МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва і архітектури

О.А. Поплавський

СИСТЕМНА ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Конспект лекцій
для студентів спеціальності
121 "Інженерія програмного забезпечення",

Київ 2020

Тема 1. Специфіка розробки програмних продуктів

Під програмним забезпеченням (ПЗ) розуміють сукупність програм і документів на них для реалізації цілей і завдань ЕОМ.

Створення програм регламентується комплексом стандартів єдиної системи програмної документації (ЄСПД), що встановлюють загальні положення, види програм і програмних документів, правила розробки, оформ-лення та обігу програм і програмної документації.

Надійність програмних засобів — це їх властивість, яка сприяє виконанню заданих функцій при збереженні в часі значення встановлених експлуатаційних показників в заданих межах, що відповідають заданим режимам і умовам використання, супроводу і відновлення цих засобів.

Теоретичною основою методів аналізу надійності програмних засобів є теорія надійності технічних систем. Однак, спроби механічного перенесення основних положень цієї теорії стосовно ПЗ виявились неправомірними внаслідок ряду специфічних особливостей цих засобів. Надійність ПЗ ви-значається їх безвідмовністю і відновлюваністю.

Безвідмовність ПЗ — це властивість зберігати працездатність при їх вико-ристанні в процесі обробки інформації на ЕОМ. Безвідмовність ПЗ можна оцінити ймовірністю їх роботи без відмов при певних умовах зовнішнього середовища протягом заданого періоду спостережень. Відмова — це подія, яка полягає в порушенні працездатності ПЗ. Під певними умовами зовнішнього середовища вважається сукупність вхідних даних і стан обчис-лювальної системи. Заданий період спостережень відповідає, як правило, необхідному інтервалу часу для виконання на ЕОМ розв'язуваної задачі.

Специфіка створення ПЗ полягає в тому, що в процесі їх відладки прак-тично неможливо виявити і ліквідувати всі існуючі в них помилки. Приховані помилки можуть викликати невірне функціонування ПЗ при певних співвідношеннях вхідних даних. Тому наявність помилки в ПЗ є головним фактором порушення нормальних умов їх функціонування. З точки зору технології розробки ПЗ помилки можна розділити на наступні типи:

• програмні, викликані неправильним записом на мові програмуванняі помилками трансляції;

• алгоритмічні, пов'язані з неповним формуванням необхідних умоврозв'язку і некоректною постановкою задач;

• системні, обумовлені відхиленням функціонування ПЗ в обчислювальній системі і відхиленням характеристик взаємодіючих об'єктіввід передбачуваних при проектуванні.

Програмний продукт (ПП) — це програмний засіб, призначений для постачання користувачу. Програмним продуктом можна вважати комплекс програм, що функціонують на ЕОМ та підтримують користувача в процесі вирішення його завдань без участі створювача.

Основні характеристики програмних продуктів

- алгоритмічна складність;
- склад функцій обробки інформації;
- обсяг файлів, що використовуються ПП;
- вимоги до ОС і технічних продуктів обробки, в тому числі дискової пам'яті, розміром оперативної пам'яті для запуску ПП, типу процесора, версії ОС і т.д.

Всі програми за характером використання поділяються на:

1. утилітарні;
2. програмні вироби.

Розробка програмних додатків - це сукупність виробничих процесів, що призводить до створення необхідного ПП, а також опис цієї сукупності процесів, починаючи з моменту зародження ідеї по створенню і до «утилізації» ПП.

Продуктом є ПП, що містить програми, які виконують необхідні функції.

Джерела помилок в програмних засобах

1 випадок. При розробці ПП людина має справу з системами. Система - сукупність взаємодіючих між собою елементів.

Логічно пов'язаний набір програм є прикладом системи. Будь-яка окрема програма також є системою. Зрозуміти систему, значить, логічно перебрати всі шляхи взаємодії між її елементами.

Відомі такі методи боротьби зі складністю систем:

а) забезпечення незалежності компонент системи;

б) використання в системах ієрархічних структур.

2 випадок. Справжня природа помилки в ПП замаскована.

Відповідно до цього процес перекладу слід розбити на етапи:

а) зрозуміти завдання;

б) скласти план (цілі і методи вирішення);

в) виконати план (перевіряючи правильність кожного кроку);

г) проаналізувати отримане рішення.

3 випадок. Оцінка вартості помилок на різних етапах створення ПЗ:

В Залежно від того де і коли при роботі над ПП виявлена помилка, ціна його може різнитися в 50 - 100 разів.

Можна виділити ряд специфічних особливостей:

1. неформальний характер вимог до ПП при постановці завдання, але формалізований об'єкт розробки - програмний засіб;
2. розробка ПП носить творчий характер, а не зводиться до будь-якої послідовності регламентованих дій;
3. продукт розробки являє собою сукупність текстів, зміст яких виражається процесами обробки даних і діями користувачів, які запускають ці процеси, що зумовлює вибір розробником специфічних прийомів, методів і продуктів;
4. ПП при своєму використанні не витрачається і не витрачає використовуваних ресурсів.

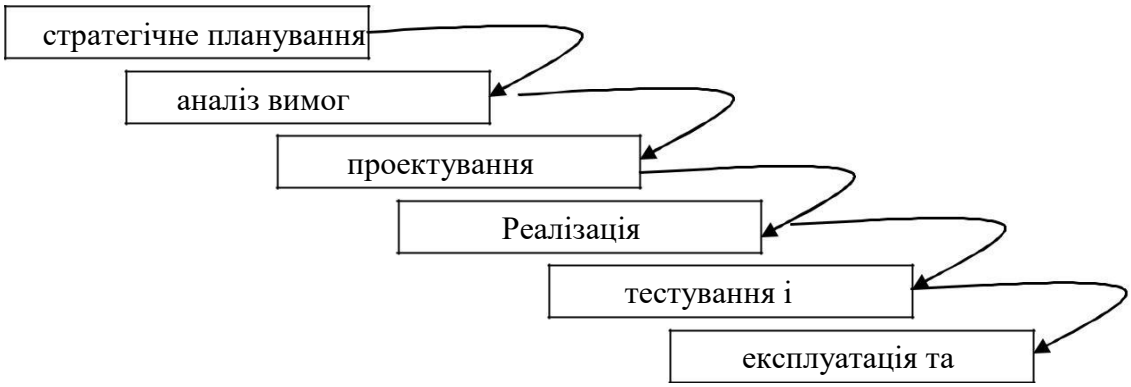
Життєвий цикл ПП

ЖЦПП - весь період розробки і експлуатації, починаючи з моменту виникнення задуму ПП і закінчуючи припиненням всіх видів його використання.

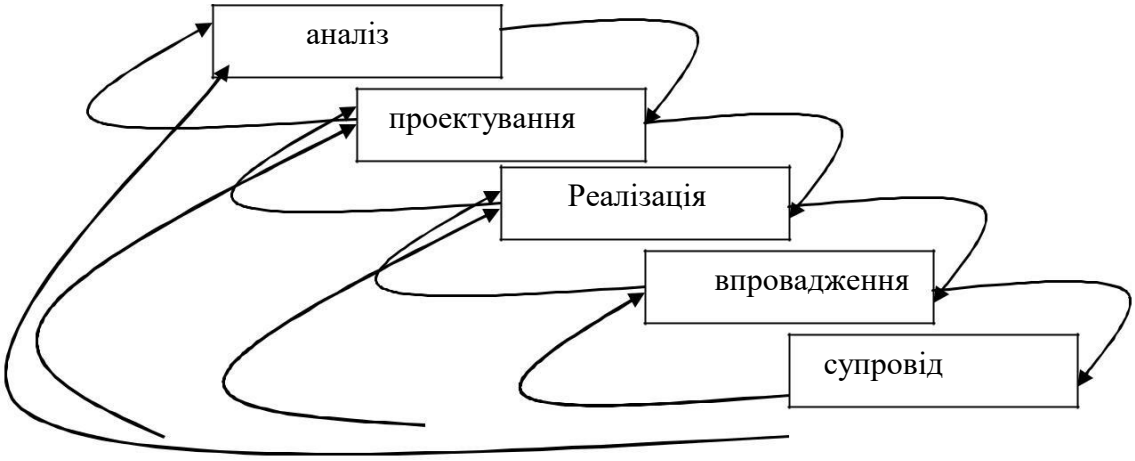
моделі ЖЦ

Історично в ході розвитку теорії проектування програмного забезпечення та у міру його ускладнення утвердилися такі основні моделі:

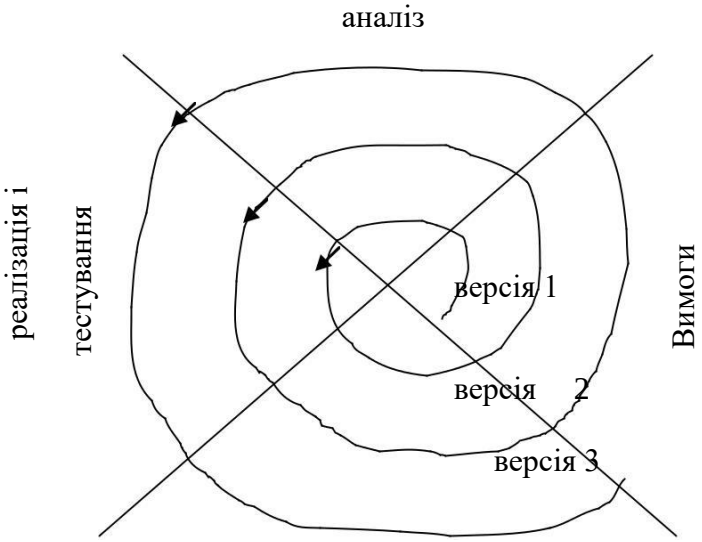
1. каскадна модель:



2. Покрокова модель:



3. Спіральна:



4. Складальне програмування: ПП конструюється з компонент, які вже існують в бібліотеці компонент.

5. Формальні перетворення: базується CASE-технологія.

Стадії ЖЦ ПП в загальному вигляді

1 стадія - розробка ПП:

а) етап зовнішнього опису ПП;

б) конструювання;

в) кодування;

г) атестація;

2 стадія - виробництво програмних виробів:

Програмний виріб - це копія або екземпляр розробленого ПП.

Виготовлення ПІ - це процес генерації або відтворення програм або програмних документів ПП з метою їх поставки користувачеві для застосування за призначенням.

Виробництво ПІ - це сукупність робіт із забезпечення виготовлення необхідної кількості ПІ у встановлені терміни.

3 стадія - експлуатація ПП:

Охоплює процеси зберігання, впровадження та супроводу, а також транспортування і застосування ПІ за своїм призначенням. Складається з 2-х паралельно проходять фаз:

- застосування ПП;
- супровід ПП.

