Тема 1 МЕТОДОЛОГІЧНІ ОСНОВИ ПРОЕКТУВАННЯ ПРОГРАМ

1.1. ЗАГАЛЬНІ ПОЛОЖЕННЯ ТЕОРІЇ ПРОЕКТУВАННЯ

Як без оформленого проекту цілком можна побудувати шпаківню, але неможливе будівництво висотної будівлі або комплексу космодрому з будівельною індустрією, житловими, стартовими та виробничими комплексами, так і без проекту можна реалізувати лише невелику програму, але не автоматизоване робоче місце фахівця, а тим більше автоматизовану систему управління великого підприємства.

Що ж роблять програмісти? Програмісти виробляють програмний продукт. У термінах автоматизованих систем програмісти створюють програмне забезпечення.

Програмний продукт— програма, яку можна запускати, тестувати, виправляти та розвивати. Така програма має бути написана в єдиному стилі, ретельно відтестована до необхідного рівня надійності, супроводжена докладною документацією та підготовлена для тиражування.

Програмний виріб- Програма на носії даних, що є продуктом промислового виробництва. Термін затверджено Державним стандартом.

Програмне забезпечення автоматизованих систем- Сукупність програм на носіях даних та програмних документів, призначена для налагодження, функціонування та перевірки працездатності автоматизованих систем.

Автоматизована система (АС)- Організаційно-технічна система, що забезпечує вироблення рішень на основі автоматизації інформаційних процесів у різних сферах діяльності (управління, проектування, виробництво тощо) або їх поєднаннях, система, що складається з персоналу та комплексу засобів автоматизації його діяльності, що реалізує інформаційну технологію виконання встановлених функцій.

Необхідність у проекті викликана складністю завдання.

Наступний приклад показує нелінійну залежність зростання складності завдання від її розміру. Необхідно в думці скласти числа 4 і 3. Відповідь, зрозуміло, - 7. Необхідно в думці перемножити числа 7 і 9. Відповідь, звичайно, - 63. Але якщо не знаєте таблицю множення, то треба виконати нестандартне перетворення у вигляді багаторазового додавання. Чи важко воно для вас? Необхідно в думці перемножити числа 289 і 347. Якщо ви не феноменальний лічильник, то чи вистачить у вашій голові оперативної пам'яті? А чи зможете ви перемножити на думці шестизначні числа? Але якщо декомпозувати це завдання на обчислення ряду творів одного з співмножників на окремі цифри іншого співмножника і потім знайдені твори скласти (при цьому записувати на папері всі проміжні результати), то з цим завданням може справитися пересічна людина.

Ще приклад, що показує один із шляхів зниження складності завдання за рахунок її декомпозиції на доступні для огляду частини. Звичайна нормальна людина із середніми здібностями може одночасно у своїй голові втримати не більше семи думок. У школі завдання з шістьма діями

вважаються завданнями підвищеної складності та позначаються символом «*». В арміях різних країн, часів і народів проводилося поділ на десятки, сотні, тисячі. У командирів у підпорядкуванні перебувало або десятьох воїнів, або десятьох молодших командирів.
Програма - дуже складний об'єкт, що містить до сотень тисяч і навіть кількох мільйонів думок. Складність програмного продукту - аж ніяк не випадкова властивість, скоріше необхідна. Його складність визначається чотирма основними причинами: складністю завдання, складністю управління процесом розробки, складністю опису поведінки окремих підсистем, складністю забезпечення гнучкості кінцевого програмного продукту.
У табл. 1.1 наведено п'ять ознак складної системи разом із прикладами. Ці ознаки інваріантні як для відчутної системи реального світу «музичний центр», так програмної системи — текстового редактора.
Таблиця 1.1.

Приклади музичного центру та текстового редактора як складних систем

Ознаки	Музичний центр	Текстовий редактор
1. Складність часто представляється як ієрархії. Складна система зазвичай складається з взаємозалежних підсистем, які також можуть бути розділені на підсистеми і т. д., аж до найнижчих рівнів абстракції	Складається з шести підсистем: підсилювача, блоку цифрового керування системою, програвача компакт-дисків, касетної деки, радіоприймача, динаміків. Кожна з підсистем може бути розділена на підсистеми. Підсилювач поділяється на фільтри, попередні каскади посилення та підсилювач потужності. Блок цифрового управління системою поділяється на процесор, панель кнопок, панель індикації та цифро аналогові, аналого-цифрові перетворювачі. Програвач компакт-дисків - на лазер, пристрій керування лазером, цифро аналоговий перетворювач і т.д.	Складається з файлів: опис глобальних констант і змінних, бібліотеки модулів підтримки дисплея, бібліотеки модулів підтримки клавіатури, бібліотеки модулів підтримки головного «меню», набору модулів самого редактора. Бібліотека модулів підтримки клавіатури у свою чергу включає модуль рядкового редактора, який використовує такі модулі файлу бібліотеки підтримки дисплея, як відображення рядка на екрані та переміщення курсору в задану позицію, а також цілий ряд внутрішніх модулів
2. Вибір нижчого рівня абстракції є значною мірою довільним і більшою мірою визначається спостерігачем	Як нижчий рівень абстракції можна вибрати вузли, що виконують закінчені функції обробки електронних або звукових сигналів: підсилювальні каскади — посилюють сигнали, фільтри — забезпечують виключення перешкод відповідних частот тощо. , Т. е. Операційні підсилювачі, транзистори, діоди та ін.	Системні аналітики як нижчий рівень абстракції в програмах використовують модулі. Кодувальники, що реалізують модулі, як нижчий рівень абстракції використовують алгоритмічні структури (оператори) мови високого рівня та структури даних
3. Внутрі елементні зв'язки зазвичай міцніші за між елементні зв'язки. Тому взаємодії елементів усередині елементів системи виявляються природно відокремленими від взаємодії між самими	Кожен вузол, як правило, має або один (керуючий), або два входи (керуючий та сигнальний) і лише один вихід (оброблений сигнал). Зв'язки між вузлами забезпечуються з'єднанням входів та виходів різних	Зв'язки між модулями реалізовані за допомогою аргументів (у кількості від 0 до 10) функцій та невеликої кількості глобальних змінних. Внутрішньо модульні зв'язки реалізовані за допомогою загальних для модуля змінних

елементами. (Відмінність між внутрішньо- та між елементними взаємодіями обумовлює поділ системи на абстрактні автономні частини, які можна вивчати окремо.)	вузлів. Вузол працює як «чорна скринька», внутрішньо елементні зв'язки якої «не видно» ззовні. Кількість внутрішньо елементних зв'язків суттєво більша, ніж між елементних	(зазвичай від 10 до кількох десятків). Оскільки змінні доступні з будь-якої точки модуля, такий зв'язок є зв'язком типу «все з усіма»
4. Ієрархічні системи зазвичай складаються з кількох підсистем різного типу, реалізованих у різному порядку та у різноманітних комбінаціях	Кожен з електронних вузлів пристрою виконаний, зрештою, з тих самих типових елементів: напівпровідникових приладів (транзисторів і діодів), опорів, конденсаторів різних номіналів і способів виготовлення. Розрізняються порядок та комбінації використання цих елементів у різних вузлах	Кожен модуль являє собою набір одних і тих же обчислювальних структур (операторів) і стандартних функцій, що по-різному взаємодіють один з одним через загальні дані в кожному з модулів
5. Складна система, що працює, неминуче виявляється результатом розвитку працюючої простої системи. Складна система, розроблена від початку до кінця на папері, ніколи не працює і не можна змусити її заробити. Зазвичай спочатку створюють просту працюючу систему, яку розвивають у наступних версіях на основі нових ідей, отриманих під час експлуатації	Прототипи музичного центру: радіо, касетний магнітофон, програвач компакт-дисків. Музичний центр є комбінацією та подальшим розвитком цих систем: покращено підсистеми посилення та фільтрації звуку, покращено динаміки, додано цифровий процесор для обробки звуку.	Спочатку з'явилися найпростіші текстові редактори як малі, так і екранні для набору та коригування текстів в режимі друкарської машинки. Потім з'явилися текстові процесори, що форматують текст та здійснюють перевірку орфографії. Далі з'явилися інтегровані системи, що включають процесори: текстові, графічні, електронних таблиць, баз даних та ділової графіки

Проектування— це розробка проекту, процес створення специфікації, яка потрібна на побудови в заданих умовах ще неіснуючого об'єкта з урахуванням первинного описи цього об'єкта. Результатом проектування є проектне рішення чи сукупність проектних рішень, які відповідають заданим вимогам. Задані вимоги обов'язково мають включати форму подання рішення.

Специфікаціяу сфері проектної діяльності - це якийсь опис у точних термінах.

Проектним документом називають документ, виконаний за заданою формою, у якому представлено будь-яке проектне рішення. У програмуванні проектні рішення оформлюються як програмної документації. Розрізняють зовнішню програмну документацію, яка узгоджується із замовником, та внутрішню проміжну документацію проекту, яка необхідна самим програмістам для їхньої роботи.

Проект(Від лат. Projectus - кинутий вперед) - сукупність проектних документів відповідно до встановленого переліку, що представляє результат проектування.

Проектною ситуацією називають реальність (ситуацію), у якій ведеться проектування. Паровоз та електровоз проектувалися у різних проектних ситуаціях, визначених рівнем знань людства. Саме тому в XIX ст. став віком паровоза.

Будь-яке завдання характеризується необхідністю перетворення деякої вихідної ситуації на ситуацію, звану рішенням. Говорячи про будь-яке завдання, завжди маємо її інформаційні елементи:

- інформація про умову (умова задачі) - що задано;
- інформація про рішення (ознаки вихідної ситуації) - що потрібно отримати;
— інформація про технологію перетворення умови на рішення — як вирішити.

Проектне завдання(англ. Engineering Task) характеризується невизначеністю апріорі інформації: що потрібно отримати, що поставлено. Більш того, спосіб розв'язання задачі є об'єктом проектування. І нарешті, розв'язання проектної задачі має бути знайдено в рамках обмежень зовнішнього середовища проектування: доступних коштів, заздалегідь заданих термінів, можливостей технічних засобів та інструментарію програмування, наукових знань, програмних заділів тощо.

Проектні завдання під силу тільки тим, хто здатний сприймати явище цілком і в найдрібніших деталях одночасно, дотепно пов'язуючи ці деталі між собою. Саме таких людей завжди називали інженерами, та й сам термін походить від латинського ingenium, що означає природний розум, а також винахідливість. Інженер-програміст - спеціаліст з вирішення проектних завдань. Інженер-системотехнік - інженер інженерів, фахівець із вирішення проектних завдань створення таких особливо складних штучних систем, як автоматизовані системи.

Джерелом, першопричиною будь-якої проектної діяльності є суб'єкт — людина чи група людей, які мають дискомфорт у існуючій ситуації.

«Лежачи на теплій печі» (перебуваючи у комфортній ситуації), можна мріяти про вирішення світових проблем і нічого не робити. Однак страх перед майбутнім дискомфортом (замерзання, голод) поверне мрійника в реальну ситуацію і вимагатиме знаходження способу вирішення і вирішення проблеми подальшого його існування (заготівля дров і продуктів).

