Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни

«ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ ЯКОСТІ ПРОГРАМНИХ ЗАСОБІВ»

для студентів спеціальності
152 «Метрологія та інформаційно-вимірювальна техніка»
спеціалізацій «Метрологія та вимірювальна техніка»,
«Метрологічне забезпечення випробувань та якості продукції»,
«Якість, стандартизація та сертифікація»

Електронне видання

ЗАТВЕРДЖЕНО
кафедрою МТЕ,
протокол № 2
від 2 жовтня 2017 р.

Харків 2017

Конспект лекцій з дисципліни «Тестування та оцінювання якості програмних засобів» для студентів спеціальності 152 «Метрологія та інформаційно-вимірювальна техніка» спеціалізацій «Метрологія та вимірювальна техніка», «Метрологічне забезпечення випробувань та якості продукції», «Якість, стандартизація та сертифікація» [Електронне видання] / Упоряд. Запорожець О.В. – Харків: ХНУРЕ, 2017. – 99 с.

Упорядник Запорожець О.В.

Рецензент: А.Б. Єгоров, к.т.н., проф. каф. МТЕ ХНУРЕ

ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ ЯКОСТІ ПРОГРАМНИХ ЗАСОБІВ

Література:

1. Кулаков, А. Ф. Управление качеством программных средств ЭВМ. – К.: Техніка, 1989. – 216 с.
2. Кулаков, А. Ф. Оценка качества программ ЭВМ. – К.: Техніка, 1984. – 167 с.
3. Майерс Г. Надежность программного обеспечения. – М.: Мир, 1980. – 360 с.
4. Майерс Г. Искусство тестирования программ. – М.: Финансы и статистика, 1982. – 176 с.
5. Канер С., Фолк Д., Нгуен Е.К. Тестирование программного обеспечения. Фундаментальные концепции бизнес-приложений. – К.: ДиаСофт, 2001. – 544 с.
6. Бейзер Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. – СПб.: Питер, 2004. – 318 с.
7. Тамре Л. Введение в тестирование програмного обеспечения. – М.: Издательский дом «Вильямс», 2003. – 368 с.
8. Басок Б.М., Кунин А.П., Красовский В.Е. Тестирование программных средств. Учебное пособие / Государственное образовательное учреждение высшего профессионального образования «Московский государственный институт радиотехники, электроники и автоматики (технический универсистет)». – М., 2005. – 84 с.
9. http://www.intuit.ru Основы тестирования программного обеспечения
10. http://www.istqb.org/downloads/syllabi/SyllabusFoundation.pdf Certified Tester. Foundation Level Syllabus

11. ДСТУ 2844-94 Програмні засоби ЕОМ. Забезпечення якості. Терміни та визначення. – К.: Держстандарт України, 1994. – 20 с.
12. ДСТУ 2850-94 Програмні засоби ЕОМ. Показники і методи оцінювання якості. – К.: Держстандарт України, 1994. – 20 с. (ISO 9126)
13. ДСТУ 2851-94 Програмні засоби ЕОМ. Документування результатів випробувань. – К.: Держстандарт України, 1994. – 11 с.
14. ДСТУ 2853-94 Програмні засоби ЕОМ. Підготовлення і проведення випробувань. – К.: Держстандарт України, 1994. – 18 с.
15. ДСТУ 3918-1999 (ISO/IEC 12207:1995) Інформаційні технології. Процеси життєвого циклу програмного забезпечення

Лекція 1
СТАНДАРТИ НА ПРОГРАМНУ ПРОДУКЦІЮ. ВИДИ ПРОГРАМНИХ ДОКУМЕНТІВ

Комп'ютеризація всіх сфер людської діяльності – це реальність сучасної економіки, прогрес якої багато в чому залежить від розвитку комп'ютерних і інформаційних технологій, включаючи ЕОМ, програмне забезпечення, професійний рівень фахівців, що розробляють і використовують обчислювальну техніку. Особливе місце тут займає програмне забезпечення. У вартості комп'ютерних систем вартість програмного забезпечення становить основну частку (до 80% і більше). Витрати на розробку програмного забезпечення безупинно ростуть. У США, наприклад, щорічно в розробку програмного забезпечення вкладається понад 20 млрд. доларів. Причина цього очевидна. Без програми ЕОМ – це дорогий, але ні до чого непридатний комплект пристроїв. Але не всі програми роблять комп’ютер корисними. Для ефективного використання ЕОМ у конкретній галузі діяльності необхідно мати відповідний асортимент високоякісних програм.

Поняття процесу програмування і його об’єкта розробки – програми якісно змінилися за останні десятиліття. Виробництво програм набуло масового характеру, істотно збільшилися їхній середній обсяг і складність. Розробка програмних комплексів стала вимагати значних зусиль великих колективів фахівців. Програми перестали бути тільки обчислювальними й почали виконувати найважливіші функції по керуванню і обробці інформації в різних галузях життєдіяльності. Принципово змінився і сам підхід до програмування. Програмні комплекси не розглядаються більше лише як результат наукової творчості і мистецтва. Домінуючим став підхід до програм як до об'єкта, що є результатом складного технологічного процесу – програмним продуктом. З’явилася необхідність визначати якість цього продукту і витрати праці, що дозволяють досягти необхідного рівня якості. Зросла необхідність у методах і засобах, що забезпечують поєднання високої якості створюваного програмного продукту з короткими строками розробки і досить високою продуктивністю праці колективів фахівців.

Створення комплексів програм на сьогоднішній день набуло характеру промислового виробництва. У результаті зростає роль технології розробки програм і різних засобів підтримки технологічного процесу їхнього проектування.

Розвиток і застосування технологій проектування комплексів програм призводить до необхідності вимірювання і порівняння їхньої ефективності насамперед по ступеню впливу на якість програмного продукту. Показники якості програм підлягають вимірюванню та чисельній оцінці, для чого вони формалізуються уведенням відповідних метрик. Застосування метрик до комплексів програм дозволяє впорядкувати їхню розробку, випробування, експлуатацію й супровід. Взаємодія розробника і замовника (або користувача) здобуває більше чіткий характер, і якість програм може оцінюватися кількісно.

Необхідно відзначити, що питанням оцінки ефективності і якості програмного забезпечення присвячено досить багато досліджень як вітчизняних, так і закордонних учених. Серед них можна виділити роботи А.Ф. Кулакова, В.В. Ліпаєва, Г. Майєрса і інших авторів. На сьогоднішній день в Україні вже існують нормативні документи, що регламентують питання оцінювання рівня якості й керування якістю програмних засобів.

1.1 Основні терміни і визначення в галузі програмних засобів (ДСТУ 2844-94)

Програма (program) – послідовність інструкцій, які може виконати ЕОМ.

Програмний засіб, ПЗ (software) – взаємопов'язана сукупність програм, процедур, правил, документації та даних, що належить до функціонування обчислювальної системи.

Програмний продукт (software product) – програмний засіб, призначений для поставки користувачеві.

Програмна продукція (software production) – сукупність програм, програмних засобів і програмних продуктів, які мають загальну класифікаційну ознаку (за належністю, місцем розробки, призначенням тощо).

Технологія програмування (software engineering) – сукупність правил, методів і засобів, а також їх опис, призначених для використання на етапах життєвого циклу ПЗ.

Користувач ПЗ (software user) – юридична чи фізична особа, яка застосовує ПЗ чи бере участь у діяльності, що прямо чи непрямо залежить від функціонування даного ПЗ.

Середовище функціонування ПЗ (environment) – заданий клас необхідних і достатніх умов функціонування ПЗ, який характеризується вимогами до технічних, організаційних, програмних та інформаційних аспектів.

Життєвий цикл ПЗ (software life cycle) – сукупність окремих етапів робіт, що проводяться у заданому порядку протягом періоду часу, який починається з вирішення питання про розроблення ПЗ і закінчується припиненням використання ПЗ.

Відмова ПЗ (failure) – подія, під час якої проявляється непрацездатність ПЗ. Ознаки непрацездатності встановлюються в нормативно-технічній документації ПЗ.

Помилка ПЗ (error) – запис елемента програми чи тексту програмної документації, використання яких призводить або може призвести до неправильного результату.

Виконання ПЗ (execution) – процес перетворення обчислювальною машиною вихідних даних згідно з директивами, які задані в ПЗ.

1.2 Види програмних документів (ГОСТ 19.101-77)

На основі комплексної стандартизації в Україні розроблені системи стандартів, кожна з яких охоплює певну сферу діяльності, проведеної в загальнодержавному масштабі або в певних галузях народного господарства.

Нормативно-технічну і організаційно-методичну основу виробництва конкретних видів, типів, груп продукції становлять галузеві системи стандартів, що регламентують технічні характеристики, вимоги до якості і надійності виробів, способи і методи досягнення та контролю цих вимог і ін. Галузеві системи включають також комплекси стандартів на терміни, визначення і позначення, що застосовуються в галузі.

Єдині державні системи стандартів забезпечують однаковість і найвищу ефективність проведення найважливіших видів робіт, спільних для різних галузей народного господарства. До таких систем належить *Єдина система програмної документації* (ЄСПД), яка призначена для встановлення єдиних правил виконання, оформлення і обігу програмної документації, які забезпечують:

– можливість взаємного обміну програмними документами між різними організаціями без їхнього переоформлення;

– зниження витрат на розробку і оформлення програмної документації за рахунок використання єдиної класифікації програм і програмних документів і єдиних стадій їхньої розробки;

– зниження витрат під час користування програмними документами внаслідок застосування їхніх єдиних раціональних форм або структури умовних позначок і правил оформлення.

Номера стандартів ЄСПД починаються на 19.

До програмних належать документи, що містять відомості, необхідні для розроблення, виготовлення, експлуатації і супроводу програмних засобів. Основні програмні документи, передбачені ГОСТ 19.101-77, наведено в табл. 1.1.

Таблиця 1.1 – Програмна документація

Вид програмного документа	Зміст програмного документа	Стандарт, що регламентує зміст і правила оформлення
Основні документи		
Специфікація	Склад програми та документації на неї	ГОСТ 19.202-78
Відомість тримачів оригіналів	Перелік підприємств, на яких зберігають оригінали програмних документів	ГОСТ 19.403-79
Текст програми	Запис програми з необхідними коментарями	ГОСТ 19.401-78
Опис програми	Відомості про логічну структуру та функціонування програми	ГОСТ 19.402-78
Програма і методика випробувань	Вимоги, які підлягають перевірці при випробуванні програми, а також порядок і методи їх контролю	ГОСТ 19.301-79

Продовження табл. 1.1

Вид програмного документа	Зміст програмного документа	Стандарт, що регламентує зміст і правила оформлення
Технічне завдання	Призначення і область застосування програми, технічні, техніко-економічні і спеціальні вимоги, що висуваються до програми, необхідні стадії і строки розробки, види випробувань	ГОСТ 19.201-78
Пояснювальна записка	Схема алгоритму, загальний опис алгоритму і функціонування програми, а також обгрунтування прийнятих технічних і техніко-економічних рішень	ГОСТ 19.404-79
Експлуатаційні документи		
Відомість експлуа-таційних документів	Перелік експлуатаційних документів	ГОСТ 19.507-79
Формуляр	Основні характеристики програми, комплектність і відомості про експлуатацію програми	ГОСТ 19.501-78
Опис застосування	Відомості про призначення програми, область застосування, застосовані методи, класи задач, що розв’язуються, обмеження для застосування, мінімальну конфігурацію технічних засобів	ГОСТ 19.502-78
Настанова системного програміста	Відомості для перевірки, забезпечення функціонування і настроювання програми на умови конкретного застосування	ГОСТ 19.503-79
Настанова програміста	Відомості для експлуатації програми	ГОСТ 19.504-79
Настанова оператора	Відомості для забезпечення процедури спілкування оператора з обчислювальною системою в процесі виконання програми	ГОСТ 19.505-79
Опис мови	Опис синтаксису і семантики мови	ГОСТ 19.506-79
Настанова з технічного обслуговування	Відомості для застосування тестових і діагностичних програм при обслуговуванні технічних засобів	ГОСТ 19.508-79

1.3 Контрольні запитання і завдання

1. Дайте визначення програми та програмного засобу.
2. Що таке програмний продукт?
3. Хто може бути користувачем програмного засобу?
4. Дайте визначення життєвого циклу програмного засобу.
5. Що таке помилка програмного засобу?
6. Назвіть основні та експлуатаційні програмні документи за ЄСПД.

Лекція 2
ПРОЦЕСИ ЖИТТЄВОГО ЦИКЛУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (ДСТУ 3918-1999 (ISO/IEC 12207:1995))

2.1 Основні процеси життєвого циклу ПЗ

До основних процесів належать:

1) *процес замовляння* – визначає дії замовника – організації, що замовляє систему, програмний продукт або програмну послугу;

2) *процес постачання* – визначає дії постачальника – організації, що надає систему, програмний продукт або програмну послугу;

3) *процес розроблення* – визначає дії розробника – організації, що визначає і проектує програмний продукт;

4) *процес експлуатації* – визначає дії оператора – організації, що надає послуги по експлуатації комп'ютерної системи в її існуючому середовищі для її користувачів;

5) *процес супроводу* – визначає дії супроводжувача – організації, що надає послуги по супроводу програмного продукту, тобто, керує внесенням змін у програмний продукт для підтримки його в належному та працездатному стані. Цей процес передбачає перенесення та вилучення програмного продукту.

Основні процеси складаються з таких дій:

1) процес замовляння:

- ініціювання;
- підготовка запиту щодо пропозицій (тендеру);
- підготовка й коригування контракту;
- нагляд за постачанням;
- приймання і завершення;

2) процес постачання:

- ініціювання;
- підготовка відповіді;
- укладення контракту;
- планування;
- виконання та контроль;
- перегляд та оцінювання;
- постачання і завершення;

3) процес розроблення:

- реалізація процесу;
- аналіз системних вимог;
- проектування архітектури системи;
- аналіз вимог до програмного забезпечення;

- проектування архітектури програмного забезпечення;
- розробка детального проекту програмного забезпечення;
- кодування та тестування програмного забезпечення;
- інтеграція програмного забезпечення;
- кваліфікаційне тестування програмного забезпечення;
- системна інтеграція;
- кваліфікаційне тестування системи;
- інсталяція (установка) програмного забезпечення;
- забезпечення приймання програмного забезпечення;

4) процес експлуатації:

- процес реалізації;
- експлуатаційні випробування;
- експлуатація системи;
- підтримка користувачів;

5) процес супроводу:

- реалізація процесу;
- аналіз проблеми та модифікації;
- реалізація модифікації;
- перегляд/приймання супроводу;
- перенесення;
- вилучення програмного забезпечення.

2.2 Процеси підтримки життєвого циклу ПЗ

До процесів підтримки належать:

1) *процес документування* – це процес фіксування інформації, що створюється в рамках життєвого циклу або діяльності;

2) *процес конфігураційного керування* – це процес, який полягає у застосуванні протягом усього життєвого циклу програмного забезпечення адміністративних та технічних процедур до: ідентифікації, визначення та встановлення базису елементів програмного забезпечення у системі; керування модифікаціями та версіями елементів; фіксування та звітування про стан елементів та запитів щодо модифікації; забезпечення повноти, несуперечливості та коректності елементів; керування зберіганням, опрацюванням та наданням елементів;

3) *процес забезпечення якості* – це процес забезпечення адекватної оцінки того, що програмні продукти та процеси протягом життєвого циклу проекту задовольняють визначені вимоги, а відповідні плани витримуються;

4) *процес верифікації* – це процес визначення того, чи задовольняють програмні продукти, що є результатом деяких дій, вимоги та умови, покладені на них попередніми діями;

5) *процес валідації* – це процес визначення того, чи відповідають вимоги та кінцева, побудована система або програмний продукт установленому для них призначенню;

6) *процес спільного перегляду* – це процес оцінювання стану діяльності та продуктів, одержаних у результаті виконання дій в рамках проекту, який здійснюється відповідним чином. Спільні перегляди здійснюються як на рівні керування, так і на технічному рівні проекту протягом дії контракту. Цей процес може застосовуватись будь-якими двома учасниками, один з яких здійснює перегляд дій іншого учасника;

7) *процес аудиту* – це процес встановлення належної узгодженості з вимогами, планами та контрактом. Цей процес може застосовуватись будь-якими двома учасниками, один з яких здійснює аудит програмних продуктів або дій іншого учасника;

8) *процес вирішення проблем* – це процес здійснення аналізу та вирішення проблем (включно з неузгодженостями), незалежно від їх природи або джерела, які виявляються під час розроблення, експлуатації, супроводу або інших процесів. Метою процесу є забезпечення вчасних, відповідальних та задокументованих способів пересвідчитися в тому, що всі виявлені проблеми проаналізовано та вирішено і тенденції визначено.

Процеси підтримки складаються з таких дій:

1) процес документування:

- реалізація процесу;
- проектування та розроблення;
- випуск;
- супровід;

2) процес конфігураційного керування:

- реалізація процесу;
- ідентифікація конфігурації;
- контроль за конфігурацією;
- облік стану конфігурації;
- оцінка конфігурації;
- керування та надання версій;

3) процес забезпечення якості:

- реалізація процесу;
- забезпечення продукту;
- забезпечення процесу;
- забезпечення якості системи;

4) процес верифікації:

- реалізація процесу;

- верифікація;

5) процес валідації:

- реалізація процесу;
- валідація;

6) процес спільного перегляду:

- реалізація процесу;
- перегляди керування проектом;
- технічні перегляди;

7) процес аудиту:

- реалізація процесу;
- аудит;

8) процес вирішення проблем:

- реалізація процесу;
- вирішення проблеми.

2.3 Організаційні процеси життєвого циклу ПЗ

До організаційних процесів належать:

1) *процес керування* складається з загальних дій та завдань, які можуть застосовуватись будь-яким учасником для керування відповідними процесами, які він проводить;

2) *процес створення та супроводу інфраструктури* – це процес запровадження та супроводу інфраструктури, необхідної для будь-якого іншого процесу. Інфраструктура може містити апаратне забезпечення, програмне забезпечення, інструментальні засоби, методи, стандарти та обладнання для розроблення, експлуатації або супроводу;

3) *процес удосконалення* – це процес встановлення, оцінювання, вимірювання, контролю та удосконалення процесу життєвого циклу програмного забезпечення;

4) *процес навчання* – це процес забезпечення та супроводу навчання персоналу.

Організаційні процеси складаються з таких дій:

1) процес керування:

- введення та визначення області застосування;
- планування;
- виконання та контроль;
- перегляд і оцінка;
- припинення;

2) процес створення та супроводу інфраструктури:

- реалізація процесу;
- встановлення інфраструктури;
- супровід інфраструктури;

3) процес удосконалення:

- реалізація процесу;
- оцінювання процесу;
- удосконалення процесу;

4) процес навчання:

- реалізація процесу;
- розробка навчальних матеріалів;
- реалізація плану навчання.

2.4 Контрольні запитання і завдання

1. Назвіть основні процеси життєвого циклу програмних засобів.
2. З яких дій складається процес розроблення програмного засобу?
3. З яких дій складається процес експлуатації програмного засобу?
4. З яких дій складається процес супроводу програмного засобу?
5. Назвіть процеси підтримки життєвого циклу програмних засобів.
6. Назвіть організаційні процеси життєвого циклу програмних засобів.

Лекція 3
УПРАВЛІННЯ ЯКІСТЮ ПРОГРАМНОЇ ПРОДУКЦІЇ

3.1 Терміни і визначення в галузі управління якістю (ISO 9000)

Якість – сукупність характеристик об'єкта, що стосуються його здатності задовольняти встановлені чи передбачувані потреби.

ПРИМІТКИ:

1. При укладанні контракту або в регламентованому навколишньому середовищі, напр. в галузі безпеки ядерних установок, потреби чітко встановлюються, тоді як в інших умовах можливі потреби повинні бути виявлені і визначені.
2. В багатьох випадках потреби можуть змінюватися з часом; це потребує проведення періодичного аналізу вимог до якості.
3. Зазвичай потреби переводяться в характеристики на основі встановлених критеріїв (див. вимоги до якості). Потреби можуть включати, наприклад, такі аспекти, як експлуатаційні характеристики, функціональна придатність, надійність (готовність, безвідмовність, ремонтопридатність), безпека, навколишнє середовище (див. вимоги суспільства), економічні та естетичні вимоги.
4. Для вираження найвищого ступеня в порівнювальному чи кількісному смислі при проведенні технічних оцінок термін "якість" не використовується ізольовано. Щоб виразити ці значення, повинен застосовуватись якісний прикметник. Наприклад, можуть використовуватися такі терміни:
    - "відносна якість", коли об'єкти класифікуються залежно від їх ступеню переваги або в порівнювальному смислі [не плутати з градацією (класом, сортом)];
    - "рівень якості" в кількісному смислі (застосовується при статистичному приймальному контролі) і "міра якості", коли проводяться точні технічні оцінки.
5. Досягнення задовільної якості включає всі стадії петлі якості як єдине ціле. Вклад в якість цих різних стадій іноді ідентифікується окремо з метою їх виокремлення, наприклад, якість, зумовлена потребами, якість, зумовлена проектуванням продукції, якість, зумовлена відповідністю.
6. В деяких довідкових джерелах якість позначається як "придатність для використання" або "відповідність меті", або "задоволення потреб споживача", або "відповідність вимогам". Все це являє собою тільки деякі сторони якості, визначеної вище.

Вимоги до якості – вираження окремих потреб або їх переведення в набір кількісно чи якісно встановлених вимог до характеристик об'єкта, щоб дати можливість їх реалізації і перевірки.

ПРИМІТКИ:

1. Суттєво, щоб вимоги до якості повністю відображали встановлені і передбачувані потреби споживача.
2. Термін "вимога" охоплює ринкові і контрактні вимоги, а також внутрішні вимоги організації. Вони можуть бути розроблені, деталізовані і актуалізовані на різних етапах планування.
3. Задані кількісні вимоги до характеристик включають, наприклад, номінальні значення, відносні значення, граничні відхилення і допуски.
4. Вимоги до якості повинні бути сформульовані на початковій стадії в функціональних термінах і документально оформлені.

Управління якістю – методи та види діяльності оперативного характеру, що використовуються для виконання вимог до якості.

ПРИМІТКИ:

1. Управління якістю включає методи і види діяльності оперативного характеру, направлені як на управління процесом, так і на усунення причин незадовільного функціонування на всіх етапах петлі якості для досягнення економічної ефективності.
2. Деякі дії з управління якістю і забезпечення якості взаємозв'язані

Оцінювання якості – систематична перевірка того, наскільки об'єкт здатен виконувати встановлені вимоги.

ПРИМІТКИ:

1. Оцінювання якості може виконуватися з метою визначення можливості постачальника в галузі якості. В цьому випадку, в залежності від конкретних умов, результат оцінювання якості може бути використаним з метою кваліфікації, схвалення. реєстрації або акредитації.
2. З терміном "оцінювання якості" може використовуватись додатковий визначник в залежності від галузі діяльності (наприклад, процес, персонал, система) і часу (наприклад, до контракту) оцінювання якості, такий, як "передконтрактне оцінювання якості процесу".
3. Загальне оцінювання якості постачальника може також включати оцінювання фінансових і технічних ресурсів.

Забезпечення якості – всі заплановані і систематично здійснювані види діяльності в рамках системи якості, а також підтверджувані (якщо це потрібно), необхідні для створення достатньої впевненості в тому, що об'єкт буде виконувати вимоги до якості.

ПРИМІТКИ:

1. Існують як зовнішні, так і внутрішні цілі забезпечення якості:
    - внутрішнє забезпечення якості: в рамках організації забезпечення якості створює впевненість у керівництва;
    - зовнішнє забезпечення якості: в контрактних чи інших ситуаціях забезпечення якості створює впевненість у споживача або інших осіб.
2. Деякі дії з управління якістю і забезпечення якості взаємозв'язані.
3. Якщо вимоги до якості не відбивають повністю потреби користувача, забезпечення якості може не створити достатньої впевненості.

Система якості – сукупність організаційної структури, методик, процесів і ресурсів, необхідних для здійснення загального управління якістю.

