**Тема 5. Якість програмного забезпечення**

**Забезпечення якості ПЗ. Тестування ПЗ.**

Як уже згадувалося вище, поточний період на ринку програмного забезпечення характеризується переходом від штучного ремісничого виробництва програмних продуктів до їх промислового створення. Відповідно зросли вимоги до якості розроблюваного програмного забезпечення, що потребує вдосконалення процесів їх розробки.

*На даний момент існує кілька стандартів, пов'язаних з оцінкою якості цих процесів, яке забезпечує організація-розробник.*

До найбільш відомих відносять:

міжнародні стандарти серії ISO 9000 (ISO 9000 - ISO 9004);

СММ - Capability Maturity Model - модель зрілості (вдосконалення) процесів створення програмного забезпечення, що запропонована SEI (Software Engineering Institute – інститут програмування при університеті Карнегі-Меллон);

робоча версія міжнародного стандарту ISO / IEC 15504: Information Technology - Software Process Assessment; ця версія більш відома під назвою SPICE - (Software Process Improvement And Capability Determination - визначення можливостей і поліпшення процесу створення програмного забезпечення).

У серії ISO 9000 сформульовані необхідні умови для досягнення деякого мінімального рівня організації процесу, але не дається ніяких рекомендацій щодо подальшого вдосконалення процесів.

**Забезпечення якості ПЗ**

*Якість програмного забезпечення* ***(Software quality)*** асоціація IEEE визначає як ступінь відповідності програмного забезпечення встановленій комбінації властивостей. У Міжнародному стандарті якості ISO 9000:2007 це поняття визначається як сукупність характеристик ПЗ, які забезпечують встановлені та очікувані вимоги.

До характеристик якості ПЗ відносять (рис. 10.1):

* *функціональність* – виконання заявлених функцій, відповідність стандартам, функціональна сумісність, безпека, точність;
* *надійність* – стійкість до відмов, можливість відновлення, завершеність;
* *ефективність* – економія часу, ефективність використання;
* *зручність у використанні* – ергономічність, інтуїтивна зрозумілість, повна документація;
* *зручність для супроводу* – стабільність, придатність для контролю та внесення змін;
* *портативність* – зручність установки, можливість заміни, сумісність.

Якість ПЗ

Функціональність

Надійність

Зручність

у використанні

Ефективність

Зручність

супроводу

Портативність

Рисунок 10.1 – Характеристики якості ПЗ

*Забезпечення якості* ***(Quality assurance, QA)*** *–* сукупність заходів, що охоплюють усі технологічні етапи розроблення, випуску та експлуатації ПЗ інформаційних систем на різних стадіях життєвого циклу ПЗ, для забезпечення його якості.

*При виконанні цих заходів для кожного продукту перевіряються****:***

* повнота та коректність документації;
* коректність процедур встановлення та запуску;
* ергономічність використання;
* повнота тестування.

У 80-х рр. ХХ ст. розмір проектів із створення програмних систем зріс настільки, що вручну стало неможливо тестувати ПЗ, тому активно стали розроблятися засоби автоматизації процесу тестування. Через дестиліття поняття якості ПЗ розширилося настільки, що було виділено окремий вид діяльності при створенні ПЗ – забезпечення якості (Quality assurance, QA). На цей час автоматизоване тестування значно поширене, а засоби автоматизованого тестування часто вбудовані у середовище програмування.

Для виконання заходів забезпечення якості ПЗ розробники часто використовують спеціальні групи контролю якості, які мають назву QA. Група QA всередині компанії фактично виконує роль вимогливого користувача. У деяких компаніях заборонено неформальне спілкування між групою розробників та групою тестувальників. Від відповідальності групи

QA залежить успіх продукту. Якщо ПЗ не сподобалося, з будь-яких причин користувачам продукт назавжди втрачає репутацію і споживача.

Незнайдені на стадії розроблення помилки коштують дорого. Традиційна стратегія розроблення ПЗ підпорядковується фундаментальному правилу, що визначає, що впродовж роботи над проектом вартість внесення змін у створюване ПЗ збільшується за експонентою (рис. 10.2).

Фактично від того, наскільки якісно виконані початкові етапи розроблення ПЗ залежить його вартість. З метою підвищення якості розроблення та уникнення ризиків, пов’язаних із нечіткими або постійно змінюваними вимогами, була створена методологія XP (eXtreme Programming).