Дискомфорт суб'єкта може бути конкретизований як потреби, задоволення якої знімає його. Для задоволення потреби потрібен певний об'єкт проектування (у разі програмний продукт), наявність якого, його і стан задовольняють потребам суб'єкта.

Відповівши питанням, які властивості має володіти об'єкт, підготуємо вихідні дані для наступного питання: як має бути влаштований об'єкт, щоб мати такі властивості? Для вирішення цього завдання також необхідно розкрити вихідну ситуацію. Причому таке розкриття потрібно різних рівнях конкретизації об'єкта. Отже, виходить, що процедура розкриття проектної ситуації може повторюватися багаторазово і різних етапах вирішення спільної проектної завдання. При цьому всі рішення взаємопов'язані: рішення, прийняті на одному етапі, мають бути враховані під час виконання інших. Якимось чином формалізувати і врахувати цей вплив важко, тому процес рішення має ітераційний характер.

Для задоволення потреби має бути реалізована деяка діяльність, кінцевим результатом якої (метою) і буде створення об'єкта та (або) приведення його в бажаний (цільовий) стан. Ця діяльність також є об'єктом, що вимагає проектування. Стосовно неї також має вирішуватися аналогічне завдання. Інакше висловлюючись, сам процес проектування є об'єктом проектування.

Метод(від грецьк. methodos — спосіб дослідження чи пізнання, теорія чи вчення) — прийом чи система прийомів практичного здійснення чогось у будь-якій предметної області, сукупність прийомів чи операцій практичного чи теоретичного освоєння дійсності, підпорядкованих вирішенню конкретних завдань. Метод включає засоби - за допомогою чого здійснюється дія та способи - яким чином здійснюється дія.

Методика (Від грец. methodike) - Упорядкована сукупність методів практичного виконання чого-небудь.

Методики проектування викладаються у вигляді описів проектних процедур та проектних операцій.

Під проектною процедурою розуміють формалізовану сукупність дій, виконання яких закінчується проектним рішенням. Наприклад, проектною процедурою є процедури розкриття проектної ситуації та розробки структури програми.

Дія чи формалізовану сукупність дій, що становлять частину проектної процедури, алгоритм яких залишається незмінним для низки проектних процедур, називають проектною операцією. Наприклад, креслення схеми, диференціювання функції.

Проектні процедури можуть включати інші проектні процедури тощо до проектних операцій. Проектні процедури можуть бути алгоритмами (тільки для тривіальних нетворчих операцій) і евроритми (якими викладаються евристичні операції).

Алгоритм- суворо однозначно визначена для виконавця послідовність дій, що призводять до вирішення завдань.

Сучасне значення слова «алгоритм» багато в чому аналогічно до таких понять, як рецепт, процес, методика, спосіб. Згідно з Д. Кнутом [17], алгоритм має п'ять важливих властивостей.

Кінцівка. Алгоритм завжди повинен закінчуватися після виконання кінцевого числа кроків.

Визначеність. Кожен крок алгоритму має бути точно визначено.
Наявність вхідних даних. Алгоритм має деяку кількість вхідних даних, що задаються до початку роботи або визначаються динамічно під час виконання.
Наявність вихідних даних. Алгоритм має одне або кілька вихідних даних, що мають певний зв'язок із вхідними даними.
Ефективність. Алгоритм зазвичай вважається ефективним, якщо його оператори досить прості для того, щоб їх можна було виконати за допомогою олівця і паперу протягом кінцевого проміжку часу.
Термін "евроритм" науки евристика утворений від легендарного вигуку Архімеда "Еврика!", що в перекладі з грецької означає "знайшов, відкрив". Алгоритм у процесі виконання не змінюється бездумним виконавцем (процесором). На відміну від алгоритму евроритм виконується людиною, яка мислить, яка може вдосконалити порядок своєї роботи в процесі її виконання. Евроритм може включати алгоритми. Наприклад, інструкція користування програмою — це евроритм, особливо якщо одні й самі дії можна виконати різними способами (через пункт меню чи натисканням кнопки).
Евристика наука, що розкриває природу розумових операцій людини під час вирішення конкретних завдань незалежно від своїх конкретного змісту. У вужчому сенсі евристика — це припущення, засновані на досвіді розв'язання родинних завдань.
Інженерія програмування(англ. software engineering, у термінах автоматизованих систем – розробка програмного забезпечення) – інженерна справа, творча технічна діяльність. Інженерія спирається на специфічні методи та методики, у тому числі евристичні. Інженерія [20] вивчає різні методи та інструментальні засоби з погляду певних цілей, тобто має очевидну практичну спрямованість. Основна ідея інженерії програмування полягає в тому, що розробка програмного забезпечення є формальним процесом, який можна вивчати, виражати в методиках та вдосконалювати.
Інженерія програмування має чітку певну дату народження — 1968 р. Причина її появи — реакція на так звану «кризу програмного забезпечення», викликану досягненням непереборного рівня складності. Характерні питання та завдання інженерії програмування, викладені Фредеріком Бруксом, актуальні й до сьогодні.
Як проектувати та будувати програми, що утворюють системи?
Як проектувати та будувати програми та системи, які є надійним, налагодженим, документованим та супроводжуваним продуктом?
Як здійснювати інтелектуальний контроль за умов великої складності?
Інженерна діяльність базується на сукупності загальнонаукових методів системного підходу, аналітико-синтетичному методі блочно-ієрархічного підходу до проектування складних систем, аналітико-синтетичному методі стадії та етапи розробки. Додатково інженери використовують методи та методики, спеціалізовані стосовно об'єкта проектування або виготовлення.
Важливими розробки процесів проектування є такі поняття, як стратегія і тактика.
Стратегія(від грец. Stratos - військо і ago - веду) - наука, мистецтво генерації найбільш істотних загальних довгострокових цілей і найбільш загального плану досягнення переваги, курсу дій та розподілу ресурсів ще до виконання реальних дій.
Тактика(від грецьк. taktika — мистецтво упорядковувати) — фіксована у своїй послідовності сукупність засобів і прийомів задля досягнення наміченої мети і мистецтво її застосування, способи дії, зорієнтовані досягнення конкретних цілей, є ланками реалізації стратегічних цілей. Метою застосування способу дії є здійснення оптимальних дій, заздалегідь не передбачених стратегічним планом ситуаціях, вже у процесі виконання реальних дій.
Стратегія охоплює теорію і практику підготовки до виконання проекту, а також загальне планування тактик ведення проектів. Стратегія визначає, куди, в якому напрямку рухатись, куди тримати курс ще до початку проекту. А тактика визначає, як, яким способом рухатися, які конкретні дії робити при труднощі під час виконання проекту.
Стратегія виконання конкретного проекту описується у програмному документі – технічному завданні.
Методологія(Від грецьк. methodos і logos - слово, вчення про методи) - система принципів і способів організації та побудови теоретичної та практичної діяльності, а також вчення про цю систему.

Методологія програмування вивчає методи з погляду основ побудови. Це об'єднана єдиним філософським підходом сукупність методів, що застосовуються у процесі розробки програмних продуктів. Будь-яка методологія створюється на основі вже накопичених у предметній галузі емпіричних фактів та практичних результатів.

Технологія(від грец. techne - мистецтво, майстерність, уміння і logos - слово, вчення) - сукупність виробничих процесів у певній галузі виробництва, а також науковий опис способів виробництва, сукупність прийомів, що застосовуються у будь-якій справі, майстерності, мистецтві.

Сучасна методологія проектування дозволила довести методи проектування до технологій із набором методик.

Технологія програмування як наука вивчає технологічні процеси та порядок їх проходження (з використанням знань, методів та засобів). Технологічний процес - послідовність спрямованих на створення заданого об'єкта дій (технологічних процедур і операцій), кожна з яких заснована на будь-яких природних процесах і людської діяльності. Знання, методи та засоби можуть використовуватись у різних процесах і, отже, у технологіях.

Технологія програмування— для інженера це наукова та практично апробована стратегія створення програм, що містить опис сукупності методів та засобів розробки програм, а також порядок застосування цих методів та засобів.

У реальних проектних ситуаціях потрібний синтез раціональної стратегії кожного конкретного проекту. Інженери часто цей синтез здійснюють на основі однієї, двох та навіть трьох технологій. Інженерна справа (діяльність зі створення та використання технологій) охоплює не тільки проектування та виробництво, а й структури організацій із взаємодією людей. У термінах інженерного відносини технологія — інструментарій інженера, інтелектуальний чи відчужений як штучних систем.

Головна різниця між технологією програмування та програмною інженерією полягає у способі розгляду та систематизації матеріалу. У технології програмування акцент робиться на вивченні процесів розробки програм (технологічних процесів) у порядку їх проходження – методи та інструментальні засоби розробки програм використовуються у цих процесах (їх застосування та утворюють технологічні процеси). У програмній інженерії вивчаються насамперед методи та інструментальні засоби розробки програм з погляду досягнення певних цілей — вони можуть використовуватись у різних технологічних процесах (і у різних технологіях програмування). Як ці методи та засоби утворюють технологічні процеси — питання другорядне.

Не слід плутати технологію програмування з методологією програмування, хоча у обох випадках вивчаються методи. У технології програмування методи розглядаються «згори» — з погляду організації технологічних процесів, методології — «знизу» — з погляду основ їх побудови.

Перед нами стоїть питання, яке вже згадувалося: «Як визначити, чи досягнута мета, чи привела діяльність до бажаного результату: чи той об'єкт створений, який нам потрібен, чи має потрібні властивості?». І тому використовуються показники якості (іноді їх називають критеріями) — величини, властивості, поняття, що характеризують систему з погляду суб'єкта, дозволяють оцінити рівень задоволення його потреб. На початковому етапі проектування аналіз потреб дозволяє визначити вид об'єкта; його функції (що об'єкт виконує при функціонуванні), властивості та стан, у яких задовольняються потреби, і навіть якість об'єкта.

При дослідженні систем вирішуються завдання аналізу та синтезу.

Аналіз(Від грец. Analysis - розкладання, розчленування) - прийом розумової діяльності, пов'язаний з уявним (або реальним) розчленуванням на частини предмета, явища або процесу.

Синтез (від грец. synthesis - поєднання, поєднання, складання) - метод наукового дослідження явищ дійсності в їх єдності та цілісності, у взаємодії їх частин, узагальнення, зведення в єдине ціле.

Синтез нерозривно пов'язані з аналізом і немає окремо від цього, і навіть пов'язані з іншими розумовими процесами. Без синтезу неможливе виконання процедур узагальнення, систематизації, порівняння (вибору), разом із якими він залишає логічний апарат мислення.

Теоретично проектування використовуються такі поняття аналізу та синтезу.

Аналіз -процес визначення функціонування за заданим описом системи.

Синтез- Процес побудови опису системи по заданому функціонуванню.

