ПРОЦЕСИ ТА СИСТЕМИ ПІДТРИМКИ ЯКОСТІ ПРОГРАМНИХ СИСТЕМ

Лекція 1. Якість програмного забезпечення

Що таке якість?

Основною проблемою в управлінні якістю є той факт, що визначення якості неясне і неоднозначне. Це викликано тим, що зазвичай термін «якість» розуміють неправильно. Причини:

1. Якість – це не окрема ідея або поняття, а багатомірна і різнопланова концепція.
2. Для будь-якого поняття і визначення існує декілька рівнів абстракції, наприклад, коли люди говорять про якість, одна частина розуміє під цим занадто широкий і розмитий смисл, а інша може вказувати на конкретне визначення.
3. Термін «якість» є невід'ємною частиною нашого повсякденного спілкування, але загальноприйняте і професійне використання може сильно відрізнятись.

Популярний погляд на якість

Загальноприйнята думка про якість – це дещо нематеріальне і «неосяжне» - про нього можуть вестись суперечки та дискусії, його можна критикувати і вихваляти, але зважити і виміряти неможливо. Вирази типу «хороша якість» і «погана якість» служать наглядним прикладом. Люди так кажуть про щось невизначене для них, що вони не можуть чітко охарактеризувати і визначити. Отже, люди сприймають і інтерпретують якість по різному. Якість не може бути контрольованою і керованою. Якість не може бути кількісно виміряна. Такий погляд різко контрастує з професійним підходом до управління якістю – якість чітко визначена величина, яку можна виміряти і проконтролювати, вона піддається управлінню і покращенню.

Інша популярна думка – якість нерозривно пов'язана з розкішшю, першим сортом і тонким смаком. Дорогий, досконально продуманий і технічно більш складний продукт розглядається як гарантія вищої якості ніж більш дешеві аналоги. За такою логікою Каділлак – якісна машина, а Шевроле – ні, незважаючи на надійність і кількість поломок. Якщо слідувати такому підходу, то якість обмежена визначеним класом дорогих продуктів. Недорогі продукти ж важко віднести до якісних.

Професійний підхід до якості

Існують чіткі та зручні для роботи визначення.

В 1979 році Філ Кросбі визначив якість як «відповідність вимогам» ("conformance to requirements").

В 1970 році Юран і Гріна визначили якість як «придатність до використання» ("fitness for use"). Ці два визначення тісно пов'язані і добре узгоджуються.

Ще декілька визначень якості:

Уотс Хемпфрі – досягнення відмінного рівня придатності до використання;

Компанія IBM - якість, яка управляється ринковими потребами ("market - driven quality").

Критерій Белдріджа для організаційної якості - "якість, яка задається споживачем"

Визначення системи менеджменту якості ISO 9001 - ступінь відповідності наявних характеристик вимогам.

«**Відповідність вимогам**» припускає, що вимоги мають бути настільки чітко визначені, що вони не можуть бути зрозумілі і інтерпретовані некоректно. Пізніше, на етапі розробки, проводяться регулярні вимірювання розробленого продукту для визначення відповідності вимогам. Будь-які невідповідності розглядаються як дефекти – невідповідність якості. Наприклад, специфікація на деяку модель радіостанції може вимагати можливість приймати визначену частоту радіохвиль на відстані більше ніж за 30 км від джерела віщання. У випадку, якщо радіостанція не може виконати дану вимогу, вона не відповідає вимогам до якості і має бути визнаною негідною і неякісною. Виходячи з таких принципів, якщо Каділлак відповідає усім вимогам до машин Каділлак, то це якісна машина. Якщо Шевроле відповідає усім вимогам до машин Шевроле, то це теж якісна машина.

«**Придатність до використання**» приймає до уваги вимоги і очікування кінцевого користувача, який очікує, що продукт або надаваний сервіс буде зручним для нього. Однак різні користувачі можуть використовувати продукт по різному і це означає, що продукт має володіти максимально різноманітними варіантами використання. Згідно визначенню Юран кожен варіант використання – це характеристика якості і усі вони можуть бути класифіковані по категоріям в якості параметрів придатності до використання.

Ці два визначення якості по суті однакові. Різниця в тому, що варіант «придатність до використання» вказує на важливу роль вимог і очікувань замовника. Роль замовника, пов'язана з якістю, не може бути переоцінена. З точки зору замовника, якість продукту, який він придбав, складається з багатьох факторів: ціна, продуктивність, надійність і т.д.

Тільки замовник може розказати про якість, оскільки це єдине, що він дійсно купляє. Замовник не купляє продукт. Він купляє гарантії того, що усі його очікування до продукту будуть реалізовані.

Висновки

Отже, визначення якості з точки зору замовника або користувача продукту:

Якість – це придатність до використання. Чи робить даний продукт те, що мені потрібно? Чи спрощує роботу? Чи можу я його використати так, як мені потрібно?

Точка зору розробника:

Якість – це відповідність специфікованим і забраним вимогам. Чи робить даний продукт усе те, що вказано у вимогах?

Проблема в тому, що специфіковані і забрані вимоги – це зазвичай частина реальних вимог і очікувань замовника.

Отже, якість – це відповідність реальним вимогам, явним і неявним. Дуже часто неявні вимоги настільки очевидні для замовника або користувача, що він навіть не припускає, що вони невідомі розробникам. На прикладі автомобілів: замовник може детально описати вимоги до дизайну, параметрам двигуна, оформленню салону, кольору кузова, але ніде не вказати, що шини мають бути круглими, а лобове скло – прозорим.

Замовник буде повністю задоволеним тільки тоді, коли куплений продукт буде повністю відповідати його реальним і життєвим вимогам – специфікованим і ні.

Існує дві професії пов'язані з якістю програмного забезпечення: тестер (Quality Control) і QA manager.

В чому різниця між Тестуванням и QA (Quality Assurance)?

Контроль якості (QUALITY CONTROL) це вимірювання якості продукту.

Забезпечення якості (QUALITY ASSURANCE) – це вимірювання і управління якістю процесу, який використовується для створення якості продукту (або якісного продукту).

Іншими словами.

Quality Assurance – попередження дефектів шляхом перевірки і тестування процесу.

Quality Control - виявлення дефектів шляхом перевірки і тестування продукту.

Що таке якість програмного забезпечення?

Якщо спитати про це достатньо широку групу людей, які мають справу з розробкою, продажем або використанням ПЗ, можна отримати наступні відповіді:

- Легко використовувати
- Хороши продуктивність
- Нема помилок
- Не псує даних користувача при збоях
- Можна використовувати на різних платформах
- Може працювати 24 години на сутки і 7 днів на тиждень
- Легко добавляти нові можливості
- Задовольняє потреби користувача

- Добре документовано

Якість ПЗ по МакКолу

Першою широко відомою моделлю якості ПЗ стала запропонована в 1977 МакКолом та іншими модель. В ній характеристики якості розділені на три групи:

Фактори, які описують ПЗ з позицій користувача. Задаються вимогами.

Критерії, які описують ПЗ з позицій розробника. Задаються як цілі.

Метрики, які використовуються для кількісного описання і вимірювання якості.

Фактори якості, яких було виділено 11, групуються в три групи по різним способам роботи людей з ПЗ. Отримана структура відображується у вигляді трикутника МакКола

Критерії якості – це числові рівні факторів, поставлених як цілей при розробці. Об'єктивно оцінити або виміряти фактори якості безпосередньо важко. Тому МакКол ввів метрики якості, які з його точки зору легше виміряти і оцінити. Оцінки в його шкалі приймають значення від 0 до 10. **Метрики**

Рис. 1. Треугольник МакКола.

- Зручність перевірки на відповідність стандартам (auditability)
- Точність керування і обчислень (accuracy)
- Ступінь стандартності інтерфейсів (communication commonality)
- Функціональна повнота (completeness)
- Однорідність використовуваних правил проектування і документації (consistency)
- Ступінь стандартності форматів даних (data commonality)
- Стійкість до помилок (error tolerance)
- Ефективність роботи (execution efficiency)
- Розширюваність (expandability)
- Широта області потенційного використання (generality)
- Незалежність від апаратної платформи (hardware independence)
- Повнота протоколювання помилок та інших подій (instrumentation)
- Модульність (modularity)
- Зручність роботи (operability)
- Захищеність (security)
- Самодокументованість (selfdocumentation)
- Простота роботи (simplicity)
- Незалежність від програмної платформи (software system independence)
- Можливість співвідношення проекту з вимогами (traceability)
- Зручність навчання (training).

Кожна метрика впливає на оцінку деяких факторів якості. Числовий вираз фактору представляє собою лінійну комбінацію значень впливаючих на нього метрик. Коефіцієнти цього виразу визначаються по різному для різних організацій, команд розробки, видів ПЗ та інш.

Якість ПЗ по Боему

У 1978 Боем запропонував свою модель, яка по суті є розширенням моделі МакКола.

Атрибути якості поділяються за способом використання ПЗ (primary use).

Визначено 19 проміжних атрибутів (intermidiate construct), які включають усі 11 факторів якості по МакКолу. Проміжні атрибути розділяються на примітивні, які в свою чергу, можуть бути оцінені за допомогою метрик.

До факторів МакКола Боем додав наступні атрибути:

- ясність (clarity)
- зручність внесення змін (modifiability)
- документованість (documentation)
- здатність до відновлення функцій (resilience)
- зрозумілість (understandability)
- адекватність (validity)
- функціональність (functionality)
- універсальність (generality)
- економічна ефективність (economy).

Лекція 2

МІЖНАРОДНИЙ СТАНДАРТ ЯКОСТІ ПРОГРАМНИХ ЗАСОБІВ ISO 9126

ISO 9126 — «**Інформаційна технологія. Оцінка програмного продукту. Характеристики якості та керівництво по їх застосуванню».**

ISO 9126 це міжнародний стандарт, який визначає оцінні якості програмного забезпечення. Стандарт поділяється на 4 частини, які описують наступні питання: модель якості, зовнішні метрики якості, внутрішні метрики якості, метрики якості у використанні.

Модель якості, встановлена у першій частині стандарту 9126-1, класифікує якість ПЗ в 6-ти структурних наборах характеристик, які в свою чергу деталізовані під-характеристиками (субхарактеристиками), такими як:

Функціональність (functionality)– набір атрибутів, який характеризує відповідність функціональних можливостей ПЗ набору потрібної користувачу функціональності. Деталізується субхарактеристиками:

- Придатність до застосування (suitability) – здатність ПС надавати належну множину функцій для розв'язання специфікованих задач і досягнення цілей користувача;
- Коректність (правильність, точність) (accuracy) – здатність ПС забезпечувати правильні або погоджені результати або впливи з необхідним ступенем точності;
- Здатність до взаємодії (в тому числі мережевої) (interoperability) – здатність взаємодіяти з однією чи більше специфікованими системами;
- Захищеність (security) – здатність забезпечувати захист інформації від несанкціонованого доступу осіб або систем і її доступність особам та системам з необхідними правами доступу.

Надійність (reliability) – набір атрибутів, які відносяться до здатності ПЗ зберігати свій рівень якості функціонування в встановлених умовах за визначений період часу. Субхарактеристики:

- Рівень завершеності (відсутність помилок) (maturity) – здатність уникати відмови із-за внутрішніх дефектів;
- Стійкість до дефектів (fault tolerance) – здатність підтримувати встановлений рівень функціонування в умовах прояви дефектів в ПС, помилок даних або порушень специфікованого інтерфейсу;
- Відновлюваність (recoverability) – здатність відновлювати функціонування на заданому рівні і відновлювати пошкоджені програми та дані;
- Доступність;
- Готовність.

Практичність (зручність застосування) (usability)– набір атрибутів, які відносяться до об'єму робіт, які необхідні для виконання та індивідуальної оцінки такого виконання визначеним або передбачуваним колом користувачів. Субхарактеристики:

- Зрозумілість (understandability) – властивості ПС, які забезпечують користувачу можливості зрозуміти, чи дійсно вона може задовільнити його потреби, як вона може бути використана для розв'язання визначених задач і які умови її використання;
- Простота використання (learnability) – властивості ПС, які забезпечують користувачу можливість засвоїти прийоми її застосування;
- Керованість (operability) властивості ПС, які забезпечують користувачу можливість керувати або контролювати її дії;
- Привабливість (attractiveness).

Ефективність (efficiency)– набір атрибутів, які відносяться до співвідношення між рівнем якості функціонування ПЗ і об'ємом використаних ресурсів при встановлених умовах. Суб:

- Часова ефективність (time behavior) - властивості ПС, які спричиняють її здатність забезпечувати належний час відклику (відповіді) і обробки завдань, а також рівень пропускної здатності при виконанні функцій у встановлених умовах застосування;
- Використовуваність ресурсів (resource utilization) властивості ПС, які спричиняють її здатність використовувати ресурси в необхідні періоди часу при виконанні своїх функцій у встановлених умовах застосування.

Супроводжуваність (maintainability) – набір атрибутів, які відносяться до об'єму робіт, які необхідні для проведення конкретних змін (модифікацій). Субхарактеристики:

- Зручність для аналізу (analyzability) властивості ПС, які спричиняють можливість діагностування її недоліків або причин відмов, а також ідентифікації частин, які мають модифікуватись;
- Змінюваність (changeability) властивості ПС, які спричиняють можливість виконання встановлених видів модифікації;
- Стабільність (stability) - властивості ПС, які спричиняють її здатність мінімізувати неочікувані ефекти модифікацій;
- Зручність для тестування (testability) - властивості ПС, які спричиняють її здатність сприяти перевірці модифікованого ПЗ..

Мобільність (portability)– набір атрибутів, які відносяться до здатності ПЗ бути перенесеним з одного оточення в інше. Суб:

- Здатність до адаптації (adaptability) - властивості ПС, які спричиняють її здатність адаптуватись для застосування в різноманітних специфікованих середовищах без використання дій або засобів відмінних від тих, що спеціально призначені для цих цілей;
- Простота встановлення (installability)
- Співвіснування (відповідність) (co-existence) властивості ПС, які спричиняють її здатність співіснувати з іншими незалежними ПС в спільному середовищі, розділяючи спільні ресурси;
- Здатність до заміщення (replaceability) - властивості ПС, які спричиняють її здатність використовуватись замість інших специфікованих ПС в середовищі їх застосування..

Підхарактеристика «**Відповідність**» не приведена і списку, але вона належить усім характеристикам. Ця характеристика має відображати відсутність протиріч з іншими стандартами або характеристиками. Наприклад, відповідність надійності і практичності.

Кожна під характеристика якості далі розбивається на атрибути. Атрибут – це сутність, яка може бути перевірена або виміряна в програмному продукті. Атрибути не визначені в стандарті із-за їх різноманітності в різних програмних продуктах.

В стандарті виділена модель характеристик експлуатаційної якості. Основні:

- Результативність – ступінь в якій користувачами досягаються задані цілі по точності і повноті розв'язування задач у встановленому контексті використання ПС
- Продуктивність – продуктивність при розв'язанні основних задач програмної системи, яка досягається при реально обмежених ресурсах в конкретному зовнішньому середовищі застосування.
- Безпека – надійність функціонування комплексу програм і можливий ризик від його застосування для людей, бізнесу і зовнішнього середовища.
- Задоволення потреб і витрат користувачів у відповідності з цілями застосування ПЗ.

Друга частина стандарту ISO 9126-2 присвячена формалізації зовнішніх метрик. Зовнішні метрики відносяться до атрибутів, які визначають поведінку ПЗ у зовнішньому середовищі і взаємодію з іншими програмами.