Рисунок 10.2 – Залежність вартості змін проектів від часу

**Тестування ПЗ**

*Тестування (Software Testing)* – діяльність, що виконується для оцінювання та поліпшення якості ПЗ. Ця діяльність базується на виявленні дефектів і проблем програмного забезпечення [14]. Тестування ПЗ включає в себе діяльність із планування робіт (Test Management), проектування тестів (Test Design), виконнання тестування (Test Execution) та аналізу отриманих результатів (Test Analysis).

Потрібно відмітити, що програмування та тестування – різні за суттю види діяльності. Якщо програмування – процес синтезу, то тестування – процес аналізу.

Тестування потребує від виконавця в пешу чергу уважності та педантичності. Тестувальник повинен шукати недоліки будь-де в системі.

Якщо початкове тестування для налагодження коду виконує сам програміст, то наступні етапи перевірки повинні готувати і виконувати інші особи, щоб перевірка була повноцінною, а не тільки для очікуваних проблемних місць.

Для планування потрібно мати уявлення про потрібні обсяги тестування. Важливі статистичні дані щодо тестування:

* на 1000 рядків коду програміст у середньому робить 100 помилок, 70% з яких усуваються на стадії налагодження коду;
* при системному тестуванні у відділі контролю якості на ліквідацію однієї помилки потрібно від чотирьох до шістнадцяти годин;
* для усунення помилки, що була виявлена у ході експлуатації, потрібно від 33 до 88 годин.

Ці дані показують, як важливо виявити та усунути проблеми ще на ранніх етапах розроблення. Цікавим підходом для підвищення якості програмування є парне програмування, що зменшує кількість помилок на 15 % порівняно із традиційним одиночним кодуванням.

Найдавнішим прийомом тестування є перевірка усіх вводів та виводів даних, передбачених та неприпустимих (complete testing). Програма повинна адекватно реагувати на помилкові ситуації, без втрати стійкості в роботі. Також при тестуванні потрібно перевірити виконання усіх передбачених функцій системи. Перелік таких тестів складається на ранніх етапах проектування паралельно із описом функцій програмного продукту.

*Види тестування* можна класифікувати за різними показниками:

1. За рівнем знання системи**:**
	* *чорний ящик (Black-box testing)* – без перегляду програмного коду;
	* *білий ящик (White-box testing)* – тестування із вивченням коду програми;
	* *сірий ящик (Gray-box testing)* – тестування із частковим вивченням коду програми.
2. За об**’**єктом тестування**:**
	* *функціональне (Functional testing) –* перевірка виконання заданих функцій;
	* *тестування продуктивності (Performance testing)* – перевірка стабільності роботи програми або її частини при заданому навантаженні (Load testing), при перевантаженні (Stress testing) та тривалому середньому навантаженні (Stability testing);
* *юзабіліті-тестування (Usability testing) –* перевірка ергономічності ПЗ;
* *перевірка інтерфейсу користувача (UI testing);*
* *перевірка безпеки (Security testing) –* тестування роботи системи з точки зору безпеки інформації;
* *тестування сумісноті (Compatibility testing)* – нефункціональне тестування, метою якого є перевірка коректної роботи ПЗ у певному оточенні.
1. За часом виконання тестування**:**
	* *альфа-тестування (Alpha testing)* – перевірка роботи ПЗ перед передачею в експлуатацію силами компанії-розробника;
	* *півгодинне тестування (Smoke testing)* – коротка перевірка функцій системи, як правило, один тест для кожної функції;
	* *тестування нової функціональності (New feature testing);*
	* *регресійне тестуванн (Regression testing)* – перевірка роботи вже протестованих модулів;
	* *тестування під час передачі ПЗ (Acceptance testing);*
	* *бета-тестування (Beta testing)* – тестування ПЗ перед випуском сторонніми силами.
2. За ступенем ізольованості компонентів**:**
	* *компонентне тестування (Component / Unit testing)* – перевірка коректності окремого модуля або компонента системи;
	* *інтеграційне тестування (Integration testing)* – перевірка роботи модулів, об’єднаних у групу;
	* *системне тестування (System / end-to-end testing)* – перевірка роботи зібраного ПЗ з метою встановлення відповідності очікуваним функціям.
3. За ступенем автоматизації**:**
	* ручне тестування (Manual testing);
	* автоматизоване тестуванняя (Automated testing);
	* напівавтоаматизоване тестування (Semiautomated testing).
4. За ступенем підготовленості до тестування**:**
	* *формальне тестування (Formal testing)* – тестування відповідно до плану, встановлених процедур;
	* *інтуїтивне тестування (Ad hoc testing)* – тестування без попереднього плану дозволяє на раніх стадіях виявляти помилки.
5. **7.** За ознакою позитивності сценаріїв**:**
	* перевірка позитивних сценаріїв (Positive testing);
	* перевірка негативних сценаріїв (Negative testing).

