десять разів, у розпорядженні провідного з'являться від 70 до 100 карток із відповідями. Якщо група підібрана добре (в тому розумінні, що в неї входять люди з різними точками зору), ймовірність того, що учасники експерименту вкажуть більшість значущих для проекту ризиків, вельми висока. Залишається скласти список названих ризиків і роздати його учасникам для внесення змін і доповнень.

В якості джерела інформації при виявленні ризиків можуть служити різні доступні контрольні списки ризиків проектів розробки ПЗ, які слід проаналізувати на придатність до даного конкретного проекту. Наприклад, Барії Боем [6] наводить список 10 найбільш поширених ризиків програмного проекту:

1. Дефіцит фахівців.
2. Нереалістичні терміни і бюджет.
3. Реалізація не відповідає функціональності.
4. Розробка неправильного користувальницького інтерфейсу.
5. "Золота сервіровка", перфекціонізм, непотрібна оптимізація та покращення деталей.
6. Безперервний потік змін.
7. Брак інформації про зовнішні компоненти, що визначають оточення системи або залучених в інтеграцію.
8. Недоліки в роботах, що виконуються зовнішніми (по відношенню до проекту) ресурсами.
9. Недостатня продуктивність одержуваної системи.
10. "Розрив" у кваліфікації фахівців різних областей знань.

Демарко і Лістер [1] наводять свій список з п'яти найбільш важливих джерел ризиків проекту розробки ПО:

1. Вади календарного планування
2. Плинність кадрів
3. Роздування вимог
4. Порушення специфікацій
5. Низька продуктивність

Не існує вичерпних контрольних списків ризиків програмного проекту, тому необхідно уважно аналізувати особливості кожного конкретного проекту. Результатом ідентифікації ризиків має стати перелік ризиків з описом їх основних характеристик: причини, умови, наслідків та збитків. Якщо

повернутися до прикладу проекту створення Автоматизованої системи продажу документації», який ми розглядали в попередніх лекціях, то список головних виявлених ризиків може виглядати наступним чином:

Таблиця 4 Список ризиків проекту створення Автоматизованої системи продажу документації»

Причина	Умови	Наслідки	Збиток
Вимоги не ясні.	Відсутність опису сценаріїв використання системи.	Затримка початку розробки прикладного ПО. Великий обсяг переробок.	Затримки в термінах здачі готового продукту і додаткові трудовитрати.
Недолік кваліфікованих кадрів.	Недолік кваліфікованих кадрів.	Велика кількість помилок. Великі витрати на їх виправлення.	Затримки в термінах здачі готового продукту і додаткові трудовитрати.
Плинність кадрів.	Часта зміна учасників команди.	Низька продуктивність при введенні нових учасників в проект.	Затримки в термінах здачі готового продукту і додаткові трудовитрати.

За процесом ідентифікації ризиків слід процес їх якісного аналізу.

Якісний аналіз ризиків

Якісний аналіз ризиків включає в себе розстановку рангів для ідентифікованих ризиків. При аналізі ймовірності і впливу передбачається, що ніяких заходів щодо попередження ризиків не проводиться.

Якісний аналіз ризиків включає:

- *Визначення ймовірності реалізації ризиків.*
- *Визначення тяжкості наслідків реалізації ризиків.*
- *Визначення рангу ризику по матриці «вірогідність — наслідки».*
- *Визначення близькості настання ризику.*
- *Оцінка якості використаної інформації.*

Для якісної оцінки ймовірності реалізації ризику та визначення тяжкості наслідків його реалізації застосовується, як правило, загальноприйняті в організації шкали. Для визначення рангу ризику використовується матриця

ймовірностей і наслідків (Малюнок 26). Ранг ризику визначається добутком ваги ймовірності та значущості наслідків. Можуть, звичайно, існувати і більш складні шкал оцінок ймовірностей, значущості наслідків і рангу ризиків. Зустрічалися шкали, які містили до 10 градацій. Найбільш прагматичний підхід — це використовувати трирівневе ранжування.

Продовжуючи розгляд прикладу проекту створення Автоматизованої системи продажу документації», матриця рангів головних виявлених ризиків може виглядати наступним чином (Таблиця 5).

Таблиця 5. Матриця рангів головних виявлених ризиків проекту створення Автоматизованої системи продажу документації»

Причина	**Ймовірність**	**Вплив**	**Ранг**
Вимоги не ясні	Дуже ймовірно	Катастрофічні	9
Недолік кваліфікованих кадрів.	Дуже ймовірно	Критичні	6
Плинність кадрів.	Можливо	Критичні	4

Рис.26. Ранг ризику і матриця ймовірностей і наслідків

Для оцінки ризиків необхідна точна і адекватна інформація. Використання неточної інформації веде до помилок в оцінці. Невірна оцінка ризику також є ризиком.

Критерії оцінки якості використовуваної при аналізі інформації виглядають наступним чином:

• Ступінь розуміння ризику.

• Доступність і повнота інформації про ризик.

• Надійність, цілісність та достовірність джерел даних.

Результатом якісного аналізу ризиків є їх детальний опис (Таблиця 6).

Таблиця 6. Приклад картки з описом ризику

Номер: R-101	**Категорія**: Технологічний.
Причина: Брак кваліфікованих кадрів **Наслідки**: Низька продуктивність розробки **Вірогідність**: Дуже ймовірно. **Близькість**: Дуже скоро.	**Симптоми**: Розробники будуть використовувати нову платформу .— J2EE. **Вплив**: Збільшення термінів і трудомісткості розробки. **Ступінь впливу**: Критична. **Ранг**: 6.

Вихідні дані: «Зміст проекту», «План забезпечення ресурсами», Протоколи нарад №21 від 01.06.2008, №27 від 25.06.2008.

Результати якісного аналізу використовуються в ході подальшого кількісного аналізу ризиків та планування реагування на ризики.

Кількісний аналіз ризиків

Кількісний аналіз проводиться щодо тих ризиків, які в процесі якісного аналізу були кваліфіковані як такі, що мають високий і середній ранг. Для кількісного аналізу ризиків можуть бути використані наступні методи:

- *Аналіз чутливості.*
- *Аналіз дерева рішень.*
- *Моделювання та імітація.*

Аналіз чутливості допомагає визначити, які ризики володіють найбільшим потенційним впливом на проект. У процесі аналізу встановлюється, якою мірою невизначеність кожного елемента проекту відображається на досліджуваній мети проекту, якщо інші невизначені елементи приймають базові значення. Результати представляються, як правило, у вигляді діаграми «торнадо». Малюнок 27 являє приклад такої діаграми, яка відображає вплив на проектні роботи різних факторів професіоналізму розробників ПЗ [7].

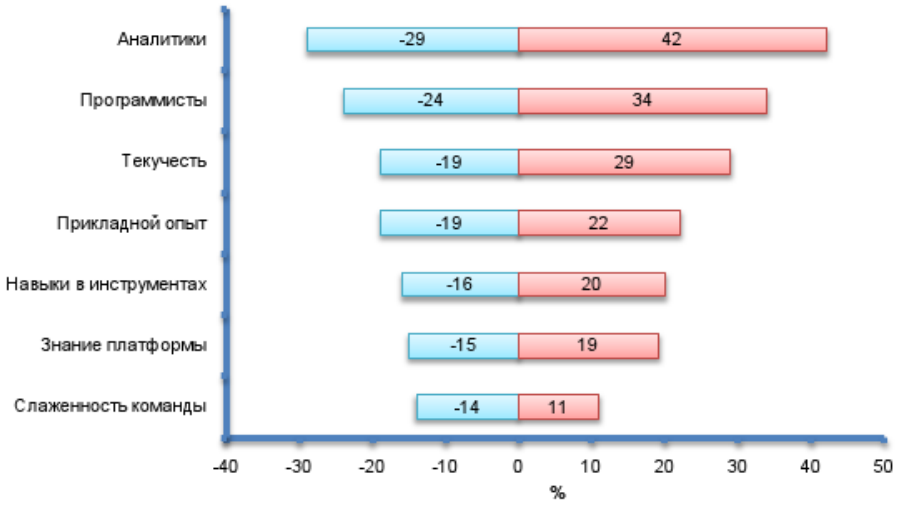


Рис .27. Вплив факторів професіоналізму розробників З роботи за проектом.

Аналіз наслідків можливих рішень проводиться на основі вивчення діаграми дерева рішень, яка описує розглянуту ситуацію з урахуванням кожної з наявних можливостей вибору і можливого сценарію. Малюнок 28 являє приклад діаграми дерева рішень на дугах якій проставлені ймовірності та витрати при розвитку подій за тим чи іншим сценарієм. Критерієм для прийняття рішення служить математичне очікування втрат від його прийняття.

Рис.28. Приклад аналіз дерева рішень при виборі купувати чи виробляти необхідну для проекту бібліотеку візуальних компонентів (VCL).

При **моделюванні ризиків** проекту використовується модель для визначення наслідків від впливу докладно описаних невизначеностей на результати проекту в цілому. Моделювання зазвичай проводиться за допомогою методу Монте-Карло.

Цікавий приклад такої моделі — система Riskology від Демарко і Лістера, який ілюструє застосування методу Монте-Карло для отримання інформації про те, який запас часу буде необхідний для того, щоб подолати вплив всіх некерованих ризиків проекту, наведено в джерелі [8]. Модель дозволяє врахувати п'ять основних (Малюнок 29) і п'ять додаткових ризиків проекту.

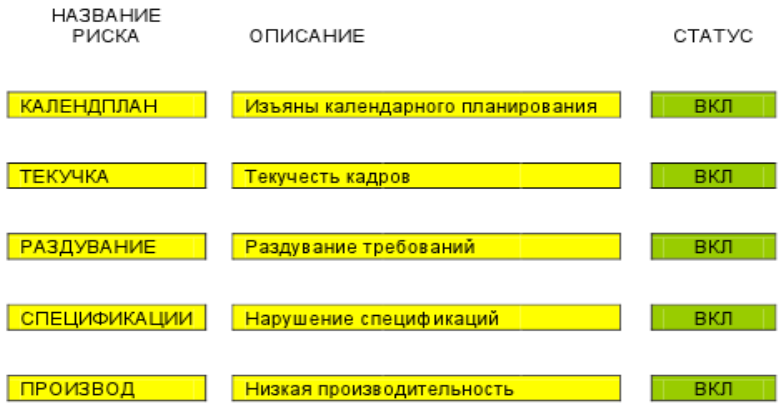


Рис. 29. П'ять основних факторів ризику програмного проекту, враховуються в моделі Riskology

Характеристики зумовлених в системі Riskology ризиків користувач може змінити, задавши значення мінімальної, максимальної та найбільш ймовірної затримки термінів здачі проекту з-за впливу даного ризику. Можна включити в модель додаткові власні ризики. Результат моделювання за методом Монте-Карло буде представлений у вигляді гістограми розподілу терміну завершення оцінюваного проекту (Малюнок 30).

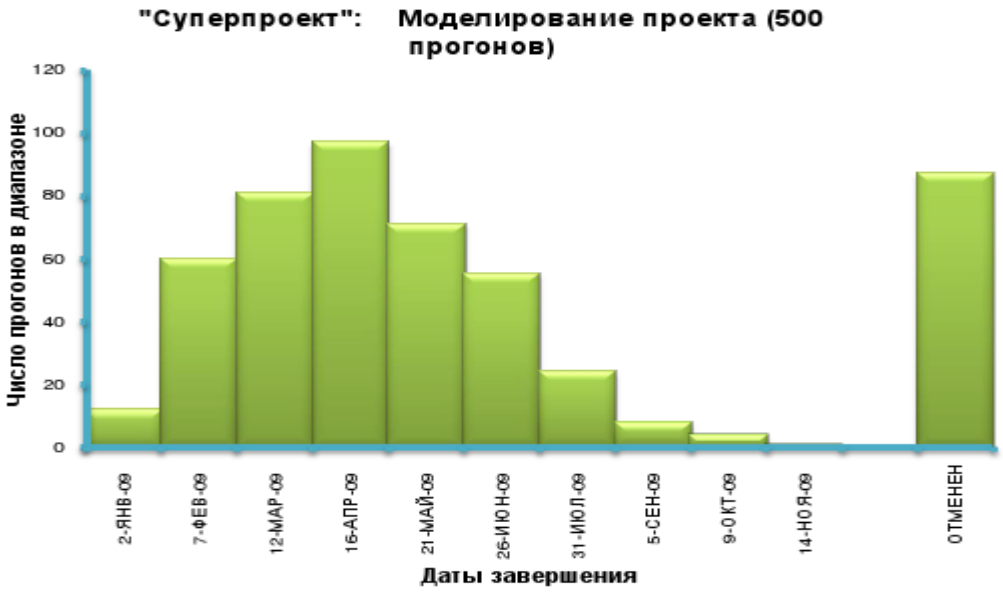


Рис.30. Гістограма розподілу можливого терміну завершення проекту, розрахована за результатами моделювання методом Монте-Карло

На діаграмі також наведено кількість випадків, приблизно 80 з 500 прогонів, в яких проект, згідно з результатами моделювання, був скасований до свого завершення.

Планування реагування на ризики

Планування реагування на ризики — це процес розробки шляхів і визначення дій по збільшенню можливостей і зниження загроз для цілей проекту. Даний процес починається після проведення якісного і кількісного аналізу ризиків. Заплановані операції з реагування на ризики повинні відповідати серйозності ризику, бути економічно ефективними у вирішенні проблеми, своєчасними, реалістичними в контексті проекту і погодженими з усіма учасниками. Згідно [3] можливі чотири методи реагування на ризики:

• *Ухилення від ризику (risk avoidance).*
• *Передача ризику (risk transference).*
• *Зниження ризиків (risk mitigation).*
• *Прийняття ризику (risk acceptance).*

Ухилення від ризику передбачає зміну плану управління проектом таким чином, щоб виключити загрозу, викликану негативним ризиком, захистити цілі проекту від наслідків ризику або послабити цілі, що знаходяться під загрозою (наприклад, зменшити зміст проекту). Деякі ризики, що виникають на ранніх стадіях проекту, можна уникнути за допомогою уточнення вимог, отримання додаткової інформації або проведення експертизи. Наприклад, ухилитися від ризику можна, якщо відмовитися від реалізації ризикованого функціонального вимоги або самостійно розробити необхідний програмний компонент, замість очікування поставок продукту від субпідрядника.

Передача ризику передбачає перекладення негативних наслідків загрози з відповідальністю за реагування на ризик на третю сторону. Передача ризику просто переносить відповідальність за його управління іншій стороні, але ризик при цьому нікуди не діється. Передача ризику практично завжди передбачає виплату премії за ризик стороні, що приймає на себе ризик. Наприклад, замовлення на стороні розробки ризикованого компонента за фіксованою ціною. В IT часто доводиться формулювати ризики у вигляді припущень, тим самим передаючи його замовнику. Наприклад, оцінюючи

проект впровадження, ми можемо записати допущення про те, що виробник не змінить вартість ліцензій на базове ПЗ.