Одним з визначень задачі оптимізації розробки програм є знаходження розумного компромісу між досягнутою метою та витраченими на це ресурсами, що може призводити як до перегляду цілей розробки, так і до зміни ліміту ресурсів. Вирішення цього завдання особливо важливе, оскільки

програми є дорогим продуктом, і, правильно провівши розробку, можна досягти значного її здешевлення.
За своєю природою програма (тобто набір інструкцій) набагато ближче до технології (точніше, до опису технологічного процесу перетворення вхідної інформації у вихідну інформацію), ніж виробу. Це означає, що з оцінки продуктивність праці програміста не потрібно шукати спосіб оцінки кількості продукції, яку він випускає, оскільки ніяка фізична продукція немає і, отже, немає її обсягу. При використанні стандартного терміна «програмний виріб» виникають методологічні, правові та суто технічні складності. Так, наприклад, програмісту не складе ніяких труднощів вставити в програму будь-яку кількість модулів з будь-яким обсягом незадіяних операторів. Товарні властивості програмного продукту – не товарні властивості цього виробу, а товарні властивості технології. Причому технології це теж об'єкти, які традиційно проектуються інженерами.
Програмний продукт є розробленою програмістом інформаційною технологією, яка матеріалізується у замовника у вигляді виробу, стаючи автоматизованими системами та інструментами їх обслуговування. Це пояснення, мабуть, знімає багато правових проблем, а також проблеми ціноутворення.

1.2. ЗАГАЛЬНІ ПРИНЦИПИ РОЗРОБКИ ПРОГРАМ

Програми розрізняються за призначенням, виконуваним функцій, форм реалізації. Однак можна вважати, що існують деякі загальні принципи, які слід використовувати для розробки програм.
Частотний принцип. Принцип заснований на виділенні в алгоритмах та даних спеціальних груп за частотою використання. Для дій, що найчастіше зустрічаються під час роботи програм, створюються умови для їх швидкого виконання. До даних, що часто використовуються, забезпечується найбільш швидкий доступ. «Часті» операції намагаються робити коротшими. Слід зазначити, що не більше 5% операторів програми надають відчутний вплив на швидкість виконання програми. Цей факт дозволяє значну частину операторів програми кодувати без урахування швидкості обчислень, звертаючи основну увагу при цьому на «красу» та наочність текстів.
Принцип модульності. Під модулем в даному контексті розуміють функціональний елемент системи, що розглядається, має оформлення, закінчене і виконане в межах вимог системи, і засоби сполучення з подібними елементами або елементами вищого рівня даної або іншої системи. Методи відокремлення складових програм в окремі модулі можуть відрізнятися значно. Значною мірою поділ системи на модулі визначається методом проектування програм, що використовується.
Принцип функціональної вибірковості. Цей принцип є логічним продовженням частотного та модульного принципів та використовується при проектуванні програм. У програмах виділяється деяка частина важливих модулів, які мають бути готові для ефективної організації обчислювального процесу. Цю частину у програмах називають ядром чи монітором. При формуванні складу монітора потрібно врахувати дві суперечливі вимоги. До складу монітора, крім чисто керуючих модулів, повинні увійти модулі, що найчастіше використовуються. Кількість модулів повинна бути такою, щоб обсяг пам'яті, яку займає монітор, був не надто великим. Програми, що входять до складу монітора постійно зберігаються в оперативній пам'яті. Інші частини програм постійно зберігаються у зовнішніх пристроях, що запам'ятовують, і завантажуються в оперативну пам'ять тільки при необхідності, перекриваючи один одного також при необхідності.
Принцип генерованості. Основне положення цього принципу визначає такий спосіб вихідного представлення програми, який би дозволяв здійснювати налаштування на конкретну конфігурацію технічних засобів, коло проблем, умови роботи користувача.
Принцип функціональної надмірності. Цей принцип враховує можливість проведення однієї й тієї роботи різними засобами. Особливо важливий облік цього принципу при розробці інтерфейсу для видачі одних і тих же даних різними способами виклику через психологічні відмінності в сприйнятті інформації.
Принцип "за замовчуванням". Застосовується полегшення організації зв'язків із системою як у стадії генерації, і під час роботи з вже готовими програмами. Принцип заснований на зберіганні в системі деяких базових описів структур, модулів, конфігурацій обладнання та даних, що

визначають умови роботи із програмою. Цю інформацію програма використовує як задану за умовчанням, якщо користувач забуде або свідомо не конкретизує її.

1.3. СИСТЕМНИЙ ПІДХІД І ПРОГРАМУВАННЯ

Системний підхід- загальнонауковий узагальнений евроритм, що передбачає всебічне дослідження складного об'єкта з використанням компонентного, структурного, функціонального, параметричного та генетичного видів аналізу.
Компонентний аналіз- Розгляд об'єкта, що включає в себе складові елементи і входить, у свою чергу, в систему вищого рангу.
Структурний аналіз виявлення елементів об'єкта та зв'язків між ними.
Функціональний аналіз- Розгляд об'єкта як комплексу виконуваних ним корисних і шкідливих функцій.
Параметричний аналіз- встановлення якісних меж розвитку об'єкта - фізичних, економічних, екологічних та ін. Що стосується програм параметрами можуть бути: час виконання якого-небудь алгоритму, розмір займаної пам'яті і т. д. При цьому виявляються ключові технічні протиріччя, що заважають подальшому розвитку об'єкта, та ставиться завдання їх усунення з допомогою нових технічних рішень.
Генетичний аналіз дослідження об'єкта з його відповідність законам розвитку програмних систем. У процесі аналізу вивчається історія розвитку (генеза) досліджуваного об'єкта: конструкції аналогів та можливих частин, технології виготовлення, обсяги тиражування, мови програмування тощо.
При блочно-ієрархічному підході (приватному евроритмі системного підходу, який використовується часто в техніці та програмуванні) процес проектування та уявлення про об'єкт розчленовується на рівні. На найвищому рівні використовується найменш детальне уявлення, що відображає найзагальніші риси та особливості проектованої системи. На кожному новому послідовному рівні розробки рівень подробиці розгляду зростає, у своїй система розглядається над цілому, а окремими блоками.
Методологія блочно-ієрархічного підходу базується на трьох концепціях: розбиття та локальної оптимізації, абстрагування, повторюваності.
Концепція розбиття дозволяє складне завдання проектування об'єкта чи системи звести до вирішення більш простих завдань з урахуванням їхнього взаємозв'язку.
Локальна оптимізація передбачає поліпшення параметрів усередині кожного простого завдання.
Абстрагованість полягає у побудові моделей, що відображають лише значущі в цих умовах властивості об'єктів.
Повторюваність- У використанні існуючого досвіду проектування.
Блочно-ієрархічний підхід дозволяє кожному рівні вирішувати завдання прийнятної складності. Розбиття на блоки має бути таким, щоб документація на будь-якому рівні була доступна для огляду та сприймання однією людиною.
Головним недоліком блочно-ієрархічного підходу і те, що у верхніх рівнях мають справу з неточними моделями об'єкта, і рішення приймаються за умов недостатньої інформації. Отже, при цьому підході висока ймовірність проектних помилок.

1.4. ЗАГАЛЬНОСИСТЕМНІ ПРИНЦИПИ СТВОРЕННЯ ПРОГРАМ

При створенні та розвитку програмного забезпечення (ПЗ) рекомендується застосовувати такі загальносистемні принципи:
1) принцип включення, що передбачає, що вимоги до створення, функціонування та розвитку ПЗ визначаються з боку більш складної, що включає його в себе системи;
2) принцип системної єдності, що полягає в тому, що на всіх стадіях створення, функціонування та розвитку ПЗ його цілісність забезпечуватиметься зв'язками між підсистемами, а також функціонуванням підсистеми управління;
3) принцип розвитку, що передбачає в ПЗ можливість його нарощування та вдосконалення компонентів та зв'язків між ними;
4) принцип комплексності, що полягає в тому, що ПЗ забезпечує пов'язаність обробки інформації як окремих елементів, так і всього обсягу даних в цілому на всіх стадіях обробки;

5) принцип інформаційної єдності, тобто у всіх, підсистемах, засобах забезпечення та компонентах ПЗ використовуються єдині терміни, символи, умовні позначення та способи подання;
6) принцип сумісності, що полягає в тому, що мова, символи, коди та засоби програмного забезпечення узгоджені, забезпечують спільне функціонування всіх підсистем та зберігають відкриту структуру системи в цілому;
7) принцип інваріантності, що визначає, що підсистеми та компоненти ПЗ інваріантні до оброблюваної інформації, тобто є універсальними або типовими.

1.5. ОСОБЛИВОСТІ ПРОГРАМНИХ РОЗРОБОК

Томас Кун 1977 р. визначив термін «парадигма» як зведення норм наукового мислення. Парадигма це правило (modus operandi) розвитку наукового знання. Воно протягом певного часу дає науковій спільноті модель постановки проблем та їх вирішення.
Коли та чи інша методологія застосовується під час стадії кодування (реалізації), часто її називають парадигмою програмування — способом мислення у програмуванні.
У програмуванні існують різні концепції мов (парадигми), які при написанні програм можуть призводити як до тих самих, і радикально різним підходам. Понад те, для низки мов необхідний «свій» тип мислення, особливі технології розробки, спеціальна школа навчання. Більшість програмістів використовують у роботі одну-дві мови програмування у межах однієї парадигми. Іноді програмісту буває важко зрозуміти чиюсь програму, реалізовану в незвичній йому парадигмі. На противагу зміні мети проекту під мову, що використовується, у ряді проектних випадків раціонально обрати іншу мову програмування.
Прийоми та способи програмування конкретного програміста визначаються мовою, що використовується. Часто осторонь залишаються альтернативні підходи до мети, отже, не використовуються оптимальні рішення у виборі парадигми, відповідної задачі. Нижче наведено список основних парадигм програмування разом із властивими їм видами абстракцій:
- Процедурно-орієнтовані - алгоритми;
- Об'єктно-орієнтовані - класи та об'єкти;
- Логічно-орієнтовані - цілі, виражені в обчисленні предикатів;
— орієнтовані правила — правила «якщо..., то...»;
— орієнтовані обмеження — інваріантні співвідношення;
- Паралельне програмування - потоки даних. Існують та інші парадигми. Чому ж їх стільки?
Почасти тому, що програмування – порівняно нова
дисципліна, а частково через бажання людей вирішувати різні завдання. Крім того, найпопулярніша в даний момент комп'ютерна архітектура не є єдиною. В даний час проводиться велика кількість експериментів з машинами, що мають нестандартні архітектури, багато з яких розраховані на застосування інших парадигм програмування, наприклад, числа Фібоначчі. Загальна природа цифрових машин дозволяє з більшою чи меншою ефективністю моделювати одну архітектуру за допомогою іншої. З архітектур найбільш вдалі ті, у яких за рахунок апаратури та програмного забезпечення досягнуто найвищої швидкості та простоти використання.
Неможливо назвати будь-яку парадигму найкращою у всіх галузях практичного застосування. Наприклад, для проектування баз знань більш придатна парадигма, орієнтована правила. Об'єктно-орієнтована парадигма є найбільш прийнятною для кола завдань, пов'язаних з великими промисловими системами, в яких основною проблемою є складність.