**Методи оцінки якості тестування. Види тестів. Верифікація програм.**

 **Методи оцінки якості тестування**

Тестування, як і будь-який процес, повино мати методи оцінки якості тестування.

Одним із способів є спосіб, якому в програму спеціально вставляється певна кількість помилок. Після проведення тестування оцінюється, скільки таких помилок було знайдено. Частка знайдених помилок показує якість тестування і допомагає оцінити, що обсяг робіт потрібно виконати для повного усунення проблем. Плануючи тестування, потрібно визначати очікувані результати тестування, які покажуть готовність створюваного продукту. Бажано затвердити їх із замовником для чіткого визначення показників завершення проекту.

 Якщо проектувальники тестів повинні мати інтегральне уявлення про систему, то виконавці тестів не потребують високої кваліфікації. Часто цю роль виконують новачки у компаніях, які не мають досвіду роботи.

Для підвищення рівня контролю якості кожного проекту повинна формуватися база даних помилок, в яку вноситься така інформація:

* хто знайшов помилку,коли;
* опис помилки;
* модуль, у якому була знайдена помилка;
* версія продукту;
* статус помилки:

− open: знайдена;

− fixed: виправлена;

− can't reproduce: неможливо повторити;

− by design: помилка проектування;

− wont fix: не помилка;

− postponed: зараз виправити важко, буде виправлена у наступній версії;

− regression: виправлена помилка з’явилася знову;

* важливість (severity) помилки:

− сrash: повна втрата даних;

− major problem: програма частково не працює, часткова втрата даних;

− minor problem: програма працює не так, як очікувалось, але дані не втрачаються;

− trivial: зараз не потрібно виправляти;

* пріоритет помилки:

− highest: не можна поставити продукт із такою помилкою, не можна перейти до наступної версії;

− high: поставити не можна, але можна перейти до наступної версії;

− medium: помилка буде виправлена;

− low: косметичні поліпшення – помилка залишиється до наступної версії.

Така база даних дозволяє проаналізувати якість роботи окремих програмістів, їх груп, оцінювати якість проекту та можливості використовуваних інструментальних засобів. Також керівники можуть відслідковувати ризики розроблення за помилками найвищих рівнів пріоритету та важливості.

Дані з бази даних помилок дозволяють приймати рішення про можливість випуску програмного продукту (помилки з важливістю "crash" або з пріоритетом "highest/high" не дозволяють поставляти ПЗ). Якщо ПЗ обов’язково потрібно поставити замовнику, а часу на виправлення помилок немає, за домовленістю можна відкинути функції, які виконуються із помилками.

 **Види тестів**

Одним з найбільш об'єктивних методів оцінки якості програм є її випробування.

*Випробування програми* може проводитися, наприклад, з метою визначення міри відповідності готової програми вимогам, що сформульовані у технічному завданні.

Так само шляхом випробування можуть бути отримані точні оцінки таких параметрів як:

* середній час вирішення завдання;
* максимальний обсяг необхідний оперативної пам'яті;
* показники завантаження зовнішніх пристроїв необхідних для оцінки вартості рішення задачі.

Іншою, не менш важливою, метою проведення випробувань є спрямований пошук помилок. З якою б метою не проводилося випробування програми, одним з найважливіших питань є підбір вхідних даних програми. Такі, спеціально підібрані з певними цілями, вихідні дані називаються тестами**.** Випробування програми тестами – тестування – починається вже в процесі її налагодження.