Тема 2. Зовнішній опис ПП

Розробка ПП починається з процесу формулювання вимог, в якому повинен бути створений документ, який визначає завдання розробників ПП. Цей документ називається зовнішнім описом ПП.

Зовнішнє опис ПП грає роль точної постановки завдання, вирішення якої повинно забезпечити розробляється ПП. Зовнішнє опис має містити всю інформацію, яку необхідно знати користувачеві для застосування ПП.

Вихідним документом для розробки зовнішнього опису ПП є визначення вимог до ПП.

Т.ч. зовнішнє опис можна представити в наступному вигляді:

Зовнішнє опис = визначення вимог + специфікація якості ПП + функціональна специфікація ПП

I. *Визначення вимог до ПП:*визначають задум ПП, характеризують умови його використання. Неправильне розуміння потреб користувача призводить до помилок в зовнішньому описі.

II. *Специфікація якості ПП:* являє собою модель, в якій міститься перелік всіх елементарних властивостей, які необхідно забезпечити в ПП і які прийнятним для користувача.

III. *Функціональна специфікація:*складається з 3-х частин:

А) опис зовнішньої інформаційного середовища;

Б) визначення функцій ПП;

В) опис небажаних ситуацій.

Вимоги до програмного забезпечення

Вимоги до програмного забезпечення – набір вимог щодо властивостей, якості та функцій програмного забезпечення, що буде розроблено, або знаходиться у розробці. Вимоги визначаються в процесі аналізу вимог та фіксуються в специфікації вимог, діаграмах прецедентів та інших артефактах процесу аналізу та розробки вимог.

Розробка вимог до програмної системи може бути розділена на декілька етапів:

- Знаходження вимог (збір, визначення потреб заінтересованих осіб та систем).
- Аналіз вимог (перевірка цілісності та закінченості).
- Специфікація (документування вимог).
- Тестування вимог.

Види вимог за рівнями

- Бізнес-вимоги – визначають призначення ПЗ, можуть описуватися в документі о баченні (англ. vision) та документі о межах проекту (англ. scope).
- Вимоги користувача – визначають набір завдань користувача, які повинна вирішувати програма, а також сценарії їхнього вирішення в системі. Ці вимоги можуть мати вигляд тверджень, варіантів використання, історій користувача, сценаріїв взаємодії.
- Функціональні вимоги – визначають «що» повинен робити програмний продукт. Ці вимоги описуються в документі **Специфікації програмного забезпечення** (англ. SRS).

Види вимог за характером

1. Функціональний характер – вимоги до поведінки системи:

- Бізнес-вимоги.
- Вимоги користувача.
- Функціональні вимоги.

2. Нефункціональний характер – вимоги до характеру поведінки системи:

- Бізнес-правила – визначають обмеження, о витікають з предметної області.
- Системні вимоги – вимоги до програмних інтерфейсів, надійності, обладнанню.
- Атрибути якості.
- Зовнішні системи та інтерфейси.
- Обмеження.

Джерела вимог

До джерел вимог відносяться:

- Законодавство.
- Вимоги стандартів.
- Бізнес-процеси.
- Очікування на бачення користувачів системи.

Методи знаходження вимог

- Спілкування з майбутнім користувачем: інтерв'ю, анкетування.
- Мозковий штурм, семінар.
- Аналіз нормативної документації та законодавства.
- Аналіз бізнес-процесів.

Документування вимог

Зазвичай вимоги використовують як засіб комунікації між різними заінтересованими особами та системами. З цього виходить, що вимоги повинні бути простими та зрозумілими як для звичайних користувачів, так і для розробників. Для цього створюються наступні документи:

- Бачення (Vision).
- Специфікація вимог до програмного забезпечення (англ. Software Requirements Specification, SRS).

Вимоги до ПЗ можуть документуватися в текстовому або графічному вигляді.

Текстові вимоги – це стислий та розгорнутий описи якогось прецеденту.

Для графічного представлення використовують наступні нотації:

- ER (IDEF1FX).
- IDEF0.
- IDEF3.
- DFD.
- UML.
- OCL.
- SysML.
- ARIS (eEPC, VAD).

Вимоги в процесах розробки

Різні методології розробки програмного забезпечення по-різному працювали з вимогами. В дуже старій, та не актуальній моделі водоспаду (англ. waterfall) етап аналізу та розробки вимог є першим. Особливістю є те, що він повністю закінчується до початку проектування та розробки ПЗ, а останні не можуть початися до завершення аналізу вимог.

В ітеративних процесах розробки фаза аналізу та розробки вимог в різному об'ємі є на кожній ітерації.

Загальносистемні принципи створення ПО

При створенні й розвитку ПП рекомендується застосовувати такі загальносистемні принципи:

- принцип включення, який передбачає, що вимоги до створення, функціонування та розвитку ПП визначаються з боку більш складної системи, що включає його в себе;

- принцип системної єдності, який полягає в тому, що на всіх стадіях створення, функціонування та розвитку ПП його цілісність буде забезпечуватися зв'язками між підсистемами, а також функціонуванням підсистеми управління;
- принцип розвитку, який передбачає в ПП можливість його нарощування та вдосконалення компонентів і зв'язків між ними;
- принцип комплексності, який полягає в тому, що ПП забезпечує зв'язаність обробки інформації, як окремих елементів, так і для всього обсягу даних в цілому на всіх стадіях обробки;
- принцип інформаційної єдності, тобто у всіх підсистемах, засобах забезпечення і компонентах ПП використовуються єдині терміни, символи, умовні позначення і способи подання;
- принцип сумісності полягає в тому, що мова, символи, коди та засоби програмного забезпечення узгоджені, забезпечують спільне функціонування всіх підсистем і зберігають відкритою структуру системи в цілому;
- принцип інваріантності визначає інваріантність підсистем і компонентів ПП до оброблюваної інформації, тобто вони є універсальними або типовими.

Основні етапи створення програм

1. Системний аналіз. У рамках цього етапу здійснюється аналіз вимог, що пред'являються до програмної системи. Він проводиться на основі первинного дослідження всіх потоків інформації при традиційному проведенні робіт і здійснюється в наступній послідовності:
    a. уточнення видів і послідовності всіх робіт;
    b. визначення цілей, які повинні бути досягнуті програмою, що розробляється;
    c. виявлення аналогів, що забезпечують досягнення подібних цілей, їх переваг та недоліків.
2. Зовнішнє специфікування. Полягає у визначенні зовнішніх специфікацій, тобто описів вхідної та вихідної інформації, форм її подання і способів обробки інформації. Реалізується у такій послідовності:
    a. постановка завдання на розробку нової програми;
    b. оцінка цілей розроблюваного програмного продукту.

Далі, при необхідності, етапи 1-2 можуть бути повторені до досягнення задовільного вигляду програмної системи з описом виконуваних нею функцій і деякої ясністю реалізації її функціонування.

3. Проектування програми. На цьому етапі проводиться комплекс робіт із формування опису програми. Вихідними даними для цієї фази є вимоги, викладені у специфікації, розробленої на попередньому етапі. Приймаються рішення, що стосуються способів

задоволення вимогам специфікації. Цю фазу розробки програми поділяють на два етапи:

a. архітектурне проектування. Являє собою розробку опису програми у найзагальнішому вигляді. Цей опис містить відомості про можливі варіанти структурної побудови програмного продукту (або у вигляді кількох програм, або у вигляді кількох частин однієї програми), а також про основні алгоритми, і структури даних. Результатом цієї роботи є остаточний варіант архітектури програмної системи, вимоги до структури окремих програмних компонентів і організації файлів для міжпрограмного обміну даними;

b. робоче проектування. На цьому етапі архітектурний опис програми деталізується до такого рівня, який робить можливими роботи з її реалізації (кодування і збірці). Для цього здійснюється складання і перевірка специфікацій модулів, складання описів логіки модулів, складання остаточного плану реалізації програми.

4. Кодування і тестування. Ці види діяльності здійснюються для окремих модулів і сукупності готових модулів до отримання готової програми.

5. Комплексне тестування.

6. Розробка експлуатаційної документації.

7. Прийомо-здавальні та інші види випробувань.

8. Коригування програм. Проводиться за результатами попередніх випробувань.

9. Здавання замовнику. Здійснюється остаточна здача програмного продукту замовнику.

10. Тиражування.

11. Супровід програми. До поняття "супровід" входять усі технічні операції, необхідні для використання даної програми у робочому режимі. Сюди входить не тільки виправлення помилок. На цьому етапі також здійснюється модифікація програми, внесення виправлень у робочу документацію, вдосконалення програми та інше. Внаслідок широких масштабів подібних операцій супровід є ітеративним процесом, який бажано здійснювати не стільки після, скільки до випуску програмного продукту для широкого використання. Роботи із супроводу часто поглинають більше половини витрат, що припадають на весь життєвий цикл програмної системи у вартісному вираженні.

Сучасні технології проектування програмного забезпечення спрямовані на часткову автоматизацію описаних вище етапів і на суміщення їх у часі з метою скорочення термінів виконання проектів.

Тема 3. Архітектура програмного продукту

Архітектура ПП - це структура компонентів програми або системи, їх взаємозв'язку, принципи та керівництва для їх проектування і розвитку в часі.

Архітектура ПП - це будова програмного продукту, тобто представлення його як системи, що складається з деякої сукупності взаємодіючих підсистем. Тут підсистемами виступають, зазвичай, окремі програми.

Розробка архітектури є 1-метапом боротьби зі складністю ПЗ, на якому реалізується принцип виділення незалежних компонент.

Основні завдання розробки архітектури ПЗ:

о виділення програмних п/с і відображення в них зовнішніх функцій (заданих в ФС);о встановлення способів взаємодії між виділеними програмними п/с.

Архітектура ПП розробляється на основі ФС - функціональної специфікації. Тобто, проводиться подальша конкретизація функціональної специфікації, яка призводить до формування архітектури.

Більшість описів архітектури є рисунками, в яких прямокутниками показані виконувані компоненти, а стрілками - взаємодія серед цих компонентів. Ці малюнки відображають існуючий досвід створення систем, подібних до тої, що проектується.

Основні питання для розробника архітектури ПП:

1.Як користувач буде використовувати ПП?

2.Як ПП буде розгортатися і обслуговуватися при експлуатації?

3.Які висуваються вимоги до таких атрибутів ПП, як: якість, безпека, продуктивність, можливість паралельної обробки, інтернаціоналізація і конфігурація?

4.Як спроектувати ПП, щоб він залишався гнучким і зручним в обслуговуванні протягом усього періоду експлуатації?

5.Які основні архітектурні елементи, які можуть впливати на ПП зараз або після його розгортання?

Необхідно пам'ятати, що архітектура повинна:

1.Формувати структуру системи, але приховувати її.