ПРИМІТКИ:

1. Масштаби системи якості повинні відповідати цілям в галузі якості.
2. Система якості організації призначена перш за все для задоволення внутрішніх потреб управління організацією. Вона ширше, ніж вимоги конкретного споживача, який оцінює тільки ту частину системи якості, яка відноситься до цих вимог.
3. У зв'язку з вимогами контракту або обов'язковими розпорядженнями по проведенню оцінювання якості може бути затребований наочний доказ застосування певних елементів системи якості.

Загальне управління якістю – підхід до управління організацією, націлений на якість, заснований на участі всіх її членів і направлений на досягнення довгострокового успіху шляхом задоволення вимог споживача і вигоди для членів організації і суспільства.

ПРИМІТКИ:

1. "Всі члени" означає персонал в усіх підрозділах і на всіх рівнях організаційної структури.
2. Сильне і наполегливе керівництво з боку вищої адміністрації, навчання і підготовка всіх членів організації є суттєвим моментом для успішної реалізації цього підходу.
3. При загальному управлінні якості концепція якості має відношення до досягнення всіх цілей управління.
4. "Вигоди для суспільства" мають на увазі виконання вимог суспільства.
5. Total quality management (TQM) (загальне управління якістю) або його складові частини іноді називають "total quality" ("загальна якість"), "CWQC" (company wide quality control) ("управління якістю в масштабах компанії") "TQC" (total quality control) ("загальне управління якістю") і т. ін.

Петля якості – концептуальна модель взаємозалежних видів діяльності, що впливають на якість на різних стадіях від визначення потреб до оцінювання їх задоволення.

ПРИМІТКА: Спіраль якості є аналогічним поняттям.

Програма якості – документ, що регламентує конкретні заходи в галузі якості, ресурси і послідовність діяльності, що відноситься до специфічної продукції, проекту або контракту.

ПРИМІТКИ:

1. Програма якості зазвичай містить посилання на частини настанови з якості, що застосовуються до окремих випадків.
2. В залежності від призначення програми вона іноді називається "програма забезпечення якості" або "програма адміністративного управління якістю".

Настанова з якості – документ, що викладає політику в галузі якості і описує систему якості організації.

ПРИМІТКИ:

1. Настанова з якості може охоплювати всю діяльність організації або тільки її частину. Найменування і галузь діяльності певної настанови відображає сферу її застосування.
2. Настанова з якості зазвичай містить або, принаймі, посилається на:
    - політику в галузі якості;
    - відповідальність, повноваження і взаємовідносини персоналу, який здійснює керівництво, виконує, перевіряє або аналізує роботу, що впливає на якість;
    - методики системи якості і інструкції;
    - положення з перегляду і коригування настанови.
3. Настанова з якості може розрізнятися за обсягом і форматом, з урахуванням потреб організації. Вона може складатися з декількох документів. І в залежності від призначення настанови вона іноді називається "настанова з забезпечення якості" або "настанова з адміністративного управління якістю".

Процес – сукупність взаємозв'язаних ресурсів і діяльності, яка перетворює вхідні елементи у вихідні.

ПРИМІТКА: До ресурсів можуть відноситись: персонал, засоби обслуговування, обладнання, технологія і методологія.

Методика – установлений спосіб здійснення діяльності.

ПРИМІТКИ:

1. В багатьох випадках методики документуються (наприклад, методики системи якості).
2. Коли якась методика документується, краще використовувати термін "письмова методика" або "документальна методика".
3. Письмова або документальна методика зазвичай включає: цілі і сферу діяльності; що повинно бути зроблено і ким; коли, де і як це повинно бути зроблено; які матеріали, документи і яке обладнання повинне використовуватись; і яким чином це повинно бути проконтрольовано і зареєстровано.

Невідповідність – невиконання встановленої вимоги.

ПРИМІТКА: Це визначення включає відсутність однієї чи декількох характеристик якості (в тому числі надійності), або елементів системи якості, або їх відхилення від встановлених вимог.

Усунення невідповідності – дія, яка здійснюється відносно невідповідного об'єкта з метою усунення невідповідності.

ПРИМІТКА: Ця дія може здійснюватись в формі, наприклад, такої корекції, як ремонт, переробка, переведення в більш низьку категорію, перетворення в брухт, дозвіл на відступ від вимог і внесення поправки в документ або вимогу.

Попереджувальна дія – дія, здійснена для усунення причин потенціальної невідповідності, дефекту або іншої небажаної ситуації з тим, щоб попередити їх появу.

ПРИМІТКА: Ці попереджувальні дії можуть спричинити зміни як в методиках, так і в системах з метою досягнення покращення якості.

Коригувальна дія – дія, здійснена для усунення причин існуючої невідповідності, дефекту або іншої небажаної ситуації з тим, щоб попередити їх повторну появу.

ПРИМІТКИ:

1. Ці коригувальні дії можуть спричинити зміни як в методиках, так і в системах з метою досягнення покращення якості на всіх етапах петлі якості.
2. Існує різниця між "корекцією" і "коригувальною дією":
    - термін "корекція" має відношення до ремонту, переробки або регулювання і відноситься до усунення наявної невідповідності;
    - термін "коригувальна дія" відноситься до усунення причин невідповідності.

3.2 Основні принципи управління якістю

Для того, щоб успішно керувати організацією та забезпечити її функціонування, необхідно направити і контролювати її діяльність систематично та відкрито. Успіху можна досягти завдяки впровадженню і актуалізуванню певної системи управління, розробленої для постійного поліпшення показників діяльності, з урахуванням потреб всіх зацікавлених сторін. Управління організаційно охоплює управління якістю поряд з іншими аспектами управління.

Установлено вісім принципів управління якістю, які вище керівництво може використовувати для поліпшення показників діяльності організації:

1) *орієнтація на замовника*. Організації залежать від замовників і тому повинні розуміти поточні і майбутні потреби замовників, виконувати їхні вимоги і прагнути до перевищення їхніх очікувань;

2) *лідерство керівництва*. Керівники встановлюють єдність мети й напрямків діяльності організації. Їм потрібно створювати і підтримувати таке внутрішнє середовище, у якому працівники можуть бути повністю залучені до виконання завдань, які стоять перед організацією;

3) *залучення працівників всіх рівнів*. Працівники всіх рівнів становлять основу організації, і їхнє повне залучення дає можливість використовувати їхні здібності на користь організації;

4) *процесний підхід*. Бажаного результату досягають ефективніше, якщо діяльністю й пов’язаними з нею ресурсами управляють як процесом. При цьому діяльність організації розглядають як сукупність взаємозв’язаних процесів;

5) *системний підхід до управління*. Ідентифікація, розуміння і управління взаємозалежними процесами як системою сприяє організації в більше результативному та ефективному досягненні її цілей;

6) *постійне поліпшення*. Постійне поліпшення діяльності організації в цілому варто вважати незмінною метою організації;

7) *прийняття рішень на підставі фактів*. Ефективні рішення приймають на підставі аналізування даних і інформації;

8) *взаємовигідні відносини з постачальниками*. Організація і її постачальники є взаємозалежними, і вигідні відносини підвищують можливість обох сторін створювати цінності.

Ці вісім принципів управління якістю формують основу стандартів на системи керування якістю, які входять у стандарти серії ISO 9000.

3.3 Петля якості. Цикл Шухарта-Демінга

Міжнародний ISO 9000 стандарт відстоює прийняття підходу з позицій процесу при розробці, впровадженні та підвищенні дієвості системи менеджменту якості для підвищення задоволеності споживача за рахунок задоволення його вимог.

Для ефективного функціонування організація може визначити й організувати численні взаємозалежні види діяльності. Процесом можна вважати

діяльність, що використовує ресурси і управляється для забезпечення можливості перетворення входів у виходи. Часто вихід одного процесу безпосередньо служить входом іншого.

Під "підходом з позицій процесу" розуміється застосування організацією системи процесів, разом з визначенням процесів, їхніми взаємодіями і управлінням ними .

Перевагою підходу з позицій процесу є передбачений ним безперервний контроль сумісності окремих процесів у рамках системи, їх з'єднання та взаємодії.

При його застосуванні до системи менеджменту якості такий підхід підкреслює важливість:

а) розуміння і виконання вимог,

б) потреби розгляду процесів з точки зору додавання цінності,

в) одержання результатів виконання і ефективності процесу,

г) постійного вдосконалення процесів на підставі об'єктивних вимірювань.

До всіх процесів застосовна і методологія, відома як "Планування – Здійснення – Перевірка – Дія" (PDCA – за першими буквами англійської назви "Plan-Do-Check-Act "). PDCA, яку називають ще циклом Шухарта-Демінга, можна стисло описати в такий спосіб:

планування: установлення завдань і процесів;

здійснення: впровадження процесів;

перевірка: відстеження та вимірювання процесів для визначення їхньої відповідності політиці, завданням і вимогам до продукту і повідомлення результатів;

дія: вживання заходів для постійного вдосконалення показників процесу.

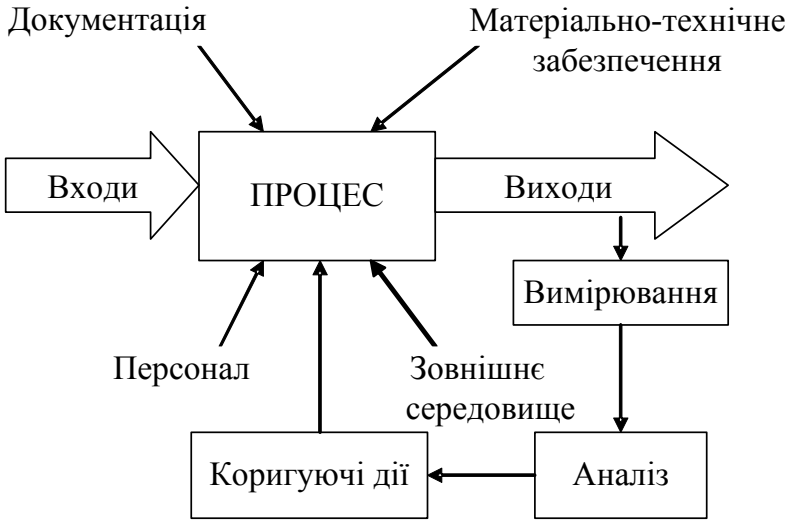

Рисунок 3.1 – Методологія PDCA стосовно до процесу

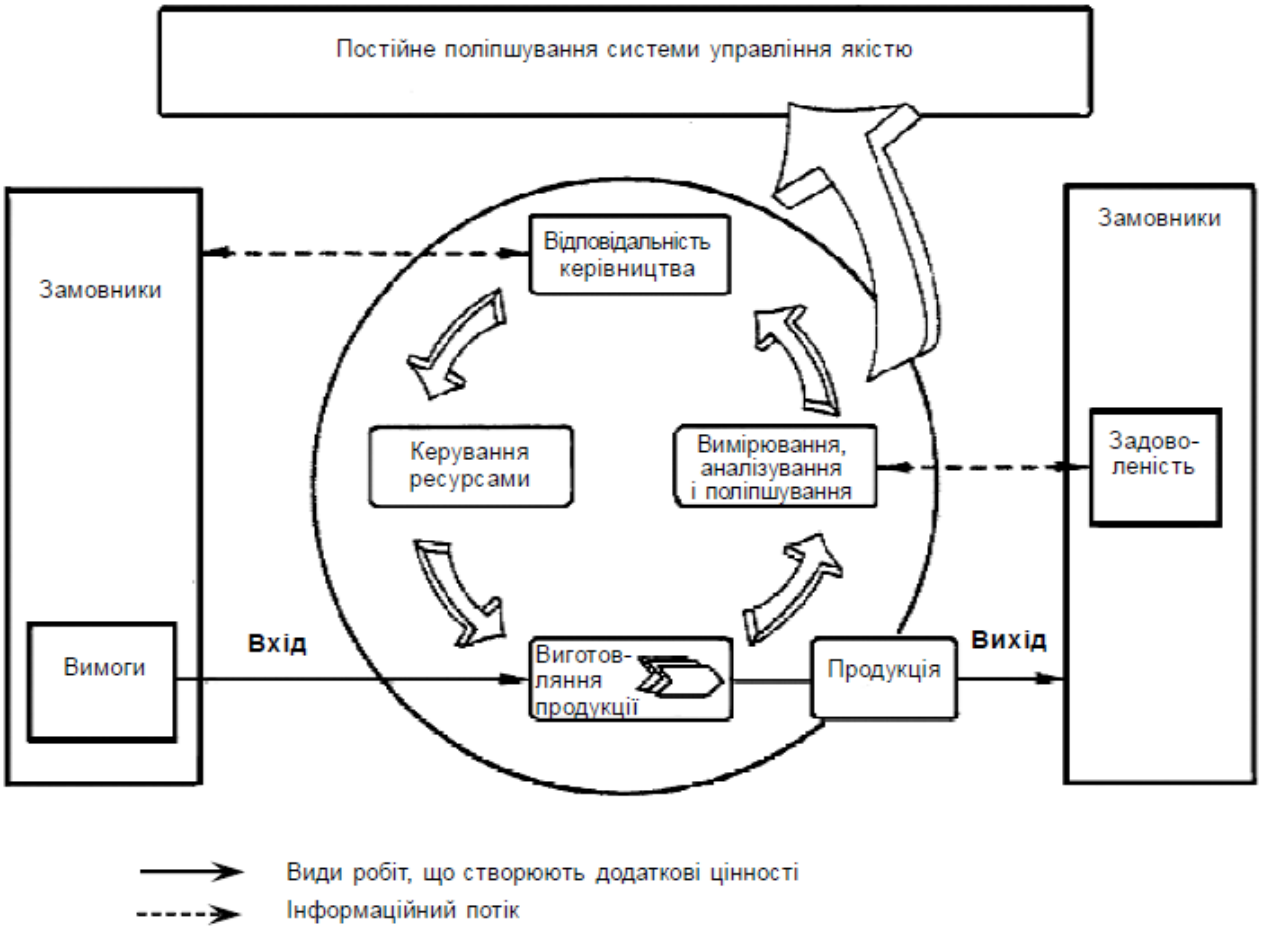

Рисунок 3.2 – Модель системи управління якістю,
в основу якої покладено процеси

Організація повинна створити, задокументувати, увести систему якості, забезпечити її функціонування і постійно підвищувати її дієвість відповідно до вимог цього міжнародного стандарту.

Організація повинна:

а) установити необхідні для системи менеджменту якості процеси і їхнє застосування у всіх своїх підрозділах;

б) визначити послідовність і взаємодію цих процесів;

в) визначити критерії та методи, необхідні для результативності як виконання цих процесів, так і управління ними;

г) забезпечити наявність ресурсів і інформації, необхідних для забезпечення виконання і контролю цих процесів;

д) здійснювати контроль, вимірювання і аналіз цих процесів;

е) виконувати дії, необхідні для одержання запланованих результатів і постійного поліпшення цих процесів.

3.4 Документальна структура системи менеджменту якості

Система менеджменту якості є системою організаційного управління підприємством, основаною на документуванні. Вся сукупність документації утворює ієрархічну пірамідальну структуру (рис. 3.5). Документи нижче розташованого рівня є деталізацією документів вище розташованого рівня.

Рисунок 3.5 – Пірамідальна модель документальної структури системи якості

3.5 Контрольні запитання і завдання

1. Дайте визначення якості, наведіть приклади.
2. Поясніть, що таке управління якістю та забезпечення якості.
3. Що таке система якості, які її складові?
4. Дайте визначення процесу, наведіть приклади.
5. Для чого здійснюються попереджувальні та коригувальні дії?
6. Назвіть основні принципи управління якістю, поясніть їх суть.
7. Які основні фази має цикл Шухарта-Демінга PDCA?
8. Які основні етапи впровадження системи менеджменту якості на підприємстві?
9. Зобразіть ієрархічну модель документальної структури системи якості.

Лекція 4
МОДЕЛЬ УПРАВЛІННЯ ЯКІСТЮ ПРОГРАМНОЇ ПРОДУКЦІЇ

4.1 Специфічні особливості програмної продукції

Програмній продукції властиві деякі характерні риси, які необхідно враховувати на всіх етапах життєвого циклу програмних засобів, а також при управлінні якістю:

1) *високі наукоємність і інтелектуальність змісту*.

Програмна продукція створюється на основі інтенсивного використання наукових знань і, у свою чергу, сама сприяє поширенню й використанню знань шляхом створення банків даних, інформаційно-довідкових, експертних систем і т. ін. До розробки ПЗ, крім фахівців в області програмування, доводиться залучати фахівців високої кваліфікації у всіляких програмних областях (хімія, фізика, системи керування, технологічні процеси і т. ін.). Висока наукоємність викликає необхідність підвищених витрат на науково-дослідні і дослідно-конструкторські роботи в процесі створення програм. Розглянута особливість утрудняє використання інженерних методів у проектуванні та управлінні якістю ПЗ;

2) *програмний засіб при використанні не витрачається і не витрачає свій ресурс*.

Відомо, що промислова продукція ділиться на два основних класи. До першого належить продукція, що витрачається при використанні (паливо, сировина і т. ін.); до другого – продукція, що витрачає при використанні свій ресурс (машини, прилади, верстати і т. ін.). Програмна продукція не може бути віднесена за даними ознаками до жодного із цих класів промислової продукції. Дана особливість істотно позначається на методах оцінки надійності ПЗ. Особливо це стосується таких показників, як довговічність, ремонтопридатність, безвідмовність;

3) *простота виготовлення*.

Виготовлення програмної продукції пов'язане з тиражуванням, зняттям копій, при цьому яких-небудь якісних змін у ПЗ не відбувається. Дана особливість істотно впливає на організацію контролю якості, основна вага якого приходиться не на процес виготовлення виробу, а на процеси розробки і випробування дослідного зразка;

4) *простота внесення змін*.

Модернізація програмного засобу вимагає знання структури змінюваного виробу, глибокого аналізу наслідків від внесення змін. Але сам процес простий і не вимагає додаткових матеріалів, пристроїв і пристосувань. Потрібна лише гарна програма-редактор.

Дана особливість при вмілому її використанні є істотною перевагою програмної продукції, життєво необхідною в динамічних сферах застосування.

Але ця ж особливість легко перетворюється в недолік, якщо потік змін стає слабко керованим і незбалансованим. Процес модернізації ПЗ повинен бути об'єктом ретельного контролю і планування;

5) *абстрагована матеріальність програмної продукції*.

За своїм формальним змістом будь-який ПЗ є інформаційним об'єктом. Але інформація, що міститься в ПЗ, сугубо специфічна. У загальному випадку інформація відбиває об'єкт пізнання. Інформація ж, що міститься в командній (операторній) частині програми, сама отримана на основі вивчення певного об'єкта пізнання і містить приписання про послідовність перетворення (обробки) даних, що відбивають стан об'єкта пізнання, у необхідний результат.

Програмний засіб, будучи частиною обчислювальної системи, управляє процесом перетворення даних. Він стає знаряддям (точніше частиною знаряддя) виробництва. Природно, виникає питання, чи можна вважати ПЗ матеріальним об'єктом. Загальноприйнятої думки з цього питання поки немає. Дана обставина приводить до необхідності використання суб'єктивних оцінок властивостей ПЗ і обмежує можливості застосування традиційних реєстраційних і вимірювальних методів визначення значень показників якості ПЗ;

6) *формування якості ПЗ у процесі розробки, систематична зміна ПЗ у результаті його модернізації (супроводу)*.

Дана обставина спричиняє необхідність здійснення процедур контролю якості програмних засобів на всіх етапах життєвого циклу, починаючи із самих ранніх стадій проектування і закінчуючи фазою супроводу, яку дуже часто називають триваючою розробкою;

7) *для переважної більшості ПЗ відсутні стандартизовані кількісні показники якості*.

Основні труднощі задачі оцінювання рівня якості ПЗ полягають в неможливості визначення єдиного (узагальненого) показника якості і наявності декількох більш-менш рівнозначних показників, кожний з яких може стати домінуючим залежно від умов і середовища функціонування ПЗ.

4.2 Концептуальна модель управління якістю програмної продукції

Саме управління якістю продукції відрізняється від управління процесом створення продукції тим, що тут об’єктом управління є не організація виробництва, а регулювання властивостей виробленої продукції. Властивості ці формуються на стадіях життєвого циклу під впливом різних умов і факторів. Керування якістю є процесом впливу на ті умови, фактори і соціально-економічні відносини, які впливають на формування і зміну споживчих властивостей продукції. Для здійснення цього процесу створюється система управління – сукупність певним чином взаємодіючих органів, засобів і методів управління.

Керування якістю програмної продукції має організаційно-методичну і соціально-економічну сторони.

Організаційно-методична сторона управління якістю програмної продукції виражається в розробці і застосуванні передових технологій програмування, закріпленні науково-технічних досягнень у відповідних стандартах і методичних документах, оснащенні розроблювачів передовою технікою і т.п.

Соціально-економічна сторона управління якістю програмної продукції виражається в створенні такої системи соціально-економічних відносин між всіма учасниками розробки, яка буде забезпечувати створення і виробництво програмної продукції необхідної і гарантованої якості.

Для більш повного розуміння сутності управління якістю програмної продукції в процесі її розробки розглянемо концептуальну модель управління. Усяке управління припускає наявність наступних основних елементів: об'єкта управління, мети управління, органа управління, давачів інформації про стан керованого процесу.

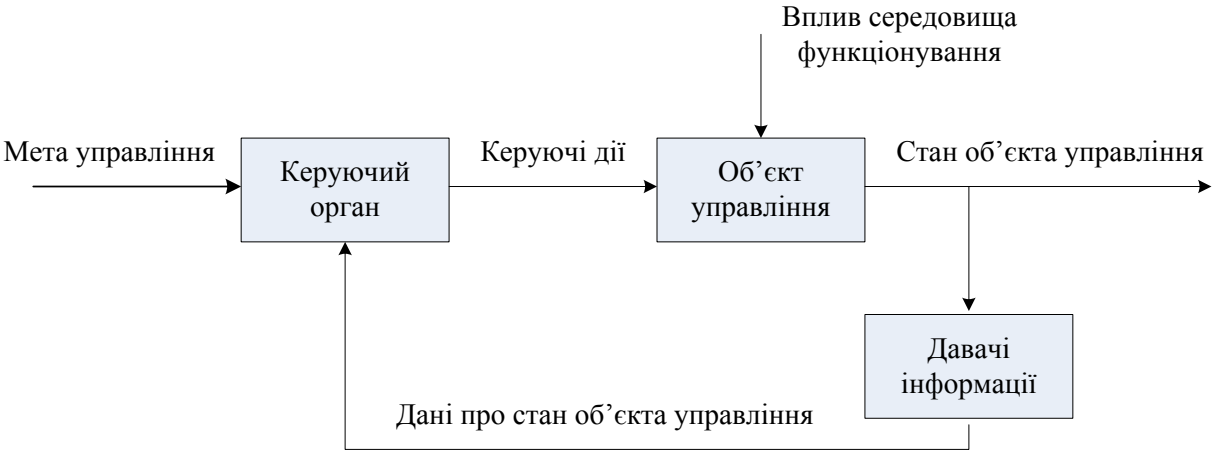

Рисунок 4.1 – Загальна схема управління якістю

Процес управління в загальному випадку зводиться до наступної схеми (рис. 4.1). Керуючий орган здійснює на об'єкт управління вплив, що відповідає меті управління. Об'єкт управління, піддавшись впливу, змінює свій стан або просторове положення. Давачі інформації визначають новий стан об'єкта управління і відповідну інформацію передають керуючому органу. Керуючий орган аналізує міру відповідності нового стану об'єкта управління меті управління і приймає відповідне рішення: якщо мета управління не досягнута, то виробляється і передається на об'єкт управління нова керуюча (коригувальна) дія; якщо ж мета управління досягнута, то керуючі впливи припиняються.

Об'єктом управління є якість предмета праці. Предметом же праці залежно від етапу розробки послідовно є технічне завдання, технічний проект, робочий проект, дослідний зразок ПЗ. Якість ПЗ в основному формується на цих етапах, тому управління якістю повинне починатися із самого початку процесу розробки ПЗ і вестися безупинно в плині всього цього процесу.

У загальному випадку керуючі впливи можуть здійснюватися не тільки безпосередньо на предмет праці, але й на засоби праці і технологічні процеси, якщо вони не сприяють досягненню мети управління, а також на фактори, що впливають на якість ПЗ.