Третя частина стандарту ISO 9126-3 присвячена формалізації внутрішніх метрик. Такі метрики вимірюють атрибути внутрішніх характеристик ПЗ.
Метрика – це комбінація конкретного методу вимірювання (способу отримання значень) атрибуту сутності і шкали вимірювання (засобу, який використовується для структурування отриманих значень).
Викладені загальні рекомендації по використанню відповідних метрик і взаємозв'язку між типами метрик.
Четверта частина стандарту ISO 9126-4 призначена для покупців, постачальників, розробників, супроводжуючих, користувачів і менеджерів якості ПЗ. В ній повторена концепція трьох типів метрик, а також анотовані рекомендовані види вимірювань характеристик ПЗ.

Лекція 3. Серія тестів

Перший цикл тестування.

Отримуємо програми і наступний опис її функціонування:

Призначення програми – скласти два введених користувачем числа. В кожному з чисел може бути одна або дві цифри. Програма відображує введені числа і після цього виводить їх суму. Введення кожного числа закінчується натисненням клавіші Ентер. Запускається програма з допомогою команди ADDER (adder.exe).

КРОК 1. Простий і найбільш очевидний тест

В програмах, представлених для першого формального тестування, часто одразу виникає збій. Тому варто виконати найпростіший тест.

Дія	Що вібувається
Вводимо ADDER і натискаємо <Ентер>	Зверху екрану з'являється знак питання. Курсор мигає
Натискаємо 2	За знаком питання з'являється цифра 2
Натискаємо <Ентер>	В наступному рядку з'являється знак питання
Натискаємо 3	За другим знаком питання з'являється цифра 3
Натискаємо <Ентер>	В третьому рядку з'являється цифра 5. На декілька рядків нижче з'являється ще один знак питання

Елементарний тест програма проходить і отже вона робоча. Але проблеми все одно присутні.

Звіт про проблеми першого тесту.

1. Помилка проектування. Нема вказівок на те з якою саме програмою ви працюєте
2. Помилка проектування. На екрані нема жодних інструкцій. Звідки знати що робити? Відобразити інструкцію на екрані нескладно, а друкована документація може загубитись.
3. Помилка проектування. Як зупинити виконання програми? Ця інструкція має бути на екрані.
4. Помилка кодування. Цифра 5 виведена в стороні від доданків.

Обов'язково необхідно представляти детальний звіт про кожну проблему. Можна згрупувати помилки в один звіт, але цього робити не варто.

КРОК 2. Що ще має бути протестовано

Необхідно скласти серії тестів. Їх краще записати. Ці записи в подальшому приймуть вигляд формалізованих описів тестів. Такі документи можуть в подальшому використовуватись для перевірки наступних версій програми.

Приклад серії тестів.

Вхідні дані	***Очікуваний результат***	***Замітки***
99 + 99	198	Пара найбільших чисел які може скласти програма

-99 + -99	-198	В документації не сказано, що неможна скаладати від'ємні числа
99 + -14	85	Перше велике число може вплинути на інтерпретацію системою другого
-38 + 99	61	Перевіримо додавання від'ємного і додатного чисел
56 + 99	155	Друге велике число може вплинути на інтерпретацію системою першого
9 + 9	18	9 – найбільше число з однієї цифри
0 + 0	0	Програми часто злітають на нулях
0 + 23	23	Програма може неправильно обробляти 0 і тому його треба перевірити в якості обох доданків
-78 + 0	-78	

Загальна кількість тестів 39601. Допустимий діапазон (-99,99) - усього 199. Отже, 199^2. Це без врахування будь-яких складних дій користувача (наприклад нанесення бекспейсів і делете).
Ефективніше усього перевіряти граничні умови. Якщо для двох тестів очікується однаковий результат, то тести належать до одного класу. В нашому випадку 81 тест відноситься до класу «пара однозначних додаткових чисел».
Для виконання слід вибирати ті тести з класу, на яких ймовірніше усього може відбутись збій програми. Класом можна назвати групу значень, які програма обробляє однаковим чином. А граничними значеннями класу є ті вхідні дані, на яких програма міняє свою поведінку. Границю завжди слід перевіряти з обох боків. Програмісти часто переконуються, що критичний фрагмент коду працює на одному із значень і забувають це зробити на другому.

КРОК 3. Перевірка недопустимих значень.

Програму тестують тому, що вона може не працювати.
На цьому етапі слід перевірити недопустимі значення.
В нашому випадку:

1. Додаткові числа і нулі обробляються правильно.
2. Не працює жоден тест з від'ємними числами. Після вводу другої цифри комп зависає. Очевидно програма не очікує вводу від'ємних чисел

КРОК 4. Трохи тестування в режимі «вільного польоту»

На цьому етапі включається інтуїція. Це етап в очікуванні виправлення вже виявлених недоліків. Розробляти нові тести немає смислу. Тому не слід втрачати час на планування. Можна провести дослідницьку роботу. Завжди записуйте те, що ви робите і що відбувається під час дослідницьких тестів.
Приклад на нашій програмі:

Тест	***Чи цікавий тест***	***Зауваження***
100 + 100	Гранична умова: числа більша за допустимий максимум	Програма прийняла 10. Коли ви ввели другий нуль, програма повела себе так, ніби була натиснутий ентер. Те саме з другим числом. В результаті – сума 20
<Enter> + <Enter>	Відсутність введених даних	Коли ви натиснули Ентер, програма надрукувала 10, останнє введене вами число
123456 + 0	Побільше цифр	Програма прийняла перші дві цифри як і у випадку з цифрою 100. В майбутній версії програма має працювати з більшими числами. Як тоді вона має реагувати на такі ситуації?

1,2 + 5	Десятковий знак	Реакція на десятковий знак така ж як і на Ентер
A + b	Недопустимі символи	Ви натиснули Ентер – після <A> програма зависла
<Ctlr>+<A> + <Ctrl>+<B>, <F1>+<Esc>	Керуючі символи і функціональні клавіші часто є джерелами пролем	Для усіх комбінацій клавіш програма виводить графічні символи, потім після натиснення на Ентер зависає

КРОК 5. Підсумки про недоліки програми

1. У програми дуже обмежений інтерфейс
2. Програма не працює з від'ємними числами. Найбільша обчислювана сума – 198, найменша – 0.
3. Третій ввідний символ програма інтерпретує як натиснутий Ентер
4. Поки не натиснуто Ентер будь-які символи сприймаються як допустимі
5. Програма не перевіряє чи дійсно введена цифра.

Якщо програміст не зовсім некомпетентний, то для таких результатів має бути причина. Скоріш за все програміст намагався зробити програму якомога меншою за розміром або швидкою.

Код обробки помилок займає пам'ять. Це ж стосується заголовків, повідомлень про помилки і інструкцій. Якщо програма дійсно має поміститись в маленьку фрагменті пам'яті то на все це місця немає. Це ж стосується часу. Час необхідне для перевірки вводу допустимих символів, для перевірки чи дійсно третя натиснута клавіша – це Ентер, і на друк повідомлень на екрані і на очистку змінних перед виконанням наступного завдання.

Для того, щоб все це вияснити не обхідно переговорити з програмістом.

Підсумки першого циклу тестування

Оскільки програма пройшла простий тест, булла розроблена серія формальних тестів для перевірки роботи з допустимими даними. Ці тести будуть використані і далі. Оскільки частину перевірок програма не пройшла, на планування подальших тестів поки доцільно не витрачати час. Замість цього було проведено ряд неформальних експериментів і виявлено, що програма дуже нестабільна. До написаних зауважень слід повернутись при наступному тестуванні програми.

ДРУГИЙ цикл тестування.

Програміст повідомив, що швидкість роботи є дуже важливою, а об'єм коду не має значення.

Програміст поставив свої резолюції.

Помилка	Резолюція
На екрані нема назви програми	Не буде виправлена
На екрані нема інструкцій	Не буде виправлена. Вивід інструкцій уповільнить роботу програми
Як зупинити програму?	Виправлена. На екрані відображається підказка «Для виходу натисніть <Ctrl>+<C>»
Сума 5 виводиться в стороні від доданків	Виправлена
Програма зависає на від'ємних числах	Виправлена. Програма буде складати і від'ємні числа
Програма інтерпретує третій введений символ як Ентер	В стадії розробки (ще не виправлена)
Збій при вводі нечислових даних	Не проблема. Коментар: не вводьте нечислові дані
Збій при вводі керуючих символів	Не проблема
Збій при натисненні функціональних клавіш	Не проблема

КРОК 1. Уважно прочитайте резолюції програміста і визначте, що треба робити, а що ні.

Із резолюцій чітко видно які тести більше проводити не треба, а які будуть замінені новими. Резолюції програміста є ключем до створення нової серії тестів

КРОК 2. Проаналізуйте коментарі до помилок, які не будуть виправлені. Можливо необхідно провести додаткове тестування.

Для того, щоб добитись виправлення помилки, необхідно продемонструвати ситуацію в якій її появлення абсолютно недопустиме. Програма зависає при введені будь-якого недопустимого символу – необхідно продемонструвати програмісту, що це неправильно.

КРОК 3. Передивіться записи першого циклу, додайте нові зауваження і переходьте до тестування.

Слід провести серію уже проведених тестів, для того щоб переконатись, що програма їх виконує. Після їх проведення ви помітили, що програма відображає підказку «Для виходу натисніть <Ctrl>+<C>» після кожної операції додавання. Можна скласти наступне зауваження:

Помилка проектування. На вивід на екран підказки тратиться зайвий час. Можна просто написати внизу екрана цю підказку перед початком роботи. Заодно і додати заголовок програми і короткі інструкції.

Оскільки програма продовжує зависати при введенні недопустимих символів, то слід описати проблему ще раз. Можливий варіант виправлення – перевіряти кожний введений символ. Недопустимі ігнорувати і виводити повідомлення про помилку.

Головне – указати, що проблема серйозна.

Хороший тестер не той, хто виявить найбільше помилок і не той, хто примусить збентежитись навіть самого першокласного програміста. Кращим є той, хто досягне виправлення найбільшої кількості помилок.

Лекція 4. Моделі життєвого циклу програмних систем

Життєвий цикл ПС визначається як «весь період існування системи від початку розробки до завершення її використання» (ДСТУ 2941-94. Розробка систем. Терміни і визначення)

ЖЦ поділяється на впорядковані стадії, основні з яких:

- Визначення потреб
- Аналіз вимог і оформлення концепції
- Розробка
- Виробництво
- Впровадження/продаж
- Експлуатація
- Супровід і підтримка
- Вилучення з експлуатації

Всередині кожної з цих стадій відбувається подальша деталізація по більш дрібним стадіям.

Моделі ЖЦ описують взаємозв’язки стадій.

Далі будемо розглядати стадії ЖЦ, які пов’язані з процесом розробки ПС, основні з яких:

- Аналіз вимог
- Проектування (попереднє і детальне)
- Реалізація
- Тестування

Найбільш відомі типи моделей ЖЦ: послідовні та ітераційні. Ці моделі на практиці можуть змішуватись, утворюючи змішані моделі ЖЦ.

Призначення моделей розробки

Моделі ЖЦ можуть використовуватись для:

- Організації, планування. Розподілення ресурсів (затрат праці і часу) і керування проектом розробки

- Організації взаємодії з замовниками і визначення складу документів (робочих продуктів), які розроблюються на кожній стадії
- Аналізу і оцінювання розподілу ресурсів і затрат на протязі ЖЦ
- Наглядного опису або в якості основи для проведення фінансових розрахунків з замовниками
- Проведення емпіричних досліджень з метою визначення впливу моделей на ефективність розробки і загальну якість програмного продукту

Рекомендації по можливому відображенню процесів ЖЦ на основні моделі розробки приведені в Керівництві по застосуванню стандарту ISO/IEC 12207 (Guide for ISO/IEC 12207 – Software life cycle processes).

Моделі послідовного виконання стадій

1. **Каскадна модель**

Каскадна (Waterfall) або стандартна модель – найбільш відома модель розробки, яка пропонується стандартами як базова. Ця модель характеризується стадіями, які виконуються послідовно. Кожна стадія має бути завершена до переходу до наступної. Створені в ній робочі продукти після їх верифікації та валідації мають бути «заморожені» і передані на наступну стадію в якості еталону. Користувач бачить працюючий програмний продукт в самому кінці розробки. Найбільш жорстке обмеження цієї моделі – необхідність «заморозки» [illegible]у, для того, щоб мінімізувати ризик, допускаються лише невеликі зміни.

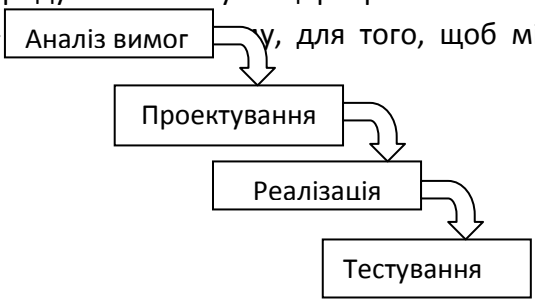


Рис 1. **Каскадна модель**

З точки зору якості ПС в цій моделі вартість виправлення дефектів на стадії тестування найбільша (у зрівнянні з іншими моделями), оскільки тестування відбувається в самому кінці розробки. Із-за нестачі часу на переробки і тестування існує значний ризик випуску ПС з серйозними дефектами.

2. **Каскадна модель із зворотнім зв'язком**

Ця модель розширює стандартну модель включенням в неї циклів зворотного зв'язку для повернення на попередню стадію при зміні вимог, проекту і по результатам інспекцій або дій по V&V

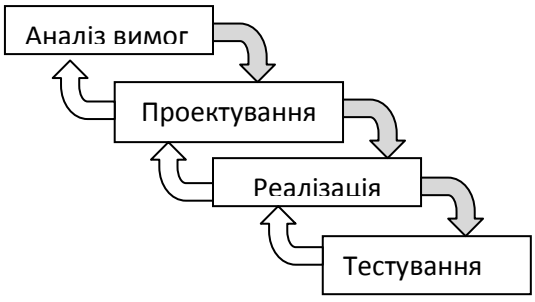


Рис 2. **Каскадна із зворотнім зв'язком**

Процеси V&V, які виконуються після завершення кожної стадії розробки, грають в цій моделі важливу роль.