Зниження ризиків передбачає зниження ймовірності і/або наслідків негативної ризикованої події до прийнятних меж. Прийняття попереджувальних заходів щодо зниження ймовірності настання ризику або його наслідків часто виявляються більш ефективними, ніж зусилля по усуненню негативних наслідків, що вживаються після настання події ризику. Наприклад, раннє дозвіл архітектурних ризиків знижує втрати при достроковому закритті проекту. Або регулярна ревізія поставок замовником може знизити ймовірність ризику його незадоволеності кінцевим результатом. Якщо у проектній команді висока ймовірність звільнення співробітників, то введення на початковій стадії проект додаткових (надлишкових) людських ресурсів знижує втрати при звільненні членів команди, оскільки не буде витрат на «в'їзд» в проектний контекст нових учасників.

Прийняття ризику означає, що команда проекту усвідомлено прийняла рішення не змінювати план управління проектом у зв'язку з ризиком або не знайшла підходящої стратегії реагування. Ми змушені приймати всі невідомі ризики». Прийняття це те, що завжди відбувається, коли ми взагалі не керуємо ризиками. Якщо ж ми керуємо ризиками, то ми можемо страхувати ризики, закладаючи резерв оцінки терміну завершення та/або трудовитрат. Проактивне ставлення до прийнятих ризиків може полягати в розробці план реагування на ризики. Цей план може бути введено в дію тільки при заздалегідь визначених умовах, якщо є впевненість і достатню кількість ознак того, що цей план буде успішно виконаний. Важливо пам'ятати про вторинних ризики (Secondary Risks), що виникають в результаті застосування реагування на ризики, які теж повинні бути ідентифіковані, проаналізовано і при необхідності включені в список керованих ризиків.

Головні ризики програмних проектів та способи реагування

Список з п'яти головних причин провалу програмних проектів — наступний:

- *Вимоги замовника відсутні / не повні / схильні до частих змін.*
- *Відсутність необхідних ресурсів і досвіду.*
- *Відсутність робочого взаємодії з замовником.*
- *Неповнота планування. «Забуті роботи».*
- *Помилки в оцінках трудоемкостей і термінів робіт.*

Це звучить банально, але скільки б про це не говорили раніше, як і раніше, доводиться стикатися з програмними проектами, в яких відсутні які-небудь певні цілі і вимоги. Цитата з життя: «Була б розроблена хороша програма, а який процес автоматизувати з її допомогою, ми знайдемо». До цього можна додати тільки одне: «Коли людина не знає, до якої пристані вона тримає шлях, для нього жоден вітер не буде попутним» (Сенека Луцій Анею, філософ, 65-3 до н. е..)

До часто упускаемым вимог можна віднести:

• Функціональні
• Програми установки, настройки конфігурації.
• Міграція даних.
• Інтерфейси із зовнішніми системами.
• Довідкова система.
• Загальносистемні
• Продуктивність.
• Надійність.
• Відкритість.
• Масштабованість.
• Безпека.
• Кросплатформенность.
• Ергономічність.

Як правило, ці вимоги «спливають» при підготовці та проведенні приймально-здавальних випробувань і можуть сильно затримати проект по часу і збільшити витрати на його реалізацію. Щоб цього не відбувалося, слід досягати угоди з замовником по всіх перерахованих пунктів краще ще на стадії ініціації проекту. Наприклад, якщо вимоги портируемости продукту на різні апаратно-програмні платформи немає, то доцільно включити в розділ концепції з допущеннями проекту.

Якщо ймовірність змін вимог проекту висока, то можливі наступні підходи для реагування на даний ризик:

• Переоцінка проекту кожен раз, коли вимоги додаються змінюються (ухилення).
• Ітераційна розробка. Контракт з компенсацією витрат на основі «Time & Materials» (передача ризику Замовнику).

• Облік в оцінках трудомісткості і термінів можливості зростання вимог, наприклад, на 50% (резервування ризику).

І ще, при зборі вимог слід дотримуватися принцип мінімалізму Вольтера: «Розповідь закінчено не тоді, коли у нього нічого додати, а тоді, коли з нього нічого більше викинути». Для більшості програмних продуктів застосуємо принцип Парето: 80% цінності продукту укладені лише в 20% вимог до нього. Якщо в проекті недостатньо кваліфікованих фахівців, то ми можемо знизити наслідки цього ризику, застосувавши такі дії:

• *Залучити експертів-консультантів на початкових етапах.*
• *Враховувати в оцінках трудомісткості витрати на навчання співробітників.*
• *Зменшувати втрати від плинності кадрів, залучаючи на початковому етапі надмірне число учасників.*
• *Врахувати в оцінках «час розгону» для нових співробітників.*

Для встановлення відкритих і довірчих відносин із замовником, необхідно робити наступні кроки:

• *Постійна взаємодія.*
• *Узгодження користувальницьких інтерфейсів і розробка прототипу продукту.*

• Періодичні поставки тестових версій кінцевим користувачам для їх оцінки.

При плануванні робіт по проекту часто «забувають»:

• *Навчання.*
• *Координація робіт.*
• *Уточнення вимог.*
• *Управління конфігураціями.*
• *Розробка і підтримка скриптів автоскладання.*
• *Розробка автотестів.*
• *Створення тестових даних.*
• *Обробка запитів на зміни.*

І ще. Не варто сподіватися, що учасники проекту будуть щотижня по 40 годин працювати саме над вашим проектом. Є безліч причин, по яких вони

не зможуть працювати за проектом 100% свого часу. До списку найбільш поширених причин цього відносяться:

• *Супровід діючих систем.*
• *Підвищення кваліфікації.*
• *Участь у підготовці техніко-комерційних пропозицій.*
• *Участь у презентаціях.*
• *Адміністративна робота.*
• *Відпустки, свята, лікарняні.*

Рекомендація, планувати, що розробники, які призначені в ваш проект на 100% будуть реально працювати над вашими завданнями в середньому від 24 до 32 годин на тиждень. Помилок в оцінках трудомісткості і термінів проекту і походів, які дають змогу їх мінімізувати, буде присвячена наступна лекція.

Управління проектом, спрямоване на зниження ризиків

На стадії ініціації проекту оцінка його трудомісткості має похибка від -50 до +100% [4]. Це, якщо оцінка хороша! А якщо погана, то невизначеність, а отже, і ризики зірвати терміни і перевищити планову трудомісткість, можуть бути в рази більше. Якщо не докладати спеціальних зусиль цей «дамоклів меч» невизначеності буде висіти над проектом на всьому його протязі (Рисунок 31). Проектом слід керувати так, щоб ризики несвоєчасної здачі і перевитрати ресурсів постійно знижувалися.

Раніше говорилося про те, що 80% цінності розробки обумовлена лише 20% вимог до продукту, без реалізації яких продукт для замовника стає просто непотрібним. Інші вимоги, як правило, так звані «прикрашення», від частини яких замовник, як правило, може відмовитися, щоб отримати проект у строк. Тому слід в першу чергу реалізовувати ключові функціональні вимоги.

Але є і ще архітектурні ризики. Відомо, що закон Парето застосуємо і до споживання обчислювальних ресурсів: 80% споживання ресурсів (час, пам'ять) припадає на 20% компонентів. Тому, необхідно реалізовувати архітектурно-значущі вимоги так само в першу чергу, створюючи «представницький» прототип майбутньої системи, який прострілює» весь стек, застосовуваних технологій. Прототип дозволить виміряти і оцінити загальносистемні властивості майбутнього продукту: доступність, швидкодія, надійність, масштабованість і ін. (рис.32).

Помилка - реалізовувати спочатку легкі вимоги, щоб продемонструвати швидкий прогрес проекту.

Рис. 31. Невизначеність не зменшується, якщо управління не спрямована на раннє дозвіл ризиків

Рис. 32. Визначення пріоритетів вимог на перші ітерації проекту

Управління, націлене на зниження ризиків, дозволяє істотно знизити невизначеність на ранніх стадіях проекту (рис. 33).

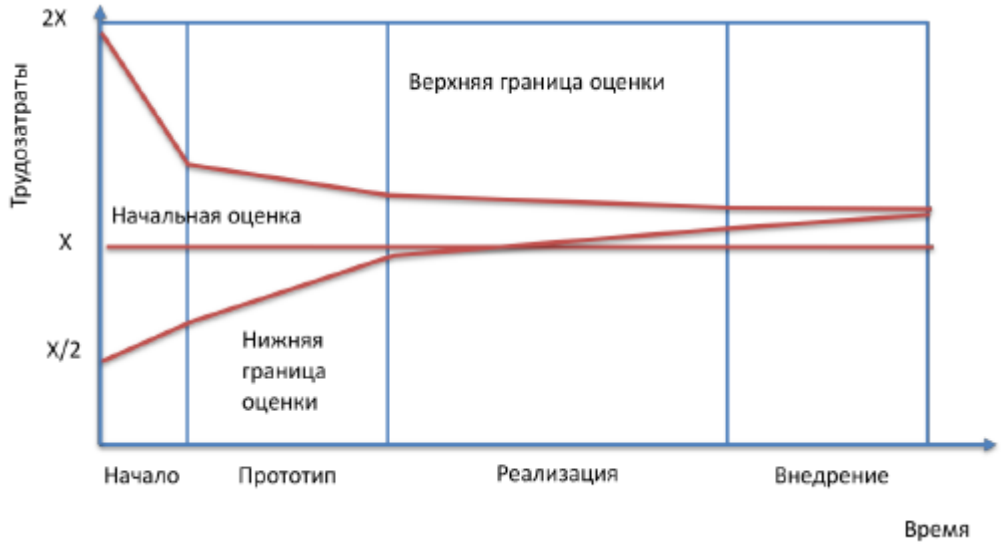


Рис. 33. Управління, націлене на зниження ризиків, дозволяє зменшувати невизначеність

Опрацювання ключових функціональних вимог і детальне планування їх реалізації дозволяє зменшити розкид початкових оцінок, приблизно, в 2 рази: від -30 до +50%. Детальне проектування і розробка прототипу майбутньої системи дозволить отримати ще більш точні оцінки загальної трудомісткості: від -10% до +15%.

Може виявитися так, що за результатами прототипування, уточнені оцінки сумарної трудомісткості виявляться неприйнятними. У цьому разі проект доведеться закрити достроково, але втрати при цьому будуть значно менші, ніж у випадку, якщо те ж саме станеться, коли проект вже в 2 рази перевищить первісну оцінку трудомісткості. Якщо із замовником не вдається знайти взаємоприйнятне рішення при первісній оцінці проекту, то розумно спробувати домовитися про виконання проекту в 2 етапи з самостійним фінансуванням:

1. Дослідження. Бізнес-аналіз, уточнення вимог, проектування і прототипироание рішення, уточнення сумарних оцінок трудовитрат. Ця робота, як правило, вимагає 10 % загальних трудовитрат і 20% часу всього проекту.

2. Безпосередньо реалізація. Якщо уточнені оцінки трудовитрат виявляться прийнятними для замовника.

Моніторинг і контроль ризиків

Управління ризиками має здійснюватися на протязі всього проекту. Моніторинг та управління ризиками — це процес ідентифікації, аналізу та планування реагування на нові ризики, відстеження раніше ідентифікованих ризиків, а також перевірки і виконання операцій реагування на ризики та оцінка ефективності цих операцій. В процесі моніторингу і управління ризиками використовуються різні методики, наприклад, аналіз трендів і відхилень, для виконання яких необхідні кількісні дані про виконання, зібрані в процесі виконання проекту.
Моніторинг і управління ризиками включає в себе наступні завдання:

• *Перегляд ризиків.*
• *Аудит ризиків.*
• *Аналіз відхилень і трендів.*

Перегляд ризиків повинен проводитися регулярно, згідно з розкладом. Управління ризиками проекту має бути одним із пунктів порядку денного всіх нарад команди проекту. Непогано починати кожен статус мітинг з питання: «Ну і які ще неприємності нас очікують?» Ідентифікація нових ризиків, перегляд відомих ризиків відбувається з використанням процесів, описаних раніше.

Аудит ризиків передбачає вивчення і надання в документальному вигляді результатів оцінки ефективності заходів реагування на ризики, що відносяться до ідентифікованим ризиків, вивчення основних причин їх виникнення, а також оцінку ефективності процесу управління ризиками.

Тренди в процесі виконання проекту підлягають перевірці з використанням даних про виконання. Для моніторингу виконання всього проекту можуть використовуватися аналіз освоєного обсягу та інші методи аналізу відхилень проекту і трендів. На підставі виходів цих аналізів можна прогнозувати потенційні відхилення проекту на момент його завершення за показниками вартості і розкладу. Відхилення від базового плану можуть вказувати на наслідки, викликані як погрозами, так і сприятливими можливостями.

Висновки

Відмовлятися від управління проектними ризиками це все одно, що в кінотеатрі не мати вогнегасників та плану евакуації на випадок пожежі.

Все, що ми робимо, керуючи проектом розробки, повинне бути спрямоване на боротьбу з ризиками не вкластися в термін, перевитрачати ресурси, розробити не той продукт, який потрібен.
Цілі управління ризиками проекту — зниження ймовірності виникнення та/або значущості впливу несприятливих для проекту подій.

Головні причини провалу програмних проектів:

• *Вимоги замовника відсутні / не повні / схильні до частих змін.*
• *Відсутність необхідних ресурсів і досвіду.*
• *Відсутність робочого взаємодії з замовником.*
• *Неповнота планування. «Забуті роботи».*
• *Помилки в оцінках трудоемкостей і термінів робіт.*

Додаткова література

1. Том ДеМарко, Тимоти Листер, «Вальсируя с Медведями. Управление рисками в проектах по разработке программного обеспечения», М., Компания p.m.Office, 2005.
2. «Microsoft Solutions Framework. Дисциплина управления рисками MSF», вер. 1.1, 2002
3. «PMBOK. Руководство к Своду знаний по управлению проектами», 3-е изд., PMI, 2004.
4. С.Макконнелл, «Сколько стоит программный проект», «Питер», 2007.
5. Ньюэл М.В., «Управление проектами для профессионалов. Руководство по подготовке к сдаче сертификационного экзамена PMP», КУДИЦ-Образ, 2006.
6. Barry W. Boehm. «A Spiral Model of Software Development and Enhancement, Computer, May 1988.
7. Barry Boehm, et al. «Software cost estimation with COCOMO II». Englewood Cliffs, NJ:Prentice-Hall, 2000
8. www.systemsguild.com/riskology © 2005 Том ДеМарко, Тимоти Листер.

Лекція 13-14. ТЕМА 6 ***Оцінка трудомісткості і термінів розробки ПО. Оцінка — імовірнісне твердження. Негативні наслідки «агресивної» розкладу. Прагматичний підхід. Метод PERT. Огляд методу функціональних точок. Основи методики COCOMO II***

Стів Макконнелл [1] пише, що ми від природи схильні вірити, що складні формули виду

$$PM = A \times SIZE^{E} \times \prod_{i=1}^{n} EM_i$$

$$A = 2{,}94$$

$$E = B + 0{,}01 \times \sum_{j=1}^{5} SF_j$$

$$B = 0{,}91$$

завжди забезпечують більш точні результати, ніж прості формули.

Трудомісткість = КоличествоФакторов х СредниеЗатратыНаФактор

Однак, далеко не завжди це так. Складні формули, як правило, дуже чутливі до точності великого числа параметрів (у наведеному прикладі формул COCOMO II міститься 21 параметр), які треба поставити, щоб отримати необхідні оцінки.