Засобами праці при програмуванні є транслятори, завантажники, збирачі програм, автоматизовані засоби налагодження, генератори тестових даних і т.п.

Метою управління є забезпечення необхідного рівня якості ПЗ, що гарантує очікуваний соціально-економічний ефект від використання даного ПЗ за своїм призначенням.

Роль керуючого органа при управлінні якістю ПЗ виконують керівники розробки, експлуатації і супроводу ПЗ, безпосередні розроблювачі або фахівці, що здійснюють супровід ПЗ, разом із засобами управління.

Давачами інформації про стан керованого процесу і якості ПЗ залежно від стадії життєвого циклу ПЗ є або самі розроблювачі (на стадії проектування і налагодження), або експерти груп контролю якості (на стадії проектування), або випробувачі (на етапах випробувань), або користувачі (на стадії експлуатації).

Задача управління якістю ПЗ є різновидом оптимізаційних задач і має наступні складові частини:

1) визначення мети управління якістю;

2) знання критеріїв оцінки якості ПЗ;

3) знання поточного положення відносно мети;

4) знання мікроструктури ПЗ і всіх факторів, що впливають на якість ПЗ;

5) знання обмежуючих умов по строках виконання і ресурсам, які є в розпорядженні;

6) визначення найкращих способів досягнення мети (оптимізації якості), узгоджених з даними 3), 4), 5).

4.3 Контрольні запитання і завдання

1. Які специфічні особливості має програмна продукція?

2. У чому полягає організаційно-методична сторона управління якістю програмної продукції?

3. У чому полягає соціально-економічна сторона управління якістю програмної продукції?

4. Зобразіть модель управління якістю програмної продукції.

5. Які основні складові має задача управління якістю програмних засобів?

Лекція 5
ОЦІНЮВАННЯ РІВНЯ ЯКОСТІ ПРОГРАМНИХ ЗАСОБІВ

5.1 Фактори, що впливають на якість програмної продукції

Якість програмної продукції в основному формується в процесі розробки. Специфіка програмної продукції проявляється в тому, що її модернізація з метою поліпшення якості досить часто інтенсивно здійснюється і на стадії супроводу. Таким чином, проблема якості багатьох програмних продуктів вирішується (але не завжди успішно) фактично в плині всього життєвого циклу.

Якість ПЗ залежить від численних факторів. Розглянемо основні з них.

1) *відповідальність керівництва*.

У відповідності зі стандартом ISO 9001 відповідальність керівництва в області забезпечення якості визначається наявністю в системі якості організації наступних елементів:

– документально оформлені політика в області якості, цілі і зобов'язання;

– відповідальність, повноваження і взаємодія всього персоналу (керівного і виконавців), що впливає на якість;

– засоби перевірки і спеціально навчений персонал;

– представник керівництва, що несе персональну відповідальність за виконання вимог до якості продукції;

– періодичний аналіз ефективності діючої в організації системи якості;

2) *якість нормативної документації на розроблювану програмну продукцію в частині оптимальності і повноти встановлених у ній вимог*.

Складання ТЗ на розробку ПЗ і визначення в ньому основного переліку вимог є першим етапом проектування ПЗ. ТЗ повинні складатися як на ПЗ, що є самостійними об'єктами постачання (програмні комплекси), так і на програми-компоненти. При розробці складних ПЗ, що не мають аналогів, безпосередньому складанню ТЗ звичайно передує науково-дослідна робота, метою якої є визначення призначення ПЗ, областей і особливостей його застосування, а також аналіз вимог потенційних користувачів.

При написанні ТЗ зазвичай виникають серйозні труднощі, головні з яких обумовлені відсутністю загальноприйнятих показників якості ПЗ і методів визначення їхніх значень. У ТЗ повинен бути визначений мінімальний обсяг вимог, що охоплює основні споживчі властивості ПЗ, причому трактування вимог повинне бути однозначним. Спроба охопити всі деталі проекту заплутує ТЗ, робить його неконкретним і сковує ініціативу розроблювачів;

3) *ефективність технологій програмування. Технологічна підготовка розробки ПЗ*.

Процес створення ПЗ є дорогим і трудомістким. Технологія програмування, управління процесом створення ПЗ повинні забезпечувати максимальний корисний ефект при певних витратах. Природно, що такий ефект може бути досягнуто тільки при використанні для розробки ПЗ найбільш

прогресивних методів і засобів. Технологічна підготовка розробки ПЗ повинна бути повною і своєчасною;

4) *регулярність і ефективність контролю за якістю розробки*.

Процес створення ПЗ повинен перебувати під постійним і ретельним контролем. Завчасно варто встановити технологію виявлення і виправлення помилок, а також часові і матеріальні ресурси на реалізацію цієї технології. Практика показує, що для випуску високоякісної продукції необхідно заздалегідь планувати до 60 % трудозатрат для забезпечення належного контролю, налагодження і випробування програм, завчасно встановлювати процедури контролю, створювати програмно-технічні засоби налагодження, тестування й випробування;

5) *кваліфікація розроблювачів*.

Якість створюваних ПЗ визначають такі властивості розроблювачів:

– рівень знань (знайомство із проблемою, мовами програмування і ЕОМ, технікою проектування, принципами обробки даних);

– наявність практичних навичок (досвіду створення аналогічних програм і програмних систем);

– рівень здібностей (творчі здібності, гострота і глибина мислення);

– рівень ініціативності (розуміння розв'язуваних задач і їхніх взаємозв'язків, ефективність використання робочого часу, прагнення довести кожне завдання до повного завершення, підтримка робочих контактів зі співвиконавцями проекту);

– рівень відповідальності (зосередженість на виконуваній роботі, постійне прагнення працювати добре, здорове самолюбство);

6) *склад і якість програмно-технічних (інструментальних) засобів, що використовуються при розробці*.

Розробка складних ПЗ пов'язана з необхідністю використання різних технічних засобів обчислювальної техніки і системного програмного забезпечення. Ці засоби служать своєрідним технологічним обладнанням і оснащенням для програмістів. Природно, що якість створюваних ПЗ залежить від надійності цього обладнання і стабільності технологічних операцій. Важливо також вчасно і повною мірою задовольняти потреби розроблювачів у цих засобах;

7) *стимулювання створення високоякісної програмної продукції*.

Незважаючи на значні успіхи в області індустріалізації програмування, характер праці програмістів є ще індивідуальним і значною мірою залежить від особистих здібностей виконавців. Продуктивність і якість праці програмістів, що працюють у тих самих умовах, можуть відрізнятися в декілька (іноді в десятки) разів. Тому при створенні програм повинна діяти ефективна система стимулювання створення високоякісних ПЗ, що передбачає оплату праці програмістів за кількістю і якістю результатів.

8) *формування і дотримання єдиних принципів розробки програмної продукції.*

На основі результатів вивчення і аналізу факторів, що впливають на якість програмної продукції, з урахуванням специфіки і досвіду створення цієї продукції в кожній організації, що розробляє програмну продукцію, повинні бути сформульовані основні принципи розробки ПЗ;

9) *маркетинг.*

Якість конкретного ПЗ залежить від ефективності системи заходів щодо вивчення ринку і споживчих властивостей цього ПЗ (ефективності маркетингу) протягом усього життєвого циклу в різних умовах застосування. Підрозділи, що здійснюють маркетинг, повинні працювати в тісній взаємодії з підрозділами служби супроводу ПЗ, тому що в службу супроводу зазвичай надходить інформація не тільки про виявлені в ПЗ помилки у процесі експлуатації, але й пропозиції про шляхи вдосконалення ПЗ;

10) *наочність результатів контролю якості*.

Для кожного ПЗ на різних етапах розробки повинні бути встановлені досить прості і ясні критерії (ознаки) високої якості і браку проектування. Інформація про хід розробки ПЗ і результати його контролю повинна бути наочною і загальнодоступною. Розроблювачі ПЗ завжди повинні бути готові не тільки на словах завірити високу якість ПЗ, але й переконливо це продемонструвати;

11) *наявність всеосяжного плану забезпечення якості розроблюваної програмної продукції.*

План містить у собі комплекс заходів щодо забезпечення і підтримки необхідного рівня якості ПЗ, розподілених по виконавцях, часу і матеріальним ресурсам. Він базується на специфікації вимог до програмної продукції, на знанні факторів якості, специфіки розроблюваних ПЗ і ресурсів, які є в наявності. План розробляють одночасно із ТЗ як додаток до нього.

5.2 Базова номенклатура показників якості програмних засобів

Показник якості ПЗ (software quality metric) – кількісна міра, що використовується для визначення ступеня, у якому ПЗ має дану якість.

Одиничний показник якості ПЗ – показник якості ПЗ, що характеризує одну з його властивостей.

Комплексний показник якості ПЗ – показник якості ПЗ, що характеризує декілька його властивостей.

Кожен із групових показників якості складається з декількох показників, що характеризують одну або кілька властивостей ПЗ. Безпосередній оцінці підлягають значення одиничних показників якості, значення групових показників якості визначаються розрахунковим шляхом.

Номенклатура властивостей і показників якості ПЗ є впорядкованою багаторівневою ієрархічною структурою. Перший (верхній) рівень становлять групи показників якості, нижній рівень – одиничні показники якості, проміжні рівні – підгрупи показників якості. Дерево властивостей і показників якості в загальному випадку є незбалансованим за висотою.

ДСТУ 2850-94 установлює два верхніх рівні комплексних властивостей ПЗ: групи властивостей і підгрупи властивостей (табл. 5.1).

Склад інших (нижніх) рівнів властивостей перебуває в сильній залежності від специфіки видів ПЗ і встановлюється для кожного індивідуального ПЗ або підкласу (групи, підгрупи) однорідних ПЗ.

Базова номенклатура показників якості (властивостей) програмних засобів, наведена в ДСТУ 2850, повністю відповідає вимогам міжнародного стандарту ISO 9126 Software Engineering – Software Product Quality.

Таблиця 5.1 –Номенклатура властивостей, що визначають якість ПЗ

Найменування властивостей, що характеризуються	Визначення властивостей, що характеризуються
1. ФУНКЦІОНАЛЬНІСТЬ (functionality)	Група властивостей ПЗ, яка обумовлює їх здатність виконувати в заданому середовищі визначений перелік функцій, які задовольняють установленим або передбаченим потребам згідно з призначенням ПЗ.
1.1. Функціональна повнота (suitability)	Підгрупа властивостей функціональності ПЗ, що характеризується наявністю і мірою достатності основних функцій для вирішення завдань згідно з призначенням ПЗ.
1.2. Здатність до взаємодії (interoperability)	Підгрупа властивостей функціональності ПЗ, що характеризується можливістю його взаємодії при функціонуванні з заданою номенклатурою інших ПЗ або систем.
1.3. Захищеність (security)	Підгрупа властивостей функціональності ПЗ, що характеризує його спроможність запобігати несанкціонованому доступу (як випадковому, так і навмисному) до програм і даних.
1.4. Масовість (rangability)	Підгрупа властивостей функціональності ПЗ, що характеризується діапазоном і граничними значеннями вхідних даних, які можуть бути перетворені в правильний результат.
1.5. Узгодженість (compliance)	Підгрупа властивостей функціональності ПЗ, яка характеризується ступенем дотримання установлених стандартів, угод, правил і рекомендацій.
1.6. Продуктивність (throughput)	Підгрупа властивостей функціональності ПЗ, що характеризується кількістю виконуваних функцій обробки даних (в тому числі і однотипних), в одиницю часу функціонування.

Продовження табл. 5.1

Найменування властивостей, що характеризуються	Визначення властивостей, що характеризуються
2. НАДІЙНІСТЬ ФУНКЦІОНУВАННЯ (reliability)	Група властивостей ПЗ, що обумовлює спроможність ПЗ зберігати працездатність і перетворювати вихідні дані в шуканий результат у заданих умовах за установлений період часу.
2.1. Безвідмовність (maturity)	Підгрупа властивостей надійності функціонування ПЗ, що характеризується частотою відмов через помилки та недосконаліть ПЗ.
2.2. Стійкість до аномалій (fault tolerance)	Підгрупа властивостей надійності функціонування ПЗ, що обумовлює його здатність виконувати свої функції в аномальних умовах.
2.3. Відновлюваність (recoverability)	Підгрупа властивостей надійності функціонування ПЗ, що обумовлює можливість відновлювати рівень якості функціональності і дані після відмов.
2.4. Правильність, точність (accuracy)	Підгрупа властивостей надійності функціонування ПЗ, що характеризується подібністю результатів обробки даних до істинних, специфікованих або теоретично вірних значень.
2.5. Реактивність (responsibility)	Підгрупа властивостей надійності функціонування ПЗ, що характеризується здатністю своєчасно перетворювати вхідні дані (запити) в шуканий результат.
3. ЗРУЧНІСТЬ ВИКОРИСТАННЯ (usability)	Група властивостей ПЗ, що забезпечують установлене або передбачене коло користувачів необхідними умовами для використання ПЗ.
3.1. Освоюваність (trainability)	Підгрупа властивостей зручності використання ПЗ, що характеризується зусиллями, необхідними для освоєння користувачами умов, процедур і правил застосування ПЗ.
3.2. Простота підготовки до роботи (preparability)	Підгрупа властивостей зручності використання ПЗ, що обумовлює легкість підготовки вхідних даних і запуску в роботу ПЗ.
3.3 Зручність інтерфейсу для користувача (operability)	Підгрупа властивостей зручності використання ПЗ, що обумовлює необхідні умови інтерфейсу користувача в процесі функціонування ПЗ.
3.4. Аналізованість результатів (result analyzability)	Підгрупа властивостей зручності використання ПЗ, що обумовлює легкість розуміння результатів функціонування (кінцевих вихідних даних).
3.5. Документованість (documentability)	Підгрупа властивостей зручності використання ПЗ, що характеризує ступінь підтримки документацією ПС процесів освоювання ПЗ, підготовки їх до роботи, взаємодії користувача з процесом обробки даних і аналізу результатів обробки.

Продовження табл. 5.1

Найменування властивостей, що характеризуються	Визначення властивостей, що характеризуються
4. РАЦІОНАЛЬНІСТЬ (efficiency)	Група властивостей ПЗ, що характеризується ступенем відповідності використаних ресурсів середовища функціонування рівню якості функціонування ПЗ за заданих умов застосування.
4.1. Зайнятість ресурсів (resource occupancy)	Підгрупа властивостей раціональності ПЗ, що характеризується обсягом ресурсів середовища функціонування, використовуваних у процесі експлуатації ПЗ.
4.2. Використовуваність ресурсів (resource utilization)	Підгрупа властивостей раціональності ПЗ, що характеризується ступенем зайнятості ресурсів обчислювальної системи, що використовуються під час виконання ПЗ за певний час.
5. СУПРОВОДЖУВАНІСТЬ (maintainability)	Група властивостей ПЗ, що характеризується зусиллями, потрібними для виконання конкретних модифікацій.
5.1. Аналізованість (analyzability)	Підгрупа властивостей супроводжуваності ПЗ, що обумовлює їх пристосованість до діагностики недоліків або відмов і виявленню частин, які мають бути змінені, і прогнозу наслідків цих змін.
5.2. Коректованість (correctability)	Підгрупа властивостей супроводжуваності ПЗ, що характеризується зусиллями, необхідними для усунення недоліків, які виявлені в ПЗ.
5.3. Модернізованість (modernizability)	Підгрупа властивостей супроводжуваності ПЗ, що обумовлює пристосованість ПЗ до модернізації.
5.4. Тестованість (testability)	Підгрупа властивостей супроводжуваності ПЗ, що характеризується зусиллями, необхідними для атестації ПЗ, який модифіковано.
6. ПЕРЕНОСИМІСТЬ (portability)	Група властивостей ПЗ, що обумовлює його пристосованість для переносу із одного середовища функціонування в інші.
6.1. Адаптованість (adaptability)	Підгрупа властивостей мобільності ПЗ, що обумовлює його спроможність адаптації до функціонування в середовищах, відмінних від тих, для яких він розроблявся.
6.2. Налагоджуваність (installability)	Підгрупа властивостей мобільності ПЗ, що характеризується зусиллями, необхідними для переносу ПЗ в одне із раніше передбачених середовищ функціонування.

5.3 Основні етапи процедури оцінювання рівня якості програмних засобів

Під оцінюванням якості конкретного ПЗ розуміють дії, спрямовані на визначення ступені задоволення ПЗ потреб відповідно до призначення.

Метою оцінювання якості може бути:

– ухвалення рішення про ступінь відповідності рівня якості ПЗ, досягнутого при розробці, необхідному (заданому) рівню;

– опис ознак і властивостей ПЗ, що виготовляються або поставляються (наприклад, у документації на ПЗ, каталогах);

– оцінка конкурентоспроможності ПЗ;

– атестація і сертифікація ПЗ.

Процес оцінювання якості ПЗ у загальному випадку складається з послідовності етапів, зображеної на рис. 5.1.

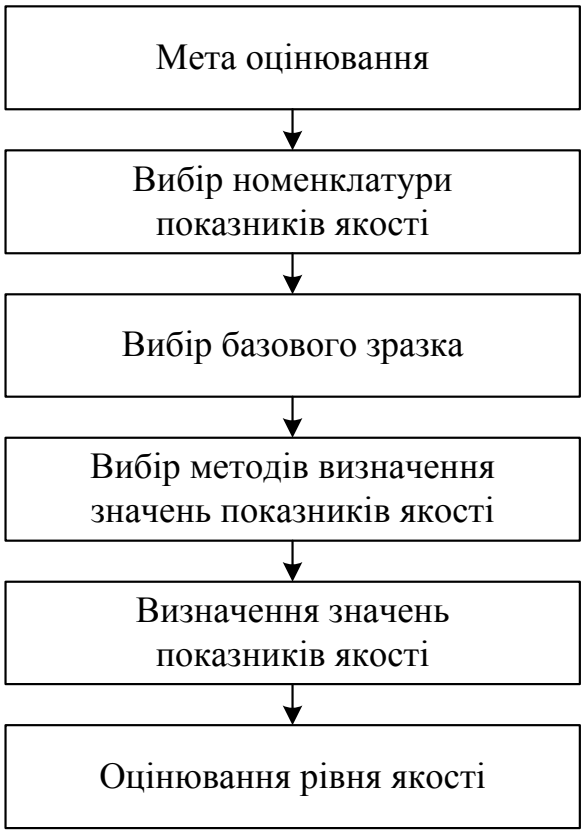

Рисунок 5.1 – Послідовність операцій оцінювання рівня якості ПЗ

Формування номенклатури показників якості (НПЯ) для індивідуальних ПЗ полягає у встановленні повного переліку показників, що характеризують істотні властивості даного ПЗ. Формування НПЯ ведеться згори вниз аж до встановлення всіх одиничних показників.

При формуванні НПЯ необхідно керуватися такими основними принципами:

– виділення групових властивостей повинне здійснюватися за чітко визначеною ознакою (час прояву, симптом і ін.);

– властивості, що входять у групу (підгрупу), повинні взаємно доповнювати одна одну, бути незалежними або слабо корельованими;

– будь-яка вихідна номенклатура показників повинна бути відкритою, тобто допускати включення або виключення з неї окремих елементів;

– властивості ПЗ, які є компонентами автоматизованих систем і систем обробки даних, повинні бути сумісними з відповідними властивостями цих систем;

– для кожної з виділених властивостей повинна існувати можливість вираження їх у шкалах «краще-гірше», якщо «більше-менше»;

– у групу (підгрупу) повинні включатися властивості, достатні для визначення відповідної групової властивості;

– сукупність властивостей, що характеризують якість оцінюваного ПЗ, повинна бути впорядкованою за певним правилом у вигляді багаторівневої ієрархічної структури – дерева властивостей;

– дерево властивостей повинне відбивати всі основні особливості використання і функціонування ПЗ;

– кожний із показників якості повинен бути корельованим з певною властивістю ПЗ і з якістю ПЗ у цілому;

– показники якості, що характеризують властивості ПЗ, повинні сприяти забезпеченню відповідності якості ПЗ вимогам користувачів і враховувати сучасні досягнення науки і техніки;

– показники якості повинні стимулювати використання найбільш прогресивних і ефективних засобів і методів розробки, виготовлення і функціонування ПЗ;

– для кожного з обраних показників слід встановити коефіцієнт (параметр) вагомості. При обмежених ресурсах основна увага приділяється показникам (властивостям), що мають більш високий параметр вагомості;

– одночасно з вибором показників якості необхідно розглядати питання про методи визначення їхніх значень. Використання показників, визначення достовірних значень яких малоймовірно, на практиці може виявитися марним.

Обрана (прийнята) НПЯ повинна задовольняти наступним критеріям: повнота, кореляція з якістю, однозначність визначень, можливість верифікації, ненадлишковість, незалежність показників, чіткість ознак комплектування, прогресивність, можливість трасування.

Під рівнем якості ПЗ розуміють відносну характеристику якості ПЗ, засновану на порівнянні фактичних значень показників якості оцінюваного ПЗ із базовими (еталонними) значеннями відповідних показників якості.

Базовим зразком називається реально досяжна сукупність значень показників якості продукції, прийнята для порівняння. Показники якості базового зразка називаються базовими значеннями показників. Сукупність базових значень показників повинна характеризувати оптимальний рівень якості даного виду продукції за деякий заданий період часу.

Базові значення показників якості ПЗ повинні відповідати:

1) значенням показників якості кращих вітчизняних і закордонних ПЗ із числа аналогів;

2) прогнозованим значенням показників якості кращих вітчизняних і закордонних зразків-аналогів до моменту завершення розробки ПЗ;

3) нормативним значенням показників якості за окремими видами ПЗ (групами однорідних ПЗ), погодженим у встановленому порядку;

4) значенням показників якості, установленим у технічних завданнях на розробку оцінюваних ПЗ.

Від вибору базового зразка і базових значень показників якості в значній мірі залежить вірогідність результатів оцінювання рівня якості і правильність прийнятих рішень. Вибір базового зразка і базових значень показників якості варто науково обґрунтовувати, а посадові особи, що приймають відповідні рішення, повинні нести персональну відповідальність за правильність прийнятих рішень.

Вибір базового зразка і базових значень показників якості для програмної продукції пов’язаний з великими труднощами, які обумовлені хоча й тимчасовими, але об’єктивними причинами:

– відсутністю загальноприйнятих показників якості, придатних для порівняльної оцінки ПЗ;

– відсутністю даних про значення показників якості більшості закордонних і вітчизняних ПЗ;

– низьким рівнем уніфікації, обмеженістю інформації про властивості і характеристики ПЗ, що утрудняє вибір зразків-аналогів ПЗ;

– відсутністю єдиної системи класифікації, що включає всі ієрархічні рівні ПЗ (підкласи, групи, підгрупи, види, підвиди ПЗ);

– слабким розвитком методів визначення оптимальних значень показників якості програмної продукції.

5.4 Методи визначення значень показників якості

При визначенні значень показників якості використовуються такі методи: вимірювальний, реєстраційний, розрахунковий і експертний, а також їхні комбінації.

Вимірювальний метод заснований на одержанні інформації про властивості і характеристики ПЗ із використанням вимірювальних, технічних і програмних засобів. Наприклад, за допомогою цього методу визначається обсяг ПЗ, число рядків вихідного тексту програм і число рядків-коментарів, число операторів і операндів, число виконаних операторів, кількість розгалужень у програмі, число точок входу (виходу), час виконання гілки програми, показники реактивності.

Реєстраційний метод заснований на одержанні інформації під час випробувань або при безпосередньому використанні ПЗ за призначенням, коли реєструються і підраховуються характерні події, наприклад, моменти часу та число збоїв і відмов, моменти часу передачі керування іншим модулям, моменти часу початку і кінця роботи.

Розрахунковий метод заснований на використанні теоретичних і емпіричних залежностей на різних стадіях розробки, а також статистичних даних, що накопичуються при випробуваннях, експлуатації і супроводі ПЗ. При проектуванні ПЗ розрахунковим методом прогнозуються показники точності, стійкості, реактивності і ін. Розрахунковий метод використовується і для визначення фактичних значень цих показників за результатами випробування або експлуатації ПЗ.