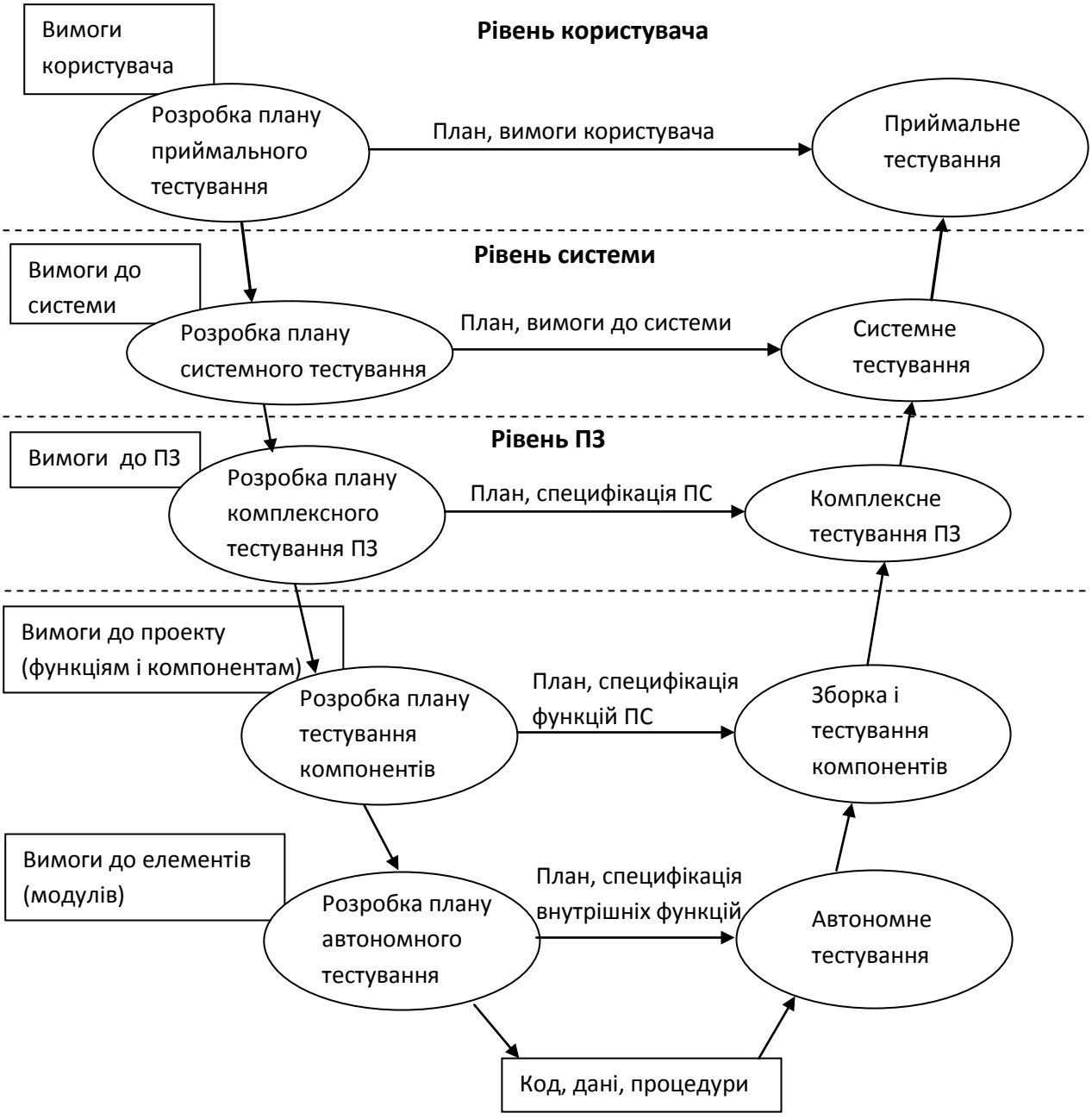
Характеристики каскадної моделі:

- Послідовне впорядкування стадій
- Формальні перевірки по завершенні кожної стадій (інспекції, технічні огляди)
- Наявність документованих вимог і проекту

Переваги каскадної моделі:

- Застосування формальних перевірок дозволяє вчасно виявляти дефекти
- Чіткі критерії початку і завершення стадій
- Чіткі вимоги і цілі проекту

3. V-подібна модель



V-подібна (V-shape) модель розширює каскадну модель і включає до неї дії по ранньому плануванню тестування.

В цій моделі тестування розглядається як неперервний процес, інтегрований в процес розробки ПС. Він включає два взаємопов'язаних під процесів – планування тестування в рамках процесів розробки системи (ліва гілка) і проведення тестування відповідних об'єктів (права гілка). На практиці задачі

тестування ПЗ і системного тестування часто об’єднуються в один процес, однак для складних ПС тестування технічних характеристик має виконуватись окремо.

Характеристики V-подібної моделі:

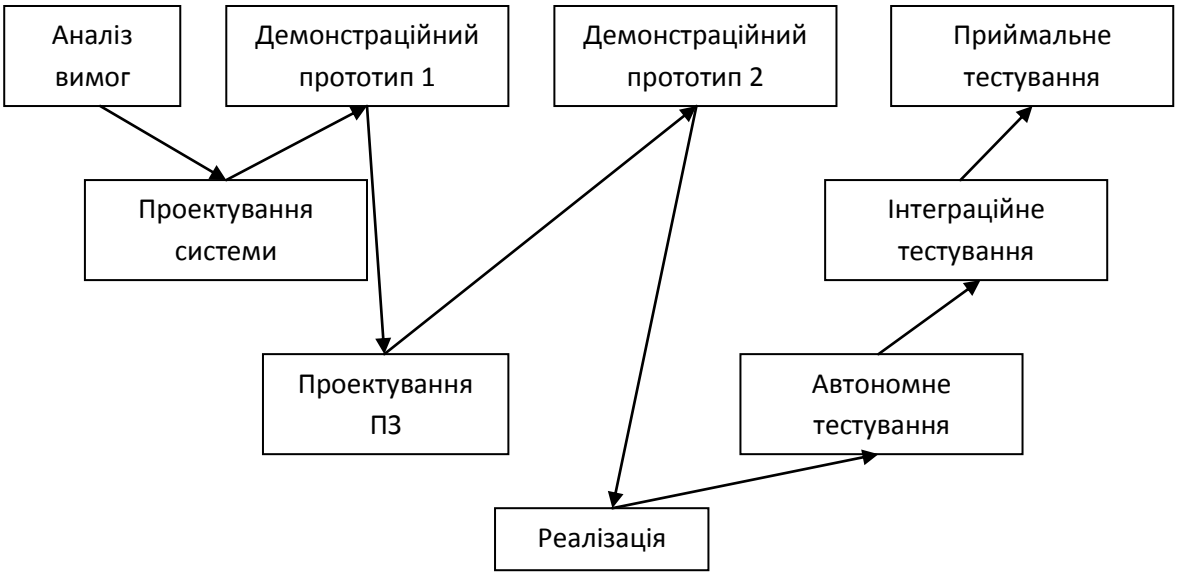
- Перевірка і оцінка тестопридатності вимог на ранніх стадіях розробки (з допомогою аналізу, який виконується під час тестування)
- Наявність документованих тестових вимог

Переваги V-подібної моделі:

- Забезпечує зворотний зв’язок з користувачем на ранніх стадіях ЖЦ
- Покращує планування і розподіл затрат на тестування
- Чіткі документовані цілі тестування

4. Каскадна модель з прототипуванням (пилоподібна модель)

Модель є модифікацією V-подібної моделі з включенням в неї прототипів для моделювання вимог і проекту



Прототипи слугують для демонстрації і після розробки проекту їх викидують, а реалізація проекту може виконуватись в іншому середовищі.

- Характеристика: для аналізу і моделювання проектних рішень застосовуються прототипи
- Переваги: усуває проблеми, пов’язані з неповнотою і нечіткістю вимог

Ризики застосування послідовних моделей:

- Вимоги не повністю зрозумілі
- Система занадто велика, щоб бути реалізованою одразу
- Швидкі зміні в технологіях
- Часта зміна вимог
- Користувач не може використовувати проміжні результати

Коли краще застосувати послідовну модель:

- Вимоги зрозумілі і не будуть суттєво мінятись
- Система має невеликий розмір і складність
- Усі можливості мають бути реалізовані одразу
- Нова система розробляється на заміну старої і необхідно повністю замінити стару систему

Ітераційні моделі

Ітераційні моделі загалом можна розділити на два класи: моделі з приростом (Incremental) і еволюційні (Evolutionary)

У відповідності з цими моделями програмний продукт розроблюється ітераціями, і кожна ітерація закінчується випуском працездатної версії програмного продукту. Основна відмінність між моделями – підхід до визначення вимог.

1. **Ітераційні моделі з приростом**

Програмний продукт розробляється ітераціями – з додаванням на кожній функціональних можливостей. При цьому спочатку визначаються усі вимоги до ПС, і можливо розроблюється попередній проект. Подальша розробка ПС розбивається на ітерації. В першій ітерації реалізується набір основних вимог, які забезпечують базову функціональність. Інші ітерації реалізуються в порядку критичності вимог для кінцевого користувача. При появі в середині ітерації нового набору вимог. вони відкладаються до реалізації наступної версії. В реальному житті це допущення може порушуватись і допускається перегляд вимог.

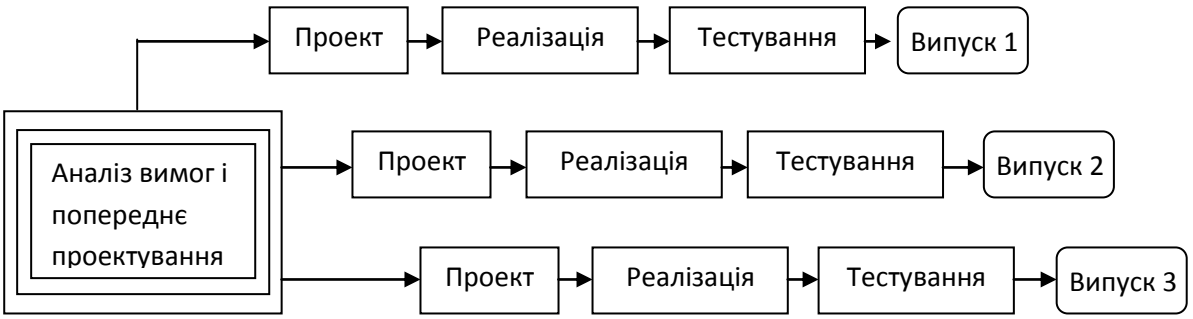


Рис 5. Ітераційна модель з перекриттям ітерацій

В різних моделях цієї групи ітерації можуть виконуватись послідовно або з перекриттям (нова ітерація починається до завершення попередньої).

Ітераційні моделі з приростом широко використовуються для розробки комерційних програмних продуктів, які розвиваються на протязі довгого періоду часу або для яких зовнішні вимоги змінюються слабо.

Характеристики ітераційних моделей з приростом:

- Аналіз і проектування виконуються для усієї системи
- Базові функціональні вимоги реалізуються першими
- Інші вимоги реалізуються в наступних версіях
- Проміжні версії придатні для використання

Переваги ітераційних моделей з приростом:

- Критичні функції реалізуються в першу чергу
- Критичні функції тестуються більш ретельно
- Найменш критичні задачі реалізуються останніми, що мінімізує наслідки відмови із-за дефектів
- Завершення першої версії остаточно затверджує вимоги і проект
- Раннє планування виконання тестування
- Раннє виявлення дефектів користувачами

Ризики, пов’язані з вибором моделі:

- Вимоги не повністю зрозумілі
- Вимоги не стабільні
- Усі можливості мають бути реалізовані одразу
- Швидкі зміни в технології

Коли краще застосовувати модель:

- Вимагається швидка реалізація основних можливостей

- Якщо проект системи можна природним чином поділити на незалежні частини

2. Еволюційні моделі

На відміну від моделей з приростом, еволюційні моделі застосовуються в тих випадках, коли усі вимоги не можуть бути визначені одразу або відомо, що вони можуть змінитись. Розробка проекту по цим моделям також виконується ітераціями. Але кожна ітерація охоплює усі стадії розробки, від аналізу вибраного набору вимог до випуску версії. На кожній ітерації виконується прототипування вимог і проекту.

До найбільш відомих еволюційних моделей відносяться спіральна модель і модель еволюційного прототипування.

1. Спіральна модель

Розроблена Боемом. Відображає керований ризиком процес еволюції проекту від аналізу до готовності продукту.

На кожному витку спіралі (стадії) виконуються наступні дії:

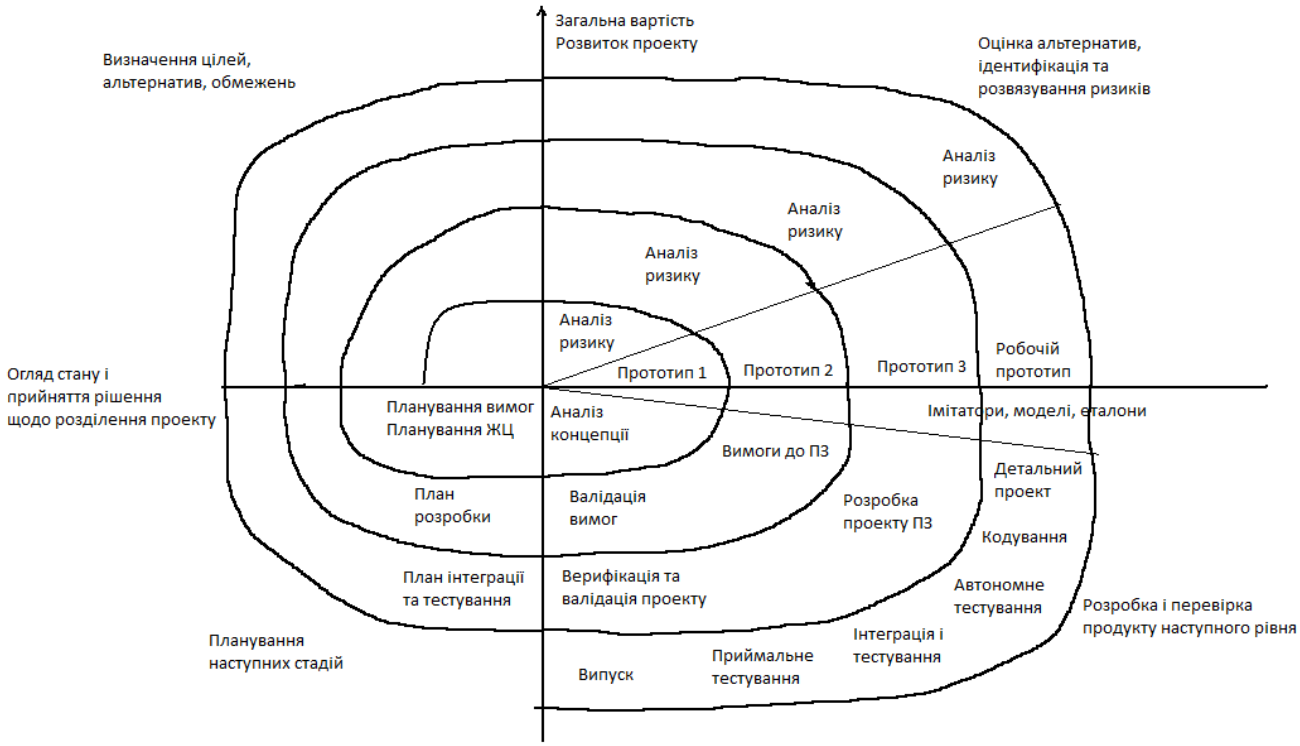
1. Визначаються цілі стадії. Розглядаються альтернативні рішення для досягнення цих цілей
2. Проводиться оцінювання цих рішень. Ідентифікуються ризики завершення стадії і виконується їх аналіз. Приймаються рішення про продовження або завершення стадії
3. Розробляються робочі продукти стадії та план для наступної стадії
4. Останній виток спіралі може мати структуру каскадної моделі.

Види розглядуваних ризиків – ризики, які стосуються технічних аспектів розробки, фінансові ризики, ризики експлуатації.

Спіральна модель застосовується для складних проектів або в тих випадках, коли проблеми проекту недостатньо зрозумілі.

Характеристики спіральної моделі:

- Перший прототип моделює концепцію. Результатом є план вимог. Перед переходом до розробки наступного прототипу виконується аналіз ризику
- Другий прототип моделює вимоги до ПЗ. Результатом є план розробки. Виконується аналіз ризику
- Третій прототип моделює проект. В результаті створюється інтегрований і протестований прототип. Виконується аналіз ризику.
- Останній прототип (робочій) використовується як основа для детального проектування, кодування та тестування.



Ризики пов'язані з вибором моделі:

- Усі можливості мають бути реалізовані одразу
- Проект неможна природним чином розділити на незалежні частини

Коли краще застосовувати модель:

- Проект крупний, складний і вимоги не можуть бути визначені одразу
- Нова технологія і вимагається її вивчення
- Користувачі не можуть чітко сформулювати вимоги
- Вимагається рання демонстрація можливостей

Подальшим розвитком цієї моделі є Win-Win Spiral Model, яка основана на залученні до розробки різних категорій учасників проекту і визначенні умови успіху системи, які обговорюються на кожній ітерації.