2.Розкривати всі можливі варіанти використання і сценарії ПП.

3.Відповідати вимогам усіх зацікавлених сторін.

4.Виконувати вимоги, як по функціональності, так і по якості.

Основні принципи проектування архітектури

При проектуванні архітектури слід керуватися наступними рекомендаціями:

1.Враховуйте можливі майбутні зміни. Продумайте, як з часом може знадобитися змінити ваш ПП, щоб він відповідав новим вимогам і завданням, і передбачте необхідну гнучкість.

2.Створюйте архітектуру, яка зменшує ризики. Використовуйте засоби проектування, системи моделювання, такі як (Unified Modeling Language, UML), і засоби візуалізації, коли необхідно виявити ризики, ухвалити архітектурні і проектні рішення, та проаналізувати їх наслідки. Проте, не створюйте дуже формалізовану модель, вона може обмежити можливості для виконання ітерацій і адаптації дизайну.

3.Використовуйте колективну роботу та візуалізацію. Для побудови хорошої архітектури критично важливим є ефективний обмін інформацією про дизайн, ухвалювані рішення і зміни. Використовуйте моделі, представлення та інші способи візуалізації архітектури для ефективного обміну інформацією і зв'язку зі всіма зацікавленими сторонами, а також для забезпечення швидкого повідомлення про зміни в архітектурі.

4.Визначіть ключові інженерні рішення. Намагайтесь зрозуміти ключові інженерні рішення і області, в яких найвірогідніше можуть виникнути помилки. На самому початку проекту приділіть достатню кількість часу і уваги для ухвалення правильних рішень. Це забезпечить створення гнучкішого дизайну, внесення змін до якого не буде потребувати повної його переробки.

Приступаючи до роботи над архітектурою ПЗ, необхідно пам'ятати про основні принципи проектування. Це допоможе створити архітектуру, яка буде відповідати перевіреним підходам, забезпечить мінімізацію витрат, простоту обслуговування, зручність використання і розширюваність.

Розглянемо основні принципи:

1.Принцип розділення функцій. ПЗ повинен бути розділений на окремі компоненти з мінімальним перекриттям функціональності. Важливим чинником є граничне зменшення кількості точок зіткнення, що

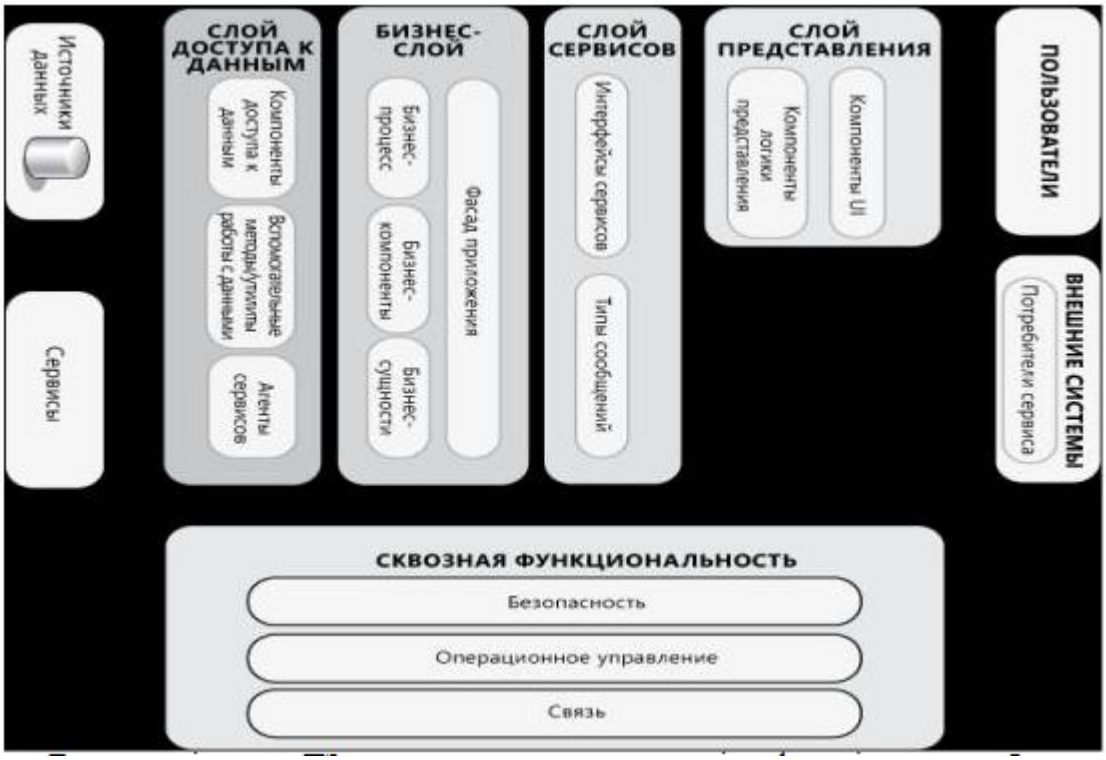
забезпечитьвисоку зв'язність (high cohesion) іслабке зчеплення (low coupling). Невірне розмежування функціональності може привести до високої зв'язаності й складнощі взаємодії, навіть не дивлячись на слабке перекриття функціональності окремих компонентів.

2.Принцип єдиності відповідальності. Кожен окремо узятий компонент або модуль повинен відповідати тільки за одну конкретну властивість/функцію або сукупність зв'язаних функцій.

3.Принцип мінімального знання (також відомий як Закон Деметера (Law of Demeter, LOD)).Будь-якійкомпоненті або об'єкту не повинні бути відомі внутрішні деталі інших компонентів або об'єктів.

4.Принцип неповторюваності (Don’t repeat yourself, DRY). Намір повинен бути відображений тільки один раз. У проектуванні архітектури ПЗ це означає, що певна функціональність повинна бути реалізована тільки в одному компоненті (об’єкті) і не повинна дублюватися ні в одному іншому місці.

5.Мінімізуйте попереднє проектування. Проектуйте тільки те, що є необхідним. Лише в окремих випадках, коли вартість розробки (або втрат у разі невдалого дизайну) дуже високі, допускається повне попереднє проектування і тестування. У інших випадках, особливо при гнучкій розробці, можна уникнути масштабного попереднього проектування. Якщо вимоги до ПП розпливчасті і чітко не визначені, або існує вірогідність зміни дизайну з часом, намагайтесь не витрачати багато сил на попереднє проектування.



Типова архітектура ПЗ, компоненти якого згруповані по функціональних областях.

Основні питання архітектора ПП

1.Визначення типу ПП

a.Застосування для мобільних пристроїв.

b.Насичені клієнтські застосування для виконання переважно на клієнтських ПК.

c.Насичені клієнтські застосування для розгортання з Інтернету з підтримкою насичених UI і мультимедійних сценаріїв.

d.Сервіси, розроблені для забезпечення зв'язку між слабо зв'язаними компонентами.

e.Веб-сервера-застосуваннядля виконання переважно на сервері в сценаріях з постійним підключенням.

f.Застосування і сервіси, що розміщуються в центрах обробки даних (ЦОД) і в хмарі.

g.Офісні бизнес-приложения(Office Business Applications, OBAs), інтегруючі технології

Microsoft Office і Microsoft Server.

2.Вибір стратегії розгортання

a.Застосування може розгортатися в різноманітних середовищах, кожна з яких матиме власний набір обмежень, таких як фізичний розподіл компонентів по серверах, обмеження по мережевих протоколах, настройки міжмережевих екранів і маршрутизаторів і т.д. Існує декілька загальних схем розгортання.

b.При виборі стратегії необхідно знайти компроміс між вимогами застосування і відповідними схемами розгортання, підтримуваним устаткуванням, і обмеженнями, що накладаються середовищем на варіанти розгортання.

3.Вибір відповідних технологій

a.Ключовим чинником при виборі технологій для застосування є тип ПП, що розробляється, а також переважні варіанти топології його розгортання і архітектурні стилі.

b.Вибір технологій також визначається політиками організації, обмеженнями середовища, кваліфікацією ресурсів і т.д.

c.Необхідно порівняти можливості вибираних технологій з вимогами до ПП і зважити на всі ці чинники.

4.Вибір показників якості

a.Такі показники якості, як безпека, продуктивність, зручність і простота використання, допомагають сфокусувати увагу на критично важливих проблемах, які повинен вирішувати дизайн ПП.

b.Залежно від конкретних вимог може знадобитися розглянути всі згадані показники якості або тільки деякі з них. Напр., питання безпеки і продуктивності необхідно врахувати при розробці кожного модуля, тоді як проблеми взаємодії чи масштабованості стоять далеко не перед усіма проектами.

c.Насамперед, необхідно зрозуміти поставлені вимоги і сценарії розгортання, щоб знати, які показники якості важливі для створюваного ПП.

d.Не можна також забувати про можливість конфлікту між показниками якості - часто вимоги безпеки йдуть врозріз з продуктивністю або зручністю використання.

5.Рішення про шляхи реалізації наскрізної функціональності

a.Протоколювання. Забезпечуйте управління і моніторинг всіх подій, які критично важливі длябізнес-логікиі системи. Протоколюйте достатню кількість відомостей для відтворення подій в системі без включення конфіденційних даних.

b.Аутентифікація. Визначіть те, як буде відбуватись аутентифікація користувачів і передача аутентифікованих посвідчень між рівнями.

c.Авторизація. На кожному рівні і на їх межах забезпечте відповідну авторизацію з необхідною деталізацією.

d.Управління виключеннями. Перехоплюйте усі виключення на функціональних, логічних і фізичних рівнях та уникайте розкриття конфіденційних відомостей кінцевим користувачам.

e.Зв'язок. Виберіть відповідні протоколи, зведіть до мінімуму кількість викликів по мережі і захистіть передачу конфіденційних даних по мережі.

f.Кешування. Визначте, що і де повинно кешуватися для поліпшення продуктивності і скорочення часу відгуку ПЗ. При проектуванні кешування не забудьте врахувати особливостіВеб-серверівтаweb-застосувань.

4. Архітектурні стилі (шаблони) ПП

Архітектурний стиль (архітектурний шаблон) – це набір принципів (високорівнева схема), що забезпечує абстрактну інфраструктуру для великого сімейства систем.

Архітектурний стиль покращує секціонування і сприяє повторному використанню дизайну завдяки забезпеченню вирішень проблем, які часто зустрічаються.

Архітектурні стилі і шаблони можна розглядати як набір принципів, що формують ПП.