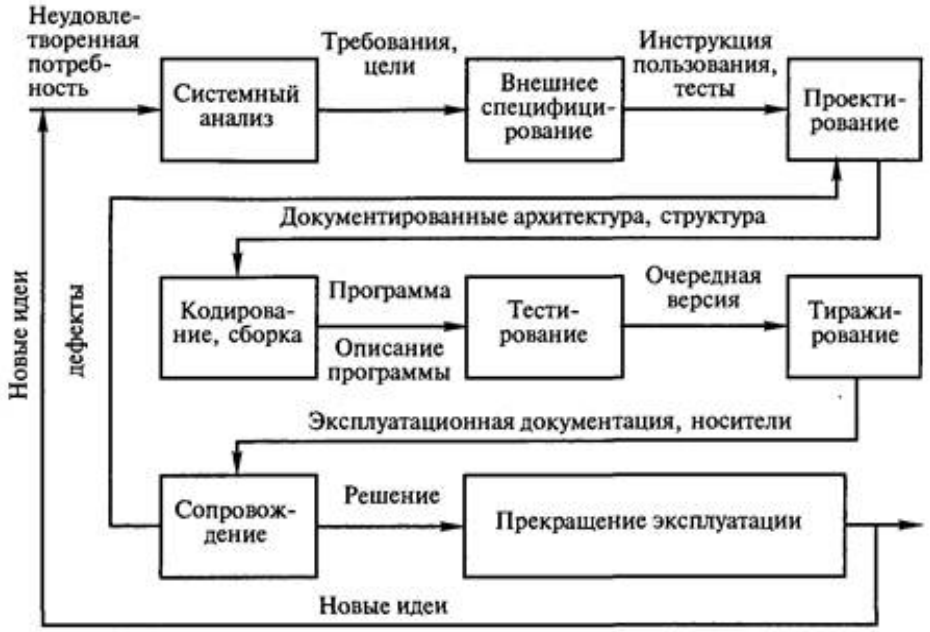
1.6. СТАНДАРТИ І ПРОГРАМУВАННЯ

Стандарти давно використовуються в техніці та програмуванні. Створення складної системи неможливо без стандартів. Вони потрібні для боротьби з хаосом і плутаниною, але водночас стандарт не повинен бути надто «вузьким» і заважати технічному прогресу.
Державні стандарти відстежують тенденції розвитку програмування та дають обов'язкові рекомендації щодо їх дотримання. Крім державних стандартів (ГОСТ), діють галузеві стандарти (ОСТ), стандарти підприємств (СТП).
Група стандартів ДЕРЖСТАНДАРТ «Єдина система програмної документації» (ЄСПД) зазнала мало змін з моменту її створення, пережила кілька поколінь ЕОМ та революційних змін технологій розробки програм. При цьому вона досі ніколи не ускладнювала новацій.

Крім вищевикладених стандартів де-юре є стандарти де-факто. Ряд стандартів встановлюється де-факто провідними фірмами-розробниками програм та обчислювальної техніки. Стандарти де-факто з'являються з урахуванням ідей якоїсь широко відомої розробки. Вигідно робити продукти у стилі розробки якоїсь фірми, оскільки користувачі вже мають навички роботи з меню у стилі Lotus, електронними таблицями, текстовими редакторами. Зазвичай стандартом де-факто визначаються операційні системи, транслятори з мов програмування, організація файлів і середній рівень якості, що досягається після закінчення тестування програм. Конкретному розробнику вигідно дотримуватися таких стандартів.
У галузі програмування загальновизнаною провідною організацією з розробки стандартів є ANSI (Американський національний інститут стандартів). Цей інститут є лідером із встановлення стандартів мов програмування, кодових таблиць клавіш і символів, що виводяться на екран, та ще багатьох інших. Необхідно також наголосити на стандартах ISO.
На жаль, найблагородніша справа стандартизації — досягнення загальної уніфікації та взаємозамінності — може стати гальмом розвитку. Вводячи новий стандарт, треба враховувати наслідки введення, особливо якщо стандарт є випереджаючим та випереджає практику розвитку або якщо стандарт є надто «вузьким» і гальмує еволюцію прогресу. В усьому світі керуються наступним ставленням до стандартів: або повністю за ними, або роби свій власний стандарт. Стандарти надають додаткові обмеження.
Програміст повинен вміти як використовувати готові стандарти, а й розробляти нові. Так, наприклад, правила однотипного оформлення вихідного тексту програми визначаються стандартом проекту, який може бути змінений на початку розробки нового проекту. Однак протягом виконання одного проекту оформлення всіх частин програми має бути однотипним. Тому найчастіше перед початком нового проекту конкретним програмістам слід розробляти свої стандарти, які не порушують ГОСТ, ОСТ та СТП та діють у межах конкретного проекту.

1.7. ОПИС ЦИКЛУ ЖИТТЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Програми створюються, експлуатуються та розвиваються в часі. Як і будь-які штучні системи вони мають свій життєвий цикл.
*Життєвий цикл*сукупність взаємозалежних процесів створення та послідовної зміни стану продукції від формування до неї вихідних вимог до закінчення її експлуатації чи споживання. Програми з мінімальною тривалістю життєвого циклу створюються для разового вирішення наукових та інших завдань. Їхній життєвий цикл — від кількох днів до кількох місяців. Раніше такі програми не мали зручного інтерфейсу, оскільки витрати на його розробку ще недавно в кілька разів перевершували витрати на розробку обчислювальної частини. Життєвий цикл програмних виробів показано на рис. 1.1.



Мал. 1.1. Життєвий цикл програмних виробів

Кожна програма починається з якоїсь незадоволеної потреби і, усвідомивши її, необхідно провести системний аналіз для виявлення цілей майбутнього програмного виробу та вимог до нього. Наступним етапом буде зовнішня специфікація, призначена для створення «ідеології» програми — загальної спрямованості у подальшому проектуванні, аж до зовнішнього вигляду програми та інструкції користування програмою. На етапі проектування програмний виріб специфікується у повному обсязі від постановки завдання до робочого проекту з описом внутрішньої структури програми та плану розробки частин програми. Потім відбувається кодування та тестування, внаслідок чого виходить готова версія програми. Програма випускається у тираж та супроводжується виробником. Супровід полягає як в усуненні помилок і випуску виправлених версій, що виявляються в процесі експлуатації, так і в удосконаленні базової версії програми, що часто призводить до перепроектування програми та випуску радикально оновлених версій. Закінчення життєвого циклу зумовлюється припиненням експлуатації розробки. Однак ідеї, висунуті в процесі експлуатації програми, зазвичай використовуються при розробці наступного, більш досконалого та сучасного виробу.
Припинення експлуатації зазвичай не одномоментний акт знищення програми в комп'ютері, а період часу, коли деякі організації або деякі користувачі ще продовжують використовувати стару розробку.

1.8. СТАДІЇ ТА ЕТАПИ РОЗРОБКИ ПРОГРАМ

ГОСТ 19.102-77 регламентує стадії та етапи програмних розробок протягом усього життєвого циклу. Даний стандарт сформувався на основі аналізу вдалих та невдалих програмних розробок та містить основні рекомендації щодо проведення нових розробок. Стандарт уже пережив кілька технологій програмування. При цьому, практично не змінюючись, він не був гальмом прогресу. Крім найменувань стадій та етапів проектування ГОСТ 19.102-77 фактично містить опис аналітико-синтетичного евроритму (алгоритма дій проектувальника з використанням методів аналізу та синтезу) за тимчасовими етапами проекту.
Стадія проекту- Одна з частин процесу створення програми, встановлена нормативними документами і закінчується випуском проектної документації, що містить опис повної, в рамках заданих вимог, моделі програми на заданому для даної стадії рівні або виготовленням програм. По досягненні закінчення стадії замовник має можливість розглянути стан проекту та прийняти рішення щодо подальшого продовження проектних робіт. Наприклад, замовник може ухвалити рішення про продовження робіт за одним із узгоджених варіантів.
Етап проекту зазвичай частина стадії проекту, виділена з міркувань єдності характеру робіт і (або) завершального результату або спеціалізації виконавців. Іноді виділяють етапи (фази), що охоплюють кілька стадій. Наприклад, етап проектування програми включає стадії ЕП та ТП. Описи етапів регламентують порядок виконання окремих видів робіт задля досягнення стадії. Одні й самі види робіт можуть тривати низку етапів.
Стадії та етапи розробки програм за ГОСТ 19.102-77 на момент написання книги дано у додатку 1.
Програмний документ «Технічне завдання» (ТЗ) крім основних вимог до програмного виробу містить проект порядку взаємодії замовника та виконавця після закінчення конкретних етапів, тобто перелік необхідних стадій та етапів та вимог до їх виконання. ТЗ може одразу не встановлювати всіх вимог, які можуть бути уточнені та узгоджені із замовником на наступних стадіях. Проте сама можливість зміни вимог має закладатися у ТЗ. У ТЗ визначається стратегія виконання проекту.
"Ескізний проект" (ЕП), як правило, необхідний для розробки кількох альтернативних варіантів реалізації майбутнього виробу та уточнення вимог на основі їх аналізу. Ступінь опрацювання при цьому має бути достатньою лише для досягнення можливості порівняння варіантів.
"Технічний проект" (ТП) виконується для отримання однозначного опису кінцевого (оптимального) варіанта побудови програмного виробу та порядку його реалізації.
«Робочий проект» (РП) необхідний реалізації виробу відповідно до раніше наміченим планом.
Стадія "Впровадження" необхідна для розмноження програмної документації в потрібній кількості, навчання користувачів, допомоги в освоєнні програми, супроводження програми.

Науково-дослідницька робота (НДР) може бути самостійним етапом. НДР переважно проводиться виявлення останніх наукових досягнень з метою їх використання у проекті, перевірки реалізованості вироби та уточнення окремих його характеристик.
Відповідно до ГОСТ 19.102-77 допускається виключати стадію ЕП, а в технічно обґрунтованих випадках - стадії ЕП та ТП. Допускається об'єднувати, виключати етапи робіт та (або) їх зміст, а також запроваджувати інші етапи робіт за погодженням із замовником. Це дозволяє розумно збудувати проект конкретної розробки (хід проекту також є об'єктом проектування).
*приклад 1.*Розробка наукомісткої підпрограми може вестись за такими стадіями:
• ТЗ (ТЗ головне плюс ТЗ на окрему НДР);
• очікування на результати НДР, що виконується в іншій організації фахівцями-математиками (термін від місяця до декількох років);
• РП (близько місяця);
• Використання.
*приклад 2.*Потрібно розробити програмний виріб середньої чи великої складності. При середній складності виробу необхідне проведення ТП, а за великої складності - ЕП і ТП. На відміну від прикладу 1, у цьому випадку ТЗ може не містити закінчених вимог.
*приклад 3.*Потрібно створити програмні засоби, які автоматизують окремі види робіт. Розробка такого проекту може проводитись за такими стадіями:
• ТЗ;
• ЕП з НДР щодо дослідження існуючих програмних засобів, що автоматизують виконання окремих видів робіт;
• РП з розробки лише документації без реалізації будь-яких програм, якщо НДР показала, що можна обійтися лише існуючими програмними засобами;
• Використання.
*приклад 4.*Розробка таких інформаційних систем, як САПР або АСУ, повинна здійснюватися відповідно до відповідних стандартів. ТП САПР чи АСУ може містити технічні завдання розробки окремих програмних виробів. Зазвичай, такі ТЗ дуже конкретні. На етапі РП САПР або АСУ спочатку ведеться контроль над розробкою програмних виробів за всіма необхідними для цього стадіями розробки програмних виробів, потім проводиться спільна перевірка всіх розроблених програм.
Зазвичай підставою для укладення договору між замовником та виконавцем є гарантійний лист замовника. З гарантійного листа укладається договір. Обов'язковим додатком договору є ТЗ.
Деякі вітчизняні та зарубіжні джерела пропонують виділяти такі етапи:
1) аналіз вимог до системи (системний аналіз). (Зазвичай проводиться на основі первинного дослідження потоків інформації при традиційному проведенні робіт з фіксацією видів цих робіт та їх послідовності.);
2) визначення цілей, що досягаються розроблюваними програмами;
3) виявлення аналогів, які забезпечують досягнення подібних цілей, їх переваг та недоліків;
4) постановка задачі на розробку нових програм, визначення зовнішніх специфікацій (тобто описів вхідної та вихідної інформації, а іноді та їх форм) та способів (алгоритмів, методів) обробки інформації;
5) оцінка досягнення цілей розробки (Далі, за необхідності, етапи 1-5 можуть бути ітеративно повторені до досягнення задовільного вигляду виробу з описом виконуваних ним функцій та деякою ясністю реалізації його функціонування.);
6) розгляд можливих варіантів структурної побудови програмного виробу: або у вигляді кількох програм, або кількох частин однієї програми; результатом цієї роботи є варіанти архітектури програмної системи та (або) вимоги до структури окремих програмних компонентів; організація файлів для між програмного обміну даними;
7) розробка остаточного варіанта архітектури системи та розробка остаточної структури програмних компонентів;
8) складання та перевірка специфікацій модулів;
9) складання описів логіки модулів;
10) складання остаточного плану реалізації програм;
11) кодування та тестування окремих модулів та сукупності готових модулів до отримання готової програми;
12) комплексне тестування;