Модель еволюційного прототипування.

Ця модель основана на застосуванні еволюційного прототипування в рамках усього ЖЦ розробки (а не тільки моделювання вимог). В літературі вона часто називається моделлю швидкої розробки програм (RAD – Rapid application development). Моделювання включає наступні кроки:

Крок	Дія
1. Аналіз застосовуваності моделі	Вивчення можливості застосування моделі для проекту
2. Обстеження замовника	Вивчення потреб користувача та розробка плану створення прототипу
3. Ітерація розробки функціонального прототипу	Створення і узгодження прототипу інтерфейсу користувача. Визначення не функціональних вимог і стратегії реалізації системи
4. Ітерація проектування і побудови	Побудова протестованої системи, яка задовольняє усім функціональним та не функціональним вимогам. На кроках 3 і 4 розробники визначаються прототипи, узгоджують строки розробки, побудову і перевірку прототипів. Ці кроки виконуються

	ітеративно і включають три ітерації: початкове ознайомлення, уточнення. Узгодження
5. Реалізація	Встановлення системи в середовищі замовника, розробка документації і навчання

Ця модель застосовується для розробки не критичних бізнес-програм. Для яких найбільш важливими є функціональні можливості. Її застосування передбачає тісну взаємодію розробника і користувача.

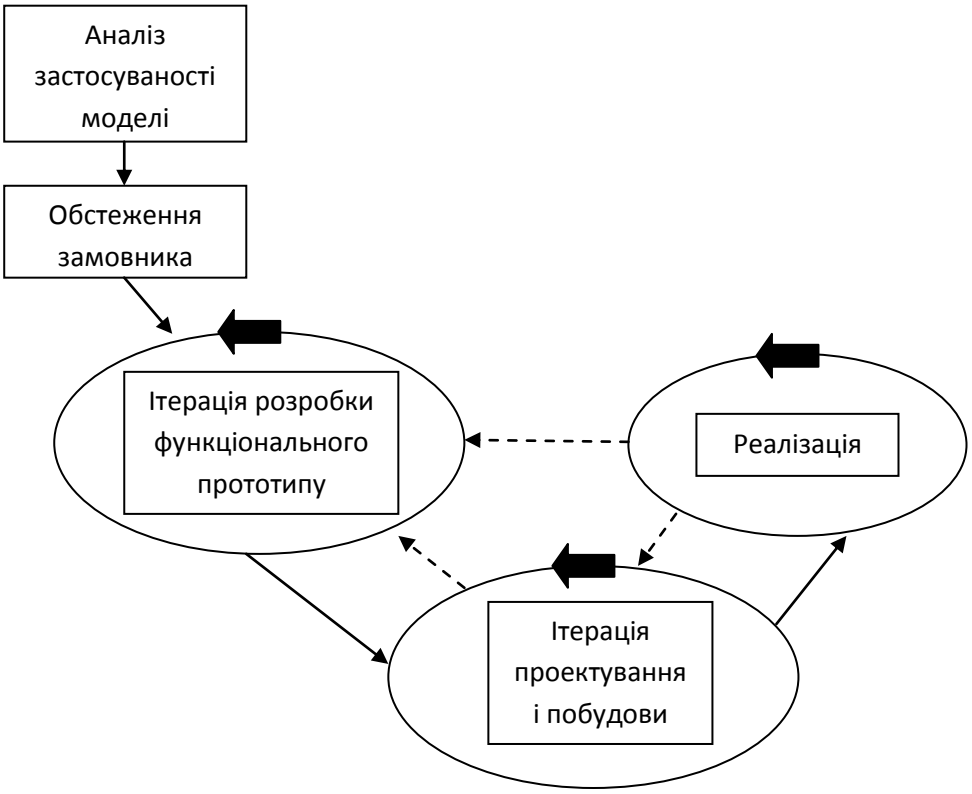


Рис. Модель еволюційного прототипування

Характеристики моделі:

- Гнучкість. Можливість швидко реагувати на зміни і розширення вимог
- Пріоритети функціональних характеристик перед технічними (якості)

Переваги моделі:

- Раннє виявлення дефектів в інтерфейсі
- Швидка демонстрація функціональних можливостей

Ризики пов’язані з вибором моделі:

- Від розробника вимагається хороше володіння CASE-засобами та інструментами
- Програми не моє бути критичною
- Вимагається наявність потужних CASE-засобів

Коли краще застосовувати модель

- Користувачі не можуть чітко сформулювати вимоги
- Вимагається рання демонстрація можливостей

Широко розповсюджені такі основні класи моделей ЖЦ:

- Каскадні
    - Стандартна
    - Із зворотнім зв’язком
    - Пилоподібна
- Ітераційні

- З прирощуваннями
- Еволюційні
  - Спіральна
  - Швидкої розробки програм (RAD)

Вибір моделі суттєво залежить від двох факторів:

А) чи можна спочатку визначити практично повний набір функцій, які необхідно реалізувати в програмному продукті

Б) чи мають усі жадані функції поставлятись замовнику одночасно

Якщо А і Б, то вибираємо каскадні моделі

А і не Б – вибирається ітераційна модель з прирощуваннями

Не А і Б, а також бажана розробка прототипів для моделювання вимог – спіральна модель

Не А і не Б – модель швидкої розробки програм, при умові, що строки розробки не будуть чітко встановлені.

Лекція 5. Технічне завдання

ДСТУ 19.201-78 «Технічне завдання. Вимоги до змісту та оформленню»

Стандарт встановлює порядок побудови і оформлення технічного завдання на розробку програми або програмного виробу для обчислювальних машин, комплексів та систем незалежно від їх призначення та області застосування.

ТЗ має містити наступні розділи:

- Вступ
- Підстави для розробки
- Призначення розробки
- Вимоги до програми
- Вимоги до програмної документації
- Техніко-економічні показники
- Стадії та етапи розробки
- Порядок контролю і прийому

ТЗ може містити додатки.

В залежності від особливостей програми допускається уточнювати зміст розділів, вводити нові розділи або об’єднувати окремі з них.

ЗМІСТ РОЗДІЛІВ

1. В розділі «Вступ» вказують найменування, коротку характеристику області застосування програми і об’єкту, в якому будуть застосовувати програму
2. В розділі «Підстави для розробки» мають бути вказані:
   - Документи, на підставі яких ведеться розробка
   - Організація, яка затвердила цей документ і дата затвердження
   - Найменування і умовне позначення теми розробки
3. «Призначення розробки» - функціональне і експлуатаційне призначення програми
4. Розділ «Вимоги до програми» має містити наступні підрозділи:
   - Вимоги до функціональних характеристик – вимоги до складу виконуваних функцій, організації вхідних та вихідних даних, часовим характеристикам тощо…
   - Вимоги до надійності – вимоги до забезпечення надійного функціонування (контроль вхідної та вихідної інформації, час відновлення після збоїв)

- Умови експлуатації – температура повітря, вологість та інші для вибраного типу носіїв даних, при яких мають забезпечуватись задані характеристики, а також вид обслуговування, кількість та кваліфікацію персоналу;
- Вимоги до складу і параметрам технічних засобів – вказують необхідний склад технічних засобів із вказуванням їх основних технічних характеристик
- Вимоги до інформаційної та програмної сумісності – вимоги до інформаційних структур на вході та виході, методам розв'язку, кодам програми, мовам програмування і програмним засобам, які використовує програма. При необхідності має забезпечуватись захист інформації та програм
- Вимоги до маркування та пакування
- Вимоги до транспортування та зберігання
- Спеціальні вимоги

5. В розділі «Вимоги до програмної документації» має бути вказаний попередній склад програмної документації і спеціальні вимоги до неї
6. В розділі «Техніко-економічні показники» мають бути вказані:
    - Орієнтовна економічна ефективність
    - Приблизна річна потреба в програмі
    - Економічні переваги розробки у зрівнянні з аналогами
7. В розділі «Стадії та етапи розробки» встановлюють необхідні стадії розробки, етапи і зміст робіт (перелік програмних документів, які мають бути розроблені, узгоджені і затверджені), строки розробки і виконавців

В додатках до ТЗ за необхідністю приводять:

- Перелік науково-дослідницьких та інших робіт, які пояснюють необхідність розробки
- Схеми алгоритмів, таблиці, розрахунки та інші документи, які можуть бути використані при розробці
- Інші джерела розробки

Лекція 5_2. **Тестування «білої скрині»**

Програміст пише програми і сам їх тестує. Ця технологія називається тестуванням «білої скрині» (white box) або «скляної скрині» (glass box).

Переваги тестування «білої скрині»:

1. **Направленість тестування**. Програміст може тестувати програму по частинах, розробляти спеціальні тестові підпрограмки, які визивають модель, що тестується, і передають йому необхідні дані. Окремий модуль завжди легше протестувати саме як «білу скриню»
2. **Повне охоплення коду**. Програміст завжди може визначити, які саме фрагменти коду працюють в кожному тесті. Він бачить які гілки залишились не протестованими і може підібрати умови при яких вони будуть виконані
3. **Керування потоком**. Програміст завжди знає, яка функція має виконуватись в програмі наступною і яким має бути її поточний стан. Тут програміст може користуватись таким зручним засобом як відладчик
4. **Відстежування цілісності даних**. Програмісту відомо, яка частина програми має змінювати кожен елемент даних. Відстежуючи стан даних, він може виявити такі помилки як зміна даних не тими модулями, їх неправильна інтерпретація або невдала організація. Програміст також може самостійно автоматизувати тестування.
5. **Внутрішні граничні точки**. В коді видно ті граничні точки програми, які скриті від погляду тестерів «чорної скрині». Наприклад, для виконання певної дії можна використати декілька алгоритмів і

невідомо який з них вибрав програміст. Переповнення буфера – програміст одразу може сказати при якій кількості даних виникне ця помилка.

6. **Тестування, яке визначається вибраним алгоритмом**. Для тестування обробки даних, яка використовує складні обчислювальні алгоритми, можуть знадобитись спеціальні технології.

Тестування «білої скрині» розглядають як частину програмування. Програміст виконує цю роботу постійно.

Структурне тестування є одним з видів тестування «білої скрині». Головна ідея – вибір правильного програмного шляху. Структурне тестування має потужну математичну теоретичну основу, але більшість тестерів використовують функціональне тестування.

Тестування програмних шляхів: *критерії охоплення*.

Протестувати усі програмні шляхи неможливо. Тому серед програмних шляхів виділяють ті, що необхідно протестувати обов'язково. Для відбору використовуються *критерії охоплення* (coverage criteria). Їх також називають *логічними критеріями охоплення* або *критеріями повноти*.

Найчастіше використовуються критерії:

- Охоплення рядків
- Охоплення розгалужень
- Охоплення умов

Коли тестування узгоджується з цими критеріями кажуть *про тестування шляхів*.

Критерій охоплення рядків – найслабший. Він вимагає, щоб кожен рядок програми був виконаний хоча б один раз. Він не підходить вже коли зустрічається хоча б один оператор умови.

Наприклад :

```
If (a < b and c = 5)  then
        Do domething;
```

Для перевірки коду необхідно проаналізувати 4 варіанти:

1) a < b and c = 5 : (do something виконується)
2) a < b and c ≠ 5 : (do something не виконується)
3) a ≥ b and c = 5 : (do something не виконується)
4) a ≥ b and c ≠ 5 : (do something не виконується)

Для виконання коду необхідно перевірити тільки 1-й варіант

При тестуванні за критерієм охоплення розгалужень програміст перевіряє варіант 1 і ще один з трьох інших варіантів.

Найбільш суворим є критерій охоплення умов. При такому необхідно перевірити усі 4 варіанти.

Критерії охоплення корисні але їх недостатньо. Приклад

A = B/C

Якщо C ≠ 0, той цей код успішно виконується і критерії охоплення виконуються. Але при C = 0 програма припинить виконання. Різниця в цих двох варіантах не в шляху, а в даних.

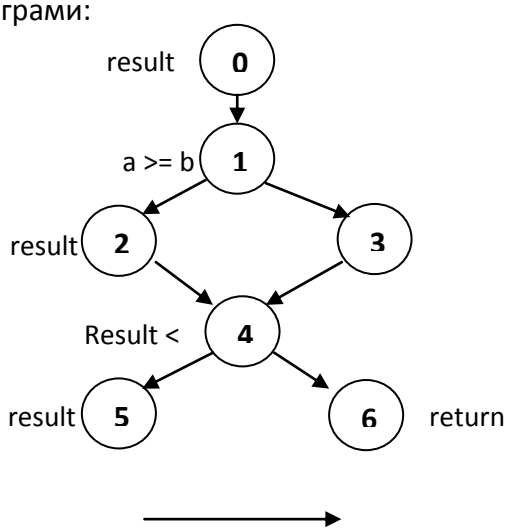
Програмний шлях називається чутливим до помилок, якщо при його проходженні помилки можуть проявитись. Якщо ж помилки обов'язково проявляться в проходженні даного шляху, то такий шлях називається таким, що знаходить помилки.

Приклад тестування потоків керування

```
Int TestFunction(int a, int b)
{
        int result = 0;
        if(a >= b)
        result = a-b;
        else
```

```
        result = b-a;
        if(result < 5)
        result = 0;
        return result;
}
```

Керуючий граф програми:



Тестовий набір, сформований згідно з критерієм покриття команд, має вигляд (X, $Y_{эт}$) = {(9,5,0), (5,9,0)}. В тестованій по даному набору програмі виконуються усі команди і виконуються наступні шляхи:

$$M = \begin{cases} 0 \to 1 \to 2 \to 4 \to 5 \to 6 \\ 0 \to 1 \to 3 \to 4 \to 5 \to 6. \end{cases}$$

Тестовий набір, сформований згідно з критерієм покриття розгалужень має вигляд (X, $Y_{эт}$) = {(9, 5, 0), (0, 255, 255)}. В тестованій по даному набору програмі виконуються усі розгалуження і виконується наступна множина шляхів:

$$M = \begin{cases} 0 \to 1 \to 2 \to 4 \to 5 \to 6 \\ 0 \to 1 \to 3 \to 4 \to 6. \end{cases}$$

Тестовий набір, сформований згідно з критерієм покриття маршрутів, має вигляд (X, $Y_{эт}$) = {(9, 5, 0), (0, 255, 255), (255, 0, 255), (5, 9, 0)}.

$$M = \begin{cases} 0 \to 1 \to 2 \to 4 \to 5 \to 6 \\ 0 \to 1 \to 3 \to 4 \to 6 \\ 0 \to 1 \to 2 \to 4 \to 6 \\ 0 \to 1 \to 3 \to 4 \to 5 \to 6. \end{cases}$$

Лекція 6. Парадигма якості в програмній інженерії

1.1 Основні поняття в області якості

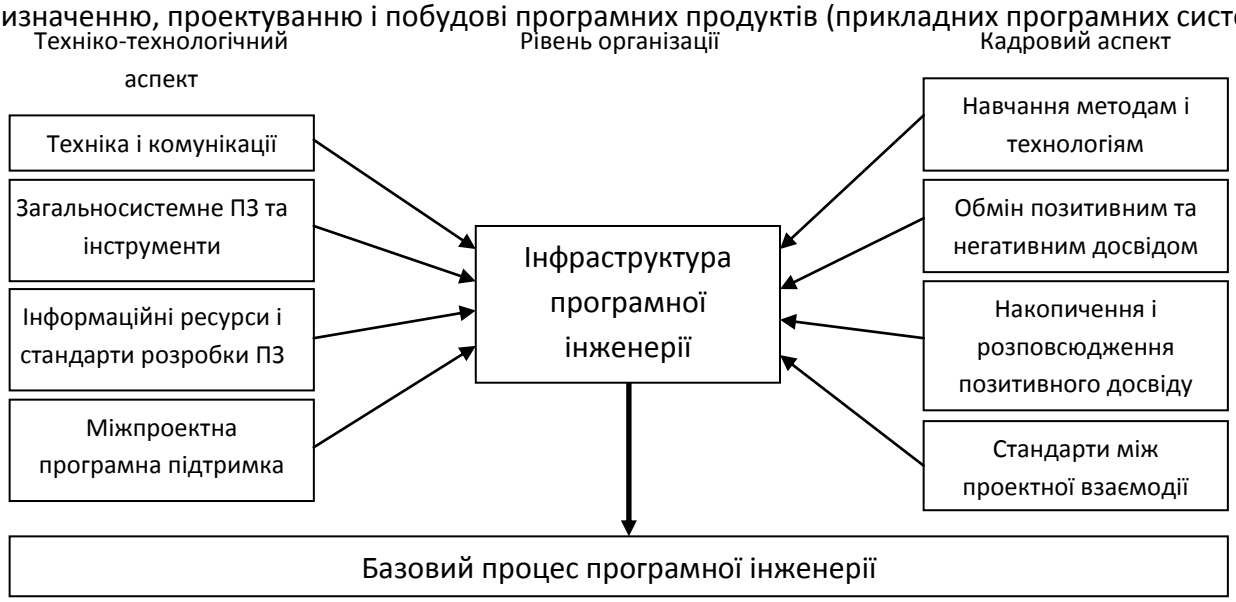
Програмна система (ПС) – група інтегрованих програмних засобів, які підтримують певний діловий процес споживача (або його частину) і використовують загальне сховище даних.