Архітектурний стиль/парадигма	Опис
Цілісна програма	Представляє вироджений випадок архітектури ПП: до складу ПЗ входить тільки одна програма
Шина повідомлень	Архітектурний стиль програмної системи, яка може приймати і відправляти повідомлення поодинці або пакетами по каналах зв'язку. Тоді ПЗ дістають можливість взаємодіяти, не маючи в своєму розпорядженні конкретних відомостей один про одного.
Клієнт/сервер	Система розділяється на дві частини, де клієнт виконує запити до сервера. У багатьох випадках в ролі сервера виступає СУБД, а бізнес-логіка представлена сторид-процедурамина сервері БД.
Об'єктно-орієнтована	Парадигма проектування, заснована на розподілі відповідальності застосування або системи між окремими багато разів використовуваними і самостійними об'єктами, що містять дані і поведінку.
Проблемно- орієнтована	Об'єктно-орієнтований архітектурний стиль, орієнтований на моделювання сфери ділової активності і визначальний бизнес-об'єктна підставі суті цієї сфери.
Багатошарова архітектура	Функціональні області ПП розділяються на багатошарові групи (рівні).
N-рівнева архітектура	Функціональні області ПП розділяються на багатошарові групи (рівні).Функціональність виділяється в окремі сегменти, багато в чому аналогічно багатошаровому стилю, але в даному випадку сегменти фізично

	розташовуються на різних комп'ютерах.
Компонентна архітектура	ПЗ розкладається на функціональні або логічні компоненти з можливістю повторного їх використання, завдяки ретельно пропрацьованим інтерфейсам.
Сервіс-орієнтована архітектура (SOA)	Описує застосування, що надають і споживаючі функціональність у вигляді сервісів за допомогою контрактів і повідомлень.

Поєднання архітектурних стилів

Архітектура програмної системи практично ніколи не обмежена лише одним архітектурним стилем, часто вона є поєднанням архітектурних стилів, створюючих повну систему. Наприклад, може існувати SOA-дизайн,що складається з сервісів, при розробці яких використовувалася багатошарова архітектура іоб'єктно-орієнтованийархітектурний стиль.

Поєднання архітектурних стилів також корисно при побудові Інтернет Веб-застосувань,де можна досягти ефективного розділення функціональності за рахунок застосування багатошарового архітектурного стилю. Таким чином можна відокремити логіку уявлення відбизнес-логикиі логіків доступу до даним. Вимоги безпеки організації можуть обумовлювати або3-рівневерозгортання застосування, або розгортання з більш ніж трьома рівнями. Рівень уявлення може розгортатися в прикордонній мережі, розташованій між внутрішньою мережею організації і зовнішньою мережею. Як модель взаємодії на рівні уявлення може застосовуватися шаблон уявлення з відділенням (різновид багатошарового стилю), така якModel-View-Controller(MVC)5. Також можна вибрати архітектурний стиль SOA і реалізувати зв'язок між Веб-сервером-серверомі сервером застосувань за допомогою обміну повідомленнями.

Створюючи настільне застосування, можна реалізувати клієнт, який відправлятиме запити до програми на сервері. В цьому випадку розгортання клієнта і сервера можна виконати за допомогою архітектурного стилю клієнт/сервер і використовувати компонентну архітектуру для подальшого розкладання дизайну на незалежні компоненти, що надають відповідні інтерфейси. Застосування об'єктно-орієнтованогопідходу до цих компонентів підвищить можливості повторного використання, тестування і гнучкість.

6.Основні класи архітектур ПП

1.Цілісна програма

Таку архітектуру вибирають зазвичай у тому випадку, коли ПП повинен виконувати одну яскраво виражену функцію і її реалізація не представляється дуже складною. Така архітектура не вимагає якого-небудьопису (окрім фіксації класу архітектури), оскільки відображення зовнішніх функцій на цю програму є тривіальним, а визначати спосіб взаємодії не потрібно (через відсутністьякої-небудьзовнішньої взаємодії програми, окрім як взаємодії її з користувачем, а останнє описується в документації до ПП).

2. Архітектура ППП

Складається з деякого набору незалежних програм, таких, що:

•будь-яказ цих програм може бути активізована (запущена) користувачем;

•при виконанні активізованої програми інші програми цього набору не можуть бути активізовані до тих пір, поки не закінчить виконання активізована програма;

•всі програми цього набору стосуються одного і того ж інформаційного середовища.

Таким чином, програми цього набору по управлінню ніяк не взаємодіють - взаємодія між ними здійснюється тільки через загальне інформаційне середовище.

3. Архітектура, заснована на шині повідомлень

Архітектура на шині повідомлень описує принцип використання ПП, який може приймати і відправляти повідомлення поодинці або більш каналам зв'язку, забезпечуючи, таким чином, застосуванням можливість взаємодії без необхідності знання конкретних деталей один про одного. Це стиль проектування, в якому взаємодії між окремими ПЗ здійснюються шляхом передачі (зазвичай асинхронною) повідомлень через загальну шину по загальних схемах і інфраструктурі.

Шина повідомлень забезпечує можливість обробляти:

Взаємодія, заснована на повідомленнях. Вся взаємодія між застосуваннями грунтується на повідомленнях, що використовують відомі схеми.

Складну логіку обробки. Складні операції можуть виконуватися як частина багатокрокового процесу шляхом поєднання ряду менших операцій, кожна з яких підтримує певні завдання.

Зміни логіки обробки. Взаємодія з шиною реалізується по загальних схемах і із застосуванням звичайних команд, що забезпечує можливість вставки або видалення застосувань на шині для зміни використовуваної для обробки повідомлень логіки.

Інтеграцію з різними інфраструктурами. Використання моделі зв'язку за допомогою повідомлень, заснованого на загальних стандартах, дозволяє взаємодіяти із застосуваннями, розробленими для різних інфраструктур, таких як Microsoft .NET і Java.

Шини повідомлень використовуються для забезпечення складних правил обробки вже протягом багатьох років. Такий дизайн забезпечує архітектуру, що підключається, яка дозволяє вводити застосування в процес або покращувати масштабованість, підключаючи до шини декілька екземплярів одного і того ж застосування.

Основні переваги архітектури шини повідомлень:

Розширюваність. Можливість додавати або видаляти застосування з шини без впливу на існуючі застосування.

Невисока складність. Застосування спрощуються, тому що кожному з них необхідно знати лише, як обмінюватися даними з шиною.

Гнучкість. Приведення набору застосувань, складових складний процес, або схем зв'язку між застосуваннями у відповідністьбизнес-требованиям,що змінюється, або вимогам користувача просто шляхом внесення змін до конфігурації або параметрів, керівники маршрутизацією.

Слабке скріплення. Окрім інтерфейсу, що надається застосуванням, для зв'язку з шиною повідомлень, немає ніяких інших залежностей з самим застосуванням, що забезпечує можливість зміни, оновлення і заміни його іншим застосуванням, що надає такий же інтерфейс.

Масштабованість. Можливість підключення до шини безлічі екземплярів одного застосування для забезпечення одночасної обробки безлічі запитів.

Простота застосування. Не дивлячись на те, що реалізація шини повідомлень ускладнює інфраструктуру, кожному застосуванню доводиться підтримувати лише одне підключення до шини повідомлень, а не безліч підключень до інших застосувань.

Тема 4. Перевірка працездатності розроблених програмних продуктів

Тестування програмних засобів

Тестування ПС - це процес виконання його програм на деякому наборі даних з метою виявлення помилок.

властивості тестів

1. Детективність;
2. покриває здатність;
3. відтворюваність.

аксіоми тестування

1. вважайте тестування ключовий завданням і доручайте його самим

кваліфікованим програмістам;

2. тест повинен бути спрямований на виявлення помилки, а не на підтвердження правильності роботи програми;
3. автор тесту - не автора програми;
4. готуйте тести як для правильних, так і для неправильних даних;
5. уникайте тестів, пропуск яких не можна повторити;
6. тести розробляються одночасно або до розробки програми;
7. необхідно передбачати очікувані результати тесту до його виконання і аналізувати причини розбіжності результатів;
8. ніколи не змінюйте програму, щоб спростити її тестування;
9. слід повторювати повне тестування після внесення змін до програми або після перенесення її в інше середовище;
10. в ті програми, в яких виявлено багато помилок, необхідно доповнити початковий набір тестів.

види тестування

1. модульне - це тестування програми на рівні окремо взятих модулів, функцій або класів.
2. інтеграційне - це тестування частини системи, що складається з двох або більше модулів.
3. системне - розглядає тестируемую систему в цілому і оперує на рівні користувальницьких інтерфейсів.

Програмні помилки

Програм без помилок не існує. Практика доводить, що винуватцями помилок у програмах найчастіше бувають самі програмісти. Один із загальних законів практичного програмування полягає в тому, що жодна програма не дає бажаних результатів при першій спробі трансляції та виконання. Певне

уявлення про справжні причини появи помилок у роботі програми дає таке процентне співвідношення джерел збоїв:

Вхідні дані	1%
Помилки користувача	5%
Апаратура	1%
Системне програмне забезпечення	3%
Розробка системи	15%
Програмування	75%

Програміст повинен не тільки писати ефективні програми, але і знаходити в них усілякі помилки. Сучасна практика навчання програмуванню орієнтована, в основному, тільки на виконання програмістом першої половини своєї роботи. Це все одно, як навчати льотчика тільки зльоту, припускаючи, що з посадкою машини він якось розбереться сам, якщо буде виконувати всі операції зльоту в зворотному порядку.

Існують два типи програмних помилок:

синтаксичні помилки - виникають через порушення правил мови програмування. Такі помилки зазвичай виявляються під час компіляції. Можуть бути виключені порівняно легко. Навіть якщо не переглядати текст програми можна бути впевненим, що компілятор на стадії трансляції знайде помилки і видасть відповідні попередження. Фактично пошук помилок здійснює компілятор, а їхнє виправлення - програміст;

семантичні (логічні) помилки - ті, що призводять до некоректних обчислень або помилок під час виконання (run-time error). Семантичні помилки усувають зазвичай за допомогою виконання програми з ретельно підібраними перевірочними даними, для яких відома правильна відповідь.

Загальні питання організації тестування

Тестування програмного забезпечення - це процес, що використовується для виміру якості розроблюваного програмного забезпечення. Зазвичай, поняття якості обмежується такими поняттями, як коректність, повнота, безпечність, але може містити більше технічних вимог, які описані в стандарті ISO 9126. Тестування - це процес технічного дослідження, який виконується на вимогу замовників, і призначений для вияву інформації про якість продукту відносно контексту, в якому він має використовуватись. До цього процесу входить виконання програми з метою знайдення помилок.

Якість не є абсолютною, це суб'єктивне поняття. Тому тестування не може повністю забезпечити коректність програмного забезпечення. Воно

тільки порівнює стан і поведінку продукту зі специфікацією. При цьому треба розрізняти тестування програмного забезпечення і забезпечення якості програмного забезпечення, до якого належать усі складові ділового процесу, а не тільки тестування.