Перший тестовий приклад пропонується програмою відразу після її трансляції. Далі тестування і налагодження тісно переплітаються між собою. Якщо черговий тест не проходить, тобто програма видає результат, що не відповідає тестовим даним, то виникає проблема знайти і виправити помилку, а це вже налагодження**.**

З іншого боку, для локалізації помилки необхідно вибрати такі вихідні дані, тобто тестові приклади, які б сприяли найбільш рельєфному прояву виявленої, але ще не локалізованої помилки.

*Тестування* – етап процесу створення програми, що полягає в підборі тестових прикладів і випробування на них програми, з метою виявлення в ній помилки.

*Налагодження* – етап процесу створення програми, що полягає в локалізації та виправленні помилок, виявлених у ході трансляції та тестування.

*Етап тестування включає три основних елементи:*

1. генерація тестових прикладів, контроль процесу тестування;
2. аналіз вихідних і проміжних результатів програми;
3. врахування внеску кожного тестового прикладу в процес тестування.

Для першого і другого елементів вихідними даними є документація програми, на основі якої генеруються тестові приклади, що забезпечують режим виконання програми, визначає ступінь відповідності отриманого результату зазначеного в прикладі.

Третій елемент визначається критерієм повноти тестування, який тісно пов'язаний з конкретним способом тестування.

*Тестування ведеться на чотирьох рівнях:*

1. рівень окремого програмного компонента (модуля);
2. рівень сполучень, на якому шукаються помилки між-компонентного інтерфейсу;
3. рівень зовнішніх функцій, на якому шукаються розбіжності програмних функцій і зовнішніх специфікацій програм;
4. комплексне тестування, випробування програмного комплексу на відповідність вихідним цілям.

Цілком природно тестування ув'язується з організацією розробки програм. Тому, два способи утворюють базис, на якому будуються різні методи тестування.

*Висхідне тестування* передбачає, що програма збирається і тестується знизу вгору. Модулі (компоненти самого нижнього рівня) тестуються автономно. Надійність тестування цих модулів визначає успіх подальшого процесу. Далі відбувається перехід до модулів, які звертаються до вже відтестованих модулів. На цьому етапі виникає необхідність перевірки інтерфейсів.

Для реалізації висхідного тестування необхідно для кожного модуля написати невелику керуючу підпрограму - драйвер. У розпорядження драйвера надаються значення вхідних змінних і структури даних. Драйвер послідовно викликає модуль, що тестується, при кожному виклику пропонуючи йому новий тестовий приклад. Основні недоліки:

* + серйозні помилки в специфікаціях, алгоритмах і інтерфейсах можуть проявлятися лише при тестуванні комплексів вищого рівня, тобто на завершальній стадії.
	+ Необхідність розробки драйверів і тестів для кожного рівня тестування, що веде до великого обсягу додаткових програм, які стають даремними при завершенні роботи над комплексом.

*Спадне проектування****.*** Автономно тестується тільки головна програма. По завершенні тестування головного комплексу до нього послідовно приєднуються комплекси та компоненти наступного рівня і т.д., до тих пір, поки не буде зібрана і випробувана вся програма.

Як же тестувати комплекси, в той час як компоненти, що входять в них, ще не перевірені і можливо ще і не написані? Для імітації функцій ще не створених модулів використовуються заглушки, які імітують роботу відсутнього модуля.

Недоліки спадного тестування збігаються з вадами спадного проектування. Підвищуються вимоги до складності і якості заглушок, які потім ліквідуються. Переваги:

* + Метод дозволяє поєднати тестування модуля, тестування сполучень і тестування вхідних функцій.
	+ Рівномірний розподіл роботи з тестування протягом усього періоду створення комплексу - це дозволяє виявити помилки в головному модулі на ранній стадії розробки.

На практиці рідко вдається використовувати один спосіб, існує ряд комбінованих способів.

*Методи тестування компонентів*

Перший метод розглядає тестування програм як «чорний ящик», при цьому внутрішня структура програми (компонента) не враховується, тести будуються на підставі функціональних властивостей програми, тобто спираючись на її функціональні специфікації. Такий підхід називається *функціональним тестуванням****.***

*Структурне тестування****,*** при його використанні враховується

внутрішня структура програми. Аналіз проходження даних від входу до виходу, ця інформація використовується для раціональної організації тестування.