Визначення значень показників якості ПЗ *експертним методом* здійснюється групою експертів-фахівців, компетентних у розв’язанні даної задачі. При цьому рішення базується на досвіді і інтуїції експертів, а не на безпосередніх результатах розрахунків або експериментів. Експертний метод застосовується у випадку, якщо задача не може бути розв’язана традиційними методами.

Для проведення експертизи встановлюються контрольовані ознаки, що включаються у вигляді питань в аналітичні карти опитування (анкети) експертів. Контрольовані ознаки повинні бути корельованими з одним або з декількома показниками якості ПЗ.

Експертні методи рекомендується застосовувати при визначенні показників аналізованості, документованості, узгодженості, завершеності, структурованості та ін.

5.5 Методи оцінювання рівня якості програмних засобів

Для оцінювання узагальненого (за НПЯ) рівня якості ПЗ на підготовчому етапі встановлюються: НПЯ, джерела і способи одержання інформації, методи визначення значень кожного із показників якості і шкали оцінок, базові

(еталонні) значення показників якості. Всі ці дані оформляються у вигляді документа “Умови оцінювання рівня якості ПЗ”.

При оцінюванні рівня якості ПЗ можуть застосовуватися диференціальний, комплексний або змішаний методи, які використовуються при оцінюванні рівня якості промислової продукції.

Диференціальним називається метод оцінювання рівня якості продукції, заснований на використанні одиничних показників її якості. При цьому визначають, чи досягнуто рівень базових значень показників якості у цілому, за якими показниками його досягнуто і за якими не досягнуто.

При диференціальному методі відносні значення окремих показників визначаються за формулами

$$Q_i = \frac{P_i}{P_{ib}}, \tag{5.1}$$

$$Q_i = \frac{P_{ib}}{P_i}, \quad i = 1,2,\ldots,N, \tag{5.2}$$

де P_i – значення i-го показника якості оцінюваного ПЗ;

P_{ib} – базове значення i-го показника;

N – кількість показників якості.

Формула (5.1) використовується у випадку дотримання умови «краще, якщо значення показника більше». У противному випадку використовується формула (5.2).

При використанні диференціального методу рівень якості вважається вище або таким, що дорівнює рівню базових значень, якщо всі значення відносних показників більше або дорівнюють одиниці. У противному випадку рівень якості оцінюваної продукції нижче рівня базових значень. Якщо частина значень $Q_i > 1$, а частина значень $Q_i < 1$, то слід застосовувати комплексний або змішаний методи оцінки.

Комплексний метод оцінки рівня якості продукції заснований на застосуванні одного узагальненого показника, що є функцією від декількох одиничних (групових) показників і коефіцієнтів їхньої вагомості.

Установлення коефіцієнтів вагомості показників якості зазвичай здійснюється експертним шляхом. При цьому широко використовуються такі шкали:

5 – надто важливо, щоб даний показник мав високе значення;

4 – важливо, щоб даний показник мав високе значення;

3 – добре було б мати високе значення даного показника;

1 – при низьких значеннях даного показника відчутних втрат немає.

У цьому випадку зведене значення коефіцієнта вагомості (параметр вагомості *M*) обчислюється за формулою

$$M_i = \frac{m_i}{\sum_{i=1}^{N} m_i}, \qquad (5.3)$$

де m_i – значення *i*-го коефіцієнта вагомості в згаданій або іншій шкалах.

Якщо $\sum_{i=1}^{N} m_i = 1$, то коефіцієнти вагомості називають параметрами вагомості.

Узагальнений показник якості може бути виражений середнім зваженим (арифметичним або геометричним) показників якості. Середній зважений арифметичний показник обчислюється за формулою

$$U = \sum_{i=1}^{N} Q_i M_i, \qquad (5.4)$$

де Q_i – відносний *i*-й показник якості, що обчислюється за формулами (5.1), (5.2);

M_i – параметр вагомості *i*-го показника, що входить в узагальнений показник, $i=1,2,\ldots,N$;

N – число показників, що складають середній зважений показник.

Якщо до моменту випробування і оцінювання рівня якості ПЗ значення параметрів вагомості використовуваних показників якості не були встановлені, то ці значення встановлюються фахівцями, які здійснюють оцінювання якості ПЗ. При повторних випробуваннях, як правило, повинні використовуватися ті самі раніше встановлені показники. Обґрунтовані зміни параметрів вагомості узгоджуються у встановленому порядку.

При багаторівневій ієрархічній структурі НПЯ розрахунок рівня якості ПЗ здійснюється знизу (від одиничних показників) вгору (до одержання узагальненого показника якості).

Змішаний метод оцінювання рівня якості заснований на спільному застосуванні одиничних і комплексних (групових) показників. При використанні змішаного методу частина одиничних показників поєднується в групи. Після такого об’єднання обчислюються відносні значення групових показників і деяких одиничних показників за формулами (5.1), (5.2) і (5.3).

Порівняння рівня якості оцінюваного ПЗ із базовими показниками якості здійснюється так, як і в диференціальному методі.

5.6 Контрольні запитання і завдання

1. Які фактори вливають на якість програмної продукції?
2. Що таке показник якості програмного засобу? Які бувають показники якості?
3. Назвіть основні групи та підгрупи властивостей програмних засобів.
4. Які основні етапи процедури оцінювання рівня якості програмних засобів?
5. Які основні принципи формування номенклатури показників якості програмних засобів?
6. Які труднощі пов’язані з вибором базових значень показників якості?
7. Які методи застосовуються для визначення значень показників якості програмних засобів?
8. У чому полягає суть диференціального метода оцінювання рівня якості програмних засобів?
9. У чому полягає суть комплексного метода оцінювання рівня якості програмних засобів? Як визначаються коефіцієнти вагомості показників якості?

Лекція 6
ОСНОВНІ ПРИНЦИПИ ТЕСТУВАННЯ

6.1 Навіщо необхідно тестувати програми

6.1.1 Сфера застосування програмних систем

Програмні системи охоплюють усе більше широкі сфери життя, від бізнес-додатків (наприклад, банківська справа) до споживчих товарів (наприклад, автомобілі). Більшість людей мало досвід використання програмного забезпечення, що не працювало як очікувалося. Програмне забезпечення, що не працює правильно, може спричинити багато проблем, включаючи втрату грошей, часу або ділової репутації, і може навіть завдати шкоди здоров'ю або смерть.

6.1.2 Причини помилок у програмному забезпеченні

Людина може припуститися помилки, що приводить до помилки (defect, fault, bug) у коді, у програмному засобі або системі, або в документі. Якщо в коді присутня помилка, система буде не в змозі робити те, що вона повинна робити (або буде робити те, що не повинна), викликаючи відмову. Помилки в програмних засобах, системах або документах можуть привести до відмов, але не завжди.

Помилки виникають через те, що людям властиво робити помилки, а також через вплив різних факторів: недолік часу, складність коду, складність інфраструктури, технології що змінюються, багато системних взаємодій.

Відмови можуть бути викликані умовами навколишнього середовища: радіація, електромагнітні поля і забруднення можуть викликати помилки в вбудованому програмному забезпеченні або впливати на виконання програмного забезпечення, викликаючи зміни в апаратурі.

6.1.3 Роль тестування в розробці, експлуатації і супроводі ПЗ

Ретельне тестування систем і документації може допомогти знизити ризик виникнення проблем і сприяє підвищенню якості програмної системи, якщо знайдені помилки виправлені перш ніж система випущена для використання.

Тестування програмного забезпечення може також вимагати виконання договірних або юридичних вимог, або промислово-визначених стандартів.

6.1.4 Тестування і якість

За допомогою тестування можна оцінити якість програмного забезпечення в термінах знайдених помилок, як для функціональних так і для нефункціональних програмних вимог і характеристик (надійність, зручність використання, раціональність, супроводжуваність і переносимість).

Тестування може дати впевненість у якості програмних засобів, якщо помилок виявлено мало або не виявлено зовсім. Прогін належним чином розробленого тесту знижує рівень ризику в системі. Коли в результаті тестування дійсно виявляються помилки і згодом вони виправляються, якість програмної системи зростає.

Необхідно засвоювати уроки з попередніх проектів. Розуміючи першопричини помилок, знайдених в інших проектах, процеси життєвого циклу можуть бути поліпшені, що у свою чергу повинно запобігти повторній появі подібних помилок і, як наслідок, поліпшити якість майбутніх систем. Це один з важливих аспектів забезпечення якості.

Тестування повинне бути інтегроване як одна з дій по забезпеченню якості (разом з використанням стандартів розробки, навчання і аналізом помилок).

6.1.5 Скільки потрібно тестувати програму?

Приймаючи рішення про те, коли програму можна вважати достатньо протестованою, необхідно враховувати рівень ризику, включаючи технічні і бізнес аспекти проекту, а також обмеження даного проекту за часом і бюджетом.

Тестування повинне забезпечити достатній обсяг інформації всім зацікавленим сторонам, щоб можна було приймати обґрунтовані рішення щодо версії програмного продукту або системи, що тестується, для наступного кроку розробки або передачі замовникові.

6.2 Що таке тестування?

Зазвичай тестування сприймають тільки як процес виконання тестів, тобто виконання програми. Однак це тільки частина тестування.

Тестування існує до і після виконання тесту: планування і контроль, вибір умов тестування, розробка test cases і перевірка результатів, оцінювання критеріїв завершення, складання звіту про процес тестування і стан системи при тестуванні, завершення. Тестування також включає розгляд документів (включаючи вихідний код) і статичний аналіз.

І динамічне і статичне тестування можуть використовуватися як засіб для того, щоб досягти подібних цілей і одержати інформацію, необхідну для поліпшення і програми, що тестується, і процесів розробки та тестування.

Тестування може мати на меті:

– виявлення помилок;

– одержання впевненості в рівні якості і забезпечення інформацією;

– запобігання помилкам.

Процес проектування тестів на ранніх етапах життєвого циклу може допомогти запобігти появі помилок у коді. Огляди документів також допомагають запобігати помилкам, що з’являються в коді.

Різні точки зору на тестування розглядають різні цілі. Наприклад, для тестування на етапах розробки (наприклад, тестування компонентів, інтеграційне і системне тестування) основна мета може полягати в тому, щоб викликати якнайбільше відмов, щоб дефекти в програмному забезпеченні були ідентифіковані і виправлені. Для приймального тестування основна мета може полягати в тому, щоб підтвердити, що система працює як очікується, одержати впевненість у тім, що програмний продукт відповідає вимогам. У деяких випадках основна мета тестування може полягати в тому, щоб оцінити якість програмного забезпечення (без наміру встановити дефекти), дати інформацію

зацікавленим сторонам щодо ризику випуску системи у встановлений строк. Тестування на етапі супроводу здійснюється для запобігання внесення нових помилок у програму в результаті змін. Основна мета тестування на етапі експлуатації може полягати в тому, щоб оцінити системні характеристики, такі як надійність або працездатність.

Налагодження і тестування – різні процеси. Тестування може демонструвати відмови, які викликані помилками. Налагодження – функція розробки, що ідентифікує причину помилки, виправляє код і перевіряє, щоб помилки були коректно усунуті. Наступне тестування відповідності програми тестувальником гарантує те, що дефекти дійсно усунуті. Обов'язки в кожному випадку дуже розрізняються, тобто тестувальники тестують, а розроблювачі налагоджують.

Майєрс дає наступне визначення: *тестування – це процес виконання програми з метою виявлення помилок*. Вірний вибір мети дає важливий психологічний ефект. Якщо поставити метою демонстрацію відсутності помилок, то підсвідомо будуть вибиратися ті тестові дані, на яких імовірність появи помилки мала.

6.3 Основні принципи тестування

За останні 40 років була запропонована багато принципів тестування, загальних пропозицій і рекомендацій.

Принцип 1 – Тестування показує наявність помилок

Тестування може показати наявність помилок, але не може довести, що в програмі немає ніяких помилок. Тестування знижує ймовірність прихованих дефектів, що залишаються в програмному забезпеченні, але, навіть якщо ніякі помилки не виявлені, це – не доказ правильності.

Принцип 2 – Вичерпне тестування неможливе

Тестування всього (всі комбінації входів і попередніх умов) нездійсненно за винятком тривіальних випадків. Замість вичерпного тестування необхідно аналізувати ризик і пріоритети, щоб зосередити зусилля на тестуванні.

Принцип 3 – Раннє тестування

Тестування повинне початися якомога раніше в життєвому циклі програми, і повинне бути зосереджене на визначених цілях.

Принцип 4 – Групування (кластеризація) помилок

Невелика кількість модулів містить більшість помилок, виявлених протягом тестування попередньої версії, вони є відповідальними за найбільш серйозні відмови.

Принцип 5 – Парадокс пестициду

Якщо ті ж самі тести повторювати багато разів, в остаточному підсумку цей набір тестів не буде більше знаходити ні однієї нової помилки. Щоб перебороти цей “парадокс пестициду”, набори тестів повинні регулярно переглядатися і обновлятися, повинні складатися нові і різноманітні тести, здатні потенційно знайти більше помилок.

Принцип 6 – Тестування залежить від контексту програми

Тестування сильно відрізняється для різних типів програм. Наприклад, програмний засіб, для якого критичними є показники безпеки, трестується зовсім по-іншому, ніж сайт електронної комерції.

Принцип 7 – Припущення про відсутність помилок невірно

Виявлення і усунення помилок не допомагають, якщо розроблена система непридатна і не задовольняє вимоги і очікування користувачів.

6.4 Основні процеси тестування

Сама видима частина тестування – виконання тестів. Але щоб оцінити ефективність процедури тестування, необхідно також ураховувати час, що буде витрачено на планування тестів, проектування наборів тестових даних, підготовку до виконання і оцінку стану.

Процес тестування складається з наступних основних дій:

1) планування і контроль;

2) аналіз і розробка;

3) виконання;

4) оцінювання критеріїв завершеності і складання звіту;

5) завершення тестування.

Хоча ці дії виконуються звичайно в зазначеній логічній послідовності, але в процесі можуть також перетинатися або мати місце одночасно.

6.4.1 Планування тестування і контроль

Планування тестування – діяльність з підтвердження призначення тестування, визначенню цілей тестування і специфікації дій процесу тестування для досягнення поставлених цілей.

Контроль тестування – діяльність, що полягає в постійному порівнянні фактичного стану процесу із планом і повідомленні про відхилення від плану. Він містить у собі дії, необхідні для виконання призначення і цілей проекту. Щоб управляти тестуванням, необхідно здійснювати моніторинг протягом усього проекту. Планування тестування враховує зворотний зв'язок від дій моніторингу і контролю.

6.4.2 Аналіз і проектування тестів

Аналіз і проектування тестів – діяльність, у результаті якої основні цілі тестування перетворюються в реальні умови тестування і набори тестів.

Аналіз і проектування тестів включає такі основні задачі:

– розгляд основних складових тесту (вимоги, архітектура, проектування, інтерфейси);

– оцінка витрат на проведення тестування;

– ідентифікація і розташування за пріоритетами умов тестування на підставі аналізу тестів, специфікації, поведінки і структури;

– проектування і розташування за пріоритетами наборів тестів (test cases);

– розподіл необхідних тестових даних для забезпечення заданих умов тестування і тестових наборів даних;

– проектування середовища тестування, визначення необхідної інфраструктури і інструментальних засобів.

6.4.3 Виконання тестів

Виконання тестів – діяльність, де визначені тестові процедури або сценарії шляхом комбінування тестів у певному порядку, включаючи будь-яку іншу інформацію, необхідну для виконання тестів, встановлене середовище тестування і виконані тести.

Виконання тестів включає такі основні задачі:

– розробка, складання і упорядкування тестових наборів даних;

– розробка і упорядкування тестових процедур, створення тестових даних, підготовка засобів тестування і запис автоматизованих сценаріїв тестування;

– створення тестових наборів даних відповідно до процедур тестування для ефективного виконання тестів;

– перевірка того, щоб середовище тестування було встановлено правильно;

– виконання тестових процедур або вручну, або використовуючи інструментальні засоби виконання тестів у запланованій послідовності;

– реєстрація результатів виконання тесту, реєстрація ідентифікаторів і версій програмного засобу, що тестується, інструментальних засобів і тестів;

– порівняння фактичних результатів з очікуваними;

– повідомлення про невідповідності і їхній аналіз із метою встановлення причин (наприклад, помилка в коді, у заданих тестових даних, у тестовій документації або помилка в способі виконання тесту);

– повторення тестування як результат дії, вжитої для усунення кожної невідповідності. Наприклад, повторне виконання тесту, що попередньо зазнавав невдачі, щоб підтвердити виправлення помилки (тестування підтвердження), виконання скоректованого тесту і/або виконання тестів, щоб переконатися, що помилки не були уведені в незмінних областях програми або що виправлення помилки не виявляє інших помилок (регресійне тестування).

6.4.4 Оцінювання критеріїв завершеності і складання звіту

Оцінка критеріїв завершеності – діяльність, у результаті якої виконання тестів оцінюється відповідно до обраних критеріїв. Така оцінка повинна виконуватися для кожного рівня тестування.

Оцінка критеріїв завершеності включає такі основні задачі:

– перевірка результатів тестування на відповідність критеріям завершеності, визначеним у плані тестування;
– оцінювання необхідності проведення додаткових тестів або внесення змін у встановлені критерії завершеності;
– складання підсумкового звіту про тестування для зацікавлених сторін.

6.4.5 Завершення тестування

На етапі завершення тестування здійснюється збір даних про закінчені тестові дії. Наприклад, коли випущена програмна система, тестування проекту закінчене (або скасоване), завершені окремий етап або закінчений виправлена версія.

Завершення тестування включає такі основні задачі:

– перевірка завершення всіх робіт, закриття звіту про події, перевірка записів про зміни для будь-якого відкритого файлу і документації про приймання системи;
– завершення і архівування тестів, середовища тестування і інфраструктури для повторного використання;
– передача тестів організації, що виконує супровід;
– аналіз отриманих уроків для майбутніх версій і проектів і вдосконалення тестів.

6.5 Психологічні аспекти тестування

Менталітет тестувальника відрізняється від менталітету розроблювача. Розроблювач відрізняється конструктивним способом мислення, а тестувальник – деструктивним. Тому розроблювачеві складно тестувати свою програму і знаходити в ній помилки. Мета тестувальника – виявити якнайбільше помилок у програмі, а мета розроблювача – продемонструвати їхню відсутність. При правильному підході розроблювач здатний тестувати свій власний код, але розподіл його функцій з функціями тестувальника робиться для того, щоб допомогти зосередити зусилля і забезпечити додаткові переваги, такі як незалежний погляд і професіоналізм в області тестування. Незалежне тестування може бути виконане на будь-якому рівні тестування.

Деякий ступінь незалежності (запобігання упередженості автора) часто є більш ефективним при виявленні помилок і відмов. Проте, познайомившись із деякими принципами забезпечення незалежності, розроблювачі можуть ефективно знаходити багато помилок у їхньому власному коді. Можна виділити кілька рівнів незалежності:

– тести розроблені людиною (людьми), яка написала програмний засіб, що піддається тестуванню (низький рівень незалежності);
– тести розроблені іншою людиною (людьми) (наприклад, із групи розробки);
– тести розроблені людиною (людьми) з іншого структурного підрозділу (наприклад, групи незалежного тестування) або фахівцями з тестування (наприклад, фахівцями з тестування зручності використання і експлуатаційних характеристик);
– тести розроблені людиною (людьми) з іншої організації або компанії (тобто outsourcing або сертифікація третьою стороною).

Люди і проекти керуються цілями. Люди прагнуть погоджувати свої плани з набором цілей, установлених керівництвом і іншими зацікавленими сторонами, наприклад, знаходити помилки або підтверджувати, що програмне забезпечення працездатне. Тому важливо усвідомити цілі тестування.

Виявлення помилок у ході тестування може бути сприйняте як критика програми і її автора. Тестування, тому, часто бачиться як руйнівна діяльність, навіть при тому, що воно дуже конструктивно в керуванні ризиками продукту. Пошук помилок у системі вимагає цікавості, професійного песимізму, критичного погляду, уваги до деталей, гарних взаємин з розроблювачами і досвіду, щоб висувати припущення про помилки.

Якщо помилки, дефекти або відмови будуть повідомлені конструктивним способом, погіршення відносин між тестувальниками і розроблювачами можна уникнути. Це відноситься як до переглядів, так і до тестування.

Тестувальник і керівник групи тестування потребують гарної комунікабельності, щоб повідомити фактичну інформацію про помилки, поліпшення і ризики конструктивним способом. Інформація про помилки може допомогти авторові програмного забезпечення або документації поліпшити свої навички. Дефекти, знайдені і усунуті в ході тестування, будуть заощаджувати час і гроші на наступних етапах розробки і знизять ризики.

Проблеми у взаєминах можуть відбутися, особливо якщо тестувальники розглядаються тільки як посильні, що приносять небажані новини про помилки. Однак, є кілька способів поліпшити відносини між тестувальниками і розроблювачами:
– починають зі співробітництва, а не бою – нагадують кожному про загальну мету поліпшення якості системи;
– повідомляють результати тестування продукту нейтральним, зосередженим на факті способом, не критикуючи тієї людини, яка створила це, наприклад, складають звіт про фактичні помилки і роблять огляд результатів;
– пробують зрозуміти, що почувають інші люди, чому вони реагують на ситуацію певним чином, як вони працюють;
– переконуються, що інша людина зрозуміла те, що йому сказали, і навпаки.

6.6 Контрольні запитання і завдання

1. Навіщо необхідно тестувати програми?
2. Які причини помилок у програмному забезпеченні? Яка роль тестування у розробці, експлуатації і супроводі програмних засобів?
3. Що таке тестування, яка його мета?
4. Назвіть основні принципи тестування.
5. З яких основних дій складається процес тестування?
6. У чому полягає планування і контроль тестування?
7. Які основні задачі аналізу і проектування тестів?
8. Назвіть послідовність дій при виконанні тестів.
9. У чому полягає оцінювання критеріїв завершеності тестування?
10. Які основні задачі включає завершення тестування?
11. Які психологічні аспекти тестування програмного забезпечення?

Лекція 7
ТЕСТУВАННЯ ПРОГРАМНИХ ЗАСОБІВ ПРОТЯГОМ ЖИТТЄВОГО ЦИКЛУ

7.1 Моделі розробки програмного забезпечення

Тестування не існує ізольовано; процеси тестування тісно пов'язані із процесами розробки програмних засобів. Існують різні моделі життєвого циклу розробки, які потребують різних підходів до тестування.

На сьогодні найбільшого поширення набули дві основні моделі життєвого циклу:

1) каскадна (послідовна) модель;
2) ітераційно-зростаюча (спіральна) модель.

7.1.1 V-модель (каскадна або послідовна модель розробки)

Її основною характеристикою є розбивка всієї розробки на етапи, причому перехід від одного етапу до наступного відбувається тільки після того, як буде повністю завершена робота на поточному. Кожний етап завершується випуском повного комплекту документації, достатньої для того, щоб розробка могла бути продовжена іншою командою розроблювачів.

Позитивні сторони застосування каскадного підходу:

1) на кожному етапі формується закінчений набір проектної документації, що відповідає критеріям повноти і узгодженості;
2) виконувані в логічній послідовності етапи робіт дозволяють планувати строки завершення всіх робіт і відповідні витрати.

У процесі використання цього підходу виявився ряд його недоліків, викликаних насамперед тим, що реальний процес створення ПЗ ніколи повністю не укладався в таку жорстку схему. У процесі створення ПЗ постійно виникала потреба в поверненні до попередніх етапів і уточненні або перегляді раніше ухвалених рішень.

Основним недоліком каскадного підходу є істотне запізнювання з одержанням результатів. Узгодження результатів з користувачами здійснюється тільки в точках, планованих після завершення кожного етапу робіт, вимоги до програми "заморожені" у вигляді технічного завдання на увесь час її створення. Таким чином, користувачі можуть внести свої зауваження тільки після того, як робота над системою буде повністю завершена. У випадку неточного викладення вимог або їхньої зміни протягом тривалого періоду створення ПЗ користувачі одержують систему, що не задовольняє їхнім потребам.

Хоча існують кілька варіантів V-моделі, звичайний тип V-моделі використовує чотири рівні тестування, що відповідають чотирьом рівням проектування:

- тестування компонентів (модулів);
- інтеграційне тестування;
- системне тестування;
- приймальне тестування.