Перше, що необхідно розуміти при оцінці проекту, це те, що будь-яка оцінка це завжди імовірнісне твердження. Якщо ми просто скажемо, що трудомісткість даного пакету робіт становить М чол.*міс. (рис.34), то це буде поганою оцінкою тому, що єдине число нічого не скаже нам про ймовірність того, що на реалізацію цього пакету буде потрібно не більше, ніж М чол.*міс. Навряд чи ми можемо вважати себе «предсказамусами», які точно знають, що станеться в майбутньому і скільки потрібно витрат на реалізацію цього пакету робіт.

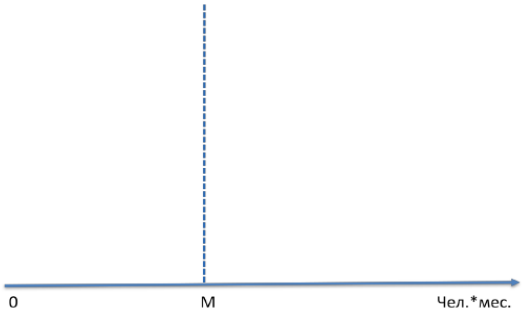


Рис. 34. Точкова оцінка трудомісткості пакету робіт нічого не скаже нам про ймовірність того, що на реалізацію цього пакету буде потрібно не більше, ніж М чол.*міс.

Для того, щоб зрозуміти, звідки береться невизначеність, розглянемо найпростіший приклад, спробуємо оцінити трудомісткість додавання поля введення телефонного номера клієнта до вже існуючої формі. Менеджер, який спостерігає роботу програмістів тільки з боку, скаже, що ця робота потребує не більше 15 хвилин робочого часу. Людина, навчений програмістських досвідом, скаже, що ця робота може зайняти від 2 до 200 годин, і щоб дати більш точну оцінку йому треба отримати відповіді на ряд питань:

• *Чи може вводитися декілька номерів?*
• *Повинна бути перевірка номерів на дійсність?*
• *Проста або складна перевірка?*
• *Якщо реалізуємо просту перевірку, то не захоче клієнт замінити її на більш складну?*
• *Повинна перевірка працювати для іноземних номерів?*
• *Можна скористатися готовим рішенням?*
• *Яким має бути якість реалізації? Ймовірність помилки після поставки?*
• *Скільки часу знадобиться на реалізацію та налагодження? (залежить від конкретного виконавця).*

Називаючи таку «розмиту» оцінку досвідчений програміст резервує всі ризики розробки, пов'язані з переліченими невизначеностями цієї вимоги, які він змушений брати на себе, не маючи в даний момент необхідної уточнюючої інформації.
Те, що наша оцінка повинна бути імовірнісним твердженням, означає, що для неї існує деякий розподіл ймовірності (Малюнок 35), яке може бути дуже широким (висока невизначеність) або досить вузьким (низька невизначеність).

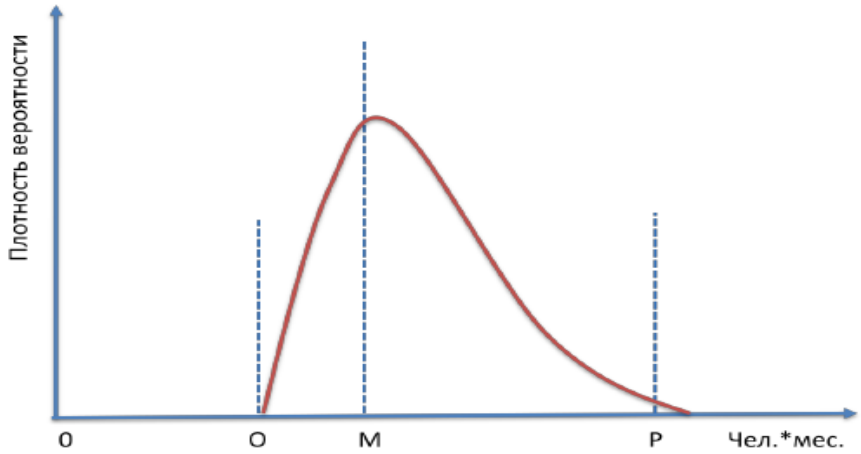


Рис. 35. Оцінка — завжди імовірнісна величина

Якщо M — найбільш ймовірне значення, то це не означає що це хороша оцінка, оскільки ймовірність того, що фактична трудомісткість перевищить цю оцінку, становить понад 50%.

Яка оцінка може вважатися гарною? Стів Макконнелл стверджує [1]: «Хорошою вважається оцінка, яка забезпечує достатньо ясне уявлення реального стану проекту і дозволяє керівнику проекту приймати хороші рішення щодо того, як управляти проектом для досягнення цілей».

Негативні наслідки «агресивної» розкладу

При побудові програм вже стало банальністю те, що розробники без достатньої підстави називають занадто оптимістичні терміни. Серед керівників навіть поширене неписане правило: множити на 2 оцінку трудомісткості, яку зробив програміст. Це песимістичний підхід. Реалісти множать на $\pi = 3.14$.

Дійсно, так іноді доводиться поступати, якщо це програміст, який тільки вчора отладил свою першу програму «Hello world!». Але якщо допомогти молодим спеціалістам навчиться аналізувати завдання, проектувати рішення, складати план роботи, ефективно його реалізувати і аналізувати отримані результати, то можна буде не згадувати, чому одно число π.

Ще один поширений джерело заниження строків — необгрунтовані очікування на застосування нових технологій і засобів розробки. Ці очікування, як правило, не виправдовуються. Згідно зі статистикою, наведеною Демарко, середня продуктивність в програмному виробництві зростає лише на 3-5% в рік.

Часто «агресивне» розклад проекту з'являється з-за того, що керівництво і/або замовник бояться переоцінити проект, вважаючи, що згідно із законом Паркінсона, роботи за проектом займуть весь відведений для нього час. Наслідком подібних побоювань є, як правило, директивне заниження строків реалізації проекту.

Нереалістичність оцінок — один з найсерйозніших демотивирующих факторів для учасників. Недооцінка призводить до помилок планування та неефективного взаємодії. Наприклад, було заплановане тестування, а реліз ще не готовий. Наслідок — простий тестувальників збільшення трудовитрат.

Якщо розклад надмірно агресивна, то з метою заощадити час, недостатньо уваги приділяється аналізу вимог і проектування. Виправлення помилок, допущених на цих етапах, призведе до суттєвих додаткових витрат.

Половина всіх помилок програмування виникають з-за стресу, викликаного надмірним тиском фактора термінів. Помилки виправляються наспіх,

обхідними шляхами. В результаті буде отримано великий проблемний код і постійно зростаючі витрати на виправлення помилок і внесення змін. Пізніше виявлення помилок призводить до того, що витрати на їх виправлення збільшуються в 50-100 разів.

Спостерігався проект, який замість спочатку занадто оптимістично запланованих шести місяців розтягнувся на три роки. Хоча, якщо б він був адекватно оцінений, то він міг би бути реалізований за один рік. Нереальні терміни, постійний тиск, понаднормові, аврали призводять до того, що витрати на проект ростуть експоненціально і необмежено.

Якщо учасники проектної команди адекватно мотивовані на виконання проектних робіт з найменшими витратами, то, на мій погляд, цього достатньо, щоб проект був реалізований в мінімально можливі терміни.

Прагматичний підхід. Метод PERT

Використання власного досвіду або досвіду колег, отриманого у схожих проектах, це найбільш прагматичний підхід, який дозволяє отримати досить реалістичні оцінки трудомісткості і терміну реалізації програмного проекту, швидко і без великих витрат.

Інженерний метод оцінки трудомісткості проекту PERT (Program / Project Evaluation and Review Technique) був розроблений у 1958 році в ході проекту зі створення балістичних ракет морського базування «Поларіс». Входом для даного методу оцінки служить список елементарних пакетів робіт. Для інженерного підходу немає необхідності точно знати закон розподілу нашої оцінки трудомісткості кожного такого елементарного пакета. Діапазон невизначеності досить охарактеризувати трьома оцінками:

Mi — найбільш ймовірна оцінка трудовитрат; Pi — песимістична оцінка трудовитрат; Oi — мінімально можливі витрати на реалізацію пакету робіт. Жоден ризик не реалізувався. Швидше точно не зробимо. Ймовірність такого, що ми вкладемося у ці витрати, дорівнює 0. Всі ризики реалізувалися. Оцінку середньої трудомісткості по кожному елементарному пакету можна визначити за формулою:

$$\mathbf{Ei = (Pi + 4Mi + Oi)/6.}$$

Для розрахунку середньоквадратичного відхилення використовується формула:

$$\mathbf{СКОi = (Pi - Oi)/6.}$$

Якщо наші оцінки трудомісткості елементарних пакетів робіт статистично незалежні, а не зіпсовані, наприклад, необгрунтованим оптимізмом то, згідно

з центральною граничною теоремою теорії ймовірностей сумарна трудомісткість проекту може бути розрахована за формулою:

$$\mathbf{E = \sum Ei}$$

А середньоквадратичне відхилення для оцінки сумарної трудомісткості буде становити:

$$СКО = \sqrt{\sum СКО_i^2}$$

Тоді для оцінки сумарної трудомісткості проекту, яку ми не перевищимо з імовірністю 95%, можна застосувати формулу:

$$\mathbf{E_{95\%} = E + 2 * СКО.}$$

Це означає, що ймовірність того, що проект перевищить цю оцінку трудомісткості складає всього 5%. А це вже цілком прийнятна оцінка, під якою може розписатися професійний менеджер.

Список елементарних пакетів робіт, який використовується при оцінці трудомісткості, як правило, береться з нижнього рівня ICP проекту. Але може бути використаний і накопичений досвід аналогічних розробок. Проілюструю це на прикладі реального проекту. В Асоціації CBOSS завданням проекту, який нам з колегами пощастило реалізовувати, була розробка на основі стандартів J2EE загальносистемного ПЗ для перекладу робочих місць CBOSS на нову триланкову архітектуру. Був розроблений набір стандартних компонентів і сервісів, з яких як з конструктора можна ефективно та якісно збирати прикладні підсистеми. Високорівнева архітектура реалізовувала стандартний шаблон MVC (Рисунок 36), кожен з компонентів якого мав «точки розширення» для прикладної розробки, які на малюнку виділено червоним світлом.

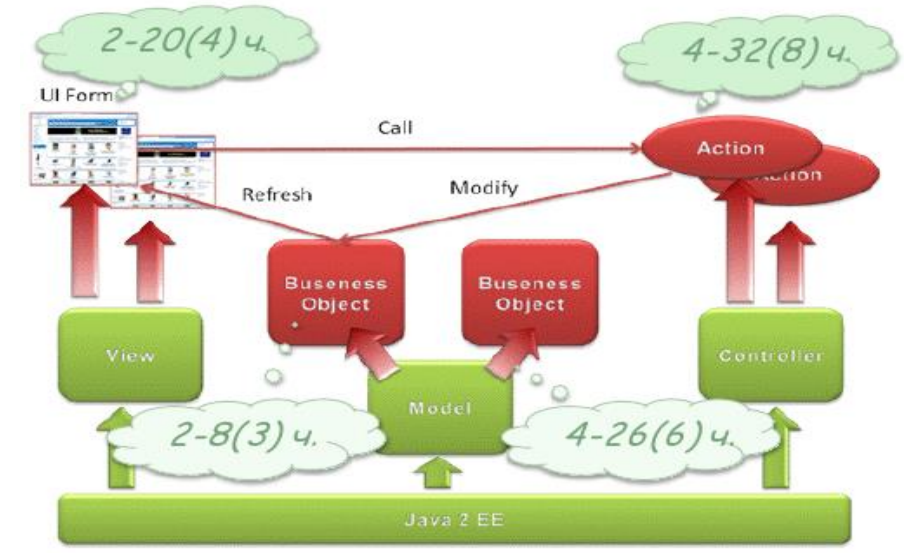


Рис.36. Високорівнева архітектура J2EE фреймворку для розробки додатків.

Такими точками розширення були:
• Настроюваний екран (UI Form), який збирався з готових візуальних компонентів.
• Обробники (Action), які обробляли на сервері додатків події від активних візуальних компонентів, що входять до складу екрану.
• Об'єкти (Business Obj), які моделювали прикладну область, і до яких зверталися обробники подій.
Так от, хоча все розробляються робочі місця розрізнялися по функціональності і складності, накопичена статистика фактичних трудовитрат на розробку прикладних систем дозволяла нам оцінювати проекти розробки нового додатка досить швидко і з високою достовірністю.

Згідно з цією статистикою, розробка і налагодження вимагала у програміста:

*• для одного екрану — **від 2 до 20 годин** (найбільш ймовірно — 4 години);*
*• для одного обробника подій — **від 4 до 32 годин** (найбільш ймовірно — 8 годин);*
*• для нового бізнес-об'єкта — **від 2 до 8 годин** (найбільш ймовірно — 3 години);*
*• для додавання нового бізнес-методу — **від 2 до 26 годин** (найбільш ймовірно — 6 годин).*
Весь проект прикладної розробки вимірювався в «папуг»:
• $К_{UI}$ — кількість користувальницьких екранів.

• KAct — кількість обробників подій.
• K_{BO} — кількість нових бізнес-об'єктів.
• K_{BM} — кількість нових або модифікуються бізнес-методів.

Якщо нове розробляється додаток містить 20 користувацьких екранів, 60 обробників подій, 16 нових бізнес-об'єкта і 40 нових бізнес-методів, які необхідно додати, як в нові, так і вже існуючих бізнес-об'єкти, тоді, згідно з нашою статистикою,

E_{UI} = (2 + 4*4 + 20) / 6 = 6.7 чол.*год., $СКО_{UI}$ = (20 - 2) / 6 = 3 чол. год
EAct = (4 + 4*8 + 32) / 6 = 11.3 чол.*год., СКОAct = (32 - 4) / 6 = 4.7 чол. год
E_{BO} = (2 +4*3 + 8) / 6 = 3.7 чол.*год., $СКО_{BO}$ = (8 - 2) / 6 = 1 чол. год.
E_{BM} = (2 + 4*6 + 26) / 6 = 8.7 чол.*год., $СКО_{BM}$ = (26 - 2) / 6 = 4 чол. год.

Для середньої трудомісткості робіт по кодуванню в проекті може бути отримана наступна оцінка:

$$E = 20 * 6.7 + 60 * 11.3 + 16 * 3.7 + 40 * 8.7 \approx 1220 \text{ чел.*час.}$$

$$СКО = \sqrt{20*3^2 + 60*4.7^2 + 16*1^2 + 40*4^2} =$$

$$= \sqrt{180+1325+16+640} \approx 46 \text{ чел.*час.}$$

Тоді для оцінки сумарної трудомісткості проекту, яку ми не перевищимо з імовірністю 95%, отримаємо

$E_{95\%}$ = 1220 + 2 *46 ≈ 1300 чол. годину.