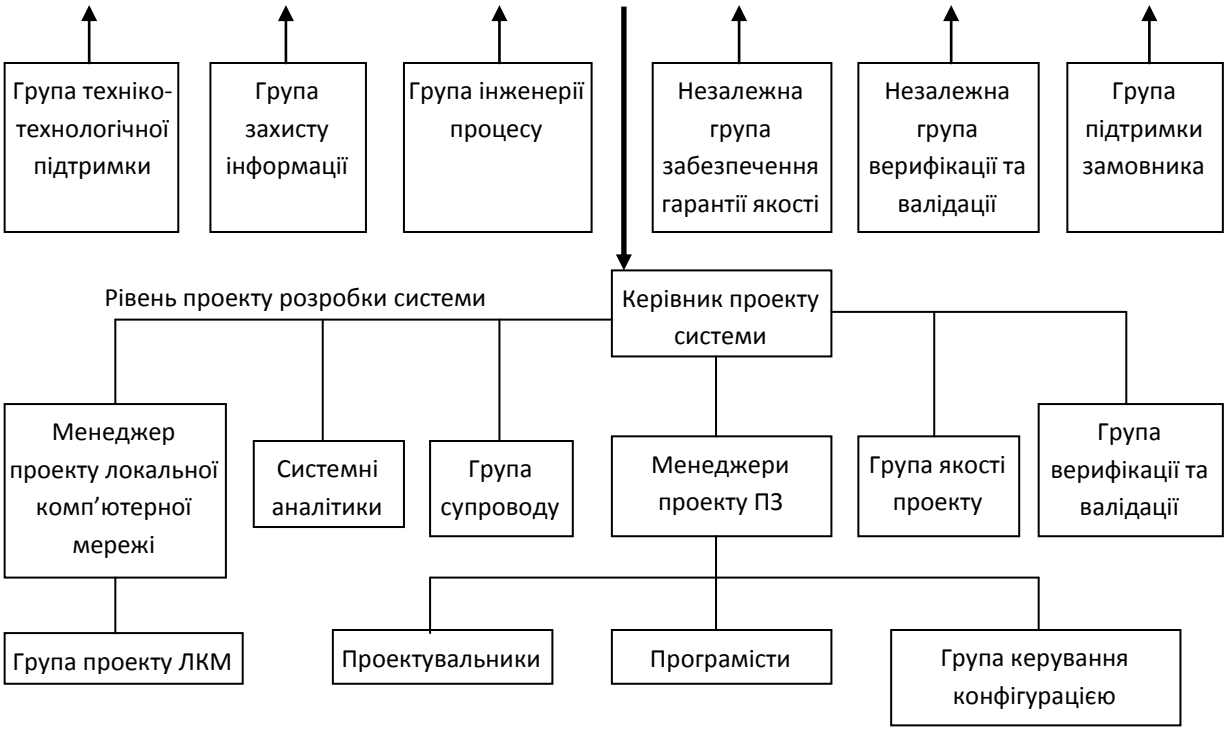
Термін ***програмне забезпечення*** (ПЗ) використовується застосовано до сукупності програмних засобів, які розробляються з метою експлуатації у складі системи.

Основа якісної розробки програмних систем – раціональна інфраструктура програмної інженерії як виду бізнесу. Спеціалісти-практики в області програмних систем і користувачі сходяться в поглядах на поняття поганого програмного продукту як такого, що:

- Не забезпечує підтримку стратегії бізнесу або потреб користувача
- Недостатньо надійний, гнучкий, ефективний і погано супроводжується
- Є коштовним і занадто довго розробляється

Інфраструктура програмної інженерії – інтегрований набір загальнодоступних технічних, технологічних і методологічних ресурсів організації розробника, які роблять можливим виконання процесу програмної інженерії колективами проектів, які відкриваються по договорах із замовниками. Тут ***проект*** – це обмежена часовими рамками діяльність, мета якої полягає в створенні унікального програмного продукту. ***Процес програмної інженерії*** – множина логічно пов'язаних видів діяльності по визначенню, проектуванню і побудові програмних продуктів (прикладних програмних систем).





Компоненти інфраструктури розробки ПС.

Техніко-технологічний аспект.

1. Техніка і комунікації:
   - Комп'ютери користувачів, файлові сервери
   - Локальні комп'ютерні мережі (ЛКМ)
   - Глобальна комп'ютерна мережа (ГКМ)
   - Електронна пошта
   - Техніка для тестування
   - Офісна техніка
   - Інші складові комплексу технічних засобів
2. Загальносистемне ПЗ та інструменти:
   - Клієнт\серверні технології
   - Операційні системи
   - Офісні системи
   - Системи документообігу
   - Утиліти
   - Засоби захисту інформації (антивіруси)
   - CASE-інструменти, системи програмування
   - СУБД
   - Графічні інструменти
3. Інформаційні ресурси і стандарти розробки:
   - Методології розробки
   - Інструменти керування проектами, конфігураціями
   - Системи підтримки використання ресурсів Інтернет
   - Нормативні документи, які стосуються технічних, програмних, комунікаційних засобів, даних і захисту інформації
   - Нормативні документи оформлення матеріалів
   - Методичні матеріали, шаблони і заготовки документів
4. Міжпроектна програмна підтримка
   - Розроблені програми (модулі), визнані здатними до загального користування, документовані та поміщені під контроль конфігурації.

Кадровий аспект.

1. Навчання методам і технологіям:
- Можливості організації по навчанню спеціалістів методам та прийомам розробки ПЗ
- Можливості вивчення спеціалістами техніко-технологічних компонент інфраструктури
2. Обмін позитивним та негативним досвідом:
- Культура «відкритого» сприйняття/передачі набутого досвіду, знань, характерних помилок. Сприяння розповсюдженню позитивного досвіду. Не приховування власних помилок і не перекладання відповідальності за них. Бажання навчатись/навчати
3. Накопичення і закріплення позитивного досвіду:
- Визначення форматів і засобів накопичення і зберігання здобутого досвіду (опитування, семінари тощо)
- Створення бібліотек активів організації за принципом «кращий об'єкт». Включення їх у сферу керування конфігурацією. Забезпечення доступності.
4. Стандарти міжпроектної взаємодії:
- Визначення стандартів (меж компетенції, знань) по процесам ЖЦ створюваної ПС. Уніфікація та стандартизація прийомів роботи з метою побудови і підтримки базового процесу програмної інженерії
- Профілювання знань для забезпечення замінюваності спеціалістів в проекті. Дотримання принципу «глибокі знання у вузькій сфері»

Ролі спеціалістів в організаційній структурі розробки

Ролі на рівні організації

1. Група техніко-технологічної підтримки:
   - Вивчення ринку послуг і попиту в організації відносно техніки та загальносистемного ПЗ
   - Придбання/встановлення/підтримка техніки
   - Придбання/встановлення/підтримка загальносистемного ПЗ
   - Навчання/консультаційні послуги співробітникам
   - Рекомендації по застосуванню техніки і технологій в проектах
2. Група захисту інформації:
   - Вивчення стану справ в області захисту інформації і накопичення досвіду
   - Забезпечення захисту інформації в організації
   - Перевірка захисту інформації в організації
   - Підтримка проектів в питаннях захисту інформації
3. Група інженерії процесу
   - Визначення, супровід та вдосконалення базового процесу програмної інженерії. Забезпечення нормативно-методичної підтримки виконання процесів ЖЦ. Організація та поповнення сховища (бібліотеки) активів організації
   - Допомога менеджерам проектів в адаптації базового процесу до потреб проектів. Підбір або виготовлення форм (шаблонів) документів для інженерії проектів
   - Підтримка процесу документування в проектах, зокрема виконання важких графічних робіт, оформлення документів згідно стандартів оформлення. Нормоконтроль та друк документів.
   - Міжпроектна координація в частині накопичення досвіду і організації навчання
   - Підтримка керування конфігурацією в проектах
4. Незалежна група якості (SQA-група):
   - Планування та виконання дій по контролю і гарантії дотримання дисципліни створення програмної продукції в проектах (організація перевірок робіт в контрольних точках проектів, визначених календарними планами)
   - Контроль документів і продуктів ПЗ в контрольних точках проектів на предмет дотримання діючих стандартів та інших нормативних документів, встановлених у вимогах замовника
   - Звітність безпосередньо перед керівником організації
5. Незалежна група верифікації та валідації (V&V-група):
   - Виконання функції верифікації (по домовленості з групою SQA)
   - Планування і проведення незалежного кваліфікаційного тестування інтегрованих компонент ПЗ або програмних продуктів з метою визначення їх відповідності потребам замовника
   - Координація планів робіт з менеджерами проектів відносно вимог до тестового середовища, строків і порядку передачі ПЗ на тестування
   - Представлення звітів (результатів) тестування менеджерам проектів для прийняття мір по виправленню ПЗ
   - Незалежність від менеджерів проектів в частині визначення об’ємів і методів тестування
   - Звітність перед керівником організації за дотримання порядку тестування і стан розроблених програмних продуктів
6. Група підтримки замовника:
   - Зв’язок із замовником з питань автоматизації ділових процесів
   - Підтримка процесів керування вимогами, навчання користувачів, супроводу (або допомога в їх виконанні на рівні окремих проектів)

Ролі на рівні проекту

1. Керівник проекту системи:
- Повна фінансова відповідальність за виконання проектних домовленостей перед замовником
- Керування розробкою складових створюваної продукції – проектів ПЗ, комплексу технічних засобів, засобів захисту інформації
- Відповідальність за дії виконавців проекту

2. Системні аналітики:
- Дослідження умов та потреб автоматизації діяльності організації-споживача
- Системний аналіз вимог споживача і формування концепції системи
- Контроль обґрунтованості проектних рішень, що приймаються

3. Група якості проекту:
- Контроль якості робочих продуктів, створених процесами ЖЦ (на відповідність стандартам, методикам тощо)
- Звітність тільки керівнику проекту
- Може бути відсутньою, якщо на рівні організації діє незалежна група якості

4. Група V&V проекту:
- Перевірка відповідності робочих продуктів, вироблених на певному етапі ЖЦ, вимог до них, встановлених на попередньому етапі
- Може виконувати тестування окремих компонент ПЗ, а також системне (інтеграційне) тестування ПЗ, виробленого в проекті
- Звітність тільки керівнику проекту

5. Менеджер проекту ПЗ:
- Повна відповідальність за усі проектні рішення та дії, пов'язані з розробкою ПЗ в проекті
- Підбір і контроль ресурсів проекту, а також графіка робіт
- У великих або розподілених програмних проектах може бути декілька менеджерів (по підсистемам або рівням проекту ПЗ)

6. Проектувальники:
- Прийняття і документування проектних рішень по архітектурі і функціям ПЗ. Узгодження рішень з менеджером проекту ПЗ.
- Дотримання стандартів якості (забезпечення досягнення характеристик якості)

7. Програмісти:
- Програмування або моделювання компонентів ПЗ по проектним специфікаціям, підготованих проектувальниками
- Дотримання стандартів якості при програмуванні (по зручності супроводу коду, зручності застосування програм)
- Відладка та автономне тестування розроблених компонент

8. Група керування конфігурацією:
- Виконання процесу конфігураційного керування версій ПЗ і робочих продуктів проекту ПЗ

9. Група супроводу:
- Виконання процесу супроводу версій ПЗ і робочих продуктів проекту ПЗ під час дослідної експлуатації і під час встановленого періоду супроводу
- Навчання користувачів
- Виконання процесу розв'язання проблем
- Можуть бути членами групи підтримки замовника

10. Група проекту ЛКМ:
- При розробці системи «під ключ» проектування і монтаж ЛКМ для встановлення в організації споживача
- Закупівля і встановлення КТЗ і загально-системного ПЗ, пуско-налагоджувальні дії.

Архітектура процесів життєвого циклу

Процес програмної інженерії має ієрархічну структуру і включає множину процесів ЖЦ програмної системи. Вимоги до процесів ЖЦ ПС визначає міжнародний стандарт ISO/IEC 12207, а в Україні йому

відповідає ДСТУ 3918-99. Процеси ЖЦ розподілені по трьом групам, які відображають функціональну направленість видів діяльності, які ці процеси регламентують:

- Основні процеси
- Процеси підтримки
- Організаційні процеси

Основні процеси ЖЦ:

1. Придбання
   1.1. Підготовка придбання
   1.2. Вибір постачальника
   1.3. Моніторинг діяльності постачальника
   1.4. Прийом споживачем
2. Поставка
   2.1. Участь в тендері
   2.2. Укладення договору
   2.3. Випуск продукту (релізу)
   2.4. Підтримка приймання продукту
3. Розробка
   3.1. Виявлення вимог
   3.2. Аналіз вимог до системи
   3.3. Проектування архітектури системи
   3.4. Аналіз вимог до ПЗ системи
   3.5. Проектування ПЗ
   3.6. Програмування ПЗ
   3.7. Інтеграція ПЗ
   3.8. Тестування ПЗ
   3.9. Системна інтеграція
   3.10. Системне тестування
   3.11. Інсталяція ПЗ
4. Експлуатація
   4.1. Функціональне використання
   4.2. Підтримка споживача
5. Супровід

Процеси підтримки інтегруються з будь-якими іншими процесами, розв'язуючи задачі, допоміжні по відношенню до задач цих процесів, і забезпечують якість їх розв'язання в конкретних проектах.