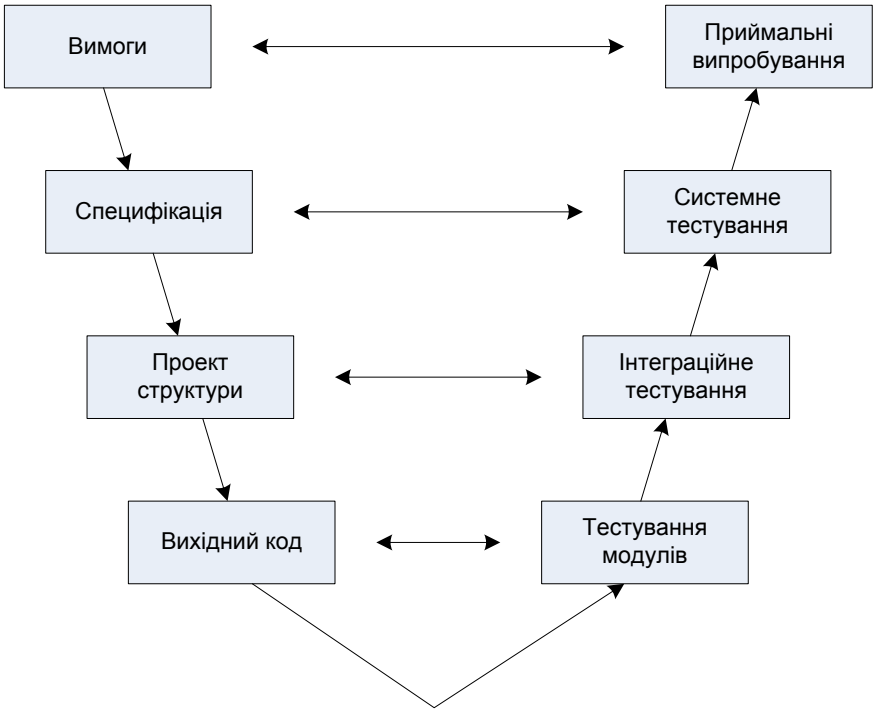

Рисунок 7.1 – V-модель розробки

На практиці V-модель може мати більше, менше або навіть різне число рівнів розробки і тестування, залежно від проекту і програмного виробу. Наприклад, інтеграційне тестування компонентів може проводитися після тестування компонентів і системне інтеграційне тестування після системного тестування.

Програмні продукти (такі як бізнес-сценарії або використання CASE-засобів, специфікації вимог, розробка документації і коду) створені на протязі розробки, часто є об'єктом тестування на одному або декількох рівнях тестування. Верифікація і валідація (і раннє проектування тестів) може виконуватись протягом розробки програмних виробів.

6.1.2 Ітераційно-зростаючі (спіральні) моделі розробки

Ітераційно-зростаюча розробка – процес установлення вимог, проектування, формування і випробування системи, реалізований як ряд більш коротких циклів розробки. Система, отримана в результаті ітерації, може бути протестована на декількох рівнях як частина розробки. Приріст, який додають до попередньо спроектованих частин, формує зростаючу систему, яка повинна також бути протестована. Регресійне тестування одержує усе більш важливе значення на всіх ітераціях після першої. Верифікація і валідація можуть бути виконані на кожному прирості.

Одним з можливих підходів до розробки ПЗ в рамках спіральної моделі життєвого циклу є методологія швидкої розробки додатків RAD (Rapid Application Development), що набула останнім часом широкого поширення. Під цим терміном звичайно розуміється процес розробки ПЗ, що містить 3 елементи:

1) невелику команду програмістів (від 2 до 10 чоловік);
2) короткий, але ретельно пророблений виробничий графік (від 2 до 6 місяців);
3) повторюваний цикл, при якому розроблювачі, у міру того, як додаток починає набувати форми, запитують і реалізують у продукті вимоги, отримані через взаємодію із замовником.

Команда розроблювачів повинна представляти із себе групу професіоналів, що мають досвід в аналізі, проектуванні, генерації коду і тестуванні ПЗ з використанням CASE-засобів. Члени колективу повинні також уміти трансформувати в робочі прототипи пропозиції кінцевих користувачів.

Життєвий цикл ПО за методологією RAD складається із чотирьох фаз:

1) фаза аналізу і планування вимог;
2) фаза проектування;
3) фаза побудови;
4) фаза впровадження.

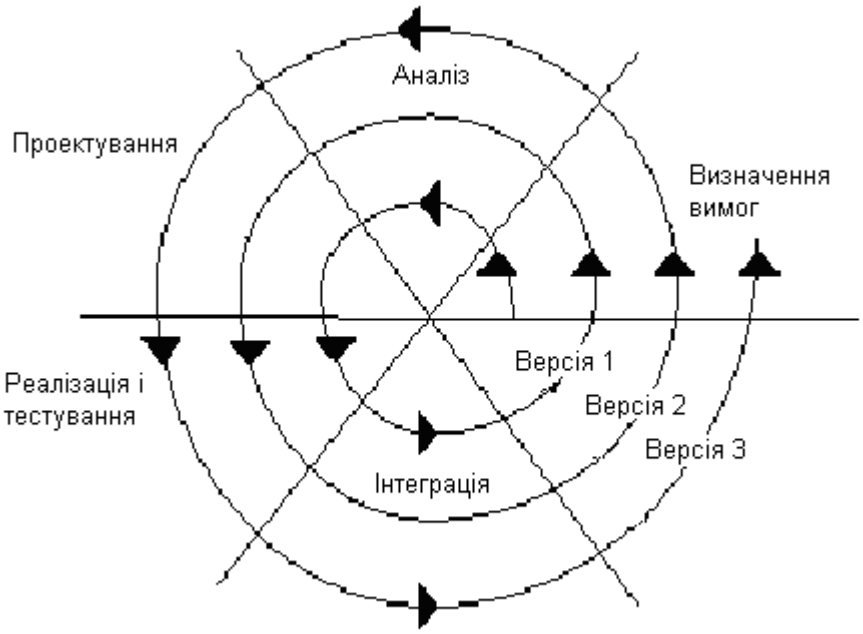

Рисунок 7.2 – Спіральна модель розробки

Розробка ітераціями відбиває об’єктивно існуючий спіральний цикл створення системи. Неповне завершення робіт на кожному етапі дозволяє переходити на наступний етап, не чекаючи повного завершення роботи на поточному. При ітеративному способі розробки відсутню роботу можна буде виконати на наступній ітерації. Головне ж завдання – якнайшвидше показати користувачам системи працездатний продукт, тим самим активізуючи процес уточнення і доповнення вимог.

Треба, однак, відзначити, що методологія RAD, як і будь-яка інша, не може претендувати на універсальність, вона гарна в першу чергу для відносно

невеликих проектів, розроблюваних для конкретного замовника. Якщо ж розробляється типова система, що не є закінченим продуктом, а являє собою комплекс типових компонентів, що централізовано супроводжується, адаптується до програмно-технічних платформ, СУБД, засобів телекомунікації, організаційно-економічних особливостей об'єктів впровадження і інтегрується з існуючими розробками, на перший план виступають такі показники проекту, як керованість і якість, які можуть суперечити простоті і швидкості розробки. Для таких проектів необхідний високий рівень планування і жорстка дисципліна проектування, строге слідування заздалегідь розробленим протоколам і інтерфейсам, що знижує швидкість розробки.

Методологія RAD незастосовна для побудови складних розрахункових програм, операційних систем або програм керування космічними кораблями, тобто програм, що вимагають написання великого обсягу (сотні тисяч рядків) унікального коду.

Не підходять для розробки за методологією RAD програми, у яких відсутня яскраво виражена інтерфейсна частина, що наочно визначає логіку роботи системи (наприклад, програми реального часу) і програми, від яких залежить безпека людей (наприклад, керування літаком або атомною електростанцією), тому що ітеративний підхід припускає, що перші кілька версій напевно не будуть повністю працездатними, що в цьому випадку виключається.

7.1.3 Тестування в моделі життєвого циклу

У будь-якій моделі життєвого циклу є кілька характеристик гарного тестування:

- для кожного етапу розробки є відповідний етап тестування;
- кожний рівень тестування має свої специфічні цілі;
- аналіз і проектування тестів для даного рівня тестування повинен початися протягом відповідного етапу розробки;
- тестувальники повинні брати участь у розгляді документів, як тільки проекти цих документів стають доступними в життєвому циклі розробки.

Рівні тестування можуть бути об'єднані або реорганізовані залежно від характеру проекту або архітектури системи. Наприклад, для інтеграції готового комерційного програмного продукту в систему, покупець може виконати інтеграційне тестування на системному рівні (інтеграція в інфраструктуру та інші системи, або запровадження в дію системи) і приймальні випробування (функціональне та/або не функціональне тестування і дослідна експлуатація).

7.2 Рівні тестування

Для кожного з рівнів тестування можна вказати такі параметри: їхні загальні цілі, тестові приклади, об'єкти тестування (тобто те, що перевіряється), типові дефекти і помилки, які будуть знайдені, засоби тестування і інструменти підтримки, певні підходи і обов'язки.

7.2.1 Тестування компонентів (модулів)

Тестування компонентів досліджує помилки і перевіряє функціонування програмних компонентів, які тестуються окремо (модулі, функції, об'єкти, класи і т.д.). Це може бути зроблено ізольовано від іншої частини системи, залежно від контексту життєвого циклу проектування і системи. Можуть використовуватися заглушки, драйвери та імітатори.

Тестування модулів може включати тестування функціональності і визначення нефункціональних характеристик, таких як поводження ресурсу (наприклад, витоки пам'яті) або стійкості до аномалій, структурне тестування (наприклад, покриття гілок). Тестові приклади одержують зі специфікації компонента, проектування програми або моделі даних.

Як правило, тестування компонента відбувається з доступом до коду, що перевіряється, і з підтримкою середовища розробки, типу структури модуля, що тестується, або засобу налагодження, і, на практиці, звичайно залучають програміста, що написав код. Помилки зазвичай виправляються, як тільки вони знайдені, формально не роблячи записів про дефекти.

Один з підходів до тестування компонентів полягає в тому, щоб підготувати і автоматизувати тестові приклади перед кодуванням. Його називають «спочатку тестування» або керована тестуванням розробка. Такий підхід є дуже ітераційним і заснований на циклах проектування тестових прикладів, потім складання і інтеграції невеликих частин коду, і виконання тестування компонентів.

7.2.2 Інтеграційне тестування

Інтеграційне тестування перевіряє інтерфейси між компонентами, взаємодії між різними частинами системи, типу операційної системи, файлової системи, апаратних засобів, або інтерфейси між системами.

Інтеграційне тестування може містити кілька рівнів, і може бути виконане на тестових об'єктах змінюваного розміру. Наприклад:

1) компонентне інтеграційне тестування перевіряє взаємодії між програмними компонентами і виконується після тестування модулів;

2) системне інтеграційне тестування перевіряє взаємодії між різними системами і може бути виконане після системного тестування. У цьому випадку організація-розроблювач може управляти тільки однією стороною інтерфейсу, а зміни можуть вносити нестабільність. Тут можливе виникнення міжплатформових проблем.

Чим більше область інтеграції, тим складніше стає ізолювати відмови певних компонентів або систем, що може привести до підвищеного ризику.

Систематичні стратегії інтеграції можуть бути засновані на системній архітектурі (низхідна і висхідна), функціональних завданнях, послідовностях діалогової обробки запитів, або деякому іншому аспекті системи або компонента. Щоб знизити ризик пізнього виявлення помилки, інтеграція повинна виконуватися крок за кроком, а не методом "великого удару".

Визначення деяких нефункціональних характеристик (наприклад, продуктивності) може бути включене в інтеграційне тестування.

На кожній стадії інтеграції тестувальники концентруються винятково на самій інтеграції. Наприклад, якщо вони інтегрують модуль А з модулем В, вони цікавляться тестуванням зв'язку між модулями, а не функціональними можливостями кожного модуля. Тут можуть використовуватися і функціональні і структурні підходи.

В ідеалі, тестувальники повинні зрозуміти архітектуру і вплив планування інтеграції. Якщо інтеграційні тести заплановані раніше, ніж складання компонентів або систем, вони можуть бути сформовані в порядку, що забезпечує найбільш ефективне тестування.

7.2.3 Системне тестування

Тестування системи займається перевіркою відповідності поведінки інтегрованої системи (продукту) вихідним вимогам проекту.

У системному тестуванні середовище функціонування програми повинно в максимально можливому ступені відповідати промисловому середовищу експлуатації, щоб мінімізувати ризик пропуску помилок, викликаних середовищем.

Системне тестування може включати тести, засновані на ризиках та/або вимогах специфікацій, бізнес-процесах, використовувати CASE-засоби або інші описи поведінки системи високого рівня, взаємодії з операційною системою і системними ресурсами.

Системне тестування повинне досліджувати і функціональні і нефункціональні вимоги системи. Вимоги можуть існувати як текст та/або моделі. Тестувальники також повинні мати справу з неповними або недокументованими вимогами. Системне тестування функціональних вимог починається з використання методів «чорної скриньки», які найбільш відповідають досліджуваному аспекту тестованої системи. Наприклад, для подання комбінацій ефектів, описаних у ділових правилах, може бути створена таблиця рішень. Структурні методи тестування (біла скринька) можуть використовуватися, щоб оцінити ретельність тестування щодо структурного елемента, такого як структура меню або навігація web-сторінки.

Системне тестування часто виконується незалежною групою тестувальників.

7.2.4 Приймальні випробування

Приймальні випробування часто є прерогативою замовників або користувачів системи; можуть також залучатися інші зацікавлені сторони.

Мета приймальних випробувань полягає в тому, щоб одержати впевненість у якості системи, частини системи або певних нефункціональних характеристик системи. Приймальні випробування не фокусуються на виявленні помилок. Приймальні випробування можуть оцінити готовність системи для розгортання і використання, хоча це не обов'язково кінцевий (заключний) рівень тестування. Наприклад, великомасштабне інтеграційне тестування системи може проводитися після приймальних випробувань системи.

Приймальні випробування можуть охоплювати більш ніж один рівень тестування, наприклад:

- приймальні випробування готового програмного продукту можуть проводитися, коли його інстальовано або інтегровано;
- приймальні випробування зручності використання компонентів можуть бути виконані в ході тестування компонентів;
- приймальні випробування нового функціонального розширення можуть виконуватися перед системним тестуванням.

Типові форми приймальних випробувань включають:

1) *тестування прийнятності для користувача*.

Звичайно перевіряється придатність системи для використання бізнес-користувачами;

2) *тестування в реальних умовах (дослідна експлуатація)*.

Прийняття системи системними адміністраторами, включаючи:

- тестування резервного копіювання/відновлення;
- аварійне відновлення;
- користувальницьке керування;
- задачі супроводу;
- періодичні перевірки уразливості захисту;

3) *контракт і правила приймальних випробувань*.

Розробка контракту приймальних випробувань, що включає критерії приймання для розробленого під замовлення програмного забезпечення. Приймальні критерії повинні бути визначені, коли контракт узгоджений.

Складання правил приймальних випробувань із урахуванням вимог будь-яких інструкцій, які потрібно дотримувати (урядових, юридичних або правил техніки безпеки);

4) *альфа- і бета-тестування*.

Розроблювачі ПЗ часто хочуть мати зворотний зв’язок з потенційними або існуючими споживачами на ринку перед тим, як програмний продукт надійде для комерційних продажів.

Альфа-тестування – імітація реальної роботи із системою штатними розроблювачами, або реальна робота із системою потенційними користувачами/замовником на стороні розроблювача. Часто альфа-тестування застосовується для закінченого продукту в якості внутрішнього приймального тестування. Іноді альфа-тестування виконується під налагоджувачем або з використанням оточення, що допомагає швидко виявляти знайдені помилки. Виявлені помилки можуть бути передані тестувальникам для додаткового дослідження в оточенні, подібному тому, у якому буде використовуватися ПЗ.

Бета-тестування – попереднє тестування програмних продуктів обраними типовими користувачами і партнерами для видалення помилок, неадекватностей і можливого вдосконалення продукту. Завдання бета-тестування – виявити

максимальну кількість помилок, одержати від бета-тестерів об’єктивну оцінку його якості. Код продукту під час бета-тестування ще не “заморожений”. Більш того, початкове бета-тестування може бути розпочато навіть при неповній готовності продукту.

7.3 Види тестування

Деякі тестувальні дії можуть бути призначені для верифікації програмної системи (або частини системи), заснованої на певних міркуваннях або меті тестування.

Кожний вид тестування зосереджений на певній меті тестування, наприклад:

- тестування функціональності програми;
- визначення нефункціональних показників якості, таких як надійність або зручність використання, структура або архітектура системи;
- підтвердження того, що помилки були виправлені (тестування відповідності);
- пошук непередбачених змін (регресійне тестування).

Модель програмного засобу може бути розроблена і/або використана в структурному і функціональному тестуванні, наприклад, у функціональному тестуванні – модель технологічного маршруту, модель переходу зі стану в стан або специфікація природною мовою; у структурному тестуванні – модель потоку керування або модель структури меню.

7.3.1 Тестування функціональності (функціональне тестування)

Функції, які система, підсистема або компонент повинні виконувати, можуть бути описані в технічних завданнях, функціональних специфікаціях, або вони можуть бути недокументованими. Функції – це опис того, “що” робить система.

Функціональне тестування засноване на функціях і особливостях (описаних у документації або зрозумілих тестувальниками) і можливості їхньої взаємодії з певними системами, і може бути виконано на всіх рівнях тестування (наприклад, тестування компонентів може бути засноване на специфікаціях компонентів).

Методи, засновані на використанні специфікації, можуть застосовуватися для одержання умов тестування і тестових прикладів з функціональних можливостей програмного засобу або системи. Функціональне тестування аналізує зовнішнє поводження програмного засобу (тестування “чорної скриньки”).

Один з видів функціонального тестування, тестування захищеності, досліджує функції (наприклад, міжмережевий екран), що стосуються виявлення погроз, таких як віруси, від зовнішніх зловмисників. Інший вид функціонального тестування, тестування функціональної сумісності, оцінює можливість програмного виробу взаємодіяти з одним або декількома зазначеними компонентами або системами.

7.3.2 Тестування нефункціональних програмних характеристик (нефункціональне тестування)

Нефункціональне тестування включає, але не обмежується цим, тестування експлуатаційних характеристик, навантажувальне тестування, стресове тестування, тестування зручності використання, тестування супроводжуваності, тестування надійності і тестування переносимості. Це – тестування того, "як" система працює.

Тестування нефункціональних характеристик може бути виконане на всіх рівнях тестування. Терміном «нефункціональне тестування» описуються тести, необхідні для вимірювання характеристик програмних систем, які можуть бути визначені кількісно в масштабі, що змінюється (наприклад, час відповіді для окремої функції). Такі тести можуть посилатися на модель якості програмного забезпечення, визначену в міжнародному стандарті ISO 9126 "Software Engineering – Software Product Quality".

7.3.3 Тестування програмної структури/архітектури (структурне тестування)

Структурне тестування (тестування «білої скриньки») може бути виконане на всіх рівнях тестування. Структурні методи найкраще використовувати після методів на основі специфікацій, щоб допомогти виміряти ретельність тестування через оцінку покриття структури.

Покриття – ступінь, у якій структура була протестована тестовим набором, виражається у відсотках від елементів, що покриваються (охоплюються). Якщо покриття не 100 %, то можуть бути розроблені додаткові тести для перевірки тих елементів, які були пропущені, і збільшення покриття.

На всіх рівнях тестування, особливо при тестуванні модулів і інтеграційному тестуванні, можуть використовуватися такі критерії для вимірювання покриття коду, як покриття операторів і рішень. Структурне тестування може ґрунтуватися на архітектурі системи, такій як ієрархія викликів.

Підходи структурного тестування можуть також застосовуватися в системі, системній інтеграції або рівнях приймальних випробувань (наприклад, до бізнес-моделей або структур меню).

7.3.4 Тестування змін (тестування підтвердження і регресійне тестування)

Після того, як дефект виявлений і усунутий, програмне забезпечення повинне тестуватися повторно, щоб підтвердити, що первісний дефект був успішно вилучений. Це називають підтвердженням. Налагодження (усунення дефекту) – це завдання розроблювача, а не тестувальника.

Регресійне тестування – повторне тестування вже перевіреної програми, після модифікації з метою виявити будь-які помилки, внесені або розкриті в результаті змін. Ці помилки можуть бути або в програмі, що тестується, або в іншому зв'язаному або незв'язаному програмному компоненті. Цей вид тестування виконується, коли програма або її середовище змінені. Визначення

ступеня (обсягу) регресійного тестування засновано на ризику не виявлення дефектів у програмному засобі, що попередньо працював.

Тести повинні мати високу повторюваність, якщо вони будуть використовуватися для тестування підтвердження і допомагати регресійному тестуванню.

Регресійне тестування може бути виконане на всіх рівнях тестування, і застосовуватися у функціональному, нефункціональному і структурному тестуванні. Виконання регресійних тестів займає багато часу і звичайно відбувається повільно, тому регресійне тестування доцільно автоматизувати.

7.4 Тестування супроводу

Після того, як програмна система розгорнута, вона функціонує протягом багатьох років або десятиліть. У цей час система і її середовище часто виправляються, змінюються або розширюються. Тестування супроводу здійснюється на існуючій працюючій системі, і викликано модифікаціями, переміщенням або вилученням з обігу програмного засобу.

Модифікації включають зміни, пов’язані із запланованим розширенням (наприклад, випуску нової версії), коригуючі і аварійні зміни і зміни середовища, типу запланованого відновлення операційної системи або бази даних, виправлень до недавно виявленої уразливості операційної системи.

Тестування супроводу, пов’язане з переміщенням (наприклад, з однієї платформи на іншу), повинне включати експлуатаційні випробування як нового середовища, так і зміненого програмного засобу.

Тестування супроводу, пов’язане із припиненням використання системи, може включати тестування перенесення даних або архівування, якщо потрібні тривалі періоди зберігання даних.

На додаток до тестування виконаних змін, тестування супроводу включає велике регресійне тестування і тих частин системи, які не були змінені. Обсяг тестування супроводу пов’язаний з ризиком зміни, розміром існуючої системи і розміром змін.

Залежно від змін, тестування супроводу може бути виконане на будь-яких рівнях тестування і включати будь-які види тестування.

Визначення того, як існуюча система може бути порушена змінами, називають аналізом впливу, і використовується для визначення обсягу регресійного тестування.

Тестування супроводу може бути утруднено, якщо специфікації є застарілими або відсутні.

7.5 Контрольні запитання і завдання

1. Назвіть основні моделі життєвого циклу програмного забезпечення. У яких випадках застосовується кожна з моделей?
2. Назвіть переваги та недоліки каскадної моделі розробки програмного забезпечення.
3. Назвіть переваги та недоліки ітераційно-зростаючої моделі розробки програмного забезпечення.
4. Назвіть основні рівні тестування програмного забезпечення. Що є метою та об’єктами тестування на кожному рівні?
5. Які основні види тестування програмного забезпечення?
6. Що перевіряється під час функціонального тестування?
7. Які дії передбачає нефункціональне тестування?
8. Як здійснюється структурне тестування?
9. Що таке регресійне тестування, з якою метою воно здійснюється?
10. Що включає у себе тестування супроводу?

Лекція 8
МЕТОДИ СТАТИЧНОГО ТЕСТУВАННЯ

8.1 Статичні методи і процес тестування

На відміну від динамічного тестування, що вимагає виконання програми, статичні методи тестування ґрунтуються на ручній перевірці (переглядах) і автоматизованому аналізі (статичний аналіз) коду або іншої проектної документації.

Перегляди – це спосіб тестування програмних продуктів (включаючи код), і вони можуть бути виконані задовго до динамічного тестування. Помилки, виявлені протягом переглядів на ранніх етапах життєвого циклу, часто набагато простіше й дешевше усунути, ніж помилки, виявлені при виконанні тестів (наприклад, помилки, знайдені у вимогах).