Хоча відносна похибка в оцінці трудомісткості кожної такої елементарної роботи становила десятки відсотків, для нашого проекту, в якому було таких «папуг» було 136, відносна похибка оцінки сумарної трудомісткості, зробленої за методом PERT, склала, приблизно, лише 4%.

Навіть якщо у нас дуже розмиті оцінки трудомісткості кожної з елементарних робіт, але вони незалежні, то помилки ми робимо як в меншу, так і більшу боку. Тому при фактичній реалізації проекту ці помилки будуть компенсуватися, що дозволяє нам оцінити загальні трудовитрати за проектом істотно точніше, ніж робота на кожну елементарну роботу. Але це твердження буде справедливим тільки в тому випадку, якщо наша ІСР містить всі необхідні роботи, які повинні бути виконані для отримання всіх продуктів проекту.

Отриману оцінку трудомісткості кодування необхідно помножити на чотири, оскільки пам'ятаємо (див. Ініціація проекту), що кодування становить

лише 25% загальних трудовитрат проекту. Тому сумарна трудомісткість нашого проекту складе приблизно 5200 чол*годину.

Як ми вже говорили раніше, якщо співробітник на 100% призначений на проект, це, як правило, не означає, що він всі 40 годин в тиждень буде витрачати на проектні роботи. Витрачати він буде 60-80% свого робочого часу. Тому, в місяць працівник буде працювати за проектом, приблизно, 165* **0.8 = 132 чол.*год/міс.** Отже, трудомісткість проекту в людино-місяцях складе, приблизно **5200 / 132 ≈ 40**.

Тоді згідно з формулою Б. Боема (Рис. 15) оптимальна тривалість проекту становитиме:

T = 2.5 * (40)1/3 = 8.5 місяців,

а середня чисельність команди — 5 осіб.

Пам'ятаємо, що споживання ресурсів в проекті нерівномірно (Рисунок 13), тому починати проект повинні 1-3 людини, а на стадії реалізації початкова чисельність команди може бути збільшена в декілька разів.

Якщо ж власний досвід аналогічних проектів відсутня, а колеги-експерти недоступні, то нам не залишається нічого іншого, як використовувати формальні методики, засновані на узагальненому галузевому досвіді. Серед них найбільше поширення одержали два підходи:

• FPA IFPUG — метод функціональних точок,

• метод COCOMO II, Constructive Cost Model.

Огляд методу функціональних точок

Аналіз функціональних точок — стандартний метод вимірювання розміру програмного продукту з точки зору користувачів системи. Метод розроблений Аланом Альбрехтом (Alan Albrecht) в середині 70-х. Метод був вперше опублікований в 1979 році. В 1986 році була сформована Міжнародна Асоціація Користувачів Функціональних Точок (International Function Point User Group — IFPUG), яка опублікувала кілька ревізій методу [2].

Метод призначений для оцінки на основі логічної моделі обсягу програмного продукту кількістю функціоналу, затребуваного замовником і поставляється розробником. Безсумнівним достоїнством методу є те, що вимірювання не залежать від технологічної платформи, на якій буде розроблятися продукт, і він забезпечує однаковий підхід до оцінки всіх проектів в компанії.

При аналізі методом функціональних точок треба виконати таку послідовність кроків (Рисунок 37):

1. Визначення типу оцінки.

2. Визначення області оцінки і меж продукту.

3. Підрахунок функціональних точок, пов'язаних з даними.
4. Підрахунок функціональних точок, пов'язаних з транзакціями.
5. Визначення сумарної кількості не вирівняних функціональних точок (UFP).
6. Визначення значення фактору вирівнювання (FAV).
7. Розрахунок кількості вирівняних функціональних точок (AFP).

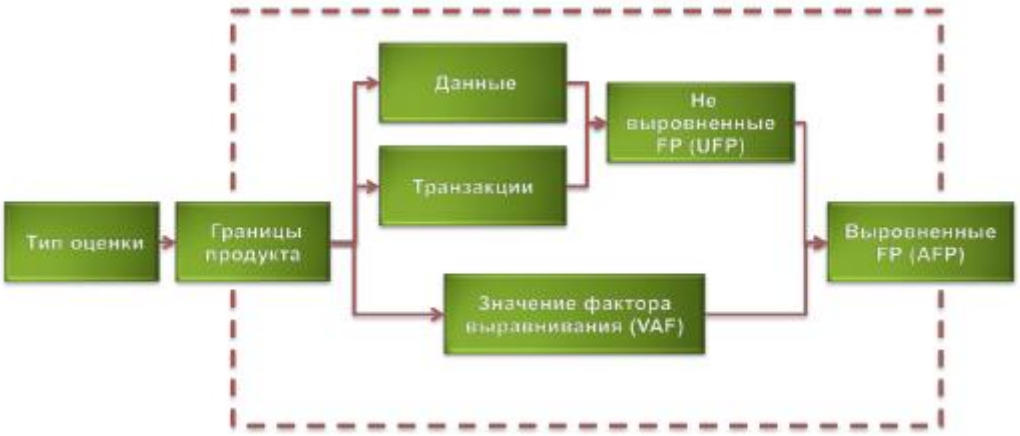


Рис. 37. Процедура аналізу за методом функціональних точок

Визначення типу оцінки

Перше, що необхідно зробити, це визначити тип виконуваної оцінки. Метод передбачає оцінки трьох типів:

1. Проект розробки. Оцінюється кількість функціональності подається користувачам у першому релізі продукту.
2. Проект розвитку. Оцінюється в функціональних точках проект доопрацювання: додавання, зміна та видалення функціоналу.
3. Продукт. Оцінюється обсяг наявного і встановленого продукту.

Визначення області оцінки і меж продукту

Другий крок — це визначення області оцінки і меж продукту. В залежності від типу область оцінки може включати:
• *Всі розроблювані функції (для розробки проекту)*
• *Все, що додаються, змінювані і видаляються функції (для проектів підтримки)*
• *Тільки функції, що реально використовуються, або всі функції (при оцінці продукту та/або продуктів).*

Третій крок. Кордону продукту (Рисунок 38) визначають:

• *Що є «зовнішніми» по відношенню до оцінюваного продукту.*
• *Де розташовується «межа системи», через яку проходять транзакції передаються або прийняті продуктом, з точки зору користувача.*
• *Дані, які підтримуються програмою, а які — зовнішні.*

Рис. 38. Кордону продукту в методі функціональних точок

До логічним даними системи відносяться:

• *Внутрішні логічні файли (ILFs) — виділяються користувачем логічно пов'язані групи даних або блоки керуючої інформації, які підтримуються всередині продукту.*
• *Зовнішні інтерфейсні файли (EIFs) — виділяються користувачем логічно пов'язані групи даних або блоки керуючої інформації, на які посилається продукт, але які підтримуються поза продукту.*

Прикладом логічних даних (інформаційних об'єктів) можуть служити: клієнт, рахунок, тарифний план, послуга.

Підрахунок функціональних точок, пов'язаних з даними

Третій крок — підрахунок функціональних точок, пов'язаних з даними. Спочатку визначається складність даних за такими показниками:

• *DET (data element type) — неповторне унікальне поле даних, наприклад, Ім'я Клієнта — 1 DET; Адресу Клієнта (індекс, країна, область, район, місто, вулиця, будинок, корпус, квартира) — 9 DET's*
• *RET (record element type) — логічна група даних, наприклад, адресу, паспорт, телефонний номер.*

Оцінка кількості не вирівняних функціональних точок, залежить від складності даних, яка визначається на підставі матриці складності (Таблиця 7).

Таблиця 7. Матриця складності даних

	1-19 DET	**20-50 DET**	**50+ DET**
1 RET	Low	Low	Average
2-5 RET	Low	Average	High
6+ RET	Average	High	High

Оцінка даних на не вирівняних функціональних точках (UFP) підраховується по-різному для внутрішніх логічних файлів (ILFs) і для зовнішніх інтерфейсних файлів (EIFs) (Таблиця 8) в залежності від їх складності.

Таблиця 8. Оцінка даних на не вирівняних функціональних точках (UFP) для внутрішніх логічних файлів (ILFs) і зовнішніх інтерфейсних файлів (EIFs)

Складність даних	**Кількість UFP (ILF)**	**Кількість UFP (EIF)**
Low	7	5
Average	10	7
High	15	10

Для ілюстрації розглянемо приклад оцінки не вирівняних функціональних точках об'єкта даних «Клієнт» (Рис.39).

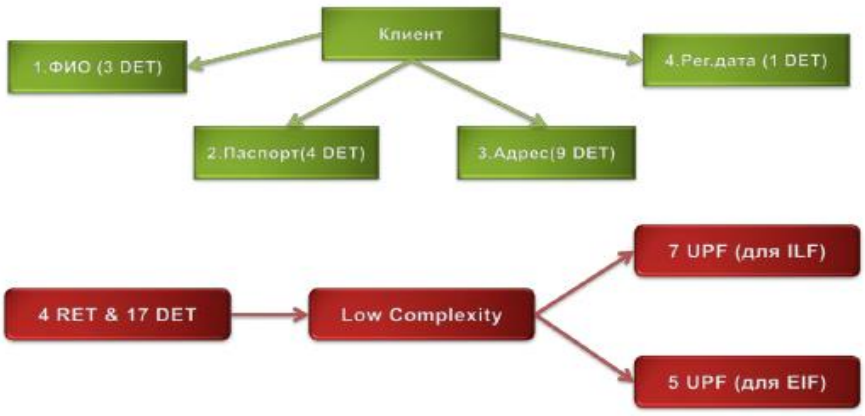


Рис. 39. Приклад оцінки не вирівняних функціональних точках об'єкта даних «Клієнт».

Об'єкт «Клієнт» містить чотири логічні групи даних, які в сукупності складаються з 15 неповторяемых унікальне полів даних. Згідно матриці (Таблиця 7), нам слід оцінити складність цього об'єкта даних, як «Low». Тепер, якщо оцінюваний об'єкт відноситься до внутрішнім логічним файлів,

згідно Таблиця 8 його складність буде 7 не вирівняних функціональних точок (UPF). Якщо ж об'єкт є зовнішнім інтерфейсним файлом, то його складність складе 5 UPF.

Підрахунок функціональних точок, пов'язаних з транзакціями

Підрахунок функціональних точок, пов'язаних з транзакціями — це четвертий крок аналізу за методом функціональних точок.
Транзакція — це елементарний неподільний замкнутий процес, що представляє значення для користувача і переводить продукт з одного консистентного стану в інший.

У методі розрізняються наступні типи транзакцій (Таблиця 9):

• *EI (external inputs) — зовнішні вхідні транзакції, елементарна операція з обробки даних або керуючої інформації, що надходять у систему ззовні.*
• *EO (external outputs) — зовнішні вихідні транзакції, елементарна операція по генерації даних або керуючої інформації, які виходять за межі системи. Припускає певну логіку обробки або обчислень інформації з одного або більше ILF.*
• *EQ (external inquiries) — зовнішні запити, елементарна операція, яка у відповідь на зовнішній запит отримує дані або керуючу інформацію з ILF або EIF.*

Таблиця 9. Основні відмінності між типами трансакцій. Легенда: 0 — основна; Д — додаткова; NA — не застосовна.

Функция	**Тип транзакции**		
	EI	**EO**	**EQ**
Изменяет поведение системы	О	Д	NA
Поддержка одного или более ILF	О	Д	NA
Представление информации пользователю	Д	О	О

Оцінка складності транзакції ґрунтується на таких її характеристиках:

• *FTR (file type referenced) — дозволяє підрахувати кількість різних файлів (інформаційних об'єктів) типу ILF та/або EIF модифікуються або прочитуваних в транзакції.*
• *DET (data element type) — неповторне унікальне поле даних. Приклади. EI: поле введення, кнопка. EO: поле даних звіту, повідомлення про помилку. EQ: поле введення для пошуку, поле виводу результату пошуку.*

Для оцінки складності транзакцій служать матриці, які представлені в Таблиця 10 Таблиця 11.

Таблиця 10. Матриця складності зовнішніх вхідних транзакцій (EI)

EI	**1-4 DET**	**5-15 DET**	**16+ DET**
0-1 FTR	Low	Low	Average
2 FTR	Low	Average	High
3+ FTR	Average	High	High

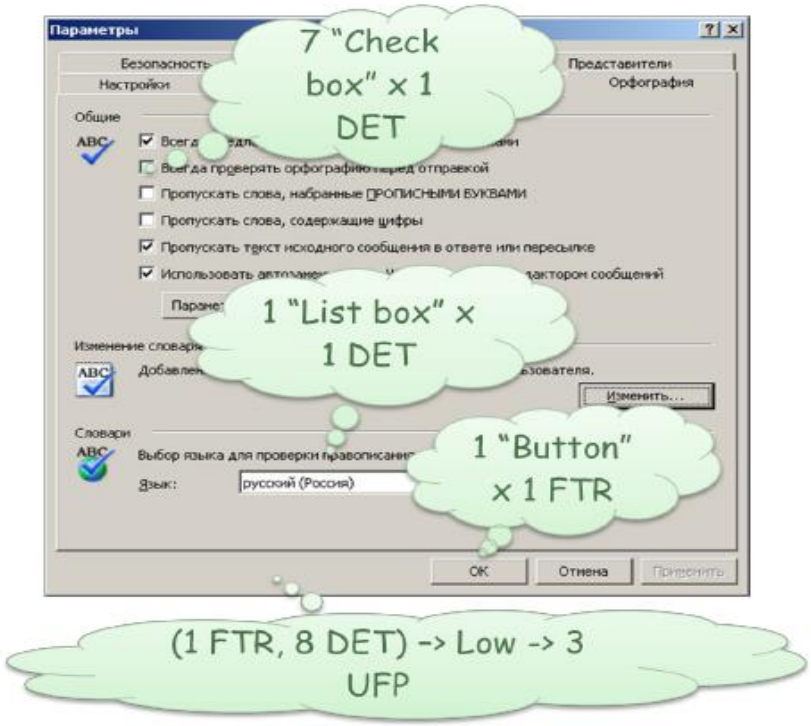
Таблиця 11. Матриця складності зовнішніх вихідних транзакцій і зовнішніх запитів (EO & EQ)

EO & EQ	**1-5 DET**	**6-19 DET**	**20+ DET**
0-1 FTR	Low	Low	Average
2-3 FTR	Low	Average	High
4+ FTR	Average	High	High

Таблиця 12. Складність транзакцій на не вирівняних функціональних точках (UFP)

Сложность транзакций	**Количество UFP (EI & EQ)**	**Количество UFP (EO)**
Low	3	4
Average	4	5
High	6	7

Як приклад, розглянемо оцінку керуючої транзакції (EI) для діалогового вікна, що задає параметри перевірки орфографії в MS Office Outlook (Рис. 40).



Малюнок 40. Діалогове вікно, що керує перевіркою орфографії в MS Office Outlook

Кожен "Check box" оцінюється, як 1 DET. Випадаючий список — 1 DET. Кожна кнопка має розглядатися як окрема транзакція. Наприклад, якщо оцінювати керуючу транзакцію по кнопці «OK», то, для даної транзакції ми маємо 1 FTR і 8 DET. Тому, згідно матриці (Таблиця 10), ми можемо оцінити складність транзакції, як Low. І, нарешті, у відповідність з матрицею (Таблиця 12), дана транзакція повинна бути оцінена в 3 не вирівняних функціональних точок (UFP).