1. Документування
2. Керування конфігурацією
3. Забезпечення гарантії якості
4. Верифікація
5. Валідація
6. Сумісний перегляд
7. Аудит
8. Керування розв'язанням проблем
9. Керування запитами на зміну
10. Забезпечення застосовуваності продукту (підтримка користувача)
11. Оцінювання проекту

Організаційні процеси ЖЦ

1. Керування
   1.1. Організаційне будівництво (формування корпоративної політики, цілей, процесів, стандартів, етики)
   1.2. Управління організацією
   1.3. Управління проектом
   1.4. Управління якістю
   1.5. Управління ризиком
   1.6. Вимірювання
2. Підтримка інфраструктури
3. Вдосконалення
   3.1. Установлення процесів
   3.2. Оцінювання процесів
   3.3. Покращення процесів
4. Забезпечення трудовими ресурсами
   4.1. Управління кадрами
   4.2. Навчання
   4.3. Управління знаннями (розповсюдження знань)
5. Управління активами організації
6. Управління програмою повторного використання
7. Доменна інженерія

Базовий процес організації

Сучасною концепцією процесу програмної інженерії є побудова базового процесу організації (БПО), який розробляється, супроводжується, оцінюється і покращується подібно до того як розробляються програмні продукти.

БПО є основою для визначення процесів усіх програмних проектів. Процеси на рівні проектів розробляються шляхом адаптації БПО до характеристик конкретного проекту.

Визначення БПО – це формалізований опис складу процесів ЖЦ, з яких мають побудуватись процеси розробки в програмних проектах, а також взаємозв'язків між елементами цих процесів.

З кожним процесом розробки пов'язуються:

- *Вимоги* до процесу, які вказують, «що» собою являє процес (що він буде робити)
- *Архітектура* і проект процесу, які описують, «як» процес буде визначений (які будуть елементи процесу і як будуть пов'язані)
- *Реалізація* опису процесу в рамках організації програмного проекту (створення елементі процесу і встановл. інтерфейсу)
- *Перевірка* і затвердження визначення процесу
- *Впровадження* процесу в середовище конкретного проекту.

При побудові БПО керівники організації-розробника ПС спочатку визначають склад основних процесів із числа рекомендованих, а потім підтримуючих і організаційних процесів. В своїх рішеннях керуються доцільністю включення певного процесу БПО з позицій необхідності виконання регламентованих процесом дій та їх достатності для досягнення цілей розробки якісного програмного продукту.

В мінімальній конфігурації процесів ЖЦ, крім процесу управління проектом, в БПО обов'язково має знайтись місце для процесу перевірки, а також процесу управління конфігурацією.

Побудований БПО має підтримувати використання моделей ЖЦ, застосування яких допускається в проектах організації. Широко розповсюджені такі основні класи моделей ЖЦ:

- Каскадні
    - Стандартна
    - Із зворотнім зв'язком
    - Пилоподібна
- Ітераційні
    - З прирощуваннями
    - Еволюційні
        - Спіральна
        - Швидкої розробки програм (RAD)

Вибір моделі суттєво залежить від двох факторів:

А) чи можна спочатку визначити практично повний набір функцій, які необхідно реалізувати в програмному продукті

Б) чи мають усі жадані функції поставлятись замовнику одночасно

Якщо А і Б, то вибираємо каскадні моделі

А і не Б – вибирається ітераційна модель з прирощуваннями

Не А і Б, а також бажана розробка прототипів для моделювання вимог – спіральна модель

Не А і не Б – модель швидкої розробки програм, при умові, що строки розробки не будуть чітко встановлені.

Керування проектами

Застосовано до програмної інженерію види діяльності по управлінню утворюють дворівневу структуру: загальне організаційне управління, керування виконанням програмних проектів.

Керування проектом (в будь-якій галузі) – це область знань, навиків, інструментарію та прийомів для досягнення цілей проектів в рамках узгоджених параметрів якості, бюджету, строків та інших обмежень.

Сучасна концепція керування проектом основана на принципі інформативного вимірювання процесів ЖЦ проекту, його ресурсів і створюваних робочих продуктів.

Керування проектом включає наступні кроки:

1. Ініціація проекту і визначення його меж. Визначення вимог до проекту за допомогою застосування методів оцінювання вимог з різних точок зору: технічної, технологічної, фінансової, соціально-політичної)
2. Планування. Передбачає:
   - Вибір моделі ЖЦ проекту і оцінювання процесів ЖЦ в контексті придатності для задовільнення вимог до проекту, адаптацію БПО.
   - Ієрархічну декомпозицію задач проекту, їх специфікацію поряд з встановленими вимогами, асоційованими робочими продуктами.
   - Оцінки об'ємів робіт, трудомісткості і вартості реалізації проекту
   - Розподіл ресурсів по задачам з врахуванням графіку проекту і з позицій раціонального використання персоналу проекту, обладнання та матеріалів.
   - Керування ризиком проекту по критеріям вартості розробки, тривалості проекту і якості програмних продуктів
   - Керування якістю - застосування процедур планування та контролю якості виконання процесів і будь-яких робочих продуктів цих процесів, а також верифікація та валідація продуктів.
   - Керування планом проекту. Необхідність такого керування обумовлена часто змінюваними вимогами замовника і умовами виконання проекту. Це робить процес планування проекту ітеративним
3. Введення в дію. Передбачає виконання обраних процесів ЖЦ у відповідності з планом поряд із змінами, моніторингом та регулюванням процесів і складанням звітів для зацікавлених сторін (керівництва організації, замовників, співвиконавців)
4. Огляд і оцінка виконання проекту. Передбачає виконання всеосяжної перевірки діяльності по проекту і оцінки його руху до встановлених цілей. Проводиться у встановлених критичних точках проекту. Оцінюються не тільки результати виконання процесів, але й ефективність застосованих методів та інструментів, а також продуктивність роботи колективу проекту і можливі труднощі. При необхідності здійснюється регулювання
5. Закриття проекту. Передбачає припинення проекту після завершення процесів та досягнення цілей. Оцінюється успішність проекту по відношенню до встановлених критеріїв. ПС передається в експлуатацію.

Процес вимірювання при керуванні проектами

Сучасний рівень розвитку програмної інженерії дозволяє вивести проблему якості ПС за рамки простого питання «працює система чи ні?» формулюючи її так: «наскільки точно створена система задовольняє встановленим вимогам до її якості». Відкладувати пошук відповіді на це питання до моменту завершення розробки – занадто великий ризик як для замовника, так і для розробника. Вимірювання мають стати невід'ємною частиною ЖЦ проекту.

Вимірювання – отримання об'єктивних даних про стан продуктів, процесів та ресурсів розробки ПС з метою побудови прогнозуючих та оціночних моделей, які застосовуються для керування проектом і вдосконалення процесів організації.

Виконання вимірювань в проекті це багатокроковий процес, який включає наступні кроки:

1. Визначення цілей програми вимірювання

2. *Побудова процесу вимірювань*. Модель розроблюється таким чином, щоб забезпечити побудову точних і аргументованих відповідей на питання, підкріплених кількісними оцінками, основаних на вимірюваних величинах.

3. *Вибір базових мір*. В їх число як мінімум мають входити:
   - Розмір і складність програмного продукту, на основі яких може бути оцінена трудомісткість та вартість розробки

- Продуктивність праці окремих спеціалістів і колективу проекту в цілому
- Міри вимірювань атрибутів якості

4. *Організація збору даних для виконання вимірювань*. Дані, що збираються мають забезпечувати не тільки високі передбачувальні можливості вимірювань. Але й бути достатньо «дешевими» з точки зору їх отримання. Збір і обробка даних для вимірювань є трудомістким процесом, який вимагає підтримку керівників організації і додаткових інвестицій в розробку ПС.
5. *Застосування моделей*.

Підходи до підвищення якості програмних систем.

Перший крок полягає у впровадженні спеціально призначених для цього підтримуючих процесів, а саме процесів гарантування якості, верифікації, валідації, сумісного перегляду та аудиту.

Процес гарантування якості (процес SQA) забезпечує гарантії, що програмні продукти і процеси в життєвому циклі проекту відповідають вимогам. Ці гарантії основані на тому, що SQA планує і здійснює контроль і здійснення допомоги в тій діяльності по проекту, що безпосередньо пов'язана із «вбудовою» в програмні продукти тих властивостей, що забезпечують якість. SQA встановлює стандарти та контролює їх дотримання в ході ЖЦ. Мета SQA – встановити чому допускаються помилки і як вони можуть бути виправлені. Об'єктом досліджень SQA є процеси ЖЦ ПС, а не програмні продукти.

Процеси верифікації та валідації визначають, чи дійсно продукти певного етапу процесу розробки і супроводу ПС відповідають вимогам, які до них висуваються. Задача цих процесів – перевірити та підтвердити, що кінцевий програмний продукт буде відповідати своєму призначенню і задовольняти користувачів.

Методи які використовують як в цілях SQA так і в цілях V&V ділять на статичні та динамічні.

Статичні методи - дослідження документації, інспекції, ревізії, аналіз потоків даних, аналіз дерев подій і дерев відмов, аналіз складності алгоритмів і т.д.

До *динамічних методів* відносять тестування, імітаційне моделювання та символічне виконання.

Керування ризиком в проекті.

Ризик виникає там, де є невизначеність, пов'язана з настанням якої-небудь небажаної події і є можливість потерпіти збитки внаслідок цієї події.

Ризик проекту ПС можна визначити як можливість зниження якості кінцевого продукту, перевищення вартості його розробки, затримки завершення розробки або зриву проекту із-за неефективності і недосконалості процесів ЖЦ ПС.

Величина ризику - це добуток серйозності наслідків небажаної події в проекті на ймовірність появи цієї події.

Ефективне керування ризиком полягає в прийнятті компромісних рішень по оцінюванню трудомісткості позбавлення від визначеного ризику з однієї сторони, і величини негативного впливу цього ризику на якість робочих продуктів і процесів – з іншої.

Підвищення зрілості організації.

Зрілість організації можна охарактеризувати як ступінь чіткості (ясності) визначення, управління, вимірювання, контролю і виконання процесу розробки ПС в організації.

Модель зрілості являє собою шаблон для оформлення маленьких еволюційних кроків для покращення процесу у вигляді 5-и рівнів зрілості.

- Рівень 1 – початковий. Включений в модель тільки з метою утворення точки підрахунку для оцінювання наступних покращень процесу . Характеризується тим, що процес розробки ПС неструктурований та хаотичний, а бюджет, графік і якість розробки непередбачувані.
- Рівень 2 – повторюваний. На цьому рівні керування проектом націлено на контроль дотримання планів по вартості, тривалості і функціональності розробки. Дисципліна розробки дозволяє застосовувати відпрацьовані засоби управління неодноразово у схожих проектах. Розробка нових

проектів ведеться на основі накопиченого досвіду і у відповідності з основними стандартами в області програмування

- Рівень 3 – фіксований. Процес програмної інженерії в організації затверджений, стандартизований і документований. Впроваджена програма навчання штату розробників ПС і менеджерів. Колективи окремих проектів слідують базовому процесу розробки в організації і налаштовують його для досягнення цілей конкретного проекту.
- Рівень 4 – керований. Досягається мета кількісної оцінки якості програмних продуктів і процесу розробки в рамках єдиної програми вимірювань. Здійснюється збір і наліз даних по проектам, що дає можливість управляти ризиком проекту і при необхідності повертати процес у встановлені рамки.
- Рівень 5 – оптимізований. Забезпечується неперервне покращення процесу завдяки наявності засобів кількісної оцінки його слабких і сильних сторін. Дані про ефективність процесу розробки використовуються для проведення аналізу в цілях переходу на нові технології і вдосконалення процесу розробки в організації. Дані про нові засоби інженерії вивчаються і розповсюджуються по організації.

Лекція 7. Інженерія якості. Ядро професійних знань

Провідні зарубіжні професіональні об'єднання і виробники програмної продукції розробили визначення ядра професійних знань (Body of Knowledge – BoK). Ці знання складають предмет програмної інженерії, а також управління проектами створення промислової продукції. Ними підготовані два керівництва (Guide), відповідно *Guide for SWEBOK* та Guide *for PMBOK*.

В області якості ПС подібне ядро знань не має чіткіх обрисів і документально не оформлено.

Термін *програмна інженерія* чітко затвердився на початку 70-х років з виходом в світ першого професійного журналу в цій області – Transactions on Software Engineering.

Програмна інженерія – область комп'ютерних наук, яка вивчає питання побудови комп'ютерних програм як інженерної регламентованої діяльності колективів розробників. Це інженерна дисципліна, яка охоплює усі аспекти створення ПЗ. Починаючи від формування вимог і закінчуючи його супроводом аж до зняття з експлуатації.

Виникнення і високі темпи розвитку програмної інженерії визначаються такими факторами:

- Накопичення значного об'єму знань в області практичного створення ПЗ, які потребують систематизації
- Поява різноманітних методів аналізу, моделювання і проектування ПЗ, а також високо технологічних засобів та інструментів розробки ПЗ, не забезпечених рекомендаціями по ефективному використанню
- Високий рівень виявлення дефектів в ПЗ, не дивлячись на використання прогресивних методів проектування і програмування.
- Не ефективна організація праці колективів розробників ПЗ (менеджерів, проектувальників, програмістів, тестерів, технологів та інших)
- Використання готових програмних компонентів, які підлягають ідентифікації та систематизованому веденню.
- Застосування ре інженерії існуючих компонентів як засобів їх адаптації до нових умов і середовищам, що швидко змінюються.

Один і метрів програмної інженерії Джексон визначив золоте правило програмування так: ***будь-яка тільки що завершена програмна система одразу потребує змін***.

Значні зусилля направлені на перетворення програмної інженерії в інженерну спеціальність.

Підтвердженням цього є створення ядра swebok, різноманітних програм навчання, інститутів і комітетів, міжнародних професійних об'єднань в області інформатики.

Практика спеціалізації професійної діяльності дозволяє рахувати професію «зрілою» тоді, коли для неї існують:

- Система початкового навчання спеціальності
- Механізми розвитку умінь і навиків персоналу, які необхідні для його практичної діяльності
- Ліцензування спеціалістів, організоване під керівництвом відповідних державних органів
- Системи професійного підвищення кваліфікації персоналу та відстежування сучасного рівня знань і технологій по спеціальності для того, щоб спеціалісти могли вижити в умовах інтенсивного розвитку спеціальності
- Етичний кодекс спеціалістів
- Професійне об'єднання.

Програмна інженерія як дисципліна тісно пов'язана з суміжними дисциплінами: комп'ютерні науки, математика, менеджмент, когнітивні науки, керування проектом , телекомунікації та мережі, електротехнічна інженерія та інші інженерні дисципліни.

Ядро знань по програмній інженерії (SWEBOK)

Для створення ядра знань по програмній інженерії в 1993 році сумісними зусиллями ACM (Association for Computing Machinery) та IEEE був створений спеціальний комітет SWECC (Software engineering coordination committee). В рамках цього комітету були організовані групи по наступним напрямам досліджень:

1. Визначення необхідного ядра знань і рекомендованих практичних засобів діяльності в програмній інженерії
2. Визначення норм професійної етики і стандартів з програмної інженерії
3. Визначення програм навчання студентів ВНЗ із спеціальності.

Ядро SWEBOK складають знання з десяти різних областей знань:

1. Програмні вимоги
2. Проектування (дизайн) ПЗ
3. Конструювання ПЗ
4. Тестування ПЗ
5. Супровід ПЗ
6. Керування конфігурацією ПЗ
7. Керування інженерією ПЗ
8. Процес інженерії ПЗ
9. Інструменти та методи інженерії ПЗ
10. Якість ПЗ

Кожній області знань присвячена окрема глава, яка структурована по розділам та рубрикам. Глави завершуються об'ємними списками літератури по предмету, яка по суті і являє матеріал, що утворює ядро знань.