Існує багато підходів до тестування програмного забезпечення, але ефективне тестування складних продуктів - це по суті дослідницький процес, а не тільки створення і виконання рутинної процедури.

Тестування пронизує весь життєвий цикл ПЗ, починаючи від проектування і закінчуючи невизначено довгим етапом експлуатації. Ці роботи безпосередньо пов'язані із завданнями управління вимогами та змінами, адже метою тестування є якраз можливість переконатися у відповідності програм заявленим вимогам.

Тестування - процес також ітераційний. Після виявлення та виправлення кожної помилки обов'язково слід повторити тести, щоб переконатися у працездатності програми. Більше того, для ідентифікації причини виявленої проблеми може знадобитися проведення спеціального додаткового тестування. При цьому потрібно завжди пам'ятати про фундаментальний висновок, зроблений професором Едсжером Дейкстри у 1972 році: "Тестування програм може служити доказом наявності помилок, але ніколи не доведе їхню відсутність!".

Існує безліч підходів до вирішення завдання тестування та верифікації ПЗ, але ефективне тестування складних програмних продуктів - це процес у вищій мірі творчий, не зводиться до прямування строгими і чіткими процедурами або до створення таких.

З точки зору ISO 9126 якість (програмних засобів) можна визначити як сукупну характеристику досліджуваного ПЗ з урахуванням наступних складових: надійність, супроводжуваність, практичність, ефективність, мобільність, функціональність.

Більш повний список атрибутів і критеріїв можна знайти в стандарті ISO 9126 Міжнародної організації зі стандартизації. Склад і зміст документації, супутньої процесу тестування, визначається стандартом IEEE 829-1998 Standard for Software Test Documentation.

Ознаки класифікації видів тестування

Існує кілька ознак, за якими прийнято робити класифікацію видів тестування. Зазвичай виділяють наступні:

За об'єктом тестування:

- функціональне тестування (functional testing);
- тестування продуктивності (performance testing):
  - навантажувальне тестування (load testing);
  - стрес-тестування (stress testing);
  - тестування стабільності (stability / endurance / soak testing);
- тестування зручності використання (usability testing);
- тестування інтерфейсу користувача (ui testing);
- тестування безпеки (security testing);
- тестування локалізації (localization testing);
- тестування сумісності (compatibility testing).

За знанням системи:

- тестування чорного ящика (black box);
- тестування білого ящика (white box);
- тестування сірого ящика (gray box).

За ступенем автоматизації:

- ручне тестування (manual testing);
- автоматизоване тестування (automated testing);
- напівавтоматизоване тестування (semiautomated testing).

За ступенем ізольованості компонентів:

- компонентне (модульне) тестування (component / unit testing);
- інтеграційне тестування (integration testing);
- системне тестування (system / end-to-end testing).

За часом проведення тестування:

- Альфа-тестування (alpha testing):
  - тестування при прийманні (smoke testing);
  - тестування нової функціональності (new feature testing);
  - регресійне тестування (regression testing);
  - тестування при здачі (acceptance testing);
- Бета-тестування (beta testing).

За ознакою позитивності сценаріїв:

- позитивне тестування (positive testing);
- негативне тестування (negative testing).

Види тестування програмного забезпечення

У залежності від переслідуваних цілей види тестування можна умовно розділити на наступні типи:

- Функціональні.
- Нефункціональні.
- Пов'язані зі змінами.

Функціональні тести базуються на функціях та особливостях, а також на взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування: компонентному або модульному (Component/Unit testing), інтеграційному (Integration testing), системному (System testing) і приймальному (Acceptance testing). Функціональні види тестування розглядають зовнішню поведінку системи. Одні з найпоширеніших видів функціональних тестів:

- Функціональне тестування (Functional testing).
- Тестування безпеки (Security and Access Control Testing) - стратегія тестування, що використовується для перевірки безпеки системи, а також для аналізу ризиків, пов'язаних із забезпеченням цілісного підходу до захисту програми, атак хакерів, вірусів, несанкціонованого доступу до конфіденційних даних. Тестування безпеки може виконуватися як автоматизовано так і в ручну, включаючи перевірку як позитивних, так і негативних тестових випадків.
- Тестування взаємодії (Interoperability Testing) - це функціональне тестування, що перевіряє здатність програми взаємодіяти з одним і більше компонентами або системами і включає в себе тестування сумісності (compatibility testing) та інтеграційне тестування (integration testing).

Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними величинами. У цілому, це тестування того, "Як" система працює. Далі перераховані основні види нефукнціональних тестів:

- Тестування продуктивності:
  - тестування навантаження (Performance and Load Testing) - визначення масштабованості додатків під навантаженням, при цьому відбувається: вимір часу виконання вибраних операцій за певних інтенсивностей виконання цих операцій; визначення кількості користувачів, що одночасно працюють з додатком; визначення меж прийнятної продуктивності при збільшенні навантаження (при збільшенні інтенсивності виконання цих операцій); дослідження продуктивності при високих, граничних, стресових навантаженнях;
  - стресове тестування (Stress Testing) дозволяє перевірити наскільки додаток і система в цілому працездатні в умовах стресу і також оцінити здатність системи до регенерації, тобто до повернення до нормального стану

після припинення впливу стресу. Стресом у даному контексті може бути підвищення інтенсивності виконання операцій до дуже високих значень або аварійна зміна конфігурації сервера. Також одним із завдань при стресовому тестуванні може бути оцінка деградації продуктивності, таким чином цілі стресового тестування можуть перетинатися з цілями тестування продуктивності;

- тестування стабільності або надійності (Stability / Reliability Testing) - перевірка працездатності програми при тривалому (багатогодинному) тестуванні з середнім рівнем навантаження. Час виконання операцій може грати в даному виді тестування другорядну роль. При цьому на перше місце виходить відсутність витоків пам'яті, перезапусків серверів під навантаженням й інші аспекти, які впливають саме на стабільність роботи;
- об'ємне тестування (Volume Testing) - отримання оцінки продуктивності при збільшенні обсягів даних у базі даних програми, при цьому відбувається: вимір часу виконання вибраних операцій за певних інтенсивностей виконання цих операцій; може проводитися визначення кількості користувачів, що одночасно працюють з додатком;

- Тестування установки (Installation testing) спрямоване на перевірку успішної інсталяції та настройки, а також оновлення або видалення програмного забезпечення. На даний момент найбільш поширена установка ПЗ за допомогою інсталяторів (спеціальних програм, які самі по собі так само потребують належного тестування). У реальних умовах інсталяторів може не бути. У цьому випадку доведеться самостійно виконувати установку програмного забезпечення, використовуючи документацію у вигляді інструкцій або readme файлів, де крок за кроком описано всі необхідні дії та перевірки;
- тестування зручності користування (Usability Testing) - це метод тестування, спрямований на встановлення ступеня зручності використання, навченості, зрозумілості та привабливості для користувачів розроблюваного продукту в контексті заданих умов.

Тестування зручності користування дає оцінку рівня зручності використання програми за наступними пунктами:

- продуктивність, ефективність (efficiency) - скільки часу і кроків знадобиться користувачеві для завершення основних завдань програми, наприклад, розміщення новини, реєстрації, покупки тощо (менше - краще);
- правильність (accuracy) - скільки помилок зробив користувач під час роботи з додатком (менше - краще);
- активізація в пам'яті (recall) - як багато користувач пам'ятає про роботу програми після припинення роботи з нею на тривалий період часу (повторне виконання операцій після перерви має проходити швидше ніж у нового користувача);
- емоційна реакція (emotional response) - як користувач почувається після завершення завдання - розгублений, знаходиться у стані стресу? Чи порекомендує користувач систему своїм друзям (позитивна реакція - краще)?

- Тестування на відмову і відновлення (Failover and Recovery Testing) перевіряє тестований продукт з точки зору здатності протистояти й успішно відновлюватися після можливих збоїв, що виникли у зв'язку з помилками програмного забезпечення, відмовами обладнання або проблемами зв'язку (наприклад, відмова мережі). Метою даного виду тестування є перевірка систем відновлення (або дублюючих основний функціонал систем), які, у разі виникнення збоїв, забезпечать збереження і цілісність даних тестованого продукту.
- Конфігураційне тестування (Configuration Testing) - ще один вид традиційного тестування продуктивності. У цьому випадку замість того, щоб тестувати продуктивність системи з точки зору навантаження, тестується ефект впливу на продуктивність змін у конфігурації. Прикладом такого тестування можуть бути експерименти з різними методами балансування навантаження. Конфігураційне тестування також може бути поєднане з навантажувальним, стрес- або тестуванням стабільності.

Після проведення необхідних змін, таких як виправлення бага/дефекту, програмне забезпечення повинне бути перетестовано для підтвердження того факту, що проблему було дійсно вирішено. Нижче перераховані види тестування, які необхідно проводити після установки програмного забезпечення, для підтвердження працездатності програми або правильності здійсненого виправлення дефекту:

- Димове тестування (Smoke Testing). Поняття димове тестування пішло з інженерного середовища. При введенні в експлуатацію нового "заліза" вважалося, що тестування пройшло вдало, якщо з установки не пішов дим. В області ж тестування програмного забезпечення, воно спрямоване на поверхневу перевірку всіх модулів програми на предмет працездатності та наявність швидко встановлюваних критичних і блокуючих дефектів. За результатами димового тестування робиться висновок про те, приймається чи ні встановлена версія програмного забезпечення на тестування, експлуатацію або на постачання замовнику.
- Регресійне тестування (Regression Testing) - вид тестування спрямований на перевірку змін, зроблених у додатку або навколишньому середовищі (лагодження дефекту, злиття коду, міграція на іншу операційну систему, базу даних, веб-сервер або сервер додатка), для підтвердження того факту, що існуюча раніше функціональність працює як і раніше.
- Тестування збірки (Build Verification Test) спрямоване на визначення відповідності випущеної версії критеріям якості для початку тестування. За своєю метою є аналогом димового тестування, спрямованого на приймання нової версії в подальше тестування або експлуатацію. Вглиб воно може проникати далі, в залежності від вимог до якості випущеної версії.
- Санітарне тестування або перевірка узгодженості / справності (Sanity Testing) - вузьконаправлене тестування достатнє для доказу того, що конкретна функція працює згідно заявлених у специфікації вимог.

Налагодження програмних засобів

ОтладкаПС - етодеятельность, направленнаянаобнаруженіеіісправленіе помилок.

Налагодження = Тестування + Пошук помилок + Редагування

видиналагодження

1. автономна: послідовне роздільне тестування різних частин програм, що входять в ПС.
2. комплексна: тестування ПС в цілому.