Визначення сумарної кількості не вирівняних функціональних точок (UFP)

Загальний обсяг продукту не вирівняних функціональних точках (UFP) визначається шляхом підсумовування по всіх інформаційних об'єктів (ILF, EIF) і елементарних операцій (транзакцій EI, EO, EQ).

$$UFP = \sum_{ILF} UFPi + \sum_{EIF} UFPi + \sum_{EI} UFPi + \sum_{EO} UFPi + \sum_{EQ} UFPi$$

Визначення значення фактору вирівнювання (FAV)

Крім функціональних вимог на продукт, накладаються загальносистемні вимоги, які обмежують розробників у виборі рішення і збільшують складність розробки. Для врахування цієї складності застосовується фактор

вирівнювання (VAF). Значення фактора VAF залежить від 14 параметрів, які визначають системні характеристики продукту:

1. *Обмін даними* (0 — продукт являє собою автономне додаток; 5 — продукт обмінюється даними з більш, ніж одному телекомунікаційного протоколу).
2. *Розподілена обробка даних* (0 — продукт не переміщує дані; 5 — розподілена обробка даних виконується кількома компонентами системи).
3. *Продуктивність* (0 — користувальницькі вимоги по продуктивності не встановлені; 5 — час відгуку сильно обмежена критично для всіх бізнес-операцій, для задоволення вимог необхідні спеціальні проектні рішення та інструменти аналізу.
4. *Обмеження щодо апаратних ресурсів* (0 — немає обмежень; 5 — продукт цілком повинен функціонувати на певному процесорі і не може бути розподілений).
5. *Транзакційна навантаження* (0 — транзакцій не багато, без піків; 5 — кількість транзакцій велике і нерівномірно, потрібні спеціальні рішення та інструменти).
6. *Інтенсивність взаємодії з користувачем* (0 — всі транзакції обробляються в пакетному режимі; 5 — більше 30% транзакцій — інтерактивні).
7. *Ергономіка* (ефективність роботи кінцевих користувачів) (0 — немає спеціальних вимог; 5 — вимоги щодо ефективності дуже жорсткі).
8. *Інтенсивність зміни даних* (ILF) користувачами (0 — не потрібні; 5 — інтенсивні зміни, жорсткі вимоги по відновленню).
9. *Складність обробки* (0 — обробка мінімальна; 5 — вимоги безпеки, логічна і математична складність, багатопоточність).
10. *Повторне використання* (0 — не вимагається; 5 — продукт розробляється як стандартний багаторазовий компонент).
11. *Зручність інсталяції* (0 — немає вимог; 5 — установка і оновлення відбувається автоматично).
12. *Зручність адміністрування* (0 — не вимагається; 5 — система автоматично самовосстанавливается).
13. *Портируемость* (0 — продукт має тільки 1 інсталяцію на єдиному процесорі; 5 — система є розподіленою і передбачає встановлення на різні «залізо» і ОС).
14. *Гнучкість* (0 — не вимагається; 5 — гнучка система запитів і побудова довільних звітів, модель даних змінюється користувачем в інтерактивному режимі).

14 системних параметрів (degree of influence, DI) оцінюються за шкалою від 0 до 5. Розрахунок сумарного ефекту 14 системних характеристик (total degree of influence, TDI) здійснюється простим підсумовуванням:

$$\mathbf{TDI = \sum DI}$$

Розрахунок значення фактору вирівнювання проводиться по формулі

$$\mathbf{VAF = (TDI *0.01) + 0.65}$$

Наприклад, якщо, кожен з 14 системних параметрів отримав оцінку 3, то їх сумарний ефект складе **TDI = 3 * 14 = 42**. В цьому випадку значення фактору вирівнювання буде: **VAF = (42 * 0.01) + 0.65 = 1.07**

Розрахунок кількості вьровненных функціональних точок (AFP)

Подальша оцінка в вирівняних функціональних точках залежить від типу оцінки. Початкова оцінка кількості вирівняних функціональних точок для програмного додатка визначається по наступній формулі:

$$\mathbf{AFP = UFP * VAF.}$$

Вона враховує тільки нову функциональностсть, яка реалізується у продукті. Проект розробки продукту оцінюється в DFP (development functional point) за формулою:

$$\mathbf{DFP = (UFP + CFP) * VAF,}$$

де CFP (conversion functional point) — функціональні точки, підраховані для додаткової функціональності, яка знадобиться при установці продукту, наприклад, міграції даних.

Проект доопрацювання і вдосконалення продукту оцінюється в EFP (enhancement functional point) за формулою:

$$\mathbf{EFP = (ADD + CHGA + CFP) * VAFA + (DEL* VAFB),}$$

де

• ADD — функціональні точки для доданої функціональності;

• CHGA — функціональні точки для змінених функцій, розраховані після модифікації;

• VAFA — величина фактору вирівнювання розрахованого після завершення проекту;

• DEL — обсяг вилученої функціональності;

• VAFB — величина фактору вирівнювання розрахованого до початку проекту.

Сумарний вплив процедури вирівнювання лежить в межах ±35% щодо обсягу розрахованого UFP.

Метод аналізу функціональних точок нічого не говорить про трудомісткості розробки оціненого продукту. Питання вирішується просто,

якщо компанія розробник має власну статистику трудовитрат на реалізацію функціональних точок. Якщо такої статистики немає, то для оцінки трудомісткості і термінів проекту можна використовувати метод COCOMO II.

Основи методики COCOMO II

Методика COCOMO дозволяє оцінити трудомісткість і час розробки програмного продукту. Вперше була опублікована Барі Боэмом [3] у 1981 році у вигляді результат аналізу 63 проектів компанії «TRW Aerospace». У 1997 методика була вдосконалена і отримала назву COCOMO II. Калібрування параметрів проводилася за 161 проекту розробки. В моделі використовується формула регресії з параметрами, обумовленими на основі галузевих даних і характеристик конкретного проекту.
Розрізняються дві стадії оцінки проекту: попередня оцінка на початковій фазі і детальна оцінка після опрацювання архітектури.
Формула оцінки трудомісткості проекту чол.*міс. має вигляд:

$$PM = A \times SIZE^{E} \times \prod_{i=1}^{n} EM_i$$

$$A = 2{,}94$$

$$E = B + 0{,}01 \times \sum_{j=1}^{5} SF_j$$

$$B = 0{,}91$$

де
• SIZE — розмір продукту у KSLOC
• EMi — множники трудомісткості
• SFj — фактори масштабу
• n=7 — для попередньої оцінки
• n=17 — для детальної оцінки

Головною особливістю методики є те, що для того, щоб оцінити трудомісткість, необхідно знати розмір програмного продукту в тисячах рядках вихідного коду (KSLOC, Kilo Source Lines Of Code). Розмір програмного продукту може бути, наприклад, оцінений експертами з застосуванням методу PERT.
Якщо ми провели аналіз продукту методом функціональних точок, то його розмір не може бути розрахований з використанням власних статистичних даних або з використанням статистики по галузі [5] (Таблиця 13).

Таблиця 13. Оцінка кількості рядків, необхідних на реалізацію одного не вирівняною функціональної точки для деяких поширених мов програмування.

Мова програмування Оцінка кількості рядків

	Найбільш ймовірна	Оптимістична	Песимістична
Assembler	172	86	320
C	148	9	704
C++	60	29	178
C#	59	51	66
J2EE	61	50	100
JavaScript	56	44	65
PL/SQL	46	14	110
Visual Basic	50	14	276

Фактори масштабу

У методиці використовуються п'ять факторів масштабу SF;, які визначаються наступними характеристиками проекту:

1. PREC — прецедентність, наявність досвід аналогічних розробок (Very Low — досвід в продукті і платформі відсутня; Extra High — продукт і платформа повністю знайомі)

2. FLEX — гнучкість процесу розробки (Very Low — процес строго детермінована; Extra High — визначені лише загальні цілі).

3. RESL — архітектура і дозвіл ризиків (Very Low — ризики невідомі/не проаналізовані; Extra High — ризики можна на 100%)

4. TEAM — спрацьованість команди (Very Low — формальні взаємодії; Extra High — повна довіра, взаємозамінність і взаємодопомога).

5. PMAT — зрілість процесів (Very Low — CMM Level 1; Extra High — CMM Level 5)

Значення фактора масштаб, в залежності від оцінки його рівня, наведені в табл. 14

Таблиця 14. Значення фактора масштабу, в залежності від оцінки його рівня
Фактор масштабу Оцінка рівня фактора

	Very Low	**Low**	**Nominal**	**High**	**Very High**	**Extra High**
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

Множники трудомісткості

У нашу задачу не входить детальний опис методу COCOMO II, тому ми розглянемо тільки випадок попередньої оцінки трудомісткості програмного проекту. Для цієї оцінки необхідно оцінити для проекту рівень семи множників трудомісткості M,-:

1. PERS — кваліфікація персоналу (Extra Low — аналітики та програмісти мають нижчу кваліфікацію, плинність більше 45%; Extra High — аналітики та програмісти мають вищу кваліфікацію, плинність менше 4%)

2. RCPX — складність та надійність продукту (Extra Low — продукт простий, спеціальних вимог щодо надійності немає, БД маленька, документація не вимагається; Extra High — продукт дуже складний, вимоги по надійності жорсткі, БД надвелика, документація потрібно в повному обсязі)

3. RUSE — розробка для повторного використання (Low — не вимагається; Extra High — потрібно переиспользование в інших продуктах)

4. PDIF — складність платформи розробки (Extra Low — спеціальні обмеження по пам'яті і швидкодії відсутні, платформа стабільна; Extra High — жорсткі обмеження по пам'яті і швидкодії, платформа нестабільна)

5. PREX — досвід персоналу (Extra Low — новий додаток, інструменти і платформа; Extra High — додаток, інструменти і платформа добре відомі)

6. FCIL — обладнання (Extra Low — найпростіші інструменти, комунікації утруднені; Extra High — інтегровані засоби підтримки життєвого циклу, інтерактивні мультимедіа комунікації)

7. SCED — стиснення розкладу (Very Low — 75% від номінальної тривалості; Very High — 160% від номінальної тривалості)

Вплив множників трудомісткості в залежності від їх рівня визначається їх числовими значеннями, які представлені у матриці, наведеної нижче (Таблиця 15).

Таблиця 15. Значення множників трудомісткості, в залежності від оцінки їх рівня
Оцінка рівня множника трудомісткості

	Extra Low	**Very Low**	**Low**	**Nominal**	**High**	**Very High**	**Extra High**
PERS	2.12	1.62	1.26	1.00	0.83	0.63	0.5
RCPX	0.49	0.60	0.83	1.00	1.33	1.91	2.72
RUSE	n/a	n/a	0.95	1.00	1.07	1.15	1.24
PDIF	n/a	n/a	0.87	1.00	1.29	1.81	2.61
PREX	1.59	1.33	1.22	1.00	0.87	0.74	0.62
FCIL	1.43	1.30	1.10	1.0	0.87	0.73	0.62
SCED	n/a	1.43	1.14	1.00	1.00	1.00	n/a

З цієї таблиці, зокрема, випливає, що якщо в нашому проекті низька кваліфікація аналітиків, то його трудомісткість зросте приблизно в 4 рази порівняно з проектом, в якому беруть участь аналітики екстра-класу. І це не вигадки теоретиків, а галузева статистика!

Оцінка багатокомпонентного продукту

Як ми зазначали раніше (див. Планування проекту), для того щоб адекватно спланувати проект і оцінити його трудомісткість, необхідно виконати попереднє проектування програмного продукту. В результаті декомпозиції ми отримуємо деяку кількість компонентів (N), які складають програмний продукт.

Слід розуміти, що сумарна трудомісткість проекту не дорівнює простій сумі трудоемкостей розробки кожного з компонентів:

$$PM \neq \sum_{k=1}^{N} PM_k$$

Проста сума не враховує взаємозв'язку компонентів і трудовитрати на їх інтеграцію.

Методика COCOMO II визначає таку послідовність обчислення трудомісткості проекту при багатокомпонентної розробці.

1. Сумарний розмір продукту розраховується, як сума розмірів його компонентів:

$$SIZE^A = \sum_{k=1}^{N} SIZE_k$$

2. Базова трудомісткість проекту розраховується за формулою:

$$PM^B = A \times (SIZE^A)^E \times SCED$$

3. Потім розраховується базова трудомісткість кожного компонента:

$$PM_k^B = PM^B \times \frac{SIZE_k}{SIZE^A}$$

4. На наступному кроці розраховується оцінка трудомісткості компонентів з урахуванням всіх множників трудомісткості, крім множника SCED.

$$PM'_k = PM_k^B \times \prod_{i=1}^{6} EM_i$$

5. І, нарешті, підсумкова трудомісткість проекту визначаться за формулою:

$$PM = \sum_{k=1}^{N} PM'_k$$

Оцінка тривалості проекту

Тривалість проекту в методиці COCOMO II розраховується за формулою:

$$TDEV = C \times (PM_{NS})^{D+0,2\times 0,01\times \sum_{j=1}^{5} SF_j} \times \frac{SCED}{100},$$

де

• З = 3,67; D = 0,28;
• PMNS — трудомісткість проекту без урахування множника SCED, що визначає стиснення розкладу.

Висновки

Оцінка трудомісткості повинна бути імовірнісним твердженням. Це означає, що для неї існує деякий розподіл ймовірності, яке може бути дуже широким, якщо невизначеність висока, або досить вузьким, якщо невизначеність низька.

Використання власного досвіду або досвіду колег, отриманого у схожих проектах, це найбільш прагматичний підхід, який дозволяє отримати досить реалістичні оцінки трудомісткості і терміну реалізації програмного проекту, швидко і без великих витрат.

Якщо власний досвід аналогічних проектів відсутня, а колеги-експерти недоступні, то необхідно використовувати формальні методики, засновані на узагальненому галузевому досвіді. Серед них найбільше поширення одержали два підходи:

• *FPA IFPUG — метод функціональних точок,*
• *метод COCOMO II, Constructive Cost Model.*

Не реалістичність оцінок один з найсерйозніших демотивирующих факторів для учасників проектної команди. Недооцінка призводить до помилок планування та неефективного взаємодії. Агресивні терміни, постійний тиск, понаднормові, аврали служать причиною того, що витрати на проект ростуть експоненціально і необмежено.

Додаткова література

1. С. Макконнелл, «Сколько стоит программный проект», «Питер», 2007.
2. Function Point Counting Practices Manual, Release 4.2, IFPUG, 2004.
3. Barry Boehm. «Software engineering economics». Englewood Cliffs, NJ:Prentice-Hall, 1981
4. Barry Boehm, et al. «Software cost estimation with COCOMO II». Englewood Cliffs, NJ:Prentice-Hall, 2000.
5. «Function Point Programming Languages Table», Quantitative Software Management, Inc., 2005.

Лекція 15-16. ТЕМА 7 *Формування команди. Лідерство і управління*

Своє уявлення про питання, пов'язані з формуванням і керівництвом командами розробників ПЗ, я докладно виклав у книзі [1]. У цій лекції зупинимося лише на ключових моментах цієї діяльності.