*Для оцінки повноти тестування використовують три критерії:*

1. тестування вважається закінченим, якщо кожен оператор був виконаний хоча б раз;
2. тестування вважається закінченим, якщо в процесі вирішення тестових прикладів по кожній дузі блок-схеми програми був здійснений хоча б один перехід;
3. тестування закінчується, якщо в процесі вирішення тестового прикладу кожен шлях від входу до виходу пройдений хоча б раз.

*Налагодження -* процес пошуку та усунення помилок у програмі, що спирається на результати самої програми, повідомлення транслятора, і оперативної системи. Важливою особливість процесу налагодження є можливість робити експерименти на ЕОМ з метою виявлення помилок. Але експеримент повинен вестися в суворій відповідності з планом.

Процес налагодження полягає в багаторазовому повторенні трьох етапів:

* 1. виявлення помилки;
	2. локалізація помилки;
	3. виправлення помилки.

*Виявлення помилки* при налагодженні здійснюється шляхом прорахунку на ЕОМ спеціально підібраних завдань, результати вирішення яких заздалегідь відомі. Якщо контрольний приклад вирішено правильно, то підбирається більш складне завдання. Якщо програма не йде, то в ній є, принаймні, одна помилка.

Наступний етап полягає у встановленні точного місця знаходження помилки. *Локалізація помилок* є процес вирішення неправильних завдань.

Етап *виправлення помилки* є найбільш простим, але основна складність не внести нову помилку при виправленні.

 **Верифікація програм**

Особливістю програмного продукту є практична неможливість в нетривіальних випадках здійснити всебічне і повне його випробування з метою виявлення помилок.

Відомий вислів Дейкстри говорить про те, що експериментальне тестування програм може служити доказом наявності в них помилок, але ніколи не доведе їх відсутність. Тому природно прагнення програмістів знайти можливість формулювати і доводити деякі твердження стосовно правильності створеної програми подібно до того, як у математиці формулюються і доводяться теореми і рішення.

Верифікація – *ідея математичного доказу коректності програм.*

Верифікація зазвичай зводиться до доказу того факту, що програма є коректною відносно її вхідної і вихідної специфікацій.

Найбільш відомий метод називається методом індуктивних тверджень**.**

У цьому методі перший крок полягає в записі тверджень щодо властивостей вхідних і вихідних даних програми, а також результатів у ряді проміжних точок, що називаються точками розрізу**.** Ці твердження формулюються в деякій формально-логічній системі. На основі цих тверджень і семантики операторної схеми програми шляхом певних перетворень формулюються *верифікаційні умови****,*** а потім ці умови доводяться. Якщо вони виявляються істинними, то програма коректна щодо вхідних і вихідних даних.

Якщо довести істинність умов неможливо, то або в програмі є помилки, або помилки є у процедурі доказу (наприклад, хибне твердження в деякій точці розрізу).

Якщо програміст хоче написати правильну програму, йому необхідно виконати такі етапи:

* написати програму;
* визначити вхідні і вихідні умови для цієї програми;
* розрізати цикли і забезпечити кожну точку розрізу індуктивним затвердженням;
* отримати верифікаційні умови;
* довести істинність верифікаційних умов; якщо це не вдається, то не виключено, що якесь індуктивне твердження записано невірно або програма містить помилку.

Очевидно, що методи верифікації знайдуть широке застосування в практиці у тому випадку, якщо вдасться їх формувати настільки явно, що можна буде здійснювати верифікацію програм за допомогою ЕОМ.

Використання в практиці програмування систем доказу правильності програм ефективно лише за допомогою ЕОМ.

Методи математичного доказів теорем в даний час інтенсивно розвиваються і удосконалюються. Їх застосування в програмуванні сприяє подальшому прогресу у цій галузі.

Ідеї доказу правильності програм роблять свій вплив на загальну культуру програмування. Необхідність формального опису вхідних і вихідних даних спонукає програміста чітко визначати інтерфейси між модулями програми. Механізм тверджень являє собою чудовий засіб специфікацій модулів. Далі, намагаючись винайти індуктивні твердження, програміст змушений більш ретельно і глибоко аналізувати свою програму і таким чином виявляти в ній помилки.