13) розробка експлуатаційної документації на програму;
14) проведення приймально-здавальних та інших випробувань;
15) коригування програм за результатами випробувань;
16) остаточне здавання програмного виробу замовнику;
17) тиражування програмного виробу;
18) супровід програми.
Сучасні технології проектування програмного забезпечення (ПЗ) спрямовані на часткову автоматизацію етапів і суміщення їх у часі з метою скорочення термінів виконання проектів.
У літературних джерелах застосовуються найменування етапів, які охоплюють ряд наведених етапів і за часом охоплюють кілька стадій. Наприклад, етап розробки програми.

1.9. ТИПОВІ ПОМИЛКИ НАВЧАЛЬНИХ ПРИ СКЛАДАННІ ТЕХНІЧНОГО ЗАВДАННЯ

У додатку 2 наведено приклад виконання навчального технічного завдання. У прикладі опущені: лист затвердження, титульний лист та додатки.
Головним, що відрізняє одне ТЗ від іншого, є сенс вимог. Усі вимоги оформлюються пропозиціями з використанням оборотів «має», «потрібно забезпечити», «необхідно виконати». Рекомендується оформляти вимоги у формі нумерованих абзаців. Це дозволить давати на них посилання.
Типовими помилками є написання неконкретних вимог, які нікого нічого не зобов'язують. Вимоги не повинні бути суперечливими. Тому дублювання тих самих за змістом вимог різними словами в різних місцях тексту часто призводить до неможливості реалізації виробу через неоднозначність їх розуміння замовником та виконавцем.
Нижче наведено типові помилки учнів при написанні вимог до функціональних характеристик:
• нерозуміння терміну «функціональні характеристики» (Сенс цього терміну характеризується такою пропозицією: «Проектований завод у нормальному режимі роботи повинен забезпечити випуск за одну 8-годинну зміну не менше 40 просапних тракторів». Ця фраза містить призначення проектованого об'єкта — заводу, так і те, що випускає завод і при яких обмеженнях стосовно програм, для правильного написання вимог до функціональних характеристик необхідно сторонніми очима майбутнього користувача розглянути, що робить корисний програмний виріб і при яких обмеженнях.);
• опис вимог до функціональних характеристик загального, універсального об'єкта (якщо за конкретними вимогами можна реалізувати цілий спектр виробів, вони не конкретні);
• написання вимог, що свідомо не реалізуються.
У наведеному у додатку 2 прикладі виконання навчального технічного завдання відсутні програми. Саме технічне завдання містить вимоги до дуже простого програмного виробу, тому ряд розділів технічного завдання написані як складнішого виробу.

1.10. МОДЕЛЮВАННЯ І ПРОГРАМУВАННЯ. ПОНЯТТЯ СПЕЦИФІКАЦІЙ

Один об'єкт чи система може у ролі моделі іншого об'єкта чи системи, якщо з-поміж них встановлено схожість у сенсі. Модель системи (або будь-якого іншого об'єкта або явища) може бути формальний опис системи, в якому виділені основні об'єкти, що становлять систему, і відносини між цими об'єктами.
Побудова моделей - поширений спосіб вивчення складних об'єктів і явищ. У моделі опущені численні деталі, що ускладнюють розуміння. Моделювання широко поширене у науці, техніці та програмуванні.
Моделі допомагають перевірити працездатність системи, що розробляється на ранніх етапах її розробки; спілкуватися із замовником системи, уточнюючи його вимоги до системи; вносити (у разі потреби) зміни у проект системи (як на початку її проектування, так і на інших фазах її життєвого циклу); передавати проект іншим виконавцям, наприклад кодувальникам, оскільки сам проект є моделлю проектованої програми.
Збираючись зробити прибудову до будинку, ви, мабуть, не почнете з того, що купите купу дощок і збиватимете їх разом різними способами, поки не отримаєте щось приблизно підходяще. Для роботи вам знадобляться проекти та схеми, так що ви швидше за все почнете з планування та

структурування прибудови. Зверніть увагу, що ваш результат буде довговічнішим. Ви не хочете, щоб робота зруйнувалася від невеликого дощу.
У світі програмного забезпечення те саме роблять для нас моделі. Вони складають проекти систем. Проект будинку дозволяє планувати його до початку безпосередньої роботи, модель дозволяє спланувати систему до того, як ви приступите до її створення. Це гарантує, що проект вдасться, вимоги буде враховано і система зможе витримати ураган наступних змін.
Будь-які досить великі програми є складними системами. Проблема складності долається шляхом декомпозиції задачі. Базова парадигма у підході до будь-якої великої задачі ясна: необхідно «розділяти та панувати».
У разі декомпозиції використовується абстрагування для отримання абстрактних моделей. Абстракція - уявне відволікання, відокремлення від тих чи інших сторін, властивостей або зв'язків предметів і явищ для виявлення суттєвих ознак. Абстрагування від проблеми передбачає ігнорування ряду подробиць для того, щоб звести завдання до більш простого завдання.
Мозок людини оперує даними через асоціації, створюючи павутину з ланцюжків, у яких залучені клітини мозку. Характерною особливістю людського мислення є мала, без особливих тренувань можливість абстрагування від предметів навколишнього світу, т. е. якщо людина справляє дію, він зазвичай робить його над конкретним предметом.
Окремі види абстракцій визначають найбільш ефективний спосіб декомпозиції стосовно конкретних цілей. Правильно впізнавши абстракції, можна отримати моделі, доступні розуміння як окремими людьми, і колективом учасників проекту.
Завдання абстрагування та подальшої декомпозиції типові для процесу створення програм. Декомпозиція використовується для розбиття програми на компоненти, які можуть бути об'єднані, дозволяючи вирішити основне завдання. Абстрагування ж передбачає продуманий вибір моделей майбутніх компонентів.
Кожна стадія проекту завершується затвердженням програмних документів. Виняток може становити розробка нехитрих програм з коротким (до півроку) життєвим циклом та з трудомісткістю не більше одного людино-місяця. Повне документування таких програм є економічно недоцільним. Програмна документація складається з урахуванням розроблених у ході проекту специфікацій.
Під специфікацією розуміється досить повний і точний опис розв'язуваної задачі на етапах проекту. Специфікація є моделлю проектованого об'єкта (програми).
Специфікація може описувати угоду між програмістами та замовниками (користувачами). Програміст береться написати програму, а користувач погоджується не покладатися знання у тому, як саме ця програма реалізована, т. е. не припускати нічого такого, що було зазначено у специфікації. Така угода дозволяє розділити аналіз реалізації від використання програми. Специфікації дають можливість створювати логічні засади, що дозволяють успішно «розділяти та панувати».
Складність проектованих систем призвела до створення спеціальних абстрактних мов, графічних нотацій і автоматизованих систем, що підтримують їх, що полегшують процес створення специфікацій.
Первинні специфікації становлять термінах розв'язуваної завдання, а чи не програми. У ході виконання проекту специфікації послідовно зазнають змін до програмних документів стадій і до документації, яка необхідна для експлуатації та супроводу програми.
Вже в первинних специфікаціях можна виділити дві частини: функціональну та експлуатаційну.
*Первинна функціональна специфікація*насамперед описує:
- Об'єкти, що беруть участь у завданні (що робить програма і що робить людина, яка працює з цією програмою);
- процеси та дії - евроритми для людини, алгоритми методів вирішення задачі в машині із зазначенням суті та порядку обробки інформації (з займаною інформацією та програмою розміром оперативної пам'яті);
- Вхідні та вихідні дані, їх організацію (наприклад, сценарій діалогу з екранними формами, організація файлів із зазначенням довжин полів записів та граничної кількості інформації у файлах).
Тобто, спочатку фіксуються зовнішні функціональні специфікації, а потім і внутрішні.
Загалом зовнішні функціональні специфікації включають:
- Опис того, що робить програма;

- визначення, що робить людина, а що машина (за якими єврористмами працює людина, звідки вона бере інформацію і як її готує до введення в ЕОМ);
- Специфікації вхідних та вихідних даних;
- Реакції на виняткові ситуації.

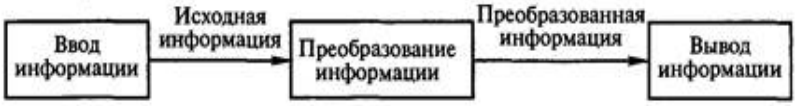
Тут бажана розробка інструкції користування майбутньою програмою, тобто все, що побачив би користувач, отримавши готову програму. До тестування добре складених зовнішніх специфікацій можна залучити потенційних користувачів до розробки внутрішніх специфікацій. Користувачеві можна показувати макети екранів у порядку виконання програми, а користувач може готувати дані для тестування всіх функцій програми та зможе апробувати методику роботи з програмою. Зовнішні специфікації зазвичай фіксуються в ТЗ чи ЕП, але можуть бути уточнені у ТП.

До внутрішніх специфікацій відносяться описи складу внутрішніх частин програми, опис їх взаємозв'язку, а також внутрішні функціональні специфікації.

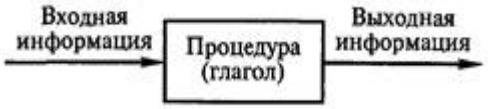
Внутрішні функціональні Специфікації включають описи алгоритмів як усієї програми, так і її частин з урахуванням специфікації внутрішніх даних програми (змінних, особливо структурованих).

Наведені далі абстракції процедури, даних та об'єктів лежать в основі багатьох методів розробки програмного забезпечення. Загалом будь-яку програму можна надати набором процедурних абстракцій (рис. 1.2). Аналізуючи рис. 1.2 можна отримати узагальнену абстракцію процедури, зображену на рис. 1.3. Абстракції процедур найповніше втілилися у технології структурного програмування.

Розділяючи в програмі тіло процедури та звернення до неї, мова високого рівня реалізує два важливі методи абстракції: абстракцію через параметризацію та абстракцію через специфікацію.