В роботі керівника проекту є дві сторони: управління та лідерство, які однаково важливі і не можуть існувати у відриві один від одного. Не можна бути лідером матеріальних ресурсів, грошових потоків, планів, графіків і ризиків. Ними необхідно управляти. Тому що у речей немає права та свободи вибору, властивих тільки людині.

Інтелектуальними людьми неможливо управляти. Творчі команди можна тільки направляти і вести. «Високопродуктивне управління в відсутність ефективного лідерства подібно впорядкування розміщення стільців на палубі потопаючого «Титаніка». Ніякої успіх в управлінні не компенсує провалу лідерства» [2].

Ефективні команди не утворюються самі по собі, вони кристалізуються навколо визнаного лідера. Як не буває лідерів без послідовників, так і не буває команд без лідерів. Тому перший крок керівника при створенні ефективної команди — це стати лідером, навколо якого зможе об'єднатися робочий колектив. Лідера можна призначити.

Лідерство, це в першу чергу, це вміння управляти своїм власним життям і тільки потім іншими людьми. Надвисоке значення коефіцієнта інтелекту IQ, на жаль, у цьому не допоможе. Особиста ефективність людини на 80% визначається його коефіцієнтом емоційного інтелекту EQ (Emotional Intelligence) [3] — здатність розуміти і ефективно взаємодіяти з іншими людьми. Хороша новина. На відміну від IQ, який формується в ранній молодості і потім практично не змінюється, EQ можна підвищувати протягом усього життя. Якщо, звичайно, докладати до цього зусиль.

Лідер повинен отримати визнання команди. Для цього необхідно:

• Визнання командою професійної компетентності та переваги лідера.

• Повна довіра команди до дій і рішень лідера, визнання його людських якостей, переконаність в його чесності, порядності, віра в його щирість і сумлінність.

Якщо керівник не зміг стати лідером, він буде змушений застосовувати у своїй практиці управлінські антипаттерны [1]. Для такого керівника характерні надмірна настороженість, скритність, нездатність делегувати повноваження. Він виходить з передумов індустріальної епохи Генрі Форда: «Працівники ледачі, тому їм необхідні зовнішні стимули для роботи. У людей немає честолюбства, і вони намагаються позбутися від

відповідальності. Щоб змусити людей трудитися, необхідно використовувати примус, контроль і загрозу покарання». Манфред Кетс де Вріс [3] називає дане відхилення «параноїдальним управлінням».

Марно намагатися мотивувати учасників команди на успіх проекту, не виключивши зі свого керівного арсеналу практики демотивації — антипаттерны, застосування яких в управлінні творчими колективами не приносить нічого, крім шкоди. Замість мотивування співробітників на успіх, застосування антипаттернов мотивує їх на уникнення ризику і негативних для себе наслідків, пригнічує свободу, самостійність, творчість та ініціативу. Це призводить до деструктивного підпорядкування, коли всі працюють строго по інструкції і тільки у відповідність з вказівками керівництва, і повної відсутності особистої відповідальності виконавців, «А які до мене претензії? Як сказали, так я і зробив!» В результаті — низька ефективність і якість роботи, погіршення морального клімату. Замість довіри і співпраці в колективі панують підозрілість і формальне взаємодія. А ви ніколи не бачили, як тимлид розмовляє з програмістом тільки «під протокол» і з підписами на кожному аркуші? Нарешті, застосування антипаттернов — це стреси, втома учасників, особисті проблеми, звільнення найбільш професійних співробітників і провал проекту.

Ефективний лідер повинен володіти наступними компетенціями:

• Бачення цілей і стратегії їх досягнення.
• Глибокий аналіз проблем і пошук нових можливостей
• Націленість на успіх, прагнення отримати найкращі результати.
• Здатність співчуття, розуміння стану учасників команди.
• Щирість і відкритість у спілкуванні.
• Навички у вирішенні конфліктів.
• Уміння створювати творчу атмосферу і позитивний мікроклімат.
• Терпимість, вміння приймати людей які вони є, прийняття їх права на власну думку і на помилку.
• Вміння мотивувати правильне професійне поведінку членів команди.
• Прагнення виявляти і реалізовувати індивідуальні можливості для професійного зростання кожного.
• Здатність активно "забезпечувати", "діставати", "вибивати" і т. д.

Не існує однієї кращої стратегії керівництва. В залежності від готовності учасників робочої групи виконувати завдання керівника, він повинен використовувати одну з 4-х стратегій [4]:

1. «Директивне управління». Керівник каже, вказує, спрямовує, встановлює. Жорстке призначення робіт, суворий контроль термінів і результатів.
2. «Пояснення». Лідер "продає", пояснює, прояснює, переконує. Поєднання директивного і колективного управління. Пояснення своїх рішень.
3. «Участь». Лідер бере участь, заохочує, співпрацює, виявляє відданість. Пріоритетне колективне прийняття рішень, обмін ідеями, підтримка ініціативи підлеглих.
4. «Делегування». Лідер делегує, спостерігає, обслуговує. «Не заважати» — пасивне управління сформувався лідера.

Застосування цих стратегій можна проілюструвати на прикладах (Рисунок 41).

1. Вас призначили керівником в новий колектив. Ви ще не одержали визнання, а справу робити треба. Стратегія: «Директивне управління».
2. Ви були учасником команди. Вас призначили керівником цієї команди. Довіра є, а впевненості в правильності ваших дій немає. Стратегія: «Пояснення».
3. Вас призначили керівником в новий колектив. Всі знають про ваших колишніх складних і успішних проектах. Всі визнають вашу перевагу, але довіри до вас немає. Ніхто не знає, якою ціною були досягнуті ваші перемоги. Стратегія: «Участь».
4. Між вами і учасниками встановлено взаємна довіра. Все досить мотивовані на успіх проекту. Кожен сам собі може бути керівником. Стратегія: «Делегування».

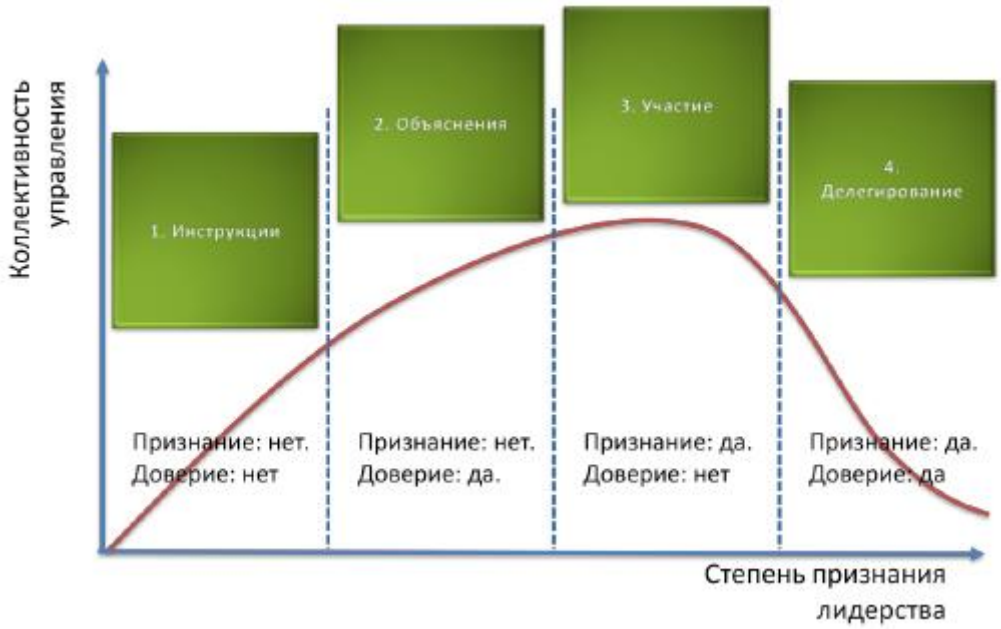


Рис. 41. Ситуаційне лідерство.

Основні зусилля керівника, якщо він прагне отримати найвищу продуктивність робочої групи, повинні бути спрямовані на вивчення і зміну об'єкту управління: людей та їх взаємодії. Отже, завдання адаптивного управління ми можемо розділити на дві підзадачі:
1. Забезпечити ефективність кожного учасника робочої групи.
2. Забезпечити ефективні процеси взаємодії.

Правильні люди

Рональд Рейган говорив: «Оточіть себе найкращими людьми, яких ви тільки зможете знайти, передайте їм в руки владу і не заважайте їм».
За роки роботи у мене сформувалося власне бачення правильного командної поведінки. Ефективний командний гравець:
• Займає активну позицію, прагне розширити свою відповідальність і збільшити особистий внесок у загальну справу.
• Постійно придбає нові професійні знання і досвід, висуває нові ідеї, спрямовані на підвищення ефективності досягнення загальних цілей, домагається поширення своїх знань, досвіду та ідей серед колег.
• Отримує задоволення від своєї роботи, пишається її результатами і прагне, щоб ці ж почуття відчували всі колеги.
• Чітко усвідомлює свої особисті і загальні цілі, розуміє їх взаємообумовленість, наполегливо прагне до їх досягнення.
• Впевнений у собі і в своїх колегах, об'єктивно оцінює їх досягнення та успіхи, уважно ставиться до їх інтересів і думок, активно шукає взаємовигідне рішення в конфліктах.
• Є оптимістом, при цьому твердо знає, що навколишній світ недосконалий; сприймає кожну нову проблему, як додаткову можливість підтвердити власний професіоналізм у своїх очах і в думці колег.

Разом з тим, зустрічаються патології поведінки, які, на мій погляд, неприйнятні в команді:

• *Непорядність*. Брехливість, відсутність совісті і почуття справедливості, здатність на низькі вчинки.
• *Синдром гострого дефіциту емпатії*. Егоцентризм. Неповага і неувага до партнерів. Схильність до негативних оцінок інших. Грубість. «Кожен сам за себе! — ніхто тобі не допоможе!» «Людина людині вовк!»

• «*Звезданутость*». Завищена самооцінка. Підкреслення власної переваги. Умничанье. Людина сильно переоцінює свій особистий вклад у загальну справу і тому вважає, що він повинен працювати не менше, ніж його «менш здібні» колеги.
• *Вульгарний анархізм*. Вольниця — це повна безвідповідальність, свобода від будь-яких зобов'язань перед іншими, нічим не стримувані прояви почуттів, дії або вчинки. «Произвольничать, діяти самовільно, в образу іншим, нахабно, зухвало» (с) Ст. Даль. Не плутати зі «свободою»!
• «*Соціальний паразитизм*». Прагнення прожити привільно за чужий рахунок там, де відповідальність розмита, а особистий внесок важко чітко виділити.

Рекомендація: лікувати патологію «хірургічно» — позбуватися від проблемних людей. Виховують у дитячому садку, ну ще трохи в початковій школі. Далі люди виховуються тільки самостійно, а оточуючі можуть лише допомагати або не заважати в цьому процесі. Переконаний, що кожна доросла людина має те, до чого він свідомо чи несвідомо прагне. Доглядати і виховувати людину — це значить захищати його від проблем, закривати йому шлях до переосмислення свого досвіду і розвитку, «заганяти хворобу всередину» за допомогою «соціального аспірину».

Чотири необхідних і достатніх умов для того, щоб співробітник ефективно вирішив поставлене завдання. Це

1. Розуміння цілей роботи.
2. Вміння її робити.
3. Можливість її зробити.
4. Бажання її зробити.

Для того щоб забезпечити виконання цих умов, керівник повинен вміти ефективно виконувати чотири функції:

1. *Направлять*. Якщо співробітник не розуміє що робити, завдання керівника — забезпечити загальне бачення цілей і стратегії їх досягнення.
2. *Навчати*. Якщо працівник не вміє, завдання керівника — «навчати», бути наставником і зразком для наслідування.
3. *Допомагати*. Якщо у співробітника не може виконати роботу, завдання керівника — «допомагати», забезпечити виконавця усім необхідним, прибрати перешкоди з його шляху.
4. *Надихати*. Якщо у співробітника не мають бажання виконати роботу, завдання керівника — «надихнути», забезпечити адекватну мотивацію учасника протягом всього проекту.

Мотивація

Відомо, що жодна задача не буде вирішена за будь, відведений на це час, якщо людина не захоче її зробити. Він завжди знайде для виправдання цього 100 «об'єктивних» причин, замість того, щоб знайти хоча б один спосіб розв'язання задачі. У кожного учасника робочої групи повинна бути особиста мета (внутрішня мотивація), яку він зможе досягти, просуваючи проект до успіху.

Почніть з себе! Вам потрібно чітко розуміти, у чому полягає ваш виграш у разі успішного завершення проекту. Добитися від учасників прихильності проекту більше, ніж маєте ви самі, вам не вдасться. Якщо в учасника немає такої значимої особистої мети, позбудьтеся від нього. Інакше вам доведеться витратити весь свій час на «промивання його мізків» і спроби мотивувати його на ефективну роботу.

Мотивація повинна починатися з підбору співробітників в команду. У старій економіці людей наймали за вміння і навчали потрібного відношенню до справи. У новій економіці необхідно чинити з точністю до навпаки: наймати за потрібне ставлення до справи і навчати необхідним умінням.

Люди не народжуються переможцями, вони ними стають. Кандидата варто наймати тільки у разі, якщо ви можете запропонувати йому можливість стати переможцем. Справжній лідер пропонує не роботу, а можливості.

Всі люди різні, а ситуацій, в яких вони можуть знаходитися в ході проекту, незліченна безліч. Бійтеся стереотипів. Якщо ви не враховуєте індивідуальні особливості конкретної особистості, то ефективність ваших взаємодій сильно знижується. Модель об'єкта управління нам невідома, отже, не може існувати вичерпний набір правил, типу «якщо..., то...», за яким зміг би діяти керівник. Тому, скільки людей і ситуацій, стільки й варіантів рішень повинен мати ефективний керівник у своєму запасі. «Якщо у керівника в руках тільки молоток, то всі навколо будуть схожі на цвяхи».

Керівник при пошуку рішення спирається на свій багаж знань і умінь. Він намагається зрозуміти кожного учасника, класифікувати стан, знайти у своєму досвіді схожу ситуацію і адаптувати раніше використане успішне рішення стосовно до даного конкретного випадку. Таким чином, керівник прагне допомогти людині (об'єкту управління) перейти в нове більш ефективне з точки зору цілей проекту стан.

Потім керівник повинен спостерігати за результатами свого впливу — це і є додатковий контур зворотного зв'язку. Необхідно пам'ятати, що зрозуміти людину можна, тільки слухаючи і чуючи, що він говорить. Керівник, який протягом тижня не поспілкувався індивідуально з кожним із своїх прямих

підлеглих, дарма отримує зарплату. І зовсім не обов'язково розмова повинна йти про статус проектних робіт. Часом, досить поговорити про погоду, кіно або футболі. Після цього керівник аналізує отримані результати і акумулює новий досвід (позитивний або негативний) у своїй «базі знань».

Чим досвідченіша керівник, тим точніше він може розпізнати і класифікувати ситуацію, тим більше в його «базі знань» прецедентів, використовуючи які він може синтезувати рішення для даного конкретного випадку. Саме тому в управлінні програмними проектами в першу чергу цінується досвід керівника і тільки потім, можливо, його звання і знання.