Перегляд може бути виконаний повністю ручним способом, але існують також засоби підтримки. Основне завдання ручного тестування – дослідити програмний виріб і записати коментарі про це. За допомогою переглядів може бути досліджена будь-яка частина проекту, включаючи технічні завдання, специфікації проекту, код, плани тестування, специфікації тестів, тестові приклади, тестові сценарії, настанови користувачів або web-сторінки.

Вигоди від переглядів включають раннє виявлення і виправлення помилок, підвищення продуктивності проектування, скорочення часу розробки, зменшення вартості і часу випробувань, зменшення витрат на різних етапах життєвого циклу, мінімізація кількості помилок і поліпшення комунікативності. Перегляди можуть знайти недоліки, наприклад, у вимогах, які, навряд чи будуть знайдені при динамічному тестуванні.

Перегляди, статичний аналіз і динамічне тестування мають ту саму мету – ідентифікація помилок. Вони доповнюють один одного: різні методи можуть ефективно знаходити різні типи помилок. У порівнянні з динамічним тестуванням, статичні методи знаходять причини відмов (дефектів), а не самі відмови безпосередньо.

Типові помилки, які простіше знайти за допомогою переглядів, ніж динамічним тестуванням: відхилення від стандартів, недосконалість вимог, помилки проектування, недоліки супроводу та неправильні специфікації інтерфейсу.

8.2 Процеси перегляду

Різні типи переглядів можуть змінюватися від дуже неформальних (наприклад, коли відсутні письмові інструкції для рецензентів) до дуже формальних (тобто добре структурованих і регульованих). Ступінь формалізації процесу перегляду пов’язана з такими факторами, як зрілість процесу розробки, наявність юридичних або регулюючих вимог або необхідність робити контрольні записи.

Спосіб виконання перегляду залежить від установленої мети перегляду (наприклад, знаходження помилок, розуміння логіки роботи або обговорення і ухвалення погодженого рішення).

8.2.1 Етапи формального перегляду

Типовий формальний перегляд включає такі основні етапи:

1. Планування: відбір учасників, розподіл ролей; визначення критеріїв входу і виходу для більш формальних типів перегляду (наприклад, інспекції); і вибір тієї частини документів, яку необхідно розглянути.

2. Початок: розподіл документів; пояснення мети, процесів і документів учасникам; перевірка критеріїв входу (для більш формальних типів перегляду).

3. Індивідуальна підготовка: робота, виконана кожним з учасників самостійно перед засіданням, відмічання потенційних помилок, питань і коментарів.

4. Засідання: обговорення або нарада, із задокументованими результатами або протоколом (для більш формальних типів перегляду). Учасники засідання можуть просто відзначати помилки, давати рекомендації з усунення помилок або приймати відповідні рішення.

5. Виправлення: усунення знайдених типових помилок, допущених автором.

6. Завершення: перевірка того, що помилки були усунуті, визначення метрик і перевірка критеріїв виходу (для більш формальних типів перегляду).

8.2.2 Ролі і обов'язки

Типовий формальний перегляд буде включати такі ролі:

Керівник (менеджер): ухвалює рішення щодо проведення перегляду, розподіляє час у календарному плані проекту, визначає, чи були досягнуті цілі перегляду.

Ведучий (модератор): людина, що веде перегляд документа або пакета документів, включаючи планування перегляду, проведення засідання й завершальні дії після засідання. Якщо буде потреба, ведучий може бути посередником між різними точками зору, часто це людина, від якої залежить успіх перегляду.

Автор: людина, яка написала документ, що розглядається, або несе за нього відповідальність.

Рецензенти: люди з певною технічною або діловою підготовкою (їх також називають перевіряючими або інспекторами), які після необхідної підготовки ідентифікують і описують помилки в розглянутому продукті. Рецензенти повинні вибиратися таким чином, щоб представити різні точки зору і ролі в процесі перегляду, і повинні брати участь у всіх засіданнях перегляду.

Секретар (реєстратор): документує всі невідповідності, проблеми і помилки, які були ідентифіковані в ході засідання.

Розгляд документів з різних точок зору та використання списків контрольних питань може зробити перегляди більше ефективними (наприклад, список контрольних питань, заснований на точці зору користувача, фахівця із

супроводу, тестувальника або експлуатаційника або контрольний список типових помилок).

8.2.3 Види переглядів

Окремий документ може бути предметом більше ніж одного перегляду. Якщо використовується більше ніж один тип перегляду, порядок дій може змінитися. Наприклад, неформальний перегляд може бути виконаний перш, ніж технічний перегляд, або інспекція може бути виконана на основі специфікацій вимог перед наскрізним контролем за участю замовників. Основні характеристики, варіанти і цілі основних видів перегляду:

Неформальний перегляд

Ключові характеристики:

- неформальний процес;
- допускається поєднання програмування і переглядів коду;
- може використовуватися документування, але не обов’язково;
- може змінюватися по повноцінності залежно від рецензента;
- основна мета: недорогий спосіб одержати деяку вигоду.

Наскрізний контроль

Ключові характеристики:

- засідання очолює автор;
- сценарії, пробні прогони, група рівноправних учасників;
- не обмежені за часом засідання;
- можлива попередня підготовка рецензентів, звіт про перегляд, список помилок і секретар (той, хто не автор);
- може змінюватися на практиці від досить неформального до дуже формального;
- основні цілі: вивчення, одержання розуміння, виявлення помилок.

Технічний аналіз

Ключові характеристики:

- документований, певний процес виявлення помилок, що включає рівних за положенням учасників і технічних експертів;
- може бути виконаний як перегляд групою рівноправних учасників без участі керівництва;
- в ідеалі очолюється навченим ведучим (який не є автором);
- виконується попередня підготовка;
- можуть використовуватися списки контрольних питань, звіт про перегляд, список знайдених невідповідностей і участь керівництва;
- може змінюватися на практиці від досить неформального до дуже формального;
- основні цілі: обговорення, прийняття рішень, оцінка альтернатив, знаходження помилок, вирішення технічних проблем і перевірка відповідності специфікаціям і стандартам.

Інспекція

Ключові характеристики:

- очолюється навченим ведучим (не автор);
- зазвичай проводиться експертиза;
- визначені ролі;
- включає метрики;
- формальний процес, заснований на правилах і списках контрольних питань із критеріями входу і виходу;
- попередня підготовка;
- звіт про інспекції, список знайдених помилок;
- формальний процес завершення;
- можливо, уточнення процесу і читач;
- основна мета: знайти помилки.

Наскрізний контроль, технічний аналіз і інспекція можуть бути виконані групою рівноправних колег на тому ж самому організаційному рівні. Цей тип перегляду називають “peer review”.

8.2.4 Фактори, що визначають успіх переглядів

Фактори, що визначають успіх переглядів:

- кожний перегляд має ясну визначену мету;
- для досягнення цілей перегляду залучено потрібних людей;
- заохочується знаходження помилок, знайдені помилки виражені об’єктивно;
- людський фактор і психологічні аспекти мають місце;
- застосовуються методи перегляду, які найбільше підходять до типу і рівня програмних продуктів і рецензентів;
- використовуються списки контрольних питань або ролі, щоб збільшити ефективність виявлення помилок;
- навчання є частиною методів перегляду, особливо більш формальних, таких як інспекція;
- керівництво підтримує ефективний процес перегляду (наприклад, включаючи адекватний час для виконання перегляду в календарний графік проекту);
- робиться наголос на навчання і удосконалення процесу.

8.3 Статичний аналіз із використанням інструментальних засобів

Ціль статичного аналізу полягає в тому, щоб знайти помилки у вихідному коді програми і програмних моделях. Статичний аналіз виконується фактично без виконання програми, досліджуваної за допомогою спеціальних інструментальних засобів; динамічне тестування виконує програмний код. Статичний аналіз може визначити місцезнаходження помилок, які важко знайти шляхом тестування. Як і перегляди, статичний аналіз знаходить помилки, а не відмови. Статичні інструментальні засоби аналізу аналізують код програми

(наприклад, потік керування і потік даних), а також згенерований код типу HTML і XML.

Переваги статичного аналізу:

1) раннє виявлення помилок до виконання тестів;
2) раннє попередження про підозрілі аспекти коду або проекту за допомогою обчислення метрик (наприклад, високий показник складності);
3) ідентифікація помилок, які складно знайти динамічним тестуванням;
4) виявлення залежностей і непогодженостей у програмних моделях;
5) покращена супроводжуваність коду і проекту;
6) запобігання помилок шляхом винесення уроків із проектування.

Типові помилки, які виявляються інструментальними засобами статичного аналізу:

1) посилання на змінну з невизначеним значенням;
2) несумісний інтерфейс між модулями і компонентами;
3) змінні, які ніколи не використовуються;
4) невиконуваний (мертвий) код;
5) порушення стандартів програмування;
6) уразливість захисту;
7) порушення синтаксису коду і програмних моделей.

Інструментальні засоби статичного аналізу звичайно використовуються розроблювачами (які перевіряють дотримання визначених правил або стандартів програмування) до і протягом тестування компонентів і інтеграційного тестування та проектувальниками протягом програмного моделювання. Інструментальні засоби статичного аналізу можуть видавати велику кількість попереджуючих повідомлень, якими необхідно керуватися, щоб найбільш ефективно використовувати інструмент.

Компілятори можуть забезпечити деяку підтримку статичному аналізу, включаючи обчислення метрик.

8.4 Контрольні запитання і завдання

1. Які помилки простіше виявити методами статичного тестування?
2. Які основні етапи формального перегляду?
3. Які ролі і обов’язки учасників перегляду?
4. Назвіть ключові характеристики неформального перегляду.
5. Які особливості наскрізного контролю?
6. Назвіть ключові характеристики технічного контролю.
7. Назвіть ключові характеристики інспекції.
8. Які фактори визначають успіх переглядів?
9. Які переваги має статичний аналіз?

Лекція 9
МЕТОДИ ПРОЕКТУВАННЯ ТЕСТІВ

9.1 Процес розробки тестів

Процес розробки тестів може бути виконаний різними способами, від дуже неформального з маленьким обсягом документації або взагалі без неї, до дуже формального. Рівень формальності залежить від змісту тестування, включаючи організацію, зрілість тестування і процесів розробки, часових обмежень і залучених фахівців.

На етапі аналізу аналізується тестова документація, щоб визначити, що потрібно тестувати, тобто ідентифікувати умови тестування. Умови тестування визначають елемент, що може бути перевірений одним або більше тестовими прикладами (наприклад функція, транзакція, показник якості або структурний елемент).

Установлення трасованості від умов тестування назад до специфікацій і вимог дає можливість робити аналіз впливу, коли вимоги змінюються і покриття вимог визначається набором тестів. На етапі аналізу виконується детальний вибір методів проектування тестів, заснований крім усього іншого на оцінці ризиків.

Протягом проектування тестів створюються і визначаються тестові приклади та тестові дані. Тестовий приклад (test case) складається з набору вхідних значень, попередніх умов виконання, очікуваних результатів і вихідних умов виконання. Стандарт IEEE 829 “Standard for Software Test Documentation” описує зміст специфікацій проектування тестів (включаючи умови тестування) і специфікації тестових прикладів.

Очікувані результати повинні бути частиною специфікації тестового приклада і включати вихідні дані, зміни даних і станів і будь-які інші результати тесту. Якщо очікувані результати не були визначені, то правдоподібні, але помилкові результати роботи програми можуть бути інтерпретовані як правильні. Очікувані результати повинні бути точно визначені до виконання тестів.

У ході виконання тестування тестові приклади повинні бути розроблені, реалізовані, розташовані за пріоритетами і організовані в специфікації процедури тестування. Процедура тестування (або сценарій виконання тестів) визначає послідовність дій при виконанні тестування. Якщо тести виконуються, використовуючи інструмент виконання тестів, послідовність дій визначена в сценарії тестування (який є автоматизованою процедурою тестування).

Різні процедури тестування і автоматизовані сценарії тестування згодом формуються в графік виконання тестів, що визначає порядок, у якому виконуються різні тестові процедури і автоматизовані сценарії, коли вони повинні бути виконані та ким. Графік виконання тестування враховує такі фактори, як регресійне тестування, установлення пріоритетів, а також технічні і логічні залежності.

9.2 Категорії методів проектування тестів

Мета методу розробки тестів полягає в тому, щоб ідентифікувати умови тестування і тестові приклади (test cases).

Методи проектування тестів традиційно діляться на дві категорії: методи «чорної скриньки» і методи «білої скриньки». Методи «чорної скриньки» (які включають методи на основі специфікації і методи на основі досвіду) – спосіб одержати і вибрати умови тестування або тестові приклади, засновані на аналізі документації та досвіді розроблювачів, тестувальників і користувачів, для компонента або системи незалежно від її внутрішньої структури. Методи «білої скриньки» (називані також структурними) засновані на аналізі структури компонента або системи.

Деякі методи чітко падають в одну категорію; інші включають елементи різних категорій.

Загальні особливості методів на основі специфікації:

1) для специфікації розв’язуваної задачі, програмного засобу або його компонентів використовуються формальні або неформальні моделі;
2) тестові приклади одержують шляхом аналізу цих моделей.
3) Загальні особливості методів на основі структури:
4) інформація про те, як сконструйована програма, використовується для одержання тестових прикладів;
5) ступінь покриття програми може бути виміряна для існуючих тестових прикладів і в подальшому можуть бути отримані додаткові тести, щоб збільшити покриття.

Загальні особливості методів на основі досвіду:

знання і досвід людей використовуються, щоб одержати тестові приклади:

– знання тестувальників, розроблювачів, користувачів і інших зацікавлених сторін про програмний засіб, його використання і його середовище;

– знання про ймовірні помилки і їхній розподіл.

9.3 Методи «чорної скриньки»

9.3.1 Еквівалентне розбиття

При використанні методології «чорної скриньки» виявлення всіх помилок у програмі можливо тільки шляхом вичерпного вхідного тестування, тобто використання всіх можливих наборів вхідних даних. Нескладно показати, що повний перебір всіх можливих наборів значень вхідних даних не можна здійснити навіть для найпростіших програм, оскільки число всіх комбінацій набуває астрономічних значень.

Таким чином, побудова вичерпного вхідного тесту є неможливою. Отже, тестування програми обмежується використанням невеликої підмножини всіх можливих вхідних даних. Тоді, звичайно, хотілося б вибрати для тестування

найбільш підходящу підмножину. Правильно обраний тест цієї підмножини повинен мати певні властивості. По-перше, кожний тест повинен включати стільки різних вхідних умов, скільки це можливо, для того щоб мінімізувати загальне число необхідних тестів. По-друге, необхідно намагатися розбити вхідну область програми на кінцеве число класів еквівалентності так, щоб можна було припустити (з певним ступенем упевненості), що кожний тест, який є представником деякого класу, еквівалентний будь-якому іншому тесту цього класу. Іншими словами, якщо один тест класу еквівалентності виявляє помилку, то варто очікувати, що і всі інші тести цього класу еквівалентності будуть виявляти ту ж помилку. Навпаки, якщо тест не виявляє помилки, то варто очікувати, що жоден тест цього класу еквівалентності не буде виявляти помилки (у тому випадку, коли деяка підмножина класу еквівалентності не попадає в межі будь-якого іншого класу еквівалентності, тому що класи еквівалентності можуть перетинатися).

Ці два положення становлять основу методології тестування за принципом «чорної скриньки», відомої як еквівалентне розбиття. Друге положення використовується для розробки набору «цікавих» умов, які повинні бути протестовані, а перше – для розробки мінімального набору тестів, що покривають ці умови.

Розробка тестів методом еквівалентного розбиття здійснюється у два етапи:

1) виділення класів еквівалентності;
2) побудова тестів.

Класи еквівалентності виділяються шляхом вибору кожної вхідної умови (як правило, це речення або фраза в специфікації) і розбиттям його на дві або більше групи.

Розрізняють два типи класів еквівалентності: правильні класи еквівалентності, що представляють правильні вхідні дані програми, і неправильні класи еквівалентності, що представляють всі інші можливі стани умов (тобто помилкові вхідні значення).

Виділення класів еквівалентності являє собою в значній мірі евристичний процес. При цьому існує ряд правил:

1) якщо вхідна умова описує *область* значень (наприклад, «ціле дане може приймати значення від 1 до 999»), то визначаються один правильний клас еквівалентності (1≤значення цілого даного≤999) і два неправильних (значення цілого даного<1 і значення цілого даного>999);
2) якщо вхідна умова описує *число* значень (наприклад, «в автомобілі можуть їхати від одного до шести чоловік»), то визначається один правильний клас еквівалентності і два неправильних (жодного та більше шести чоловік);
3) якщо вхідна умова описує *множину* вхідних значень і є підстава думати, що кожне значення програма трактує особливо (наприклад, «відомі стилі плавання КРОЛЛЬ, БРАС, БАТЕРФЛЯЙ і НА СПИНІ»), то визначається

правильний клас еквівалентності для кожного значення і один неправильний клас еквівалентності (наприклад, «НА БОЦІ»);

4) якщо вхідна умова описує ситуацію «повинно бути» (наприклад, «першим символом ідентифікатора повинна бути буква»), то визначається один правильний клас еквівалентності (перший символ – буква) і один неправильний (перший символ – не буква);
5) якщо є будь-яка підстава вважати, що різні елементи класу еквівалентності трактуються програмою неоднаково, то даний клас еквівалентності розбивається на менші класи еквівалентності.

Другий крок полягає у використанні класів еквівалентності для побудови тестів. Цей процес містить у собі:

1) призначення кожному класу еквівалентності унікального номера;
2) проектування нових тестів, кожний з яких покриває як можна більше число непокритих правильних класів еквівалентності, доти поки всі правильні класи еквівалентності не будуть покриті тестами;
3) запис тестів, кожний з яких покриває один і тільки один з непокритих неправильних класів еквівалентності, доти поки всі неправильні класи еквівалентності не будуть покриті тестами.

Розбиття також може бути виконано не тільки для входів, але й для виходів, внутрішніх значень, пов’язаних з часом значень (наприклад, до або після події) і для параметрів інтерфейсу (наприклад, протягом інтеграційного тестування).

Еквівалентне розбиття може застосовуватися на всіх рівнях тестування.

9.3.2 Аналіз граничних значень

Одним з методів тестування програмних засобів з використанням методології чорної скриньки є аналіз граничних значень. Як показує досвід, тести, що досліджують граничні умови, приносять більшу користь, ніж тести, які їх не досліджують.

Граничні умови – це ситуації, що виникають безпосередньо на, вище або нижче границь вхідних і вихідних класів еквівалентності. Аналіз граничних значень відрізняється від еквівалентного розбиття у двох відношеннях:

1) вибір будь-якого елемента в класі еквівалентності в якості представницького при аналізі граничних значень здійснюється таким чином, щоб перевірити тестом кожну границю цього класу;

2) при розробці тестів розглядаються не тільки вхідні умови (простір входів), але й простір результатів (тобто вихідні класи еквівалентності).

Аналіз граничних значень значною мірою ґрунтується на здатностях людського інтелекту. Проте можна виділити кілька загальних правил для цього методу:

1) побудувати тести для границь області і тести з неправильними вхідними даними для ситуацій незначного виходу за межі області, якщо вхідна умова описує область значень. Приміром, якщо правильна область вхідних значень є [–1;+1], те написати тести для ситуацій: –1; 1; –1.001; 1.001;

2) побудувати тести для мінімального і максимального значень умов і тести, більших і менші цих значень, якщо вхідна умова задовольняє дискретному ряду значень. Наприклад, якщо вхідний файл може містити від 1 до 255 записів, то одержати тести для 0; 1; 255 і 256 записів;

3) використовувати правило «1» для кожної вихідної умови. Наприклад, якщо програма обчислює щомісячну витрати, мінімум яких 0.00 грн., а максимум 3000 грн., то побудувати тести, які викликають витрати з 0.00 грн. і 3000 грн. Крім того, побудувати, якщо це можливо, тести, які викликають від'ємні витрати і витрати більше 3000 грн. Важливо перевірити границі простору результатів, оскільки не завжди границі вхідних областей представляють такий же набір умов, як і границі вихідних областей (наприклад, при розгляді підпрограми обчислення синуса). Не завжди також можна одержати результат поза вихідною областю, але проте варто розглянути цю можливість;

4) використовувати правило «2» для кожної вихідної умови. Наприклад, якщо система пошуку відображає на екрані останні запитані реферати, але ніяк не більше чотирьох, то побудувати тести, такі, щоб програма відображала нуль, один і чотири реферати, і тест, що міг би викликати виконання програми з помилковим відображенням п'яти рефератів;

5) якщо вхід або вихід програми є впорядкована множина (наприклад, послідовний файл, лінійний список, таблиця), то зосередити увагу на першому й останньому елементах цієї множини;

6) спробувати свої сили в пошуку інших граничних умов.

Аналіз граничних значень може бути застосований на всіх рівнях тестування. Він відносно простий у застосуванні, має високу ймовірність виявлення помилок; корисні детальні специфікації.

Цю методику часто розглядають як розширення методу еквівалентного розбиття.

9.3.3 Тестування таблиці рішень (метод функціональних діаграм)

Функціональна діаграма – це текст на деякій формальній мові, на якому транслюється специфікація, складена на природній або напівформальній мовах. Діаграмі можна зіставити комбінаторну логічну мережу. Побудова тестів цим методом здійснюється в кілька етапів:

1) специфікація розбивається на «робочі» ділянки. Це пов'язано з тим, що функціональні діаграми стають занадто громіздкими при застосуванні даного методу до великих специфікацій;
2) в специфікації визначаються причини і наслідки. *Причина* є окремою вхідною умовою або класом еквівалентності вхідних умов. *Наслідок* є вихідною умовою або перетворенням системи (тобто остаточна дія програми, викликана певною вхідною умовою або їхньою комбінацією). Причини і наслідки визначаються шляхом послідовного читання специфікації. При цьому виділяються слова або фрази, які описують причини і наслідки. Кожній причині і наслідку приписується окремий номер;

3) аналізується семантичний зміст специфікації, яка перетворюється в булівський граф, що зв'язує причини і наслідки. Це і є функціональна діаграма;
4) діаграма забезпечується примітками, що задають обмеження і описують комбінації причин і (або) наслідків, які є неможливими через синтаксичні або зовнішні обмеження;
5) шляхом методичного простежування станів умов діаграми, функціональна діаграма перетвориться в таблицю рішень. Кожний стовпчик таблиці рішень відповідає тесту;
6) стовпчики таблиці рішень перетворюються в тести.

Базові символи для запису функціональних діаграм представлені на рисунку 9.1.

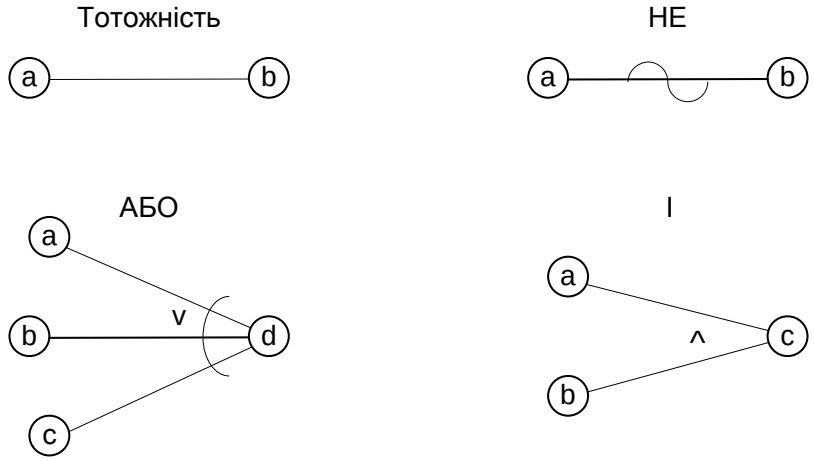

Рисунок 9.1 – Базові логічні відношення функціональних діаграм

Кожний вузол діаграми може перебувати у двох станах – 0 або 1; 0 позначає стан «відсутній», а 1 – «є присутнім». Функція ***тотожність*** установлює, що якщо значення *a* є 1, то й значення *b* є 1; у противному випадку значення *b* є 0. Функція ***не*** встановлює, що якщо значення *a* є 1, то значення *b* є 0. Функція ***або*** встановлює, що якщо значення *a,* або *в*, або *c* є 1, то й значення *d* є 1; у противному випадку значення *d* є 0. Функція ***і*** установлює, що якщо значення *a* і *b* є 1, то й значення *c* є 1; у противному випадку значення *c* є 0. Останні дві функції дозволяють мати будь-яке число входів.