Документування програмних засобів

В Відповідно до Єдиної системою програмної документації (ГОСТ 19101-77) встановлено такі документи для програмного забезпечення:

Специфікація містить перелік і короткий опис всіх файлів ПО, в тому числі и файлів документації на нього. Є обов'язковим компонентом.

Відомість власників оригіналів (код виду документа - 05) містить список підприємств на яких зберігаються оригінали програмних документів.

Текст програми (код виду документа - 12) містить текст програми з необхідними коментарями.

Опис програми (код виду документа - 13) містить відомості про логічну структуру і функціонуванні програми.

Відомість експлуатаційних документів (код виду документа - 20) містить перелік документів з кодами 30, 31, 32, 33, 34, 35, 46.

Формуляр (код виду документа - 30) містить основні характеристики ПО, комплектність і відомості про експлуатацію програми.

Опис застосування (код виду документа - 31) містить відомості про призначення ПО, області застосування, застосовувані методи, класі вирішуваних завдань, мінімальної конфігурації технічних засобів.

Керівництво системного програміста (код виду документа - 32) містить відомості для перевірки, забезпечення функціонування і налаштування програми на умовах конкурентного застосування.

Керівництво програміста (код виду документа - 33) містить відомості для експлуатації програмного забезпечення.

Керівництво оператора (код виду документа - 34) містить відомості для забезпечення процедури спілкування оператора з обчислювальною системою в процесі виконання програмного забезпечення.

Опис мови (код виду документа - 35) містить опис синтаксису і семантики мови.

Керівництво з технічного обслуговування (код виду документа - 46) містить відомості для застосування тестових і діагностичних програм при обслуговуванні технічних засобів.

Програма і методика випробувань (код виду документа - 51) містить вимоги, що підлягають перевірці при випробуванні програмного забезпечення, а також порядок і методи їх контролю.

Пояснювальна записка (код виду документа - 81) містить інформацію про структуру і конкретних компонентах програмного забезпечення, в тому числі схеми алгоритмів, їх загальний опис, а також обґрунтування прийнятих технічних і техніко-економічних рішень.

Зміст пояснювальної записки по ГОСТ 19.404-79 має включати такі розділи:

1. Вступ;
2. Призначення і область застосування;
3. Технічні характеристики;
4. Очікувані техніко-економічні показники;
5. Джерела, використовувані при розробці.

Документацію, створювану в процесі розробки можна розбити на дві групи:

I. Документи управління розробкою ПС: управляють і протоколюють процеси розробки та супроводу ПС, забезпечуючи зв'язки всередині колективу розробників і між колективами розробників.

II. Документи, що входять до складу ПС: описують програми з точки зору їх застосування користувачами і супровідник.

Тема 5. Поняття якості програмних продуктів та всесвітні стандарти її забезпечення

Якість програмного забезпечення(Software Quality) 1) це ступінь, в якій програмне забезпечення має необхідної комбінацією властивостей. [1061-1998 IEEE Standard for Software Quality Metrics Methodology] 2) це сукупність характеристик програмного забезпечення, що відносяться до його здатності задовольняти встановлені і передбачувані потреби. [ISO 8402: 1994 Quality management and quality assurance]

Забезпечення якості (Quality Assurance - QA) - це сукупність заходів, що охоплюють всі технологічні етапи розробки, випуску та експлуатації програмного забезпечення (ПО) інформаційних систем, основаних на різних стадіях життєвого циклу ПО, для забезпечення якості продукту, що випускається.

Контроль якості (Quality Control - QC) - це сукупність дій, що проводяться над об'єктом тестування в процесі розробки для отримання інформації про актуальний стан об'єкта тестування в розрізах: "готовність Продукту до випуску", "Відповідність зафіксованим вимогам", "Відповідність заявленому рівню якості продукту".

Тестування програмного забезпечення (Software Testing) - це одна з технік контролю якості, що включає в себе активності з планування робіт (Test Management), проектування тестів (Test Design), виконання тестування (Test Execution) і аналізу отриманих результатів (Test Analysis).

Верифікація (verification)- це процес оцінки системи або її компонентів з метою визначення чи задовольняють результати поточного етапу розробки умов, сформованим на початку цього етапу [IEEE]. Тобто чи виконуються наші цілі, терміни, завдання по розробці проекту, визначені на початку поточної фази.

Валідація (validation) - це визначення відповідності розробляється ПО очікуванням і потребам користувача, вимогам до системи [BS7925-1].

У 1979 році Crosby визначив якість як «відповідність вимогам» ("conformance to requirements"), а Juran і Gryna в 1970 визначили якість як «придатність до використання»

"Відповідність вимогам" передбачає, що вимоги повинні бути настільки чітко визначені, що вони не можуть бути зрозумілі і інтерпретовані некоректно.

Пізніше, на етапі розробки, проводиться регулярне вимірювання розробленого продукту, для визначення відповідності вимогам. Будь-які невідповідності повинні розглядаються як дефекти - відсутність якості. Наприклад, специфікація на певну модель радіостанції може вимагати можливості приймати певну частоту радіохвиль на відстані більш ніж 30 кілометрів від джерела мовлення. У разі, якщо радіостанція не здатна виконати дану вимогу, вона не задовольняє вимоги до якості і повинна бути визнана непридатною і неякісною. Виходячи з тих же принципів, якщо Кадиллак відповідає всім вимогам до машин Кадиллак, значить це якісна машина. Якщо Шевроле відповідає всім вимогам до машин Шевроле, отже, це теж якісна машина. Ці дві машини можуть бути абсолютно різними за стилем, швидкості і економічності, але якщо обидві вимірювати за стандартними для них наборам, тоді вони обидві будуть якісними машинами.

Визначення «Придатність до використання» приймає до уваги вимоги та очікування кінцевих користувачів продукту, які очікують, що продукт або сервіс, що надається буде зручним для їх потреб. Однак різні користувачі можуть використовувати продукт по-різному, це означає, що продукт повинен володіти максимально різноманітними варіантами використання. Згідно з визначенням Juran кожен варіант використання це характеристика якості і всі вони можуть бути класифіковані за категоріями в якості параметрів придатності до використання.

Ці два визначення якості («відповідність вимогам» і «придатність до використання») 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, ніколи не може бути переоцінена. З точки зору замовника, якість продукту, який він придбав, складається з безлічі різних факторів, таких як: ціна, продуктивність, надійність і т.д.

Тільки ваш замовник може розповісти вам про якість, тому що це єдине що він дійсно купує. Замовник не купує продукт. Він купує ваші гарантії того, що всі його очікування до продукту будуть реалізовані.

Давайте ще раз спробуємо дати визначення якості з точки зору замовника або користувача продукту.

Якість - це придатність до використання. Чи робить даний продукт то, в чому маю потребу полегшує він мою роботу, чи можу я його використовувати так, як мені зручно.

А тепер подивимося на точку зору розробника.

Якість - це відповідність специфікованою і зібраним вимогам робить даний продукт все те, що зазначено у вимогах.

Проблема в тому, що специфіковані і зібрані вимоги це зазвичай лише частина всіх реальних вимог і очікувань замовника. Де ж правильне визначення якості?

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. Для прикладу повернемося до наших автомобілів - замовник може детально описати вимоги до дизайну, параметрам двигуна, оформлення салону, кольором кузова, але ніде не вказати, що шини повинні бути круглими, а лобове скло - прозорим.

Замовник буде задоволений тільки тоді, коли куплений товар буде повністю задовольняти його реальним і життєвим вимогам, як специфіковані, так і немає.

Основним стандартом якості в області інженерії програмного забезпечення в даний час є є стандарт ISO / IEC 9126. Він визначає номенклатуру, атрибути і метрики вимог якості програмного забезпечення. Відносно недавно цей стандарт став одним з визначальних чинників при моделюванні якості програмного забезпечення і залишається їм до цих пір.

На додаток до нього випущений набір стандартів ISO / IEC 14598, який регламентує способи оцінки цих характеристик. В сукупності вони утворюють модель якості, відому під назвою SQuaRE (Software Quality Requirements and Evaluation).

Якість програмного забезпечення

Кожен день у своїй роботі ми стикаємося з досить абстрактним поняттям «якість ПО» і якщо поставити запитання тестувальників або програмісту - «що таке якість?», То у кожного знайдеться своє тлумачення. Розглянемо визначення "якості ПЗ" в контексті міжнародних стандартів:

[1061-1998 IEEE Standard for Software Quality Metrics Methodology] Якість програмного забезпечення - це ступінь, в якій ПО володіє необхідною комбінацією властивостей.

[ISO 8402: 1994 Quality management and quality assurance] Якість програмного забезпечення - це сукупність характеристик ПО, що відносяться до його здатності задовольняти встановлені і передбачувані потреби.

На даний момент найбільш поширена і використовується багаторівнева модель якості програмного забезпечення, представлена в наборі стандартів ISO 9126. На верхньому рівні виділено 6 основних характеристик якості ПЗ, кожну з яких визначають набором атрибутів, що мають відповідні метрики для подальшої оцінки (див. Рис. 1) .

Модель якості програмного забезпечення (ISO 9126-1)

Capability Maturity Model

Офіційна історія CMM (Capability Maturity Model, що зазвичай перекладають як "модель зрілості процесу розробки ПО", хоча більш вірним за змістом був би переклад "модель вдосконалення можливостей") починається в 1991 році, коли американський інститут SEI (Software Engineering Institute - Інститут системного програмування при університеті Карнегі-Меллон) опублікував першу версію цього стандарту.

Початковою метою розробки стандарту було створення методики, що дозволяє великим урядовим організаціям США вибирати найкращих постачальників ПО. Для цього передбачалося створити вичерпний опис способів оцінки процесів розробки ПЗ та методики їх подальшого удосконалення. В результаті, авторам вдалося домогтися такого ступеня подробиці і деталізації, що стандарт виявився придатним і для звичайних компаній-розробників, які бажають поліпшити існуючі процеси.

Головним поняттям стандарту є зрілість організації. Незрілої вважається організація, в якій процес розробки програмного забезпечення залежить тільки від конкретних виконавців і менеджерів, і рішення часто просто імпровізують "на ходу". У цьому випадку велика ймовірність перевищення бюджету або завалювання термінів здачі проекту, і тому менеджери змушені займатися тільки дозволом найближчих проблем.

З іншого боку, в зрілої організації є чітко визначені процедури створення програмних продуктів і управління проектами. Ці процедури в міру необхідності уточнюються і удосконалюються в пілотних проектах або за допомогою аналізу вартість / прибуток. Оцінки часу і вартості виконання робіт ґрунтуються на накопиченому досвіді і досить точні. Нарешті, в компанії існують стандарти на процеси розробки, тестування і впровадження ПО, правила оформлення кінцевого програмного коду, компонент, інтерфейсів і т.д. Все це становить інфраструктуру і корпоративну культуру, яка підтримує процес розробки програмного забезпечення.