Програміст складається з чотирьох компонентів: тіло, серце, розум і душа.

1. Тілу необхідні гроші і безпека.
2. Серцю — любов і визнання.
3. Розуму — розвиток і самовдосконалення.
4. Душі — самореалізація.

Надайте все це вашим співробітникам, і ефективність їх праці зросте багаторазово. В іншому випадку люди, які хочуть перемагати, знайдуть все це в іншій команді, а у вашій залишаться тільки невдахи.

Ефективна взаємодія

Раніше ми вже говорили, що процес виробництва програмного забезпечення, що застосовується в проекті, повинен ґрунтуватися на итеративности, инкрементальности, самоврядності команди і адаптивності. Головний принцип: не люди повинні будуватися під обрану модель процесу, а модель процесу повинна підлаштовуватися під конкретну команду, щоб забезпечити її найвищу продуктивність.

У програмній інженерії багато вже визнали, що найбільш ефективні виробничі процеси складаються в самоврядних і самоорганізуються робочих командах. Ідеї командного менеджменту на Заході зародилася на початку 80-х років. Ефективність команд в нових економічних умовах одними з перших оцінили такі гіганти, як Procter & Gamble і Boeing [5]. Доктрина командного менеджменту припускає ясність загальних цінностей і цілей, самоорганізацію і самоврядування спільною діяльністю, взаємний контроль, взаємодопомога і взаємозамінність, колективну відповідальність за результати праці, всебічний розвиток і використання індивідуального і групового потенціалів.

Керівнику недостатньо стати лідером, треба ще зуміти згуртувати команду. Експерти в області командного менеджменту виділяють 4 обов'язкові послідовні стадії, через які повинна пройти робоча група перш, ніж вона стане ефективною командою.

1. ***Forming***. Формування. Характеризується надлишком ентузіазму, пов'язаної з новизною. Люди повинні подолати внутрішні суперечності, перехворіти конфліктами перш, ніж сформується справді спаяний колектив. На цьому етапі багато залежить від керівника. Він повинен чітко поставити цілі членам команди, вірно визначити роль кожного у проекті.
2. ***Storming***. Розбіжності і конфлікти. Самий складний і небезпечний період. Мотивація новизни вже зникла, а сильні і глибокі стимули у команди ще не з'явилися. Неминучі складності або невдачі породжують конфлікти і «пошук винних». Учасники команди методом проб і помилок виробляють найбільш ефективні процеси взаємодії. Керівнику на цьому етапі важливо забезпечити відкриту комунікацію в команді. Конфлікти не слід ховати або розрубувати. Спори необхідно розрулювати спокійно, терпляче і ретельно.
3. ***Norming***. Становлення. У команді зростає довіра, люди починають помічати у колег не тільки проблемні, але і сильні сторони. Закріплюються і відточуються найбільш ефективні процеси взаємодії. На зміну битві амбіцій приходить продуктивне співробітництво. Чіткіше стає поділ праці, зникає дублювання функцій. Керівник перестає перебувати в стані постійного авралу, робота з побудови команди на цьому етапі — вже не гасіння пожежі, а скрупульозна праця з відпрацювання загальних норм і правил.
4. ***Performing***. Віддача. Команда працює ефективно, високий командний дух, люди добре знають один одного і вміють використовувати сильні боку колег. Всі прагнуть дотримуватися вироблених загальних процесів. Високий рівень довіри. Це найкращий період для розкриття індивідуальних талантів.

Часто трапляється, що робоча група в'язне на одній із стадій і ніколи не досягає плато найвищої продуктивності.

Якщо команда пройшла всі стадії формування і вийшла на фазу «Performing», не варто вважати, що менеджер проекту може делегувати свої повноваження і відправитися у відпустку. Завдання менеджера на цьому етапі — «точити пилу». Це значить підтримувати необхідний рівень мотивації, бути штурманом, шукати нові шляхи і відкривати нові можливості. Постійно спостерігати і оцінювати ефективність усіх процесів, задіяних у проекті. Шукати відповідь на питання: «Що загрожує проекту?» «Що зайве ми робимо?» «Що можна робити простіше?» Працювати на

скорочення непотрібних зусиль замість того, щоб прагнути до нових героїчним подвигам».

Якщо керівник не буде докладати додаткові зусилля команда, рано чи пізно, почне «сповзати» з плато найвищої ефективності в стан застою і стагнації (Малюнок 42). Пам'ятайте, що оточення і команда змінюються по ходу проекту. Колишня мотивація слабшає або перестає діяти. Змінюйте правила і процеси. Відмовляйтеся від того, що перестало діяти або стало працювати неефективно. «Струшуйте» (Reforming) і повертайте команду в стадію «Forming». Це дозволить їй знову, пройшовши через всі етапи становлення, вийти на новий більш високий рівень продуктивності. Зрозуміло, робити це варто, після здачі чергового релізу програмного продукту, ну і, можливо, в разі глибокої кризи проекту.

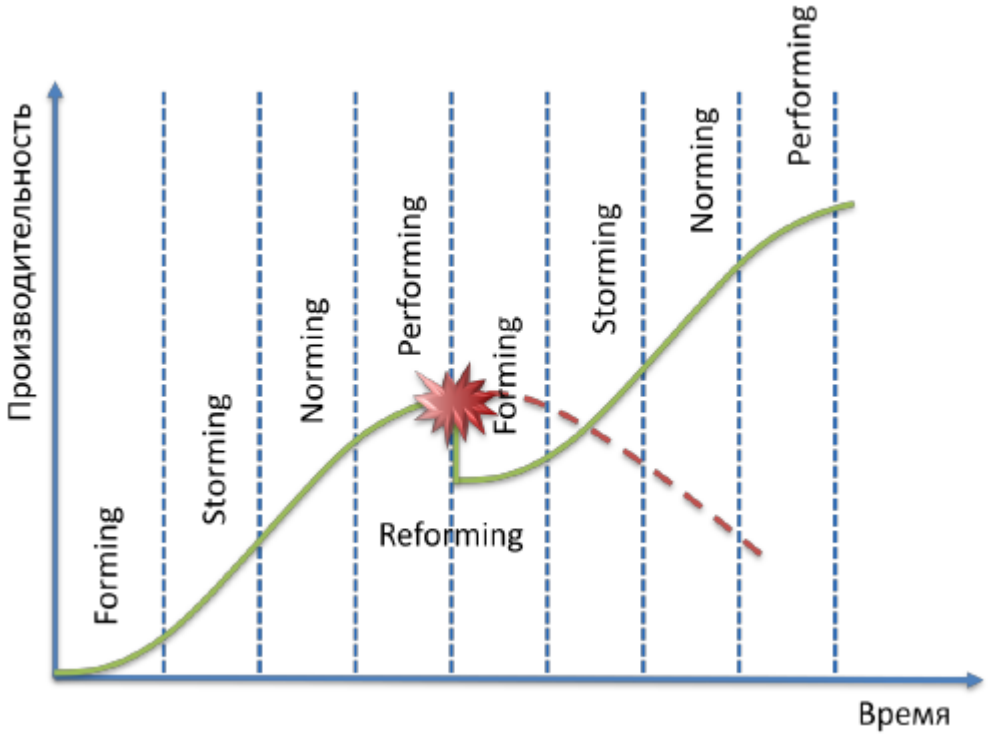


Рис.42. Reforming. «Струшування» і переклад команди проекту на новий, більш високий, рівень продуктивності.

Висновки

Ефективні команди не утворюються самі по собі, вони кристалізуються навколо визнаного лідера. Для того щоб стати лідером, необхідно:

• Визнання командою професійної компетентності та переваги лідера.

• Повна довіра команди до дій і рішень лідера, визнання його людських якостей, переконаність в його чесності, порядності, віра в його щирість і сумлінність.

Мотивація повинна починатися з підбору співробітників в команду. У старій економіці людей наймали за вміння і навчали потрібного відношенню до справи. У новій економіці необхідно чинити з точністю до навпаки: наймати за потрібне ставлення до справи і навчати необхідним умінням.
Робоча група перш, ніж вона стане ефективною командою, повинна пройти чотири обов'язкових послідовних стадії: 1) **Forming**, 2) **Storming**, 3) **Norming**, 4) **Performing**.

Якщо керівник не буде докладати додаткові зусилля команда, рано чи пізно, почне «сповзати» з плато найвищої ефективності в стан застою і стагнації. Завдання менеджера на цьому етапі — «точити пилу»: підтримувати необхідний рівень мотивації, бути штурманом, шукати нові шляхи і відкривати нові можливості. Чотири стадії розвитку команди повинні циклічно повторюватися, щоб забезпечити безперервне зростання продуктивності.

Додаткова література

1. С. Архипенков, "Руководство командой разработчиков программного обеспечения. Прикладные мысли", 2008
2. Стивен Р. Кови, «7 навыков высокоэффективных людей. Мощные инструменты развития личности», 2-е изд., М., Альпина Бизнес Букс, 2007.
3. Манфред Кетс де Врис, «Мистика лидерства. Развитие эмоционального интеллекта», М., Альпина Бизнес Букс, 2005.
4. Hersey P., Blanchard K.H. "Management of Organizational Behavior", 6th ed., Englewood Cliffs: Prentice-Hall, 1993.
5. Л. Томпсон, «Создание команды», М., Вершина, 2005.

Лекція 17. **ТЕМА 8** ***Реалізація проекту. Робоче планування***

Управління — це розчленування, аналіз, визначення послідовності дій, конкретна реалізація. Управління фокусується на нижньому рівні: як мені зробити це найкращим чином? Ця компетенція керівника визначає ефективність руху по вибраному шляху. Як правило, менеджери, що вийшли з програмістів, без особливого праці оволодівають необхідними управлінськими навичками.
Базове розклад, складений на етапі планування проекту, служить орієнтиром для моніторингу стану справ на макрорівні. Для оперативного управління

проектом використовується робочий план. Робоче планування рекомендується виконувати методом «набігаючої хвилі»: робота, яку треба буде виконати в найближчій перспективі, детально планується на нижчому рівні ICP, а далеко відстоїть робота планується на порівняно високому рівні ICP.

Елементарна робота, як правило, являє собою окремий функціональний вимога до програмного продукту або запит на зміну, над яким послідовно працюють: бізнес-аналітик, розробник, розробник і тестувальник документаліст. Трудомісткість елементарної роботи кожного з виконавців має бути від 4 до 20 чол. годину. Якщо трудомісткість завдання не вкладається в ці межі, слід провести декомпозицію роботи.

Для робочого планування доцільно використовувати систему управління завданнями або багтрекинга, оскільки вона дозволяє задавати послідовність переходів завдання від виконавця до виконавця, управляти пріоритетами робіт і адекватно відстежувати їх статус: аналіз, проектування, кодування, тестування, документування. Робота повинна вважатися закінченою тільки тоді, коли реалізація вимоги протестована і документована.

Залежно від рівня професіоналізму та зрілості команди проекту розподіл робіт може здійснюватися або директивно з жорсткою постановкою терміну і контролем виконання кожного завдання, або ці повноваження делегуються виконавцям. У цьому випадку вони самі обирають завдання послідовно у відповідність з пріоритетами, а їх виконання періодично аналізується на статус мітингу. Можна рекомендувати щотижневі збори за статусом проекту всієї команди або, якщо проект досить великий, то його ключових приватників: керівників проектів і лідерів команд. Гарний час для цього ранок понеділка, оскільки учасники проекту, особливо студенти, які поєднують навчання і роботу, часто працюють у вихідні, але, зрозуміло, не тому, що аврал, а тому що їм так зручніше. Обговорюються, як правило, всього три питання:

• *Загрози і проблеми.*
• *Аналіз результатів за тиждень.*
• *Уточнення пріоритетів завдань на новий тиждень.*

Як правило, немає сенсу оцінювати відсоток реалізації роботи в проміжному стані, оскільки, якщо завдання передана до тестування, то це зовсім не означає, що 70% роботи зроблено. На етапі тестування може бути виявлена помилка проектування і вся робота почнеться заново. Рекомендація — використовувати правило «50/100». Якщо робота по завданню розпочато,

то слід враховувати її, як виконані на 50%. А 100% повчає тільки протестована і документована робота.

Принципи кількісного управління

«Тим, що не можна виміряти, не можна керувати». Виміру за проектом необхідно виконувати регулярно, не рідше одного разу на 1-2 тижні. Для кожного вимірної показника повинні бути визначені його планові значення. Для кожного планового значення повинні бути визначені три області критичності відхилень:

• Допустимі відхилення. Передбачається, що ніяких керуючих впливів не потрібно.

• Критичні відхилення. Потрібен ретельний аналіз причин відхилення і за необхідності застосування коригувальних дій.

• Недопустимі відхилення. Потрібен терміновий аналіз причин відхилення і обов'язкове застосування коригувальних дій.

Вимірювання необхідно проводити регулярно. Мета — виявити причини настали або можливих критичних і неприпустимих відхилень. Результатом аналізу повинні стати планування коригувальних дій щодо компенсації неприпустимих відхилень, їх реалізація та моніторинг результативності застосування цих коригувальних дій.

Всі вимірювання необхідно зберігати в репозитарії проекту. Вимірювання, накопичені в ході проекті, є найбільш достовірною основою при детальній оцінці та плануванні робіт на наступних ітераціях проекту.

Оскільки головне завдання менеджера утримати проект у межах «залізного» трикутника, то, в першу чергу, необхідно аналізувати відхилення проекту щодо термінів і витрат. Робиться це за допомогою методу освоєного обсягу [1]. Доводилося стикатися з думкою, що цей метод не застосуємо в управлінні програмними проектами. Це дійсно так, якщо ми використовуємо «водопадную» модель процесу розробки. Але якщо ІСР проекту, орієнтована на инкрементальную розробку, то це означає, що на верхніх рівнях декомпозиції знаходяться компоненти проектного продукту і їх функціонал, а не виробничі процеси. Отже, якщо в проекті реалізовані, протестовані і документовані 50 % функціональних вимог, то є всі підстави вважати, що залишилася приблизно половина проектних робіт.

Суть методу оцінки проекту з освоєному обсягом полягає в наступному. Спочатку оцінюється відхилення від графіка SV (Shedule Variance), у грошових одиницях:

$$SV = EV - PV,$$

де

• EV (Earned Value) — освоєний обсяг. Планова вартість виконаних робіт. Обсяг виконаних робіт, виражений в термінах схваленого бюджету, виділеного на ці роботи для планової операції та елементу ієрархічної структури робіт;

• PV (Planned Value) — плановий обсяг. Планова вартість запланованих робіт. Затверджений бюджет, виділений на планові роботи, що виконуються в рамках планової операції або елементу ієрархічної структури робіт.

Наприклад. Нехай ми на поточний момент реалізували (протестували і документували) 20 функціональних вимог, на кожне з яких було заплановано витратити за 40 чол. годину. по 1000 грн, то освоєний обсяг буде

EV = 20 * 40 * 1000 = 800 000 грн.

Якщо ж на поточний момент планувалося реалізувати тільки 15 вимог, то плановий обсяг буде

PV = 15 * 40 * 1000 = 600 000 грн.