*Мал. 1.2.*Абстракція програми як набору процедур, що обробляють дані



*Мал. 1.3.*Анотація процедури

Абстракція через параметризацію дозволяє, використовуючи параметри, уявити фактично необмежений набір різних обчислень однією процедурою, що є абстракція всіх цих наборів. Наприклад, необхідна процедура сортування масиву цілих чисел А. При подальшій розробці програми можливе виникнення потреби у сортуванні іншого масиву, але з іншим ім'ям. Використання абстракції через параметризацію узагальнює процедуру сортування та робить її більш універсальною.

Абстракція через специфікацію дозволяє абстрагуватися від процесу обчислень, описаних у тілі процедури, рівня знання лише те, що дана процедура має у результаті реалізувати. Це досягається шляхом завдання для кожної процедури специфікації, що описує ефект її роботи, після чого зміст звернення до цієї процедури стає зрозумілим через аналіз цієї специфікації, а не самої процедури.

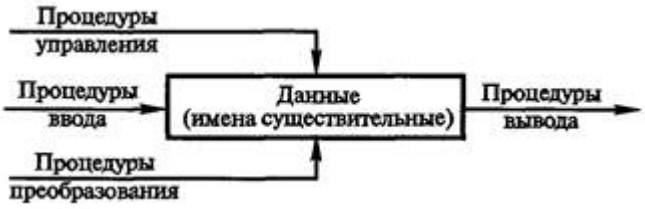
При аналізі специфікації для з'ясування сенсу звернення до процедури слід дотримуватися наступних двох правил.

Правило 1.Після виконання процедури вважатимуться, що виконано кінцеве умова. Виконання кінцевої умови за дотримання початкових умов — це те, заради чого і побудована процедура. Якщо це процедура пошуку максимального значення масиві, то кінцева умова — факт, що максимальний елемент знайдено. Якщо це процедура обчислення квадратного кореня, то кінцева умова знаходження квадратного кореня.

Правило 2 Можна обмежитися лише тією інформацією, яку передбачає кінцева умова.

Ці правила демонструють дві переваги абстракції через специфікацію. Перше у тому, що програмісти, використовують цю процедуру, нічого не винні знайомитися з текстом її тіла. Отже, їм не потрібно усвідомлювати, наприклад, подробиці алгоритму відшукання квадратного кореня, встановлюючи, чи справді повернутий результат — кількість, яку шукає.
Друге правило показує, що насправді маємо справу з абстракцією: абстрагуючись від тіла процедури можна не звертати уваги на несуттєву інформацію. Саме таке «ігнорування» інформації та відрізняє абстракцію від декомпозиції. Звичайно, аналізуючи тіло процедури, можна витягти деяку кількість інформації, яка не випливає з кінцевої умови (як, наприклад, те, що знайдений елемент перший або останній у розглянутому вище прикладі). У специфікації подібна інформація про результат, що повертається, відкидається.
Абстракції через параметризацію та через специфікацію є потужним засобом створення програм. Вони дозволяють визначити два види абстракцій: процедурну абстракцію та абстракцію даних. У загальному випадку кожна процедурна абстракція та абстракція даних використовують обидва способи.
Наприклад, абстракцію SQRT (витяг квадратного кореня) можна порівняти з операцією: вона абстрагує окрему подію або завдання. Ми будемо посилатися до таких абстракцій, як до процедурних абстракцій. Зазначимо, що абстракція SQRT включає як абстракцію через параметризацію, так і абстракцію через специфікацію.
На процедурній абстракції засновано розробку структури програми в технології структурного програмування. Базова компонента технології структурного програмування — модуль, якому зазвичай відповідає підпрограма (процедура чи функція мовами програмування високого рівня). Розглядаючи програму не як набір процедур, а передусім деякі набори даних, кожен із яких має дозволену групу процедур, отримуємо абстрактне уявлення програми, представлене на рис. 1.4. Аналіз рис. 1.4 дозволяє отримати абстракцію даних, показану на рис. 1.5.
У технології абстрактних даних Дейкстри застосовується функціональна модель як набору діаграм потоків даних (далі — ДПД; DFD — Data Flow Diagram), які описують зміст операцій та обмежень. ДПД відображає функціональні залежності значень, що обчислюються в системі, включаючи вхідні значення, вихідні значення та внутрішні сховища даних. ДПД - це граф, на якому показано рух значень даних від їх джерел через процеси, що їх перетворюють, до їх споживачів в інших об'єктах. Фрагменти ДПД показано на рис. 1.6.

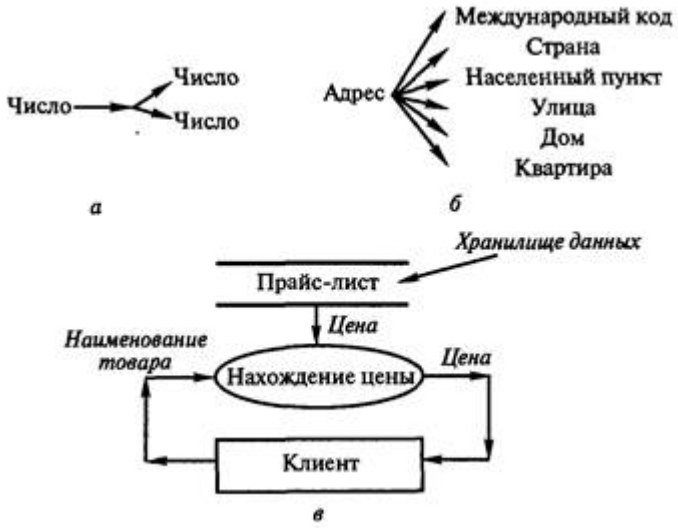
Мал. 1.4. Абстракція програми як набору даних, що обробляються процедурами



*Мал. 1.5.*Абстракція даних

Потік даних з'єднує вихід об'єкта (або процесу) із входом іншого об'єкта (або процесу). Він представляє проміжні дані обчислень.
Сховище даних- Це пасивний об'єкт у складі ДПД, в якому дані зберігаються для подальшого доступу. Сховище даних допускає доступ до даних у порядку, відмінному від того, в якому вони були туди поміщені. Агрегатні сховища даних, як, наприклад, списки та таблиці, забезпечують доступ до даних у порядку їх надходження або за ключами.
ДПД показує всі шляхи обчислення значень, але з показує, у порядку ці значення обчислюються. Рішення про порядок обчислень пов'язані з керуванням програмою, що відображається у динамічній моделі. Ці рішення, що виробляються спеціальними функціями, або предикатами,

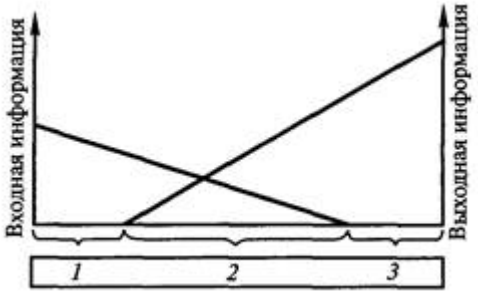
визначають, чи буде виконано той чи інший процес, але при цьому не передають процесу жодних даних, тому їх включення до функціональної моделі необов'язково. Тим не менш, іноді буває корисно включати зазначені предикати у функціональну модель, щоб в ній були відображені умови виконання відповідного процесу.



*Мал. 1.6.*Фрагменти діаграми потоків даних (ДПД):

а- Копіювання даних (числа); б – розщеплення даних; в – активний об'єкт «Клієнт» за допомогою операції «Знаходження ціни» працює зі сховищем «Прайс-лист»

Дейкбуд запропонована відносно технологія, що рідко застосовується, заснована на абстракції даних. Ця технологія є альтернативою структурному програмуванню. У чистому вигляді вона успішно застосовувалася розробки СУБД та інших виробів, орієнтованих перетворення інформації з однієї форми на іншу.
Якщо структурному програмуванні головними є функції та процедури (дії), то технології абстрактних даних Дейкстри на чолі ставляться дані.
За цією технологією спочатку дуже ретельно специфікуються вихід, вхід, проміжні дані; велика увага приділяється типізації даних з використанням структур для об'єднання близької за змістом інформації у єдині дані. Зазвичай розписується схема ієрархії даних. Тут зручно застосовувати моделі як діаграм потоків даних.
Нижче наводиться опис методики отримання програми з раціональною структурою даних, яка ґрунтується на абстракції даних Дейкстри. Ця методика ставить на перше місце дані, що з деякими труднощами забезпечується методикою розподілу програми на смислові модулі шляхом виділення смислових підфункцій у технології структурного програмування.
*Крок 1*Грунтуючись на потоці даних у задачі, виділіть 3-10 значеннєвих частин обробки даних.
*Крок 2*Визначте головний вхідний та вихідний потоки даних завдання.
*Крок 3*Простежте, як слід вхідний потік від частини до частини, від входу до кінця обробки знайдіть цю точку. Простежте від кінця до початку, отже вихідний потік; знайдіть абстрактну точку, де з'явився (рис. 1.7).
Знайдені точки поділяють завдання на дві чи три найбільш незалежні (за даними) частини.
*Крок 4.*Подайте незалежні частини підпрограмами та визначте їх функції. Ці підпрограми будуть підпорядкованими до модуля, розбиття якого виконується.
*Крок 5.*Визначте сполучення підпрограм за даними.



*Мал. 1.7.*Перетворення головного вхідного потоку інформації у вихідний потік:

1 - ділянка відповідає перетворення інформації вхідного потоку на проміжну інформацію; 2 — ділянка відповідає отриманню вихідної та проміжної інформації з вхідної та проміжної інформації; 3 - ділянка відповідає отриманню вихідної інформації з проміжної інформації

Сучасна об'єктно-орієнтована технологія, що витісняє технології структурного програмування та абстракції даних, поєднує в собі абстракції процедур і даних у новій абстракції — об'єкті (рис. 1.8). Поняття абстракції даних розширено доти, що як внутрішні дані, і код процедур розглядаються як новий тип даних — об'єкт.
Об'єктна модель базується на двох постулатах: є об'єкти; об'єкти взаємодіють передачею повідомлень.
Методи структурного проектування допомагають спростити процес розробки складних систем з допомогою використання алгоритмів як готових будівельних блоків. Об'єкт - більший будівельний блок. Він може включати як дані (поля), так і процедури (методи). Узбуйнення будівельних блоків стало необхідністю при створенні великих програм.
При процедурному програмуванні акцент робиться на обробці (алгоритмі), яка потрібна на виконання необхідних обчислень. Парадигма: виріши, які потрібні процедури; використовуй найкращі доступні алгоритми.