Такі базові посилки стандарту CMM. Можна сказати, що стандарт в цілому складається з критеріїв оцінки зрілості організації та рецептів поліпшення існуючих процесів. У цьому спостерігається принципова відмінність з моделлю, прийнятою в ISO 9001, так як в ISO 9001 сформульовані тільки необхідні умови для досягнення деякого мінімального рівня організованості процесу, і не дається ніяких рекомендацій щодо подальшого вдосконалення процесів.

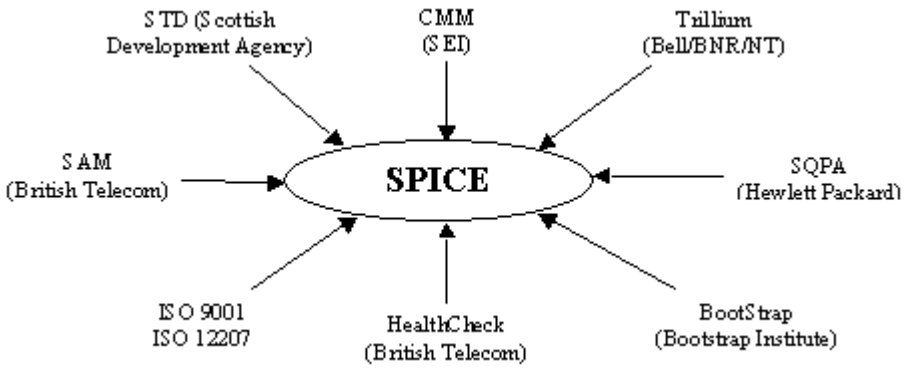
У моделі CMM визначено п'ять рівнів зрілості організацій. В результаті атестації компанії присвоюється певний рівень, який в подальшому може підвищуватися або (теоретично) знижуватися. На малюнку 1 показані перераховані деякі технології, впровадження яких необхідно для досягнення різних рівнів зрілості організації. Відзначимо, що кожен наступний рівень включає в себе всі ключові характеристики попередніх.

Малюнок 1. П'ять рівнів зрілості в моделі CMM.

SPICE

У 1991 році Міжнародна організація по стандартизації ініціювала роботу зі створення єдиного стандарту оцінки програмних процесів. Стандарт отримав ім'я SPICE (скорочення від Software Process Improvement and Capability dEtermination, визначення можливостей і поліпшення процесу створення програмного забезпечення). Офіційно стандарт називається "ISO / IEC 15504: Information Technology - Software Process Assessment" і на даний момент існує в якості робочої версії, останній випуск якої відбувся в травні 1998 року.

Завданням SPICE є створення міжнародного стандарту, в якому був би врахований весь накопичений досвід в області розробки програмного забезпечення. На малюнку 2 показані найбільш значні стандарти, ідеї яких були використані при створенні SPICE:



Малюнок 2. Попередники стандарту SPICE.

Отже, стандарт SPICE успадкував багато рис більш ранніх стандартів, в тому числі і вже згадуваних ISO 9001 і CMM. Для цього довелося вдатися до підвищення рівня деталізації стандарту. Наслідком такого ґрунтовного підходу є великий обсяг стандарту: документація до нього містить близько 500 сторінок!

Найбільше SPICE нагадує CMM. Точно так же, як і в CMM, основним завданням організації є постійне поліпшення процесу розробки ПО. Крім того, в SPICE теж використовується схема з різними рівнями можливостей (в SPICE визначено 6 різних рівнів), але ці рівні застосовуються не тільки до організації в цілому, а й до окремо взятих процесів. У таблиці 1, запозиченої з статті [6], наведено список рівнів здібностей моделі SPICE і характерні для них процедури управління. Відзначимо, що на даний момент не існує російського перекладу стандарту SPICE, тому використані терміни не є загальноприйнятими або офіційно зареєстрованими

рівні	**Назва**
рівень 0	**Процес не виконується**
рівень 1	**Здійснюється процес**
1.1	Вимірювання продуктивності процесу
рівень 2	**керований процес**
2.1	управління продуктивністю
2.2	Управління виробництвом товарів
рівень 3	**встановлений процес**
3.1	документування процесу
3.2	Відстеження ресурсів процесу

рівень 4	**передбачуваний процес**
4.1	Вимірювання процесу
4.2	управління процесом
рівень 5	**оптимізує процес**
5.1	зміна процесу
5.2	постійне вдосконалення

Таблиця 1. Рівні здібностей процесу в стандарті SPICE

Решта частини стандарту - сьома і восьма - присвячені відповідно поліпшенню процесу, і визначення можливостей процесу.

У зв'язку зі своєю відкритістю, стандарт SPICE популярний і по ньому існує багато вільно доступних матеріалів. Наприклад, на офіційному сайті SPICE будь-яка організація може зареєструватися для участі в SPICE Trials (пробних застосуваннях). На сайті групи користувачів SPICE (див.http://www.iese.fhg.de/SPICE/) Зібрана велика кількість інформації про самому стандарті, доступних ресурсах і його застосуваннях на практиці. З серпня 1999 року виходить журнал SPICE.World, цілком присвячений SPICE (існує електронна версія цього журналу - см.http://www.spiceworld.hm).

Забезпечення якості програмних засобів

I. Серія стандартів ISO 9000

Основою регламентування показників якості програмних засобів є міжнародний стандарт ISO 9126: тисяча дев'ятсот дев'яносто одна (ГОСТ ISO / IEC 9126-93) «Інформаційна технологія. Оцінка програмного продукту. Характеристики якості і керівництво по їх застосуванню ».

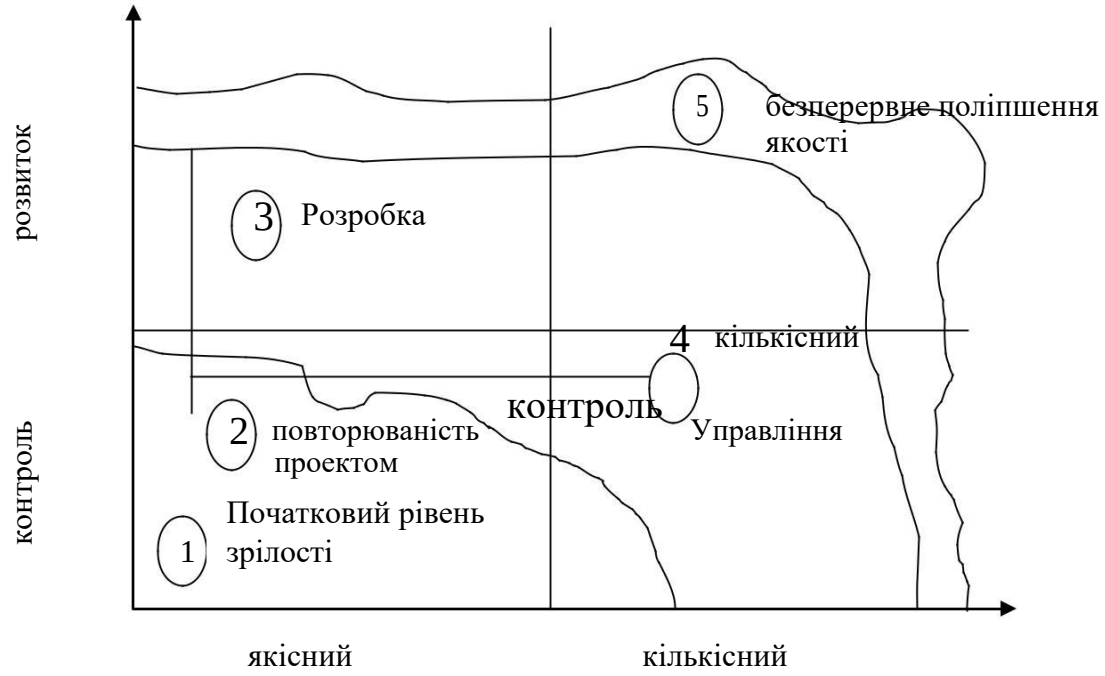
II. Процес сертифікації програм на базі інформації про їх використання

Даний пакет дає надійні гарантії якості для комерційних програмних пакетів.

Запрошуючи третю сторону для видачі сертифікатів якості на ПО, розробники знімають з себе відповідальність.

III. CMM

Використовується для впорядкування роботи субпідрядників для розробки ПО, тому що у багатьох компаніях проекти виконуються із значним запізненням і перевищенням запланованого бюджету.

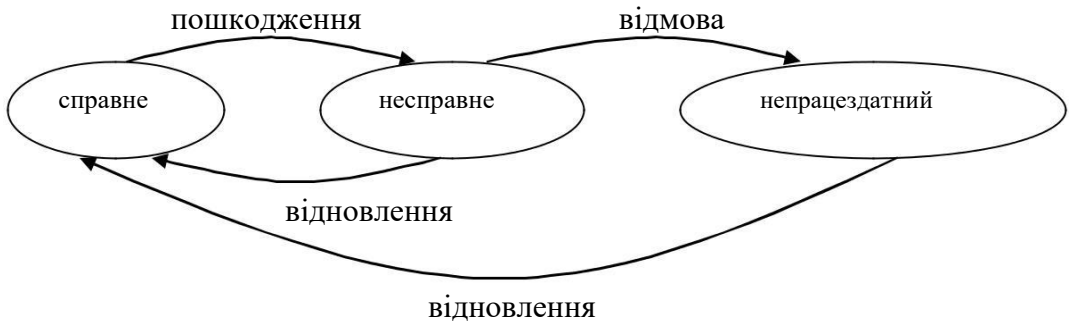


Надійність програмного забезпечення

Надійне ПС - це його здатність безвідмовно виконувати певні функції при заданих умовах протягом заданого періоду часу.

Відмова - порушення працездатності виробу і його відповідності вимогам технічної документації.

В Загалом вигляді діаграма станів і переходів при відмовах:



Пошкодження - несправність вже з'явилася, але ще не проявилася зовні.

Відновлення - повернення в справний стан шляхом:

види помилок

1. синтаксису мови;
2. під час виконання;
3. багатокористувацького доступу;
4. інструментарію та інших компонентів системи.

Тема 6. Проектування програмного продукту при об'єктному підході

Як показала практика, традиційні методи процедурного програмування не здатні впоратися ні з наростаючою складністю програм і їх розробки, ні з необхідністю підвищення їх надійності. В середині 80-х років XX століття для розробки ПО великого обсягу було запропоновано використовувати *об'єктний* підхід, а в якості технології вибрано *об'єктно-орієнтоване програмування* (ООП) - методика розробки програм, в основі якої лежить поняття *об'єкта*.