Програмні вимоги:

- Основи вимог
- Процес інженерії
- Витяг вимог
- Аналіз вимог
- Специфікація вимог
- Перевірка вимог
- Практичні міркування

Проектування:

- Основи проектування
- Ключові питання
- Структура і архітектура
- Аналіз і оцінка якості проекту
- Нотації дизайну
- Стратегії і методи проектування

Конструювання:

- Основи конструювання
- Управління конструюванням
- Практичні міркування

Тестування:

- Основи тестування
- Рівні тестування
- Засоби тестування
- Метрики тестування
- Процес тестування
- Основи супроводу

Супровід:

- Ключові питання
- Процес супроводу
- Практичні засоби

Управління конфігурацією:

- Управління процесом
- Ідентифікація конфігурації
- Контроль конфігурації
- Облік стану конфігурації
- Аудит конфігурації
- Управління випуском та поставкою

Управління інженерією:

- Ініціювання та визначення рамок проекту
- Планування проекту
- Огляд і оцінка проекту
- Закриття проекту
- Вимірювання в інженерії ПЗ

Процес програмної інженерії:

- Реалізація і зміна процесу
- Визначення процесу
- Оцінювання процесу
- Вимірювання процесу і продукту

Інструменти і методи програмної інженерії:

- Інструменти розробки
  - Управління вимогами
  - Проектування
  - Конструювання
  - Тестування
  - Супровід
  - Управління конфігурацією
  - Управління інженерією
  - Підтримка процесів
  - Забезпечення якості
  - Інші інструменти
- Методи інженерії ПЗ
  - Евристичні методи
  - Формальні методи
  - Методи прототипування

Якість:

- Основи якості
- Процеси керування якістю
- Практичні міркування

Класифікація інструментів по SWEBOK

Інструменти роботи з вимогами:

- Засоби моделювання
- Засоби трасіровки

Інструменти проектування

- UML
- Бізнес-проектування
- Проектування БД

Інструменти конструювання

- Редактори програм
- Компілятори і генератори коду
- Інтерпретатори
- дебаггери

Інструменти тестування

- генератори тестів
- засоби виконання тестів
- інструменти оцінки тестів
- засоби керування тестами
- інструменти аналізу продуктивності

Інструменти супроводу

- засоби візуалізації
- інструменти реінженерії

Інструменти управління конфігурацією

- інструменти відслідковування дефектів і проблем
- інструменти управління версіями
- інструменти зборки та випуску

Управління інженерією

- інструменти планування та відстежування , прогнозування вартості
- Інструменти керування ризиками
- Засоби кількісної оцінки

Інструменти підтримки процесів

- Інструменти моделювання процесів
- Засоби керування процесами
- Інтегровані CASE-середовища і рольові платформи розробки
- Процес-орієнтовані середовища розробки

Інструменти забезпечення якості

- Інструменти інспекції, підтримка оглядів та аудитів
- Інструменти статичного аналізу

Додаткові аспекти

- Засоби інтеграції інструментів: програмні платформи (Java, microsoft .NET), платформи розподілених обчислень (CORBA, WebServices)
- Мета інструменти: засоби генерації інших інструментів, компілятор компіляторів тощо
- Засоби оцінки інструментів

Ядро знань по керуванню проектами (PMBOK)

PMBOK визначає 39 процесів ЖЦ проекту, об'єднаних в 5 базових груп процесів і 9 ключових областей знань, типових практично для будь-яких проектів.

Групи процесів:

1. Ініціація
2. Планування
3. Виконання
4. Моніторинг і керування
5. Завершення

Ключові області знань:

1. Управління інтеграцією проекту
2. Управління змістом проекту
3. Управління тривалістю (строками) проекту
4. Управління вартістю проекту
5. Управління якістю проекту
6. Управління людськими ресурсами
7. Управління комунікаціями проекту
8. Управління ризиками проекту
9. Управління закупівлями (поставками) проекту

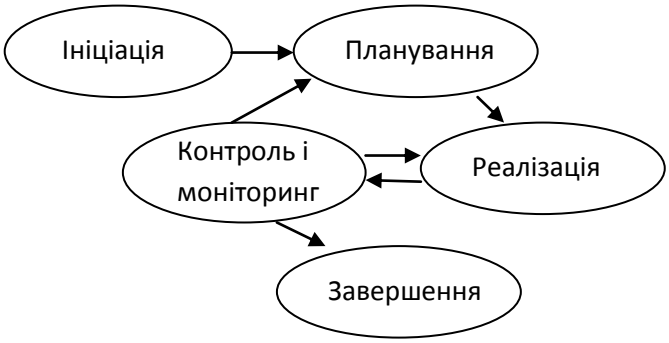


Схема взаємодії груп процесів

Парадигми та стилі програмування

Імперативне програмування (основане на машині Тьюрінга-Поста – абстрактному обчислювальному пристрої, яке виконує послідовність інструкцій програми, яка переходить із одного стану в інший):

1. Неструктурне програмування -весь код програми представлений одним неперервним блоком (bat-файли, Fortran, Basic).
2. Процедурне програмування – базується на концепції виклику процедур (підпрограм, функцій, методів). Algol, Ada, Basic, C, Cobol, Fortran, Pascal, PL/1, Matlab.
3. Модульне і збіркове програмування – строго визначені механізми вводу/виводу даних: через аргументи на вході і значення повернення на виході. Мови імперативного та об'єктно-орієнтованого програмування.
4. Структурне програмування – направлення процедурного програмування, особливості якого – структурування програми шляхом її розділення на секції одного з видів: послідовність, розгалуження, цикл. Усі сучасні мови імперативного програмування.

Подійно-кероване програмування (event-oriented, event-based, event-driven) – найбільш розповсюджена сучасна парадигма програмування.

Узгоджене програмування (concurrent programming) та ***паралельне програмування***. MPI, OpenMP.

Об'єктно-орієнтоване програмування – основане на представленні предметної області у вигляді системи взаємопов'язаних абстрактних об'єктів та їх реалізацій. Найважливіші принципи ООП – наслідування, інкапсуляція, абстракція, поліморфізм.

1. Програмування на класах: Smalltalk, C++, Java, C#, Python, PHP, Object Pascal (Delphi), VB.NET, Xbase++, UML та інші.
2. Програмування по прототипах. В цьому стилі поняття класу відсутнє, а повторне використання відбувається шляхом клонування існуючого екземпляру об'єкту – прототипу. Self, JavaScript, Squeak, Cecil, NewtonScript, Io, MOO, REBOL, Kevo.

Декларативне програмування. В такій програмі чітко формулюється ціль і результат її роботи, а не алгоритм отримання результату. HTML, XML, SQL. Domain Specific Language: DSL-мови.

1. Функціональне програмування – застосовується для розв'язку задач, які важко сформулювати в термінах послідовних операцій – розпізнавання образів, спілкування на природній мові, реалізація експертних систем, автоматизоване доведення теорем тощо. LISP, ML, Miranda, Haskell, XSLT.
2. Логічне програмування. Є ефективним для реалізації задач штучного інтелекту і для описання складних систем, наприклад диспетчерських систем.
3. Програмування в обмеженнях (constraint programming) – програмування в термінах постановок задач, де постановка задачі – кінцевий набор змінних, множин значень і набір обмежень. Більшість таких систем – це інтерпретатор мови Пролог із вбудованим механізмом для розв'язку певного класу задач (логічне програмування в обмеженнях). Constraint Logic Programming (CLP). CLP(X), X вказує на клас задач, що розв'язуються. B-Prolog, CHIP V5, Ciao Prolog, ECLiPSe, GNU Prolog. Може бути реалізовано в рамках імперативного програмування як бібліотеки (Java, C++).
4. Доказательное програмирование и синтез програм. Побудова програм паралельно з доведенням їх правильності (синтез завідомо правильних програм).

Парадигми прикладного програмування нового покоління.

Сценарна парадигма. В наш час популярність сценарних мов пов'язана з розвитком Інтернет-технологій. Скриптові мови використовуються для створення динамічних, інтерактивних веб-сторінок, зміст яких модифікується в залежності від дій користувача і стану інших сторінок і даних. Perl, Pyton, PHP, ASP.

Компонентно-орієнтоване програмування. (Component based development CBD). В основі – індустріальний підхід до розробки програмних систем, не з нуля, а шляхом швидкої зборки з готових програмних компонент. Головна ідея – розповсюдження класів в бінарному вигляді і представлення доступу до методів класу через строго визначені інтерфейси, що дозволяє зняти проблему несумісності компіляторів. COM (DCOM,COM+), CORBA, .Net.

Сервісно-орієнтоване програмування. Веб-сервіси, інтегровані за допомогою стандартних протоколів SOAP, WSDL. Open Net (Sun), .Net (Microsoft), e-services (HP), Web Services (IBM).

Аспектно-орієнтоване програмування. Мета – інструментальна підтримка програміста в чіткому поділі компонентів і аспектів за допомогою механізму, який дозволяє абстрагувати і складати компоненти і аспекти для розробки системи в цілому. AspectJ, HyperJ (Java).

Генеруюче програмування. В її основі суміщення розробки програмних компонент для забезпечення їх повторного використання і наступної розробки ПС із застосуванням компонент, що використовуються повторно. Головним елементом є не унікальний програмний продукт, а родина продуктів. Елементи родини не створюються з нуля. А генеруються на основі загальної генеруючої моделі.

Агентно-орієнтоване програмування. Інтелектуальний програмний агент – сутність, здатна формулювати цілі, навчатись, планувати свої дії і приймати рішення при обставинах, що динамічно змінюються. Простий приклад – пошук необхідних даних в Інтернет, який вимагає як правило великих

затрат часу на вибірку, аналіз і відсіювання зайвої інформації. Найбільш відомі агентні архітектури – PRS, JAM, TOURINMACHINE, COSY, INTERRAP.

Автоматне програмування. (Switch-технологія). Це стиль програмування, оснований на застосуванні кінцевих автоматів для опису поведінки програм. Автомати задаються графами переходів.

Парадигми теоретичного програмування

1. Алгебраїчне програмування базується на теорії переписування термів.
2. Інсерційне програмування. Програма в цій парадигмі розглядається як агент, який володіє поведінкою, котрий занурюючись в середовище, міняє його поведінку по відношенню до зовнішнього спостерігача.
3. Композиційне програмування, експлікативне програмування.

Лекція 8. Моделі і метрики якості програмних систем

Атрибути програмної системи, які характеризують її якість, вимірюються за допомогою метрик якості.
Метрика – це комбінація конкретного методу вимірювання атрибута і шкали вимірювання.
Метрика визначає міру – змінну, котрій присвоюється значення в результаті вимірювання.
Стандарт ISO 9126-2 визначає метрику якості програмної системи так: «модель вимірювання атрибута, який пов'язаний з деякою характеристикою якості ПС. Є індикатором одного або декількох атрибутів.
Метрику можна побачити в лівій частині рівнянь типу X=A*B» A і B – базові метрики.
Приклади метрик:

Повнота функціональної реалізації: X=1-A/B (0≤X≤1). A – число нереалізованих функцій, B – число функцій, описаних в специфікації вимог.

Точність обчислювання даних: X=A/B. A – число елементів даних, для яких забезпечується встановлений рівень точності. B – число елементів даних, для котрих в специфікації встановлені рівні точності.

Інтенсивність виявлення дефектів. $F = x_1/M(x)$. X_1 – кількість знайдених дефектів. M(x) – очікувана кількість дефектів.

Стандарт ISO 9126-2 визначає 5 видів шкали вимірювань:

- Номінальна шкала (класифікаційна). Якісна шкала
- Порядкова шкала (впорядковує характеристики). Якісна шкала
- Інтервальна шкала
- Відносна шкала
- Абсолютна шкала

Класифікація мір якості:

1. Міри розміру
   1.1. Функціональний розмір
   1.2. Розмір програми
   1.3. Об'єм ресурсів, які використовує програма
2. Міри часу
   2.1. Час функціонування системи
   2.2. Час виконання задачі
   2.3. Час використання
3. Міри зусиль
   3.1. Продуктивність праці
   3.2. Трудомісткість
4. Міри інтервалів між подіями (наприклад час між відмовами)

5. Розрахункові міри
   5.1. Кількість знайдених помилок
   5.2. Структурна складність програми (кількість програмних шляхів, циклічна складність)
   5.3. Число несумісних елементів
   5.4. Число змін
   5.5. Число знайдених відмов
   5.6. Ергономічні лічильники
   5.7. Лічильники-оцінки (очки, бали)
6. Базові міри розміру, часу і розрахункові можуть комбінуватись.

Класифікація метрик якості:

- Об'єктивні/суб'єктивні
- Примітивні/обчислювані
- Динамічні/статичні
- Передбачаючі/пояснюючі

По відношенню до об'єкту вимірювань (ПС) метрики діляться на зовнішні, внутрішні і метрики використання ПС.

Зовнішні метрики використовують міри працюючого на комп'ютері програмного продукту, отримані в результаті вимірювання його поведінки в ході тестування і функціонування.

Внутрішні метрики забезпечують можливість оцінювати якість проміжних і кінцевих продуктів ПС безпосередньо по їх властивостям, без виконання на комп'ютері.

Метрики якості у використанні вимірюють ступінь, з яким програмний продукт, який експлуатується в певному середовищі, задовольняє потреби користувача.

УЗАГАЛЬНЕНА МОДЕЛЬ ЯКОСТІ.

Стандарт ISO 9126-1 пропонує дворівневу ієрархічну модель

Ієрархічні моделі якості ПС

Модель МакКола (1977 р)

Модель Боема (1978 р)

Модель Ваттса (1987 р)

Модель Дьюча та Вілліса (1988 р)

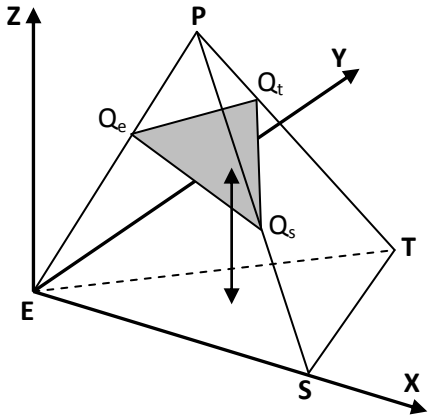
ISO 9126 (1991 р)

COQUALMO (Constructive Quality Model) – запропонована Боемом для визначення трудомісткості та вартості розробки ПС.

Неієрархічні моделі якості ПС

1999 р, А. Абран і Л. Бугліоне запропонували модель QEST (Quality Factor + Econimic, Social and Technical Dimensions).

В 2001 р Абраном та Кецесі запропонована схема GDQA (Graphical Dynamic Quality Assessment).



БАЄСІВСЬКИЙ ПІДХІД.

Часто основне мірило якості – кількість знайдених дефектів.

Очікувану кількість дефектів необхідно прогнозувати. Для прийняття рішень в умовах невизначеності використовується формула повної ймовірності і формула Баєса.

Простий приклад. Нехай A_1 – розроблено програмний продукт високої якості, A_2 – низької якості; B_1 – розробку виконав спеціаліст високої кваліфікації, B_2 – низької. Апріорні ймовірності: $P(B_1)=0.3$, $P(B_2) = 0.7$.Умовні ймовірності: $P(A_1|B_1) = 0.6$, $P(A_1|B_2) = 0.2$

Тоді по формулі повної ймовірності:

$$P(A_1) = \sum_{i=1}^{2} P(A_1|B_i) \cdot P(B_i) = 0.6 * 0.3 + 0.2 * 0.7 = 0.32$$

Парадигма «вбудови» якості в програмній інженерії

Виникла в Японії в 60-х роках. QFD – Quality Function Deployment. Традиційна ідея підвищення якості за рахунок контролю створюваної продукції була замінена на ідею виробництва бездефектної продукції за рахунок вбудови елементів забезпечення якості в процесу виробництва.