Для ілюстрації розглянемо діаграму, що відображає специфікацію:

Перший символ команди повинен бути буквою «А» або «В», а другий – цифрою. У цьому випадку файл обновляється. Якщо перший символ неправильний, то видається повідомлення X12, а якщо другий символ неправильний – повідомлення X13.

Причинами є	а наслідками
1 – перший символ є буквою «А»;	70 – файл обновляється;
2 – перший символ є буквою «В»;	71 – видається повідомлення X12;
3 – другий символ є цифрою,	72 – видається повідомлення X13.

Функціональна діаграма представлена на рисунку 9.2.

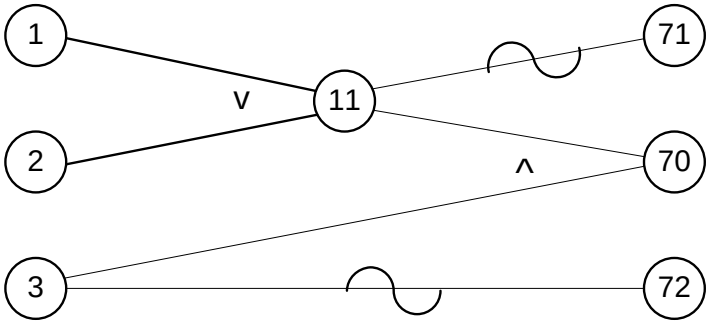

Рисунок 9.2 – Приклад функціональної діаграми

У більшості програм певні комбінації причин неможливі через синтаксичні або зовнішні обмеження (наприклад, символ не може приймати значення «А» і «В» одночасно). У цьому випадку використовуються додаткові логічні обмеження, зображені на рис. 9.3.

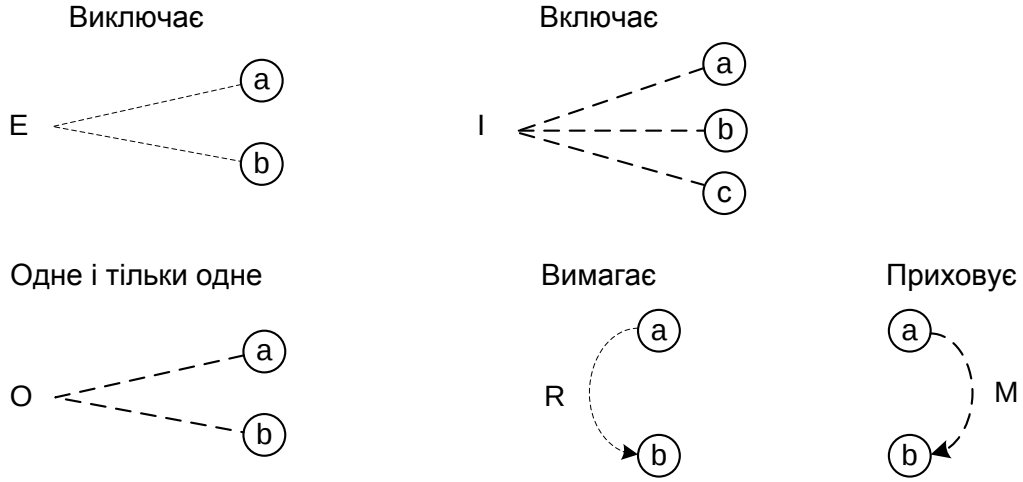

Рисунок 9.3 – Символи обмежень

Обмеження E встановлює, що причини не можуть набувати значення 1 одночасно; *обмеження I* встановлює, що причини не можуть набувати значення 0 одночасно; *обмеження O* встановлює, що одна й тільки одна із причин повинна набувати значення 1; *обмеження R* встановлює, що якщо *a* набуває значення 1, то й *b* повинна набувати значення 1. Часто виникає необхідність в обмеженнях для наслідків. *Обмеження M* встановлює, що якщо наслідок *a* має значення 1, то наслідок *b* повинен мати значення 0.

Найбільш трудомістким етапом є перетворення отриманої діаграми в таблицю рішень. Процедура генерації полягає в наступному:

– вибрати деякий наслідок, що повинен бути в стані 1;

– знайти всі комбінації причин (з урахуванням обмежень), які встановлять цей наслідок в 1, прокладаючи із цього наслідку зворотну трасу через діаграму;

– побудувати стовпчик у таблиці рішень для кожної комбінації причин;

– для кожної комбінації причин визначити стани всіх інших наслідків і помістити їх у відповідний стовпчик таблиці рішень.

При виконанні цього кроку необхідно керуватися трьома положеннями:

1) якщо зворотна траса прокладається через вузол ***або***, вихід якого повинен мати значення 1, то одночасно не слід установлювати в 1 більше одного входу в цей вузол. Таке обмеження на установку вхідних значень називається *чутливістю шляху.* Мета даного правила – уникнути пропуску помилок через те, що одна причина маскується іншою;
2) якщо зворотна траса прокладається через вузол ***i***, вихід якого повинен мати значення 0, то всі комбінації входів, що приводять вихід в 0, повинні бути в остаточному підсумку перераховані. Однак, коли досліджується ситуація, де один вхід є 0, а один або більше інших входів є 1, не обов’язково перераховувати всі умови, при яких інші входи можуть бути 1;
3) якщо зворотна траса прокладається через вузол ***i***, вихід якого повинен мати значення 1, то необхідно вказати лише одну умову, відповідно до якої всі входи є нулями. (Коли вузол ***i*** перебуває в середині графа, і його входи виходять із інших проміжних вузлів, може існувати надзвичайно велика кількість ситуацій, при яких всі його входи набувають значення 0).

Таблиця рішень					Тести
1	1	0	0	1	A3
2	0	1	0	0	B7
3	1	1	1	0	E2
70	1	1	0	0	AD
71	0	0	1	0	
72	0	0	0	1	

Застосування функціональних діаграм – систематичний метод генерації тестів, що представляють комбінації умов. Альтернативою є спеціальний вибір комбінацій, але при цьому існує ймовірність пропуску багатьох «цікавих» тестів, визначених за допомогою функціональної діаграми.

Розробка функціональних діаграм є гарним способом виявлення неповноти і неоднозначності у вихідних специфікаціях. Цей метод дозволяє побудувати набір «корисних» тестів.

9.3.4 Тестування переходів зі стану в стан

Система може видавати різну відповідь залежно від поточних умов або передісторії (її стану). У цьому випадку розглянутий аспект системи можна показати як діаграму переходів. Це дозволяє тестувальнику розглядати програму в термінах її станів, переходів між станами, входів або подій, які викликають зміну стану (переходів) і дій, які можуть бути результатом цих переходів. Стани системи або об'єкта при тестуванні є поділюваними, ідентифікованими і число їх кінцеве. Таблиця станів показує відношення між станами і входами, і може виділяти можливі переходи, які є неправильними. Можуть бути розроблені тести, що покривають типову послідовність станів, кожний стан, що дозволяють здійснювати кожний перехід, здійснювати певні послідовності переходів або перевіряти недійсні переходи.

Тестування переходів зі стану в стан часто використовується в межах індустрії вбудованого ПЗ і технічній автоматизації взагалі. Однак цей метод підходить також для моделювання бізнес-об'єктів, що мають певні стани, або тестування потоку екранного діалогу (наприклад, для internet-додатків або бізнесів-сценаріїв).

9.3.5 Тестування сценаріїв

Тести можуть розроблятися для перевірки сценаріїв. Сценарій описує взаємодію між суб'єктами, якими є користувачі і програмна система.

Кожний сценарій має попередні умови, які необхідні для його успішної реалізації. Кожний сценарій закінчується з вихідними умовами, які є спостережуваними результатами і кінцевим станом системи після того, як сценарій був закінчений. Сценарій звичайно має основну тенденцію (тобто найбільш імовірну), і іноді альтернативні гілки.

Сценарії описують "технологічний процес" системи на підставі її фактичного ймовірного використання, так що тестові приклади, отримані зі сценаріїв, найбільш корисні в розкритті дефектів технологічного процесу протягом реального використання системи. Сценарії є дуже корисними для того, щоб розробити приймальні тести за участю клієнта/ користувача. Вони також допомагають розкривати дефекти інтегрування, викликані взаємодією і взаємним впливом різних компонентів, які індивідуальне тестування компонента не виявило б.

9.4 Методи «білої скриньки»

Однією з основних стратегій тестування програмних засобів є стратегія «білої скриньки», називана ще стратегією тестування, керованого логікою програми. Тестування методами «білої скриньки» засновано на ідентифікованій структурі програмного забезпечення або системи:

- рівень компонентів: структура – це безпосередньо код, тобто оператори, рішення або гілки;

– рівень інтегрування: структурою може бути дерево викликів (діаграма, у якій модулі викликають інші модулі);
– системний рівень: структурою може бути структура меню, бізнес-процес або структура web-сторінки.

При використанні цієї стратегії тестувальник одержує тестові дані шляхом аналізу логіки програми (на жаль, тут часто не використовується специфікація програми). При такому підході виявлення всіх помилок у програмі можливе тільки шляхом вичерпного тестування маршрутів. Мається на увазі, що програма перевірена повністю, якщо за допомогою тестів вдається здійснити виконання цієї програми по всіх можливих маршрутах її потоку (графа) передач керування. Нескладно показати, що навіть для найпростіших програм число маршрутів, які не повторюють один одного, набуває астрономічних значень. Крім того, вичерпне тестування маршрутів не гарантує відсутності помилок у програмі (наприклад, невідповідність програми опису, відсутність деяких маршрутів в алгоритмі, наявність помилок, що залежать від вхідних даних і т.п.).

Тестування за принципом білої скриньки характеризується ступенем, у який тести виконують або покривають логікові (вихідний текст) програми. Якщо відмовитися повністю від тестування всіх шляхів, то можна показати, що критерієм покриття є виконання кожного оператора програми принаймні один раз (*покриття операторів*). На жаль, це досить слабкий критерій, тому що виконання кожного оператора принаймні один раз є необхідною, але недостатньою умовою для прийнятного тестування за принципом білої скриньки. Розглянемо невелику програму, схема алгоритму якої наведена на рисунку 9.4.

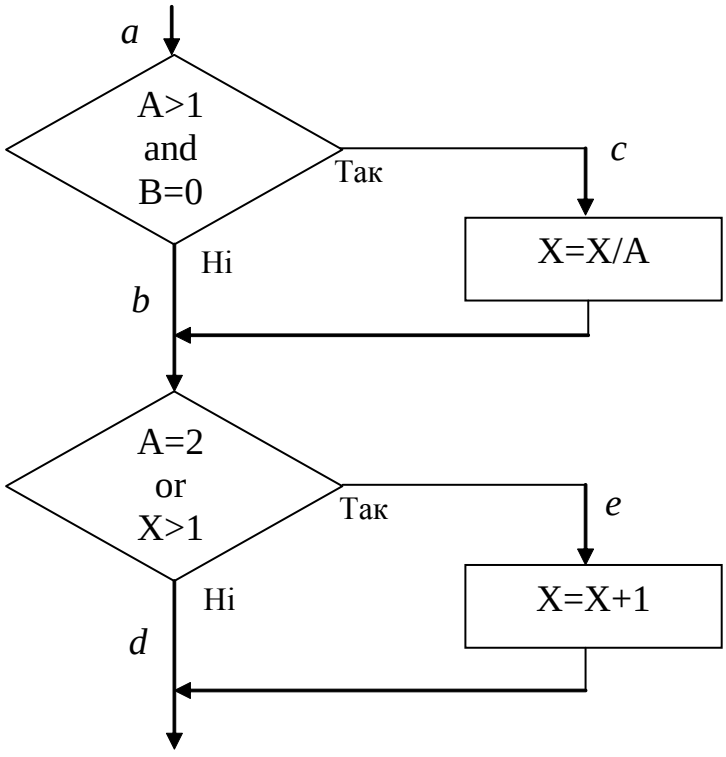

Рисунок 9.4 – Схема алгоритму програми

Можна виконати кожний оператор, записавши один-єдиний тест, що реалізував би шлях *ace*. Іншими словами, якби в точці *a* були встановлені значення $A=2$, $B=0$ і $X=3$, кожний оператор виконувався б один раз (насправді X може мати будь-яке значення).

На жаль, цей критерій гірше, ніж він здається на перший погляд. Наприклад, нехай перше рішення записане як *або*, а не як *і*. При тестуванні за даним критерієм ця помилка не буде виявлена. Якщо друге рішення записане в програмі як $X>0$, то ця помилка також не буде виявлена. Крім того, існує шлях, у якому X не змінюється (шлях *abd*). Якщо тут є помилка, то й вона не буде виявлена. Таким чином, критерій покриття операторів є настільки слабким, що його звичайно не використовують.

Більш сильний критерій покриття логіки програми відомий як *покриття рішень* або *покриття переходів*. Відповідно до даного критерію повинне бути записане достатнє число тестів, таке, що кожне рішення на цих тестах набуде значення *істина* й *хибність* принаймні один раз. Іншими словами, кожний напрямок переходу повинен бути реалізований принаймні один раз. Прикладами операторів переходу є оператори IF, WHILE, SWITCH.

Покриття рішень зазвичай задовольняє покриттю операторів. Оскільки кожний оператор лежить на деякому шляху, що виходить або з оператора переходу, або із точки входу програми, при виконанні кожного напрямку переходу кожний оператор повинен бути виконаний. Однак існують деякі виключення. Перше – патологічна ситуація, коли програма не має рішень. Друге зустрічається в програмах або підпрограмах з декількома точками входу; даний оператор може бути виконаний тільки в тому випадку, якщо виконання програми починається з відповідної точки входу. Оскільки покриття операторів вважається необхідною умовою, покриття рішень, що представляється більше сильним критерієм, повинне включати покриття операторів. Отже, покриття рішень вимагає, щоб кожне рішення мало як результат значення *істина* й *хибність* і при цьому кожний оператор виконувався б принаймні один раз.

У програмі, схема алгоритму якої наведена на рис. 8.4, покриття рішень може бути виконано двома тестами, що покривають або шляхи *ace* і *abd*, або шляхи *acd* і *abe*. Якщо ми вибираємо останнє альтернативне покриття, то входам двох тестів є $A=3$, $B=0$, $X=3$ і $A=2$, $B=1$, $X=1$.

Покриття рішень – більш сильний критерій, ніж покриття операторів, але й він має свої недоліки. Наприклад, якщо в другому рішенні існує помилка ($X<1$ замість $X>1$), те вона не буде виявлена цими двома тестами.

Кращим критерієм у порівнянні з попереднім є *покриття умов*. У цьому випадку записують число тестів, достатнє для того, щоб всі можливі результати кожної умови в рішенні виконувалися принаймні один раз. Оскільки, як і при покритті рішень, це покриття не завжди приводить до виконання кожного оператора, до критерію потрібне доповнення, яке полягає в тому, що кожній точці входу в програму або підпрограму повинне бути передане керування при виклику принаймні один раз.

Розглянута програма має чотири умови: $A>1$, $B=0$, $A=2$ і $X>1$. Отже, потрібне достатнє число тестів, таке, щоб реалізувати ситуації, де $A>1$, $A\leq 1$, $B=0$ і

$B \neq 0$ у точці *a* і $A=2$, $A \neq 2$, $X>1$ і $X \leq 1$ у точці *b*. Тести, що задовольняють критерію покриття умов, і відповідні їм шляхи:

$A=2$, $B=0$, $X=4$ *ace*;
$A=1$, $B=1$, $X=1$ *abd*.

Хоча аналогічне число тестів для даного приклада вже було створено, покриття умов звичайно краще покриття рішень, оскільки воно може (але не завжди) викликати виконання рішень в умовах, не реалізованих при покритті рішень.

Критерій покриття умов не завжди задовольняє критерію покриття рішень. Якщо тестується рішення IF(A&B), то при критерії покриття умов були потрібні б два тести – A є *істина*, B є *хибність* і A є *хибність*, B є *істина*. Але в цьому випадку не виконувалася б THEN-гілка оператора IF. Тести критерію покриття умов для розглянутого приклада (рис. 8.4) покривають результати всіх рішень, але цей тільки випадковий збіг. Наприклад, два альтернативних тести

$A=1$, $B=0$, $X=3$,
$A=2$, $B=1$, $X=1$

покривають результати всіх умов, але тільки два із чотирьох результатів рішень (вони обоє покривають шлях *abe* і, отже, не виконують результат *істина* першого рішення і результат *хибність* другого рішення).

Очевидні наслідком із цієї дилеми є критерій, названий *покриттям рішень/умов*. Він вимагає такого достатнього набору тестів, щоб всі можливі результати кожної умови в рішенні виконувалися принаймні один раз, всі результати кожного рішення виконувалися принаймні один раз і кожній точці входу передавалося керування принаймні один раз.

Недоліком критерію покриття рішень/умов є неможливість його застосування для виконання всіх результатів всіх умов; часто подібне виконання має місце внаслідок того, що результати умов у виразах *і* та *або* можуть приховувати і блокувати дію інших умов. Наприклад, якщо умова *і* є *хибність*, то ніяка з наступних умов у виразі не буде виконана. Аналогічно якщо умова *або* є *істина*, то ніяка з наступних умов не буде виконана. Отже, критерії покриття умов і покриття рішень/умов недостатньо чутливі до помилок у логічних виразах.

Критерієм, що вирішує ці й деякі інші проблеми, є *комбінаторне покриття умов*. Він вимагає створення такого числа тестів, щоб всі можливі комбінації результатів умови в кожному рішенні і всі точки входу виконувалися принаймні один раз. Легко бачити, що набір тестів, який задовольняє критерію комбінаторного покриття умов, задовольняє також і критеріям покриття рішень, покриття умов і покриття рішень/умов.

Відповідно до цього критерію, у розглянутому прикладі повинні бути покриті тестами наступні вісім комбінацій:

1. $A>1$, $B=0.5$.
2. $A>1$, $B \neq 0.6$.
3. $A \leq 1$, $B=0.7$.
4. $A \leq 1$, $B \neq 0.8$.
5. $A=2$, $X>1$.
6. $A=2$, $X \leq 1$.
7. $A \neq 2$, $X>1$.
8. $A \neq 2$, $X \leq 1$.

Помітимо, що комбінації 5–8 є значеннями другого оператора IF. Оскільки *X* може бути змінене до виконання цього оператора, значення, необхідні для його перевірки, варто відновити виходячи з логіки програми для того, щоб знайти відповідні вхідні значення.

Для того, щоб протестувати ці комбінації, необов'язково використовувати всі вісім тестів. Фактично вони можуть бути покриті чотирма тестами:

A=2, *B*=0, *X*=4	покриває 1, 5;
A=2, *B*=1, *X*=1	покриває 2, 6;
A=1, *B*=0, *X*=2	покриває 3, 7;
A=1, *B*=1, *X*=1	покриває 4, 8.

Насправді представлені вище тести не покривають всіх шляхів, вони пропускають шлях *acd*.

Таким чином, для програм, що містять тільки одну умову на кожне рішення, мінімальним є критерій, набір тестів якого

1) викликає виконання всіх результатів кожного рішення принаймні один раз;
2) передає керування кожній точці входу принаймні один раз.

Для програм, що містять рішення, кожне з яких має більше однієї умови, мінімальний критерій складається з набору тестів, що викликають виконання всіх можливих комбінацій результатів умов у кожному рішенні і передають керуванні кожній точці входу програми принаймні один раз.

9.5 Методи тестування на основі досвіду

У тестуванні на основі досвіду тести створюються виходячи з навичок і інтуїції тестувальника і його досвіду роботи з подібними програмами і технологіями. Ці методи можуть бути корисні для ідентифікації спеціальних тестів, які складно одержати формальними методами. Однак, ступінь ефективності цієї методики може варіюватися в широких межах, залежно від досвіду тестувальників.

Найпоширеніша методика тестування на основі досвіду – припущення про помилку. Процедуру для методу припущення про помилку описати важко, тому що цей метод у значній мірі є інтуїтивним. Основна ідея його полягає в тім, щоб перелічити в деякому списку можливі помилки або ситуації, у яких вони можуть з'явитися, а потім на основі цього списку написати тести. Ці списки помилок можуть бути сформовані на підставі досвіду, доступних даних про дефекти і помилки та загальноприйнятих знань про помилки в програмному забезпеченні.

Тестування на основі досвіду представляє підхід, що є найбільш корисним у тому випадку, коли специфікації неповні або неадекватні, підтискають строки, або щоб збільшити ефективність або доповнити інший, більш формальний метод тестування.

9.6 Вибір методів тестування

Вибір використовуваного методу тестування залежить від ряду факторів, включаючи тип системи, регулюючі стандарти, вимоги споживача або контракту, рівень ризику, тип ризику, мету тестування, доступну документацію, знання тестувальників, час і бюджет, життєвий цикл, моделі сценарію та попередній досвід знаходження помилок.

Жоден з розглянутих методів тестування не може забезпечити повний набір тестів, тому що орієнтований на виявлення певного типу помилок. Прийнятна стратегія тестування полягає в спільному використанні методів білої та чорної скриньки.

9.7 Контрольні запитання і завдання

1. Які існують категорії методів проектування тестів?
2. Чим відрізняється тестування чорної і білої скриньки?
3. У чому суть методу еквівалентного розбиття?
4. Як здійснюється тестування програм шляхом аналізу граничних значень?
5. Викладіть суть методу функціональних діаграм.
6. Як відбувається тестування сценаріїв?
7. Які існують критерії покриття логіки програми?
8. У чому полягає підхід до тестування на основі досвіду?

Лекція 10
ІНТЕГРАЦІЙНЕ ТЕСТУВАННЯ

Розглядаючи проблеми тестування програмних засобів, необхідно враховувати такі фактори, як організація процедури тестування та розмір програм, що тестуються. Перехід до великих програм (500 операторів і більше) вимагає спеціальних способів структурування процесу тестування.

Перш ніж починати тестування програми в цілому, необхідно протестувати окремі невеликі модулі, що складають цю програму. Такий підхід мотивується трьома причинами. По-перше, з\являється можливість управляти комбінаторикою тестування, оскільки спочатку увага концентрується на невеликих модулях програми. По-друге, полегшується задача налагодження програми, тобто виявлення місця помилки і виправлення тексту програми. Нарешті, по-третє, допускається паралелізм, що дозволяє одночасно тестувати кілька модулів.

Мета інтеграційного тестування – віднайти помилки в інтерфейсах і міжмодульних зв'язках. Процес тестування інтеграції розглядається в трьох аспектах: способи побудови наборів тестів; порядок, у якому модулі тестуються і збираються в систему; деякі практичні рекомендації з реалізації тестування.

10.1 Покрокове і монолітне тестування

Реалізація процесу інтеграційного тестування опирається на два ключових положення: побудова ефективного набору тестів і вибір способу, за допомогою якого модулі комбінуються при побудові з них робочої програми. Існує два способи комбінування модулів у програму. Перший з них, називаний *монолітним методом* тестування або методом «великого удару», полягає в автономному тестуванні кожного модуля з наступним формуванням робочої програми. При *покроковому методі* тестування кожний модуль для тестування підключається до набору раніше протестованих модулів. Процедура покрокового тестування може здійснюватися двома способами – згори вниз (*низхідне* тестування) і знизу нагору (*висхідне* тестування).

Для тестування будь-якого модуля потрібні спеціальний модуль-*драйвер*, що імітує функції модуля, який викликає даний модуль, і один або кілька модулів-*заглушок*, що імітують функції модулів, які викликаються даним модулем.