Отже, ми випереджаємо графік (відхилення від графіка позитивне) на величину

SV = EV - PV = 800 000- 600 000 = 200 000 грн.

Як пересчитываете відхилення від графіка, виражену в грошових одиницях, скорочення термінів проекту ілюструється на рис. 43.

Якщо ми випереджаємо графік, то це не обов'язково означає що проект йде успішно. Добре це чи погано-залежить від значення іншого показника методу освоєного об'єму: CV (Cost Variance) — відхилення по витратам, яке оцінюється за формулою:

CV = EV - AC

де

• AC (Actual Cost) — фактичні витрати. Фактична вартість виконаних робіт. Фактичні витрати на виконання робіт за певний період в рамках планової операції або елементу ієрархічної структури робіт.

Наприклад, якщо ми для того що скоротити час робіт за проектом працювали 25% часу понаднормово та у вихідні дні з подвійною оплатою, то фактичні трудовитрати склали:

AC = 20 * (30 * 1000 + 10 * 2000) = 1 000 000 грн.

Тому відхилення за витратами в нашому випадку буде

CV = EV - AC PV = 800 000 - 1 000 000 = - 200 000 грн.

Від'ємне значення відхилення за витратами означає, що ми перевищили бюджет, що, в загальному випадку, не дуже добре. Але якщо термін завершення проекту для нас має вищий пріоритет, і наші прогнозовані витрати по завершенню проекту не перевищують планових з урахуванням управлінського резерву (рис. 43), то в цьому випадку можна вважати, що проект виконується успішно.

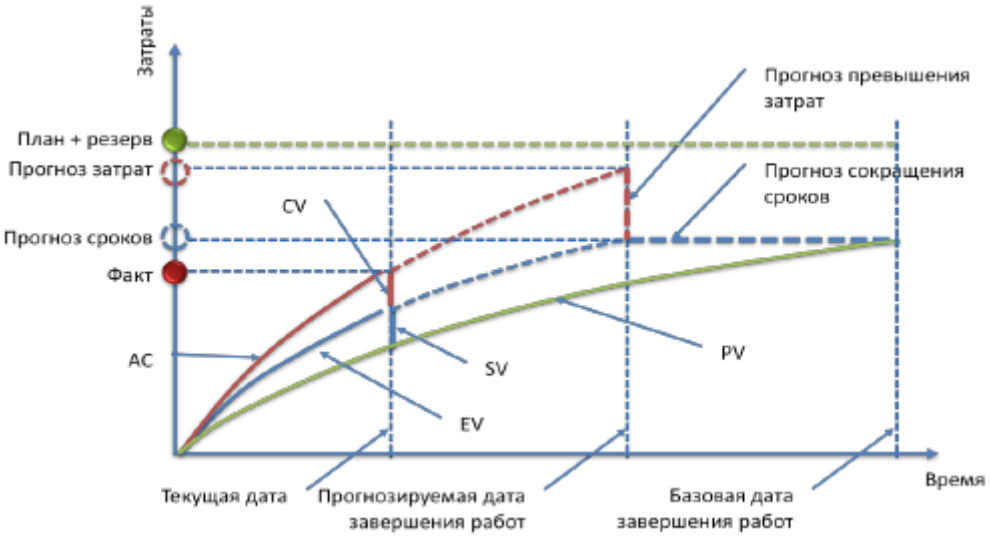


Рис. 43. Оцінка і прогноз показників по методу освоєного об'єму

Відхилення від бюджету і за термінами в абсолютних грошових одиницях недостатньо для характеристики проектів різних масштабів. Більш наочні відносні показники: індекс виконання термінів SPI (Schedule Performance Index)

$$\mathbf{SPI = EV / PV}$$

і індекс вартості виконання CPI (Cost Performance Index)

$$\mathbf{CPI = EV/ AC,}$$

які характеризують проект незалежно від його розміру. Якщо значення обох індексів більше 1, то це свідчить про благополучному стані у проекті.

Які ще вимірні показники доцільно застосовувати в управлінні програмним проектом?

В першу чергу це показник прогресу проекту, частка реалізованих і перевірених високорівневих вимог до проекту, наприклад відношення числа завершених сценаріїв використання продукту до їх загального числа.

Інший показник — стабільність проекту, загальна кількість прийнятих (затверджених спонсором або замовником) змін в плані управління проектом.

Чим вище нестабільність в проекті, тим більше складність у керуванні роботами і нижче продуктивність учасників.

Якщо хтось думає, що це код вирішення проблеми, це не так. Код це нове джерело проблем. Тому завжди слід вимірювати поточний розмір проекту — кількість рядків вихідного коду, доданих, змінених і віддалених у ході виконання проекту розробки ПО. Чим більше обсяг вихідного коду, тим більше часу буде потрібно на внесення змін і виправлення помилок.

При збільшенні обсягу проектного продукту трудовитрати на кожну нову рядок вихідного коду збільшуються. Якщо за номінал взяти продуктивність проектної команди при виробництві продукту в 10 KSLOC, то та ж команда на проекті в 100 KSLOC покаже продуктивність в 1.3–1.7 разів меншу, а на проекті 1000 KSLOC слід очікувати, що продуктивність знизиться в 1.6–3.0 рази [2].

Великий обсяг коду так само потребує більшої кількості людей на його супровід. Оскільки, навіть якщо буде виявлятися лише кілька критичних помилок у рік, то для того, щоб їх виправити в прийнятні терміни, наприклад, за 24 години, в продукті загальним обсягом 1000 KSLOC один програміст з цим не впорається. Це пов'язано з тим, що для того, щоб виправити помилку в обмежені терміни необхідно оперативно виявити та усунути її причину, а для цього треба добре знати архітектуру та код програмного продукту. Щоб ефективно супроводжувати продукт такого об'єму необхідно мати в «гарячому» резерві приблизно 20 розробників, тому що 50 KSLOC, на мій погляд, це граничний обсяг коду, який може утримувати в голові і ефективно супроводжувати одна людина. І ще проблема: чому цих людей займати у вільний від виправлень помилок, якщо немає нових проектів розвитку продукту.

Наступний важливий показник стану проекту — це середня продуктивність, відношення поточного розміру проекту до фактичних витрат за проектом. С. Макконнелл [2] наводить такі показники (мінімальне, максимальне і середнє значення продуктивності в KSLOC на один чол.*міс. фактичних витрат для стандартних типів проектів обсягом у 100 KSLOC:

- *300-7000 (800) — інтранет система.*
- *200-7000 (600) — бізнес система.*
- *100-2000 (300) — система Інтернет.*
- *50-600 (100) — системне ПЗ, телекомунікації.*
- *20-300 (50) — системи реального часу.*

Висока продуктивність у проекті — це далеко не завжди хороша ознака. Доводилося зустрічатися з проектами, в яких внаслідок активного застосування методу «copy+past», середня продуктивність в розробці бізнес системи досягала 2000 SLOC/чол.*міс. Однак для реалізації необхідного функціоналу було написано в 3-4 рази більше коду, ніж це могло б бути адекватною опрацювання архітектури.

Ще одна група кількісних показників, які слід спостерігати в ході реалізації проекту, характеризує якість програмного продукту:

• Дефектність продукту — кількість виявлених дефектів на одиницю обсягу продукту (наприклад, KSLOC).

• Частка не усунених дефектів — відношення кількості незакритих максимально критичних і критичних дефектів до кількості виявлених невідповідностей.

• Середні витрати на супровід — середні трудовитрати на виправлення одного дефекту. Високе значення цього показника може свідчити про неякісну архітектурі програмного продукту.

• Документованість коду — визначає відсоток рядків вихідного коду коментарі по відношенню до загальної кількості рядків.

Слід підкреслити, що треба спостерігати за середніми за проектом значеннями показників, і ні в якому разі не намагатися вимірювати індивідуальні характеристики продуктивності і якості. Головні причини, чому цього не слід робити, полягають у тому, що, по-перше, в цьому випадку замість злагодженої командної роботи ми отримаємо особисту конкуренцію, а, по-друге, найбільш «просунуті» розробники стануть працювати на формальні показники, а не на досягнення цілей проекту.

Якщо команда дійсно відбулася, то для неї характерна колективна відповідальність за досягнення загальних цілей. І, як пише, Т. Демарко [3], «менеджер проекту повинен займати чергу, щоб покритикувати працівника, який не виконує свої обіцянки», оскільки в правильній команді для цього завжди знайдеться маса бажаючих.

Завершення проекту

Головна мета цієї фази — перевірити і передати замовникові результат проекту. Для цього необхідно виконати приймально-здавальні роботи у відповідності з процедурою приймання, яка повинна бути визначена заздалегідь на самій ранній стадії проекту.

Результати проекту повинні бути передані у впровадження або доопрацювання, або належним чином законсервовані для подальшого використання. Не повинно залишатися «завислих» робіт за проектом. Всі

лінійні керівники всіх учасників повинні бути сповіщені про завершення робіт за проектом, і звільнення співробітників.
Важлива задача, яка повинна бути вирішена на даній фазі, реалізація зворотного зв'язку за проектом. Мета — зберегти результати, знання і досвід, отримані в проекті, для більш ефективного і якісного виконання аналогічних проектів в майбутньому. Необхідно архівувати всі результати, документувати досвід, уроки по проекту та пропозиції щодо поліпшення технології виконання робіт та управління проектами.
Всі проекти і в особливості провальні проекти повинні завершуватися підсумковим звітом, якщо компанія не хоче «наступати на одні і ті ж граблі». Пам'ятаємо про те, що «вчорашні проблеми, це сьогоднішні ризики».
Підсумковий звіт повинен містити наступну інформацію:
• Підсумки проекту:
• Досягнення цілей проекту
• Додаткові корисні результати
• Фактичні терміни
• Фактичні витрати
• Обґрунтування відхилення від цілей
• Відхилення результатів від вимог
• Уроки проекту
• Проблеми проекту і способи їх вирішення
• Матеріали програмні компоненти для подальшого використання
• Пропозиції щодо зміни технологій або стандартів компанії
На фазі завершення бажано реалізувати і план мотивації учасників проектної команди, оскільки відкладене винагороду мотивує істотно слабкіше.

Висновки

Для оперативного управління проектом використовується робочий план. Елементарна робота, як правило, являє собою окремий функціональний вимога до програмного продукту або запит на зміну, над яким послідовно працюють: бізнес-аналітик, розробник, розробник і тестувальник документаліст.
Виміру за проектом необхідно виконувати регулярно, не рідше одного разу на 1-2 тижні. Для кожного вимірної показника повинні бути визначені його планові значення і допустимі відхилення.
До складу вимірюваних показників повинні входити наступні характеристики проекту:
• Освоєний і плановий обсяги робіт і фактичні витрати по проекту.

• Показники прогресу і стабільності проекту.
• Розмір продукту.
• Продуктивність.
• Показники якості програмного продукту.

За результатами проекту обов'язково повинна бути реалізована зворотний зв'язок. Мета — зберегти результати, знання і досвід, отримані в проекті, для більш ефективного і якісного виконання аналогічних проектів в майбутньому.

Додаткова література

1. «PMBOK. Руководство к Своду знаний по управлению проектами», 3-е изд., PMI, 2004.
2. С. Макконнелл, «Сколько стоит программный проект», «Питер», 2007.
3. Том Демарко, Тимоти Листер, «Человеческий фактор: успешные проекты и команды», Спб. Символ-Плюс, 2005.

Висновок. Ростіть професіоналів

Робота менеджера проекту подібна праці садівника.

Подібно до того, як садівник любовно відбирає найбільш підходящі рослини для свого майбутнього саду, менеджер набирає людей, найбільш відповідних цілям проекту.

Подібно до того, як садівник шукає кращу грунт для кожної рослини з урахуванням його особливостей, менеджер для кожного учасника проектної команди шукає найбільш підходящу для нього завдання.

Подібно до того, як садівник ретельно плекає своїх вихованців, оберігає їх від шкідливих впливів, стежить за тим, щоб жодна рослина не затінювало інше, а тільки доповнювала його і сприяло його зростанню, менеджер проекту терпляче працює з кожним учасником проектної команди, сприяючи його правильному розвитку, охороняючи від зовнішніх і внутрішніх потрясінь, для того, щоб максимально розкрити його індивідуальні здібності і збільшити віддачу від них, із задоволенням відзначає кожне нове досягнення.

«Що посієш, те й пожнеш» — цей закон однаково застосовується як до праці садівника, так і до праці менеджера проекту. Зневажливе ставлення до людей породить лише відповідь зневагу. Вкладіть у людей частину своєї душі, і вам воздасться сторицею.

ЛІТЕРАТУРА

1. IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.
2. IEEE Std 1074-1995, IEEE Standard for Developing Software Life Cycle Processes.
3. «Руководство к своду знаний по программной инженерии». The Guide to the Software Engineering Body of Knowledge, SWEBOK, IEEE Computer Society Professional Practices Committee, 2004.
4. David Rubinstein, «Standish Group Report: There's Less Development Chaos Today». 2007
5. Брукс Фредерик, «Мифический человеко-месяц, или Как создаются программные комплексы», Пер. с англ., СПб., Символ-Плюс, 1999.
6. «PMBOK. Руководство к Своду знаний по управлению проектами», 3-е изд., PMI, 2004.
7. Уолкер Ройс, «Адаптивный стиль управления программными проектами». Открытые системы. 2006. № 1.
8. Ершов А. П., «О человеческом и эстетическом факторе в программировании». Информатика и образование. 1993. № 6.
9. Paulk, Mark C., and others, Capability Maturity Model for Software, Version 1.1 (CMU/SEI-93-TR-24). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1993.
10. Филипп Крачтен, «Введение в Rational Unified Process», Вильямс, 2002 г.
11. «MSF, Microsoft, Microsoft Solutions Framework», Отдел MSF, Microsoft, 2002.
12. M. Pomeroy-Huff, J. Mullaney, R. Cannon, M. Sebern, «The Personal Software Process (PSP) Body of Knowledge», version 1.0, SPECIAL REPORT CMU/SEI, 2005
13. Watts S. Humphrey, «The Team Software Process (TSP)», Technical Report CMU/SEI, 2000
14. Kent Beck, and others, «Manifesto for Agile Software Development», 2001
15. А. Коуберн, «Люди как нелинейные и наиболее важные компоненты в создании программного обеспечения», Humans and Technology Technical Report, Oct.1999 (русский перевод — К.Максимов, А.Максимова)
16. А. Коуберн, «Каждому проекту своя методология», Humans and Technology Technical Report, TR 99.04, Oct.1999 (русский перевод — К.Максимов, А.Максимова).

17. С. Макконнелл, «Остаться в живых. Руководство для менеджеров программных проектов», «Питер», 2006.
18. С. Архипенков., Лекции по управлению по управлениб программными проектами. Москва. 2009. 127с.
19. С. Архипенков. Руководство комагдой разработчиков программного обеспечения. Прикладные мысли.Москва. 2008. 80с.

Додаткова

20. С. Макконнелл, Сколько стоит программный продукт. Питер. 2007.295с.
21. Том Демарко., Вальсируя с медведями. Управление рисками в проектах по разработке программного обеспечения. 201с.