*Мал. 1.8.*Абстракція об'єкту

Парадигма програмування на основі абстрактних даних Дейкстри: визнач організацію даних та вияви всі стани значень даних, вважай, що процедури – це щось, що змінює дані.
У структурному програмуванні основний структурної одиницею є модуль (The module).
Парадигма: розбий програму на систему ієрархічно підлеглих модулів (процедур) так, щоб забезпечити максимальну якість тестування під час виконання розробленого плану реалізації програми. Важливим є те, в якому з модулів та на якому рівні ієрархії модулів описувати ті чи інші дані. p align="justify"> При структурному підході інформаційні потоки протікають в одному напрямку: від вихідних даних до результату.
Розміщений в окремому файлі набір пов'язаних процедур разом із даними, які вони обробляють, називають програмною одиницею (Unit). Часто слово "Unit" перекладають як модуль. Так виник термін «модульне програмування». У модульному програмуванні акцент змістився від проектування процедур у бік організації даних. Крім іншого, це було відображенням факту збільшення розмірів програм. Парадигма: розв'яжи, які потрібні модулі; розбий програму так, щоб приховати дані в модулях.
В об'єктно-орієнтованому програмуванні абстракція даних є основним аспектом якісного проектування. Парадигма: програма представляється набором об'єктів, які, взаємодіючи один з одним за допомогою повідомлень, змінюють себе та навколишні об'єкти. Принцип модульного програмування використовується в об'єктно-орієнтованому програмуванні. Зазвичай, у одному Unit описується або один клас, або кілька класів, успадкованих від одного класу. Механізми Unit реалізують приховування інформації. Об'єктно-орієнтований підхід характеризується різноспрямованістю та різнорідністю інформаційних потоків.

1.11. МНЕМОНІКА ІМЕН У ПРОГРАМАХ

Запропонована методика складання імен (ідентифікаторів) носить рекомендаційний характер. Для використання цієї методики у конкретному проекті необхідна її адаптація. Складені відповідно до методики імена можна використовувати в програмах для іменування констант, змінних, типів, процедур, об'єктів, файлів і т. д. Після адаптаційної переробки методика може стати складовою стандарту проекту. Імена, що використовуються у програмних продуктах, повинні:
- відповідати призначенню (з імені має однозначно слідувати його призначення та, навпаки, з призначення - його ім'я);
— мати впізнаваність (ця властивість імені дозволяє покращити читання вихідних текстів програм);
— забезпечувати запам'ятовуваність (ім'я необхідно легко запам'ятати для того, щоб щоразу не повертатися до документації або тексту програми, в якому це ім'я визначено);
- бути короткими (надто довгі імена не запам'ятовуються і, незважаючи на підвищену впізнаваність порівняно з короткими іменами, ускладнюють читання вихідного тексту програми);
— мати унікальність (як наслідок, «відповідати призначенню»: імена мають складатися таким чином, щоб у всій системі не було двох однакових глобальних імен).
Звичайно, хотілося б домогтися одночасного виконання всіх викладених вище вимог у сукупності, але оскільки вимога стислості суперечить вимогі «володіти впізнаваністю», а іноді й «забезпечувати запам'ятовуваність», необхідно знаходити оптимальний компроміс у дотриманні всіх вимог.
В ідеальному випадку імена не слід запам'ятовувати: їх потрібно складати таким чином, щоб кожен раз, знаючи, для якого об'єкта складаєте ім'я, ви приходили б до одного і того ж варіанта імені.
Текст даних рекомендацій не лімітує використання великих і малих літер, спосіб поділу слів і те, які слова використовувати в іменах, - додаткові обмеження накладає конкретну мову програмування. Також не розглядається відповідний для конкретної мови програмування символ роздільника слів.
Імена констант і змінних відносяться до даних, а імена даних утворюються від іменників. Імена процедур повинні бути активними, тобто базуватися на активному дієслові, за яким слід іменник. Імена об'єктів зазвичай утворюються від іменників, але в поодиноких випадках можуть включати дієслова та прикметники. Повне ім'я методу складається з імені об'єкта, якому належить метод, символ «.» роздільника та власне імені методу об'єкта. Для таких повних імен дуже важко досягти стислості. Метод є процедурною абстракцією, та її власне ім'я утворюється від дієслова.
Отже, ім'я складається із слів. Нехай довжина імені – це кількість слів, використаних у цьому імені. Ім'я А є батьківським стосовно імені Б, якщо довжина імені Б більша, ніж довжина імені А, і перші (ліворуч) слова імені Б збігаються зі словами імені А в тому самому порядку. Ім'я А можна розглядати як загальний префікс для імен групи Б. Наприклад, ім'я debug є батьківським для імен debug_info, debug_mode, debug_log, debug_error_get, debug_error_set тощо. Ім'я debug_error, у свою чергу, є батьківським для debug_error_get.
Ім'я А є дочірнім по відношенню до імені Б, якщо ім'я Б є батьківським по відношенню до імені А. Імена належать одній групі, якщо ці імена мають однакову довжину та одного спільного предка А. Довжина імені А на одиницю менша за довжину імен цієї групи. У такому разі А буде ім'ям цієї групи. Наприклад, імена debug_error_get, debug_error_set є іменами групи debug_error. Імена debug__info, debug_mode, debug_log та debug_error, у свою чергу, є іменами групи debug.
Нехай потужність групи А – загальна кількість імен у цій групі. Префікс імені - це слово, яке записується найпершим у імені та не враховується при визначенні довжини, спорідненості та приналежності до групи. Префікси використовують, наприклад, для вказівки типів змінних чи полів: i_count, b_valid, is_protected. i, b, is префікси.
Вимоги ієрархічної організації імен можуть частково порушуватись або взагалі не використовуватися при складанні локальних імен (імен для локальних змінних, імен полів таблиць, імен властивостей та методів об'єктів тощо). Однак у випадку, якщо локальних імен багато, є сенс застосовувати ці вимоги і до локальних імен.
Якщо ім'я А є дочірнім по відношенню до імені Б, ім'я Б є позначенням деякого об'єкта. Це означає, що всі слова імені, крім останнього імені, можуть бути утворені тільки іменниками. Тільки останнє слово в імені може бути іменником, дієсловом або прикметником. Це правило, однак, іноді може порушуватися. Наприклад, є певна дія та набір глобальних налаштувань

(констант), які контролюють цю дію. У такому разі в іменах цих констант передостаннім словом буде дієслово.
У деяких випадках імена об'єктів можуть бути дієсловами. Наприклад, підсистему очищення бази даних було б логічно назвати слово «clear» (очистити). У такому разі дієслово «очистити» стоятиме в середині імені.
*приклад 1.*change_user_password – погане ім'я. Перше слово - дієслово і воно не може позначати ім'я об'єкта. Крім того, може виявитися, що для кожного користувача в системі зберігається кілька паролів, наприклад, пароль для доступу до свого облікового запису (account) та пароль для входу в чат. У такому випадку у двох різних місцях може знадобитися ввести дві функції (змінити пароль для доступу до account та змінити пароль для входу в чат) з однаковими іменами, що суперечить пункту «відповідати призначенню» загальних вимог до імен.
*приклад 2.*passport_password_change або passport_password_change — добрі імена. У системі є підсистема керування обліковими записами користувачів, звана passport. Частина цієї підсистеми, що займається керуванням паролями, називають passport_password. Одну з функцій цієї частини – зміна пароля – називають passport_password_change.
Довжина імені має бути мінімальною. Не використовуйте в зайвих іменах слів. Кожне слово, використане в імені, має означати конкретний об'єкт, якому належить це ім'я чи конкретну дію чи властивість, якому відповідає це ім'я. Імена об'єктів, дій та властивостей, у свою чергу, повинні складатися з імен, довжина яких дорівнює одному слову.
*приклад 1.*ConvertIntegerDateToSQLStrDate - погане ім'я. Як ви вважаєте, чи згадаєте ви його з точністю до символу через день?
Слова "Convert" і "To" найчастіше можна взагалі опустити, оскільки очевидно, що якщо є два формати дати, то, отже, відбувається перетворення з одного формату на інший.
Слово "Date" повторювати двічі не потрібно, оскільки і вихідне дане, і результат є датою.
Усі SQL-запити у програмі – це рядки. Тому слово "Str" - зайве.
*приклад 2.*IntegerDateSQL - прийнятне ім'я. Існує підсистема керування датами, і ця функція конвертує дату, представлену у вигляді цілого числа в рядок, який можна використовувати SQL-запиті. Недоліком цього імені є відсутність активного дієслова. Порівняйте це ім'я із вихідним варіантом прикладу 1.
У системі може з'явитися група імен потужністю 1.
*приклад 1.*PASSPORT_DEAD_REMOVE_TIMEOUT - погане ім'я, якщо з померлими користувачами (так на сайті названі користувачі, які занадто довго не з'являються) не можна робити жодних інших операцій, крім видалення.
*приклад 2.*PASSPORT_DEAD_TIMEOUT – гарне ім'я.
*приклад 3.*passport_is_login_valid – погане ім'я. Слово "is" - зайве.
*приклад 4.*passport_login_valid – гарне ім'я. Є підсистема керування обліковими записами passport. У ній є частина, яка займається керуванням логінами користувачів passport_login. Функція passport_login_valid перевіряє, чи є логін правильним.
У деяких випадках, однак, можуть бути створені групи довжиною 1, наприклад, якщо передбачається, що до цієї групи в майбутньому будуть додані нові імена.
Скорочення у словах у випадку неприпустимі. Якщо використовуєте в іменах слова зі скороченнями, то може скластися ситуація, коли довго згадуватимете, яким саме способом скоротили це слово і чи скорочували його взагалі. Тим більше, що те саме слово можна скоротити різними способами.
Це стосується і використання множини іменників, дієслів у другій і третій формі і т. п. Скрізь, де можливо, потрібно використовувати початкову форму слова, для того щоб уникнути різночитання. Крім того, якщо допускаєте різні скорочення (або будь-яку іншу плутанину з формами одного і того ж числа), то може виявитися, що з призначення імені не випливає однозначно саме ім'я через те, що не виконується пункт «відповідність до призначення» загальних вимог до імен.
Слова не в початковій формі можуть бути використані тільки в тому випадку, якщо вони використовуються багато разів і при цьому у всіх місцях однаково.
Наслідок: як останнє слово імені може бути використане тільки загальноприйняте скорочення (або початкова форма), яке СКРІЗІ (і багато разів) у програмі використовується саме в такому варіанті. Якщо скорочення використовується рідко, краще використовувати початкову форму слова.
*приклад 1.*StrToFloat – погане ім'я. У деяких мовах програмування є зарезервоване слово «String». У такому разі виходить, що в деяких випадках до рядків звертаємося на повне ім'я, а в деяких —