Об'єкт - Це певна структура, відповідна об'єкту реального світу, його поведінки. Кожен об'єкт має *стан*, Має чітко визначеним *поведінкою* и *унікальною ідентичністю*. сукупність *атрибутів* (Властивостей) і їх значень характеризує об'єкт. Об'єкти, що описуються однаковими наборами атрибутів, об'єднуються в *класи*. Дані класу називаються *полями*, Процедури і функції - *методами*. Усе *екземпляри* одного класу (об'єкти, породжені від одного класу) мають один і той же набір властивостей і загальна поведінка, тобто однаково реагують на однакові повідомлення. Кожен об'єкт має певний час життя. У процесі виконання програми або функціонування реальної системи, можуть створюватися нові об'єкти і знищуватися вже існуючі. Оскільки об'єкт - це динамічна структура, *змінна-об'єкт* містить не дані, а *посилання* на дані об'єкта. Тому програміст повинен подбати про виділення пам'яті для цих даних. Виділення пам'яті при створенні об'єкта здійснюється за допомогою спеціального методу класу - *конструктора* (*constructor*), А звільнення пам'яті при його знищенні - за допомогою *деструктора* (*destructor*).

Найважливішими поняттями ООП є *інкапсуляція*, *успадкування*, *поліморфізм*, Які дозволяють конструювати складні об'єкти з порівняно простих. Програма, написана з використанням ООП, складається з безлічі об'єктів, що взаємодіють між собою шляхом передачі *повідомлень*.

В основі об'єктного підходу до розробки ПЗ лежить об'єктна декомпозиція, т. Е. Уявлення розроблюваного програмного продукту у вигляді сукупності об'єктів, в процесі взаємодії яких через передачу повідомлень і відбувається виконання необхідних функцій. Об'єктно-орієнтований підхід має наступні переваги:

- зменшення складності програмного забезпечення;
- підвищення надійності програмного забезпечення;
- забезпечення можливості модифікації окремих компонентів програмного забезпечення без зміни інших його компонентів;
- забезпечення можливості повторного використання окремих компонентів програмного забезпечення.

Систематичне застосування об'єктного підходу дозволяє розробляти добре структуровані, надійні в експлуатації, досить просто модифікуються програмні системи, тому ООП є одним з найбільш інтенсивно розвиваються напрямків теоретичного і прикладного програмування. Однак при об'єктному підході відразу можна виконати декомпозицію тільки дуже простого ПО. На зорі епохи ООП були запропоновані методи аналізу і проектування в рамках об'єктного підходу, що використовують різні моделі і нотації.

Сперечатися про переваги і недоліки цих методів і моделей можна було нескінченно. Ця ситуація отримала назву «*війни методів*». Кінець «війні методів» поклало поява в 1995 р першої версії мови **UML** (*Unified Modeling Language* - Уніфікована мова моделювання), який був створений провідними фахівцями в цій галузі (Граді Бучем, Іваром Якобсоном і Джеймсом Рамбо) і в даний час фактично визнаний стандартним засобом опису проектів, створюваних з використанням об'єктно-орієнтованого підходу

Логічна модель описує ключові абстракції ПО (класи, інтерфейси), т. е. кошти, що забезпечують необхідну функціональність. **модель реалізації** визначає реальну організацію програмних модулів в середовищі розробки.**модель використання** є опис функціональності програмного продукту з точки зору користувача. **модель процесів** відображає організацію обчислень і оперує поняттями «процеси» і «нитки». Вона дозволяє оцінити продуктивність, масштабованість і надійність програмного забезпечення. І наостанок, **модель розгортання** показує особливості розміщення програмних компонентів на конкретному обладнанні.

Алгоритмічна і об'єктна декомпозиції. Класи і об'єкти

Принципово можна виділити 2 види розбиття предметної області на елементи, що становлять:

•Алгоритмічна декомпозиція (основні елементи програми - будівельні блоки - алгоритми).

•Об'єктна декомпозиція (основні елементи програми - види абстракцій (класи) і представники цих класів (об'єкти)).

Відповідно до алгоритмічної декомпозиції предметної області при аналізі завдання ми намагаємося зрозуміти, які алгоритми необхідно розробити для її вирішення, які специфікації цих алгоритмів (вхід, вихід), і як ці алгоритми зв'язані один з одним. У мовах програмування даний підхід повною мірою підтримується засобами модульного програмування (бібліотеки, модулі, підпрограми).

В рамках об'єктної декомпозиції ми намагаємося виділити основні змістовні елементи завдання, розбити їх на типи (класи). Далі для кожного

класу абстракцій ми визначаємо його властивості (дані) і поведінку (операції), а також, як ці класи абстракцій взаємодіють один з одним.

На сьогоднішній день об'єктний підхід і його основи - об'єктна модель і об'єктна декомпозиція - підтримуються усіма сучасними об'єктно-орієнтованимимовами програмування (Object Pascal, C++, Java, C#.).

Складові частини об'єктного підходу

Розглянемо стисло складові частини об'єктного підходу, грамотне виконання яких, як правило, приводить до створення якісного програмного продукту.

Об'єктний підхід:

•OOA (object oriented analysis) - об'єктно-орієнтованийаналіз.

•OOD (object oriented design) - об'єктно-орієнтованепроектування.

•OOP (object oriented programming) - об'єктно-орієнтованепрограмування.

Що означають ці ключові поняття [1]:

Об'єктно-орієнтований аналіз - це методологія, при якій вимоги до системи

сприймаються з погляду класів і об'єктів, виявлених в предметній області. Об'єктно-орієнтоване проектування - це методологія проектування, що сполучає в

собі процес об'єктної декомпозиції і прийоми уявлення логічною і фізичною, а також статичної і динамічної моделей проектованої системи.

Об'єктно-орієнтоване програмування - це методологія програмування, заснована на представленні програми у вигляді сукупності об'єктів, кожен з яких є екземпляром певного класу, а класи утворюють ієрархію спадкоємства.

У російськомовній і україномовній літературі, як правило, під абревіатурою ООП розглядають всі 3 складових об'єктного підходу. Далі і ми слідуватимемо цьому принципу.

Курси з циклу "Методи програмування" і, конкретніше, "Об'єктно-орієнтованепрограмування" переважно концентруються на OOP. Даний курс, принаймні, його теоретична частина основна увага приділяє OOA і OOD.

Принципи об'єктного підходу

Розглянемо найбільш важливі принципи об'єктного підходу.

Абстрагування застосовується при вирішенні багатьох завдань.Будь-якапобудована нами модель дозволяє абстрагуватися від реального об'єкту, підміняючи його вивчення дослідженням формальної моделі. Абстрагування в ООП дозволяє виділити основні елементи наочної області, що володіють однаковою структурою і поведінкою. Таке розбиття на класи дозволяє істотно полегшити аналіз і проектування системи.

Інкапсуляція - найважливіший елемент об'єктного підходу. Суть його полягає в приховуванні деталей внутрішньої реалізації. Інкапсуляція робить позитивний вплив на захист даних і істотно підвищує шанси безболісної заміни однієї з частин системи її новою версією за умови збереження інтерфейсу.

Ієрархія допомагає розбити завдання на рівні і поступово її вирішувати, поступово збільшуючи деталізацію розгляду. Ієрархія упорядковує абстракції. На щастя, різні ієрархії можна виявити практично вбудь-якійсистемі. Агрегація і спадковість - приклади таких ієрархій. Вони підкреслюють той факт, що нові абстракції можуть бути створені на основі тих, що вже існують.

Поліморфізм дозволяє мати природні імена і виконувати дії, релевантні ситуації, розбираючись на етапі роботи програми, яким з методів необхідно викликати. Поліморфізм нерозривно пов'язаний із спадкоємством і пізнім скріпленням.

Приклад: ООП і структури зберігання. Стек

Постановка завдання: Необхідно розробити структуру зберігання Стек. Примітки:

•Не враховувати необхідність перерозподілу пам'яті.

•Вважати, що елементи цілого типу. Аналіз і проектування.

Дані:

•MemSize - максимальна кількість елементів.

•DataCount - кількість елементів в стеку.

•pMem - покажчик на пам'ять для зберігання значень. Операції:

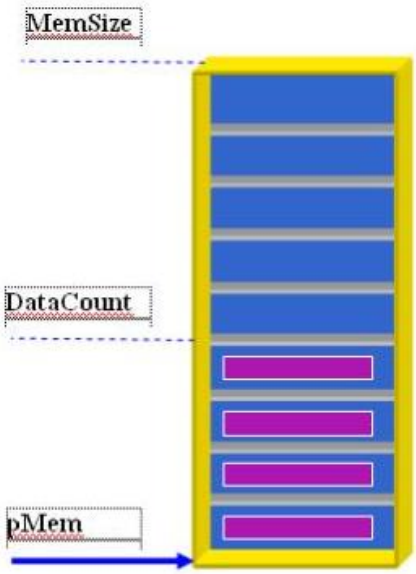
•IsFull - перевірка на повноту.

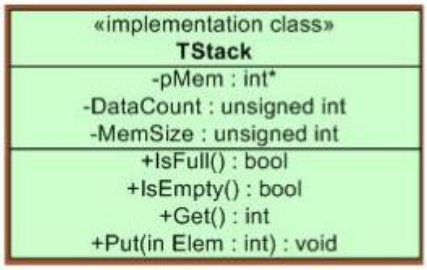
•IsEmpty - перевірка на порожнечу.

•Get - узяти елемент з вершини.

•Put - покласти елемент в стек.

Розглянемо модель і фінальний результат нашого проектування (використовується нотація UML):





Повторне використання

Ідея повторного використання.

Повторне використання - застосування вже існуючих напрацювань в тому, що розробляється ПЗ.

Повторне використання - важливий елемент проектування. Необхідно проектувати нові елементи системи з тим, щоб їх надалі можна було застосовувати при розробці інших систем. Крім того, при проектуванні системи необхідно розглядати можливість використання того, що вже є і працює.

Девіз: не треба винаходити велосипед, якщо він вже винайдений.

Достоїнства повторного використання.

Достоїнства повторного використання (по Соммервілю [1]):

•Підвищення надійності.

•Зменшення проектних рисок.

•Ефективне використання фахівців.

•Дотримання стандартів (приклад: призначений для користувача інтерфейс).

•Прискорення розробки.

Повторне використання досягається за рахунок наступних прийомів (видів

використання):

•Компонентна розробка. Частина компонентів вже розроблені раніше, мають чітко описаний інтерфейс. Вони використовуються як "цегла" в новій системі.

•Використання патернів (шаблонів) проектування. Застосовуються відомі підходи до вирішення деяких проблем, що зустрічалися раніше.

•Використання стандартних прикладних (MKL, MFC) і системних (API) бібліотек.