Вимірювання як процес життєвого циклу

Процес вимірювання входить в групу організаційних процесів ЖЦ і пов'язаний з процесом верхнього рівня – процесом керування.

4 основні причини, які обумовлюють вимірювання процесів, продуктів і ресурсів в ЖЦ це необхідність їх:

1. Охарактеризувати
2. Оцінити
3. Передбачити
4. Вдосконалити

Визначення Н. Фентона: Вимірювання - це процес, в ході якого атрибутам сутностей реального світу присвоюються числові або символьні значення, що дозволяє охарактеризувати атрибути за допомогою ясно визначених правил.

Види сутностей, які представляють інтерес з позиції вимірювання: продукти, процеси, ресурси, артефакти, дії, агенти, організації, середовища, обмеження.

Мистецтво вимірювання полягає в прийнятті рішень про те, які атрибути необхідно використати для того, щоб дати правильне представлення про відповідні сутності.

Приклади атрибутів для класу сутностей:

Система: розмір, щільність дефектів

Компонент: довжина (число рядків коду, число операторів), об'єм повторного використання

Рядок коду: тип оператору, мова програмування

Дефект: тип, місце розташування, серйозність, зусилля на усунення, вік (кількість часу від моменту відкриття дефекту)

Процес розробки: час розробки, контрольні строки, зусилля на розробку, відповідність стандартам, ефективність.

Між суб'єктивними та об'єктивними слід вибирати об'єктивні вимірювання (і відповідні їм метрики).

Існують різні моделі і механізми трансформації інформаційних потреб у вимірювані атрибути і множину придатних до вимірювання мір і метрик. Найбільш відомі – QFD, GQM (Goal-Question-Measure) парадигма «ціль-питання-метрика».

Керований цілями процес вимірювання базується на 3 принципах і включає 10 етапів робіт.

Три принципу вимірювань:

- Цілі вимірювання отримуються із ділових цілей
- Контекст вимірювання забезпечується побудовою ментальних моделей
- Неформальні цілі перетворюються в інформаційні структури вимірювань.

Етапи процесу вимірювання:

1. Визначити ділові цілі
2. Ідентифікувати об'єкти дослідження
3. Визначити підцілі
4. Ідентифікувати сутності і атрибути, пов'язані з підцілями
5. Формалізувати цілі вимірювання
6. Сформулювати питання, які вимагають відповіді у кількісній формі і визначити наглядні засоби від ображення залежностей (діаграми), які допоможуть досягти цілей вимірювання

7. Визначити елементи даних, які збираються для конструювання діаграм
8. Дати визначення використаних мір і зробити їх основаними на конкретних метриках
9. Ідентифікувати дії, які будуть зроблені для виконання вимірювань (визначення мір)
10. Підготувати план реалізації вимірювань

Не дивлячись на існування різноманітних методологій проведення вимірювань, статистика показує, що біля 80% програм вимірювань були невдалими, а деякі навіть шкідливі із-за зловживання вимірюваннями.

Загальні рекомендації по впровадженню вимірювань:

- Необхідно починати з малої кількості простих метрик, поступово збільшуючи об'єм по мірі накопичення досвіду
- Встановлювати чіткі цілі і плани вимірювань. Програму вимірювань оформити у вигляді самостійного проекту по вдосконаленню процесу вимірювань
- Створити середовище, яке забезпечує безпеку процесу збору даних і правильність самих даних. Розробники повинні мати стимули збирати коректні дані
- Вповноважити і запропонувати розробникам використовувати інформацію вимірювань для аналізу власних дій

Своєчасно доводити інформацію до усіх зацікавлених сторін, які беруть участь в програмі вимірювань.

Лекція 9. Контроль і гарантія якості

В архітектурі процесів ЖЦ ПС це завдання двох процесів:

- Забезпечення гарантії якості (SQA)
- Управління якістю (Quality management)

Існує дві категорії об'єктів забезпечення гарантії якості і пов'язаних з ними задач: робочі продукти і процеси ЖЦ

Гарантуючи якість роб. продуктів необхідно впевнитись в тому, що:

- Усі плани належним чином документовані, узгоджені і виконувані
- Робочі продукти і пов'язана з ними документація узгоджуються з умовами договорів і відповідають вимогам планів
- Продукти повністю задовольняють поставленим до них вимогам і є прийнятними для замовника.

Гарантуючи якість процесів необхідно впевнитись в тому, що:

- Усі процеси ЖЦ узгоджуються з планами
- Застосовані засоби програмної інженерії, середовище розробки, середовище тестування узгоджуються з договором
- Замовник забезпечується необхідною підтримкою згідно з договорами
- Метрики продуктів і процесів відповідають затвердженим стандартам і процедурам
- Назначений штат виконавців має досвід і знання, необхідні для досягнення цілей проекту.

Вимоги до підготовки компетентних спеціалістів.

В 1995 р. У. Хамфрі запропонував концепцію «персонального процесу учасника розробки ПЗ» (PSP, “Personal Software Process”).

PSP – це схема і сукупність методів індивідуальної професійної діяльності виконавців процесів ЖЦ, основаної на принципах планування, обліку, самоконтролю і особистої відповідальності за якість рішень, що приймаються.

Підвищення якості продукту досягається по ключовим аспектам:

- Аналізуючи усі допущені дефекти, виконавець визначає причини власних помилок і намагається працювати уважніше
- Аналізуючи дефекти, виконавець починає усвідомлювати вартість їх усунення і необхідність визначення найбільш ефективних способів виявлення і локалізації дефектів
- Виконавець використовує ефективні практичні прийоми PSP для попередження появи помилок в майбутньому.

PSP визначається на семи рівнях (звичні прийоми роботи і вивчення основ PSP - адаптація PSP до усіх крупних проектів).

Колективний процес розробки ПЗ (TSP, Team Software Process). Під час 4-денного періоду налаштування усі члени команди визначають стратегію роботи по проекту на 3-4 місяці, складають колективний та індивідуальні плани робіт.

People CMM (people capability maturity model – модель зрілості процесу управління кадрами).

Рівень 1 (початковий):

- Неузгодженість дій
- Перекладання відповідальності
- Дотримання ритуалу
- відособленість учасників проекту

Рівень 2 (керований):

- виробничі перенавантаження
- неділова атмосфера
- неочевидні цільові показники
- нестача необхідних знань або досвіду
- погана взаємодія
- поганий моральний стан

Рівень 3 (визначений):

- спеціалізація персоналу по видам діяльності і рівням кваліфікації
- планування робіт із врахуванням рівнів кваліфікації
- «вузька» спеціалізація процесів
- Створена інфраструктура для вимірювання кваліфікації

Рівень 4 (передбачуваний):

- Керування трудовими ресурсами на кількісній основі
- Рівень компетентності персоналу відповідає специфіці спеціального процесу, що виконується
- Обґрунтована довіра менеджерів до результатів роботи

Рівень 5 (оптимізований):

- Вдосконалення персоналу
- Вдосконалення на індивідуальному та колективному рівнях
- Постійний пошук механізмів вдосконалення індивідуальної роботи

Ключові метрики для контролю розробки.

- Метрики трудомісткості та вартості розробки
- Метрики розміру і складності програмного продукту
- Метрики помилок

Трудомісткість та вартість розробки.

Основні методи оцінки: SLIM, COCOMO, FPA.

Розмір.

SLOC (число рядків інструкцій коду)

FPA (function point analysis) – методологія аналізу показників функціонального розміру.

Складність.

Найбільш відомі методи: метрики Холстеда, метрика «цикломатичне число» МакКейба

Метрики Холстеда:

n_1 – кількість різних операторів

n_2 – кількість різних операндів

N_1 – загальна кількість операторів

N_2 – загальна кількість операндів

Розмір програми: $N=N_1+N_2$

Розмір словника: $n= n_1+n_2$

Прогнозований розмір: $N' = n_1 \cdot \log_2 n_1 + n_2 \cdot \log_2 n_2$

Об'єм програми: $V = N \cdot \log_2 n$

Трудомісткість розробки: $E = n_1 \cdot N_2 \cdot N \cdot \log_2 n / 2 \cdot n_2$

Час, необхідний на розробку: $T = E/18$

Кількість помилок: $B = V/3000$.

Метрики МакКейба основані на аналізі графу потоків управління програми.

$C = e – n + 2$.

C – цикломатична складність

e – кількість ребер на графі

n – кількість вузлів.

Рекомендоване значення C для забезпечення супроводжуваності і можливості тестування коду – 10.

Наприклад – реалізація задачі «Попадання точки в трикутник»

Цикломатичне число: e=25, n=24, $C = 25-24+2 = 3$.

Метрики Холстеда:

Число рядків: 21.

$n_1 = 9$, $n_2 = 17$, $N_1 = 29$, $N_2 = 60$.

$N = 29+60 = 89$. $n = 9+17=26$.

$N' = 9 \cdot \log_2 9 + 17 \cdot \log_2 17 = 9 \cdot 3,17 + 17 \cdot 4,09 = 98$

$V = 89 \cdot \log_2 26 = 418,3$ (bit)

$E = 9 \cdot 60 \cdot 89 \cdot 4,07 / 2 \cdot 17 = 6643,6$ (bit). $T = 369$ sec

$B = 418,3/3000 = 0,14$

Метрики, які використовуються при ООП:

Цикломатична складність – оцінка складності алгоритму методу

Зважене число методів в класі – визначається кількістю методів, реалізованих в класі або сумою оцінок цикломатичної складності кожного.

Кількість віддалених методів, що викликаються

Відклик на клас – сума числа локальних і віддалених методів. Чим більше число методів, що можуть бути викликані з класу повідомленнями – тим складніше клас

Недостатність зв'язності методів – число пар методів, що не мають загальних атрибутів

Зчеплення між класами – кількість окремих ієрархій класів, які залежать від даного класу. Чим слабше зчеплення – тим краще

Глибина дерева наслідування

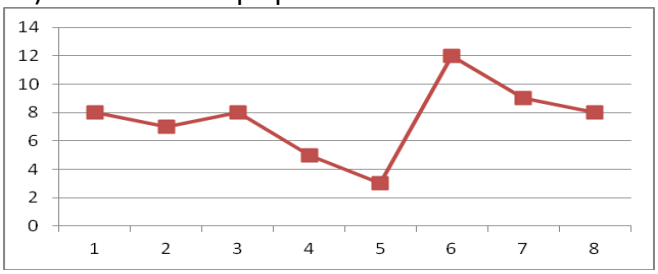
Кількість наслідників – число безпосередніх підкласів (чим більше – тим більше ймовірність невдалої абстракції класу).

Метрики помилок

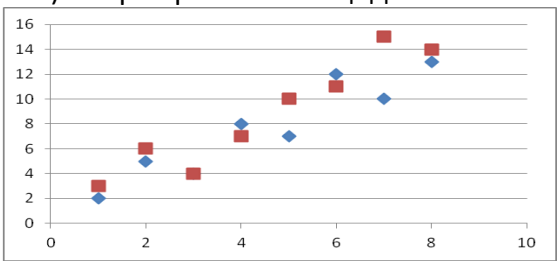
Не досягнуто повної узгодженості між основними поняттями: відмова, дефект, помилка. В англ.. літературі – anomaly, error, fault, failure, incident, flaw, problem, gripe, glitch, defect, bug.
Відмова (failure) – подія переходу ПС із працездатного стану в непрацездатний або отримання результатів за межами допустимих значень.
Дефект (defect) – запис елементу програми або тексту документу, використання якої може призвести до неправильної інтерпретації цього елемента комп'ютером (fault) або людиною (error).
Схема відмови програми:
дефект в коді (defect) – [помилка (fault)] – аномалія (anomaly) = відмова (failure)

Графічні інструменти аналізу якості.

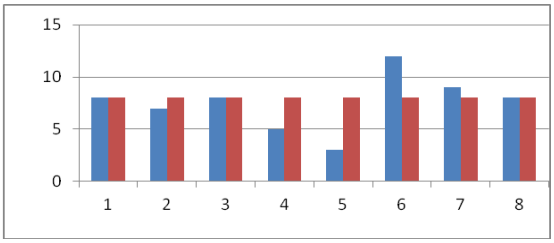
Таблиця розшарування даних. Множина даних розшаровується виходячи з двох критеріїв (наприклад «тип дефекту» та «дата виявлення»).
Діаграма виконання (Run Chart) або часовий графік.



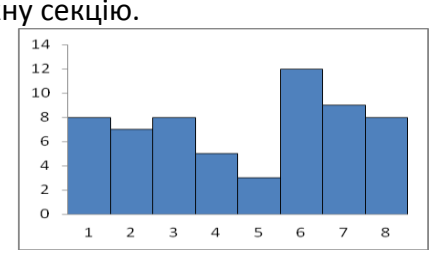
Діаграма розсіяння (Scatter Diagram). Перевірка гіпотез щодо залежності між двома величинами.



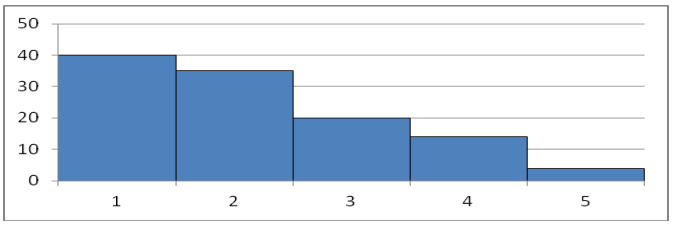
Діаграма стовпчиків (Bar chart).



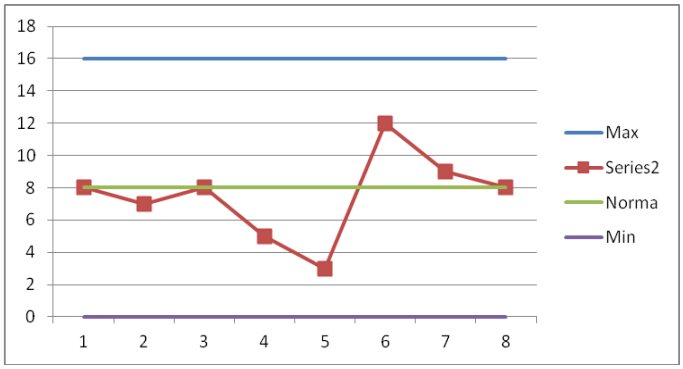
Гістограма. Створюється шляхом групування результатів вимірювань по секціям і підрахунку кількості попадання виміряних значень в кожну секцію.



Діаграми Парето



Контрольні карти (X-карти)



Причинно-наслідова діаграма (діаграма Ісікави, «риб'яча кість»). C&E – Cause and Effect diagram

Література

1. Андон Ф.И., Коваль Г.И., Коротун Т.М. Основы инженерии качества программных систем. 2-е издание. — К.: Академпериодика, 2007. — 672 с.
2. Соммервилл Я. Инженерия программного обеспечения. — М.: Вильямс, 2002. — 624 с.
3. Канер С., Фолк Дж., Нгуен Х.К.. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. – К.: Диасофт, 2001. — 544 с.
4. ДСТУ 19.201-78. Технічне завдання. Вимоги до змісту та оформленню.
5. Дастин Э., Рэшка Дж., Пол Дж. Автоматизированное тестирование программного обеспечения. – М.: "ЛОРИ", 2003. — 568 с.
6. ДСТУ 2844-94. Програмні засоби ЕОМ. Забезпечення якості. Терміни та визначення.
7. ISO/IEC 9126-1: 2001. Software engineeering. Product quality. Part 1: Quality Model.
8. Брауде Э. Дж. Технология разработки программного обеспечения. СПб.: Питер, 2004 — 655 с.
9. http://software-testing.ru