на скорочення. Проте скорочення "Str" - загальноприйняте в системах програмування фірми "Borland". При використанні цих систем, але не в SQL запитах, таке скорочення резонно використовувати, і ім'я StrToFloat стає хорошим ім'ям.
*приклад 2.*StringToFloat - хороше ім'я (якщо не враховувати наявність зайвого слова "To", але "Te" добре показує, що це ім'я процедури конвертора типів).
*приклад 3.*mp_pagelist — хороше ім'я, якщо в групі «mp» багато імен або це скорочення використовується в такому ж написанні і такому ж сенсі багато разів в інших іменах (mp — скорочення від «main page»).
*приклад 4.*PASSPORT_PASSWORD_LENGTH_MIN - хороше ім'я, скорочення в кінці - загальноприйняте і скрізь у системі використовується саме в такому варіанті написання.
*Приклад 5.*TPassportPrivileges — погане ім'я для таблиці, де зберігається список привілеїв. Вочевидь, що у таблиці зберігається багато всяких привілеїв, і множина у разі зайве.
*Приклад 6.*TPassport_Privilege — хороше ім'я, проте якби не йшлося про бази даних, префікс «T» відповідав би типу, а не змінній.
Не допускається використання префіксів без особливої необхідності.
Випадки, у яких використання префіксів виправдане:
• якщо це імена змінних, а імена типів змінних, можна використовувати префікс «T»;
• якщо імена ваших сутностей перемішуватимуться зі сторонніми іменами;
• у разі локальних імен.
Імена, які використовуються в обмеженому контексті, можуть бути дуже короткими. Традиційно імена i та j використовуються для позначення лічильників, p і q – для покажчиків, s – для рядкових, а ch – для літерних змінних. Ці традиційні найкоротші імена можуть відповідати префіксам, що пояснюють тип змінних.
*приклад 1.*is_passport_privilege_valid – погане ім'я. Префікс у глобальному імені зайвий.
*приклад 2.*passport_privilege_valid – гарне ім'я.
*приклад 3.*i_order - хороше ім'я для поля в таблиці, що вказує на порядок чогось. Префікс «i» характеризує цілий тип.
Додаткові рекомендації щодо складання імен:
• не починайте і не закінчуйте імена символом підкреслення;
• не використовуйте імена, що складаються лише з малих літер (виключення становлять імена констант та макровизначень);
• не слід в одній і тій же програмі використовувати імена, що відрізняються лише написанням літер — малої або великої;
• залежно від можливостей мови програмування можна розділяти частини імен символом підкреслення або написанням великої літери чергової частини імені;
• використання малих літер на початку кожного слова імені ускладнює трансляцію тексту програми з однієї мови програмування на низку інших мов.
Ряд імен, що розуміються, важко ретельно коментувати. Такий коментар краще наводити праворуч від опису імені.
При описі логічної організації змінних, файлів чи класів слід застосовувати додаткові коментарі.
Рефакторинг(Від англ. Refactoring) - Оптимізація, поліпшення реалізації програми без зміни її функціональності.
Стосовно вже кимось або колись написаних програм може здійснюватися реакторинг імен, структури даних програми, структури програми та коду. Одночасно з рефакторингом коду може бути здійснений і рефакторинг опису алгоритму природною мовою.
Щодо імен під рефакторингом розуміється зміна імен таким чином, щоб вони відповідали новим вимогам. Імена – це дуже важлива частина програми. Багато програмістів схильні применшувати значущість імен.
Незрозумілі імена — це нечитана програма, а програму, що не читається, важко супроводжувати.
Розглянемо випадки, у яких може знадобитися рефакторинг імен:
Випадок 1- Зміна правил. Можна скласти різні правила освіти імен і слідувати спочатку одним правилам, потім, уточнивши ці правила, слідувати іншим. Незмінним має залишатися лише одне правило: всі імена у проекті мають бути побудовані за одними правилами.
*Випадок 2*частковий рефакторинг імен. Якщо з будь-яких причин змінили правила складання імен, слід оновити всі імена, які не підходять під нові правила.

*Випадок 3*неможливість однозначного передбачення майбутнього. Будь-який проект розвивається. На етапі створення може виявитися, що спроектована структура не повна або спочатку передбачалася одна структура, а потім стала очевидною інша, краща структура.
*Випадок 4*рефакторинг імен, на вашу думку, не потрібен: ви вже закінчуєте роботу над проектом, не плануєте в майбутньому його підтримувати і вам все одно, що про вас подумають люди, яким доведеться розбиратися у вашому коді.
Кваліфіковані програмісти витрачають деякий час на очищення свого коду для простоти подальшого використання. Ясність коду визначається ясністю імен даних, зрозумілістю призначення та послідовності дій, ясністю імен процедур та об'єктів.

1.12. ПРОБЛЕМА ТИПОВИХ ЕЛЕМЕНТІВ У ПРОГРАМУВАННІ

Під типовими елементами, чи «кубиками», розуміються якісь окремо виготовлені типові частини, у тому числі можна було збирати безліч програм. Проблема «кубиків» притаманна як програмуванню, й у різних галузях вона проявляється по-різному.
Область машинобудування поруч із такими «кубиками», як болти, гайки, оперує безліччю нетипових елементів. Наприклад, ліве крило і праве крило автомобіля хоч і дуже схожі один на одного, але не можуть бути взаємозамінними і можуть використовуватися лише в конкретній моделі автомобіля. У галузі машинобудування значні зусилля проектувальників витрачаються проектування елементів. Кількість елементів, з яких складаються конструкції, зазвичай не перевищує декількох сотень.
Найбільш повно вирішено проблему «кубиків» у галузі радіоелектроніки. Резистори, ємності, лампи, транзистори, мікросхеми, ряди функціональних блоків є стандартизованими та взаємозамінними. У цій галузі проектувальники вирішують завдання синтезу штучних систем із десятків і навіть сотень тисяч елементів.
Програми є найскладнішими штучними системами, у яких загальна кількість елементів (операторів) може сягати кількох мільйонів. Нові технології програмування використовують дедалі нові, зазвичай, більші, типові елементи побудови програм.
Першими укрупненими типовими елементами були підпрограми. До цього часу бібліотеки математичних методів зазвичай постачаються у вигляді набору підпрограм. Складність навіть найпростішого, дуже поширеного такого типового елемента, як редактор текстів, характеризується вже десятком підпрограм.
Для тиражування таких елементів програм, як редактор текстів, система ієрархічного меню, елементів діалогу типу заповнення бланків фірма Borland Inc. запропонувала застосовувати TPU - Turbo Pascal Unit (модуль OBJ у ряді мов). TPU-файл дозволив використовувати механізм приховування в секції Implementation нецікавих внутрішніх підпрограм та внутрішніх даних і, навпаки, поза файлом механізм приховування забезпечив відкритість виклику лише корисних для користувача процедур та використання внутрішніх глобальних змінних, описаних у секції Interface. Після цього нововведення програмісту для використання, наприклад редактора, написаного не ним, треба знати лише інформацію, описану в секції Interface. Механізм приховування інформації в межах файлу був введений ще в низку компіляторів різних мов. Реалізація механізму приховування спростила завдання використання "кубиків".
Об'єктно-орієнтовані мови програмування дали чотири нових механізми використання кубиків:
1) механізм класів, що породжують під час виконання будь-яку кількість однотипних об'єктів, наприклад, ряд однотипних кнопок;
2) можливість тиражування об'єктів від програми, що породила, у всі нові програми;
3) динамічно лінкувані бібліотеки з класами, що породжують об'єкти;
4) механізм складання програм із «кубиків» — об'єктів у процесі їх виконання.
Перший механізм полегшив розвиток систем візуального програмування, під час роботи у яких значну частину програми можна створити шляхом відбору мишкою стандартних «кубиків».
Другий механізм призвів до виникнення об'єктно-орієнтованих СУБД, які постачають програмам як дані, а й код, обробляє ці дані.
Третій і четвертий механізми дозволили спробувати будувати гнучкі програми, що мають властивість можливого розвитку при зміні умов їх експлуатації. Вперше можливість реалізації отримали ідеї еволюційного програмування. Ідеї минулих технологій еволюційного програмування були явно недостатніми для забезпечення гнучкості програм, що для тривалого

розвитку у мінливому зовнішньому середовищі вимагало неможливого однозначного передбачення майбутнього. Згідно з третім механізмом виникли COM-технології. На основі четвертого механізму з бази готових «кубиків П-об'єктів» створюються нові «кубики» і з них нові програми.
Таким чином, теоретично програмування проблема «кубиків» залишається найважливішою проблемою, поетапне вирішення якої дозволило створювати найбільші за кількістю складових частин штучні системи — програми. Другий аспект проблеми «кубиків» — здешевлення програмних розробок за рахунок повторного використання у нових програмах частин, створених у попередніх розробках. Якщо є «кубики», то технології програмування необхідно включити методи, що полегшують синтез цілісних систем з окремих «кубиків».

ВИСНОВКИ

• Проектування – високоінтелектуальний процес. Для поняття теорії проектування необхідно оперувати безліччю термінів та визначень, такими як проектна ситуація, технологія, оптимізація програмних розробок. Усе це говорить необхідність ретельно підходити до вивчення словникового апарату теорії проектування.
• Програми в основному складні системи з мільйонів машинних інструкцій. Складність визначається чотирма основними причинами: складністю завдання; складністю управління процесом розробки; складністю опису поведінки окремих підсистем; складністю забезпечення гнучкості кінцевого програмного продукту.
• Під час розробки програмного забезпечення слід використовувати такі загальні принципи: частотний; модульності; функціональної вибірковості; генерованості; функціональної надмірності; "за замовчуванням".
• Однією з найважливіших складових успішного проектування є системний підхід, який передбачає всебічне дослідження складного об'єкта.
• Під час створення та розвитку ПЗ рекомендується застосовувати такі загальносистемні принципи: включення; системної єдності; розвитку; комплексності; інформаційної єдності; сумісності; інваріантності.
• У програмуванні існують різні парадигми, що призводять до різних підходів під час написання програм: процедурно-орієнтований; об'єктно-орієнтований; логічно-орієнтований; орієнтований правила; орієнтований обмеження; паралельне програмування, і навіть багато інших.
• Необхідно пам'ятати, що проектування невід'ємне від різних стандартів (ГОСТ, ANSI, проекту) та їх слід дотримуватись як при оформленні документації, так і для уніфікації вашого проекту.
• Програми створюються, експлуатуються та розвиваються у часі, проходячи свій життєвий цикл. Характерна риса життєвого циклу ПЗ - відсутність етапу утилізації.
• У процесі виконання проекту передбачаються окремі моменти часу, що характеризуються закінченим оформленням результатів усіх робіт, виконаних розробниками до цього моменту. Відповідно до ГОСТ можливі такі стадії розробки: ТЗ; ЕП; ТП; РП; Використання. Можливі також і нестандартні етапи та стадії. Набір етапів та стадій відображає результати проектування самого процесу проектування.
• Моделі відіграють важливу роль у проектуванні програм. При побудові моделей використовується абстрагування та декомпозиція.
• Кожна стадія проекту завершується затвердженням програмних документів. Документи включають опис (специфікації). Специфікації є моделі. Специфікації поділяються на зовнішні та внутрішні.
• Раціональний вибір стандартних елементів («кубиків») має два аспекти: зручність при повторному використанні та можливість здійснення синтезу з малих елементів загальніших елементів.
• Імена, що використовуються в програмах, повинні відповідати призначенню, мати впізнаваність, забезпечувати запам'ятовуваність, бути короткими, мати унікальність.

Контрольні питання

1. Дайте визначення проектування.
2. Що таке евристика?
3. У чому схожість і відмінність алгоритму та евроритму?
4. Що розв'язує задачу оптимізації розробки програм?
5. Назвіть п'ять ознак складної системи.
6. На чому ґрунтується частотний принцип розробки програм?

7. Які види аналізу застосовуються при системному підході?
8. Що таке принцип сумісності?
9. Для чого необхідна стандартизація проектування та програмування?
10. Назвіть основні етапи життєвого циклу програмних виробів.
11. Назвіть основні стадії та етапи розробки програм за ГОСТами.
12. У чому сутність моделювання?
13. Які типи абстракцій ви знаєте?
14. Що таке первинна функціональна специфікація?
15. Які механізми використання «кубиків» надали об'єктно-орієнтовані мови програмування?
16. Що таке рефакторинг?
17. Навіщо потрібний рефакторинг імен?
18. Чому важко визначити ідеальні імена?