Деякі узагальнення для методів комбінування модулів:

1) монолітне тестування вимагає більших затрат праці на створення драйверів і заглушок, ніж покрокове. Пояснюється це тим, що при тестуванні згори вниз модулі, що тестуються, виконують функції драйверів, а при тестуванні знизу нагору – функції заглушок;

2) при покроковому тестуванні раніше виявляються помилки в інтерфейсах між модулями, оскільки раніше починається складання програми. На противагу цьому при монолітному тестуванні модулі «не бачать один одного» до останньої фази процесу тестування;

3) налагодження програм при покроковому тестуванні легше. Якщо є помилки в міжмодульних інтерфейсах, то при монолітному тестуванні вони можуть бути виявлені лише тоді, коли зібрана вся програма. У цей момент локалізувати помилку досить важко, оскільки вона може перебувати в будь-якому місці програми. Навпаки, при покроковому тестуванні помилки такого типу в основному пов’язані з тим модулем, що підключається останнім;

4) результати покрокового тестування більш досконалі, оскільки при спільному тестуванні модулів можливе створення таких ситуацій, які не враховувалися при їхньому автономному тестуванні. При монолітному тестуванні модуля результати обмежуються тільки цим модулем;

5) при монолітному тестуванні менше витрати машинного часу, ніж при покроковому;

6) використання монолітного метода надає більші можливості по паралельній організації роботи на початковій фазі тестування. Це може мати важливе значення при виконанні великих проектів, в яких багато модулів і багато виконавців.

Зазначені характеристики монолітного і покрокового тестування дозволяють зробити висновок про те, що метод покрокового тестування є кращим.

10.2 Низхідне і висхідне тестування

Низхідне тестування починається з верхнього, головного модуля програми. Строгої, коректної процедури підключення чергового модуля, що послідовно тестується, не існує. Єдине правило, яким варто керуватися при виборі чергового модуля, полягає в тому, що ним повинен бути один з модулів, який викликається модулем, що вже пройшов тестування.

На рис. 10.1 зображений приклад програми, що ілюструє даний метод.

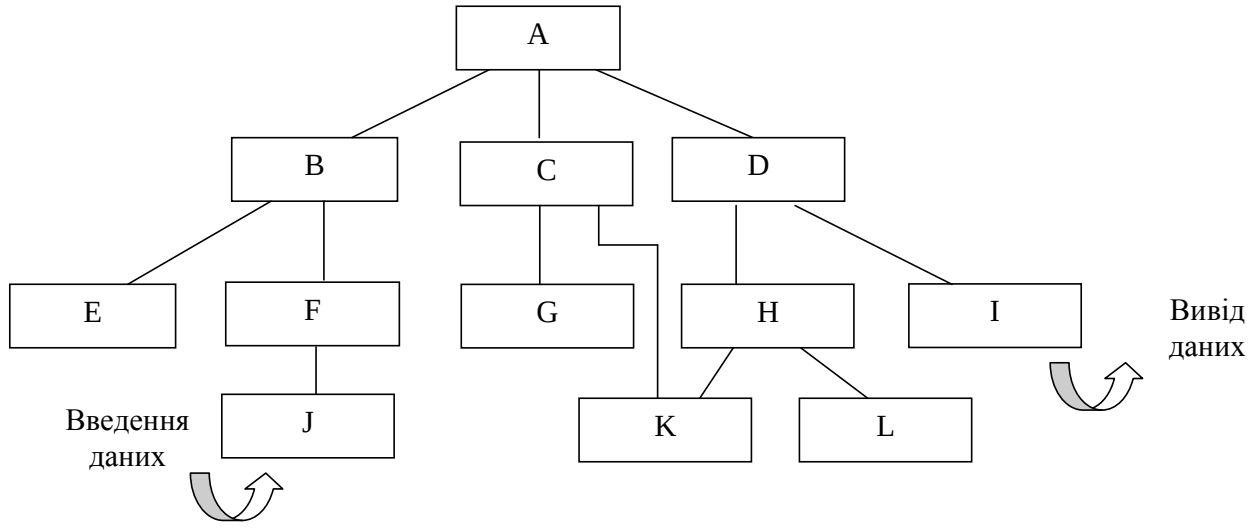

Рисунок 10.1 – Ілюстрація методу низхідного тестування

Спочатку тестується модуль A (основна програма), а замість модулів B, C і D використовуються заглушки. Потім заглушка B заміняється модулем, який у свою чергу використовує заглушки E і F (рис. 10.2).

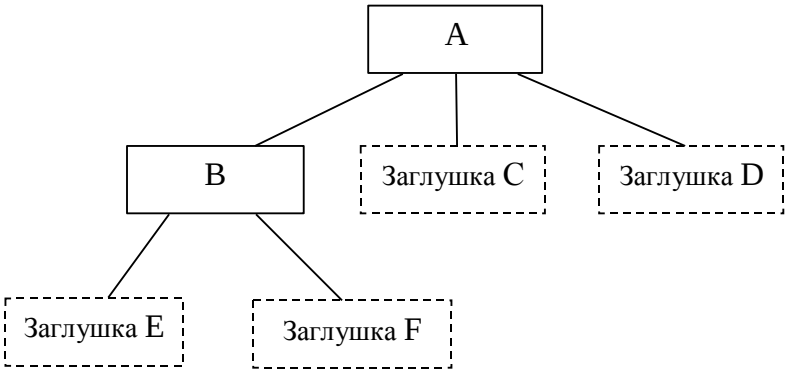

Рисунок 10.2 – Проміжний стан при низхідному тестуванні

Заглушки призначені для виконання наступних дій:

1) імітація функцій модулів, що заміщаються;

2) передача тестових даних модулям, які викликають заглушку;

3) вивід результатів тестування.

У загальному випадку створення модулів-заглушок є нетривіальною задачею.

Якщо послідовно тестуються всі модулі, то можливі такі варіанти:

A B C D E F G H I J K L

A B E F J C G K D H L I

A D H I K L C G B F J E

Послідовність, у якій тестуються модулі, може бути досить довільною, але рекомендується дотримуватися двох основних правил:

1) якщо в програмі є критичні в якому-небудь розумінні частини, то доцільно вибирати послідовність, що включала б ці частини якомога раніше. Критичним може бути складний модуль або модуль з великим числом передбачуваних помилок;

2) модулі, що включають операції вводу-виводу, також необхідно підключати в послідовність тестування якомога раніше. Це дозволить зменшити обсяг робіт з написання заглушок і зняти з них функції введення тестових даних і виведення результатів.

Низхідне тестування має ряд серйозних недоліків. Нехай стан програми, що перевіряється, відповідає показаному на рис. 10.3. На наступному кроці потрібно замінити заглушку самим модулем H. Для тестування цього модуля потрібно спроектувати набір тестів, які представлені у вигляді реальних даних, що вводяться через модуль J. При цьому виникають три проблеми:

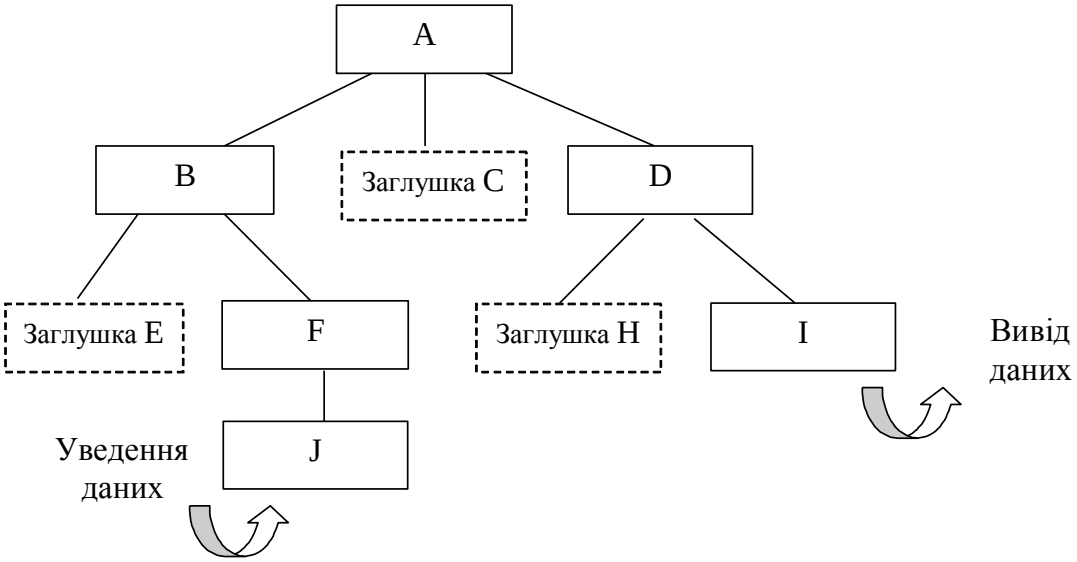

Рисунок 10.3 – Проміжний стан при низхідному тестуванні

1) між модулями H і J є проміжні модулі (F, B, A і D), тому може виявитися неможливим передати модулю такий тест, який би відповідав кожній попередньо описаній ситуації на вході модуля H;

2) навіть якщо є можливість передати всі тести, то через «дистанцію» між модулем H і точкою уведення в програму виникає досить важке інтелектуальне завдання – оцінити, якими повинні бути дані на вході модуля J, щоб вони відповідали необхідним тестам модуля H;

3) результати виконання тесту демонструються модулем, розташованим досить далеко від модуля, який тестується в цей момент. Отже, установлення відповідності між тим, що демонструється, і тим, що відбувається в модулі насправді, досить складно, а іноді просто неможливо.

Висхідне тестування в багатьох відношеннях протилежне низхідному. Дана стратегія починає тестування з термінальних модулів (тобто модулів, які не викликають інші модулі). Для кожного модуля потрібен свій модуль-драйвер, тобто модуль, що містить фіксовані тестові дані, викликає модуль, що тестується, і відображає вихідні результати. Драйвери простіше розробляти, ніж заглушки. При виборі послідовності підключення модулів керуються тими ж правилами, що й у методі низхідного тестування.

Недолік розглянутої стратегії полягає у тому, що концепція побудови структури робочої програми на ранній стадії тестування відсутня. З іншого боку, тут відсутні проблеми, пов’язані з неможливістю або труднощами створення всіх тестових ситуацій, характерні для низхідного тестування.

10.3 Контрольні запитання і завдання

1. Яка мета інтеграційного тестування?
2. Що таке покрокове і монолітне тестування? Які переваги і недоліки має кожен з цих методів?
3. Що таке низхідне і висхідне тестування?
4. Які функції виконують заглушки при низхідному методі тестування?
5. У якій послідовності повинні підключатися модулі при низхідному тестуванні?
6. Які переваги та недоліки має низхідне тестування?
7. Які функції виконують драйвери при висхідному методі тестування?
8. Які переваги та недоліки має висхідне тестування?

Лекція 11
УПРАВЛІННЯ ТЕСТУВАННЯМ

11.1 Організація тестування

11.1.1 Організація тестування і незалежність

Ефективність виявлення помилок шляхом тестування і переглядів може бути підвищена, використовуючи незалежних тестувальників. Варіанти забезпечення незалежності:

1) ніяких незалежних тестувальників. Розроблювачі перевіряють свій власний код;

2) незалежні тестувальники в складі команди розроблювачів;

3) незалежна група тестувальників у межах організації, підзвітна керівництву проектом;

4) незалежні тестувальники з бізнес-організації або співтовариства користувачів;

5) незалежні фахівці з тестування для певних цілей тестування (наприклад, тестувальники зручності використання, тестувальники захищеності або тестувальники-сертифікатори);

6) незалежні тестувальники із зовнішньої організації.

Для великих, складних або критичних з погляду безпеки проектів, звичайно найкраще мати численні рівні тестування, з деякими або всіма рівнями, зробленими незалежними тестувальниками. Розроблювачі можуть брати участь у тестуванні, особливо на більше низьких рівнях, але брак об'єктивності часто обмежує їхню ефективність. Незалежні тестувальники можуть мати повноваження, що дозволяють їм визначати процеси і правила тестування, але тестувальники повинні брати на себе таку роль тільки при наявності чіткого розпорядження керівництва.

Вигоди від незалежності включають:

1) незалежні тестувальники бачать різні помилки, яких можуть не бачити розроблювачі, вони неупереджені;

2) незалежний випробувач може перевірити припущення людей, зроблені в ході складання специфікації і виконання програми;

Недоліки включають:

1) ізоляція від групи розробки (якщо тестування є повністю незалежним);

2) незалежні тестувальники можуть виявитися критичним параметром як остання контрольна точка;

3) розроблювачі можуть втратити відчуття відповідальності за якість.

Завдання тестування можуть бути виконані людьми, що відіграють певні ролі в процесі –менеджера проекту, менеджера з якості, розроблювача, бізнес-експерта, фахівця із проблемної області, інфраструктури або IT-технологій.

11.1.2 Задачі керівника тестування і тестувальника

Дії і завдання, виконувані керівником тестування і тестувальником, залежать від проекту і контексту програми, людей, які виконують дані ролі, і організації.

Іноді керівника тестування називають тест-менеджером або тест-координатором. Роль керівника тестування може бути виконана керівником проекту, керівником розробки, менеджером із забезпечення якості або менеджером групи тестування. Зазвичай тест-менеджер виконує функції планування, моніторингу і управління тестовими роботами й завданнями.

Типові завдання тест-менеджера можуть включати:

1) координування стратегії і плану тестування з керівництвом проекту;

2) запис або перегляд стратегії тестування для проекту, і політики тестування для організації;

3) сприяння перспективи тестування іншим проектним роботам, наприклад плануванню інтеграції;

4) планування тестів – розгляд змісту і розуміння цілей тестування і ризиків – включаючи обрані підходи, оцінку часу, зусиль і вартості тестування, придбання ресурсів, визначення рівнів тестування, циклів і планування контролю подій;

5) ініціалізація, підготовка і виконання тестів, моніторинг результатів тестування і перевірка критеріїв завершення;

6) адаптація планування, заснована на результатах тестування і досягнутому прогресі, виконання будь-яких дій, необхідних для усунення проблем;

7) встановлення адекватного управління конфігурацією тестування для забезпечення трасованості;

8) введення придатних метрик для того, щоб виміряти прогрес тестування і оцінити якість тестування та продукту;

9) прийняття рішення про те, що повинне бути автоматизоване, до якого ступеня, і яким чином;

10) вибір інструментальних засобів підтримки тестування, організація навчання тестувальників використанню інструментальних засобів;

11) прийняття рішення про реалізацію середовища тестування;

12) складання підсумкових звітів про тестування на підставі інформації, зібраної протягом тестування.

Типові завдання тестувальника можуть включати:

1) перегляд і вдосконалення планів тестування;

2) аналіз, перегляд і оцінка користувальницьких вимог, специфікацій і моделей тестованості;

3) створення специфікацій тестів;

4) встановлення середовища тестування (часто координується із системним адмініструванням і керуванням мережею);

5) підготовка і одержання тестових даних;

6) виконання тестів на всіх рівнях тестування, реєстрація ходу тестування, оцінювання результатів і документування відхилень від очікуваних результатів;

7) використання інструментальних засобів адміністрування або управління тестуванням і засобів моніторингу тестування при необхідності;

8) автоматизація тестування (може підтримуватися розроблювачем або експертом з автоматизованого тестування);

9) вимірювання продуктивності компонентів і систем (якщо це необхідно);

10) перегляд тестів, розроблених іншими тестувальниками.

Люди, які працюють над аналізом тестів, проектуванням тестів, визначенням типу тестів або автоматизацією тестування, можуть бути фахівцями в цих областях. Залежно від рівня тестування і ризиків, пов’язаних із продуктом і проектом, різні люди можуть відігравати роль тестувальника, зберігаючи деякий ступінь незалежності.

Зазвичай тестувальниками на рівні тестування компонентів і рівні інтеграційного тестування є розроблювачі, тестувальниками на рівні приймальних випробувань є бізнес-експерти і користувачі, і тестувальниками на рівні дослідної експлуатації є оператори.

11.2 Планування тестування і оцінка

11.2.1 Планування тестування

Планування може бути задокументоване в головному плані тестування і в окремих планах для різних рівнів тестування, таких як системне тестування і приймальні випробування. Основні документи по плануванню тестування освітлені в стандарті IEEE 829 “Standard for Software Test Documentation”.

На планування впливає політика тестування організації, область тестування, цілі, ризики, обмеження, критичність, тестованість і доступність ресурсів. Планування тестування – безперервна діяльність і виконується у всіх процесах життєвого циклу. Зворотний зв’язок від дій по тестуванню використовується, щоб розпізнати ризики, що змінюються, і відкоригувати план.

11.2.2 Роботи із планування тестування

Роботи із планування тестування можуть включати:

1) визначення масштабу і ризиків, ідентифікацію цілей тестування;

2) визначення загального підходу до тестування (стратегії тестування), включаючи визначення рівнів тестування і критеріїв входу та виходу;

3) інтегрування та координування робіт з тестування в процеси життєвого циклу програми: замовлення, поставка, розробка, експлуатація і супровід;

4) ухвалення рішення про те, що тестувати, як будуть розподілені ролі в діях по тестуванню, як повинні бути виконані роботи з тестування і як будуть оцінюватися результати тестування;

5) планування аналізу тестів і розробка дій;

6) планування реалізації, виконання і оцінка тестування;

7) призначення ресурсів для різних дій;

8) визначення кількості, рівня деталізації, структури і шаблонів для документації по тестуванню;

9) вибір метрик для моніторингу і керування підготовкою і виконанням тестів, розв’язання проблем і ризиків;

10) встановлення рівня деталізації для тестових процедур, щоб забезпечити достатній обсяг інформації для підтримки підготовки і виконання тестів.

11.2.3 Критерії завершення

Мета критеріїв завершення полягає в тому, щоб визначити, коли припинити тестування (наприклад наприкінці рівня тестування) або коли набір тестів досягає певної мети (наприклад, забезпечує покриття умов і переходів).

Зазвичай критерії завершення можуть включати:

1) оцінки завершеності, такі як покриття коду, функціональність або ризик;

2) оцінки щільності помилок або показників надійності;

3) вартість;

4) залишкові ризики, такі як невиправлені помилки або брак тестового покриття в деяких областях;

5) графіки, засновані на часі продажів.

11.2.4 Оцінювання тестування

Існують два підходи для оцінки витрат на тестування:

1) підхід, заснований на використанні метрик: оцінка витрат на тестування заснована на метриках минулих або подібних проектів або заснована на типових значеннях;

2) експертний підхід: оцінка задач власником цих задач або експертами.

Як тільки витрати на тестування оцінені, можуть бути ідентифіковані ресурси і складений графік робіт.

Витрати на тестування можуть залежати від ряду факторів, включаючи:

1) характеристики продукту: якість специфікації та іншої інформації, використовуваної для моделей тестування, розмір програми, складність предметної області, вимоги до надійності і захищеності і вимоги до документації;

2) характеристики процесу розробки: стабільність організації, використовувані інструментальні засоби, процес тестування, навички залучених людей і брак часу;

3) результат тестування: кількість помилок і кількість необхідних виправлень.

11.2.5 Підходи до тестування (стратегії тестування)

Один з способів класифікувати стратегії тестування заснований на моменті часу, у який починається проектування тестів:

1) профілактичні підходи, де тести розробляються якомога раніше;

2) реактивні підходи, коли тести розробляються після того, як програмний продукт або система були зроблені.

Типові підходи або стратегії включають:

1) аналітичні підходи, такі як тестування на основі ризику, де тестування спрямоване на області найбільшого ризику;

2) засновані на моделі підходи, наприклад стохастичне тестування, що використовують статистичну інформацію про інтенсивності відмов (як у моделі зростання надійності) або коефіцієнті завантаження (функціональний розріз);

3) методичні підходи, такі як підхід, заснований на помилці (включаючи припущення про помилку і атаки відмов); підхід, заснований на досвіді; використання контрольного списку питань; підхід, заснований на використанні показників якості;

4) підходи, що відповідають процесам або стандартам, наприклад визначені стандартами, специфічними для даної галузі, або різними динамічними методологіями;

5) динамічні і евристичні підходи, такі як дослідницьке тестування, де тестування більше реактивне, ніж попередньо заплановані події, і де виконання і оцінка – паралельні задачі;

6) консультативні підходи, коли тестове покриття управляється насамперед рекомендаціями і правилами технології та/або фахівцями із проблемної області, які працюють поза групою тестування;

7) регресійні підходи, які включають багаторазове використання існуючих тестових матеріалів, широку автоматизацію функціонального регресійного тестування і стандартних тестових сценаріїв.

Різні підходи можуть поєднуватися, наприклад, динамічний підхід на основі ризику.

Вибір підходу до тестування повинен ураховувати контекст, включаючи:

- ризик відмови проекту, небезпеки продукту нанесення збитків людям, навколишньому середовищу і компанії;
- навички і досвід людей у запропонованих методах і інструментальних засобах;
- мета зусиль по тестуванню і задачі групи тестування;
- регулюючі аспекти, такі як зовнішні і внутрішні правила для процесу розробки;
- характер продукту і бізнесу.

11.3 Моніторинг і управління тестуванням

11.3.1 Моніторинг виконання тестування

Мета моніторингу тестування полягає в тому, щоб одержати зворотний зв’язок і візуальний контроль тестових дій. Інформація для моніторингу може бути зібрана вручну або автоматично і може використовуватися, щоб виміряти критерії завершення, такі як покриття. Також можуть використовуватися

метрики, щоб оцінити виконання запланованого графіка і бюджету. Загальні метрики тестування включають:

- відсоток від роботи, виконаної для підготовки тестового набору (або відсоток від запланованих підготовлених тестових наборів даних);
- відсоток від роботи, виконаної для підготовки середовища тестування;
- виконання тестового набору даних (наприклад, число тестів виконаних/не виконаних і пройшли/не пройшли);
- інформація про помилки (наприклад, щільність помилок, знайдені і виправлені помилки, інтенсивність відмов і результати повторного тестування);
- тестове покриття вимог, ризиків або коду;
- суб'єктивна впевненість тестувальників у продукті;
- строки проміжних етапів тестування;
- вартість тестування, включаючи порівняльну вартість вигоди знаходження наступної помилки або виконання наступного тесту.

11.3.2 Звіти про тестування

Звіт про тестування містить підсумкову інформацію про тестування:

- що трапилося протягом періоду тестування, наприклад моменти, коли критерії завершення були виконані;
- проаналізована інформація і метрики, щоб обґрунтувати рекомендації і рішення про майбутні дії, такі як оцінка помилок, що залишилися, економічна вигода продовження тестування, ризики, що залишилися, і рівень довіри до протестованої програми.

Структура підсумкового звіту про тестування наведена в стандарті IEEE 829 "Standard for Software Test Documentation".

Протягом і наприкінці рівня тестування повинні бути зібрані метрики, щоб оцінити:

- адекватність цілей тестування для цього рівня тестування;
- адекватність обраних підходів до тестування;
- ефективність тестування щодо його цілей.

11.3.3 Управління тестуванням

Управління тестуванням описує будь-яке керівництво або коригувальні дії, початі в результаті аналізу отриманої інформації і зібраних метрик. Дії можуть охоплювати будь-яку діяльність, пов'язану з тестуванням, і можуть зачіпати будь-яку іншу задачу життєвого циклу програми.

Приклади керуючих дій тестування:

- прийняття рішень, заснованих на інформації моніторингу тестування;
- перерозподіл пріоритетів тестування, коли відбувається ідентифікований ризик (наприклад, пізня поставка програмного забезпечення);
- зміна графіка тестування через готовність середовища тестування;

– визначення критерію входу, який вимагає повторного тестування окремих частин програми (підтвердження відповідності) розроблювачем перед прийняттям їх до складання.

11.4 Управління конфігурацією

Мета управління конфігурацією полягає в тому, щоб установити і підтримати цілісність продукту (компонентів, даних і документації), програмного засобу або системи через проект і життєвий цикл програми.

Для тестування керування конфігурацією може гарантувати що:

– всі елементи програми ідентифіковані, контролюється версія, відслідковуються зміни, підтримується трасованість протягом процесу тестування;
– на всі ідентифіковані документи і програмні елементи є однозначні посилання у випробувальній документації.

Для тестувальника управління конфігурацією допомагає однозначно визначати (і відтворювати) елемент, який тестується, документи з тестування, тести і засоби тестування.

Протягом планування тестування процедури управління конфігурацією і інфраструктура (інструментальні засоби) повинні бути обрані, задокументовані і здійснені.

11.5 Ризик і тестування

Ризик може бути визначений як шанс виникнення випадку небезпеки, загрози або поява ситуації з небажаними наслідками, потенційної проблеми. Рівень ризику визначається ймовірністю несприятливої ситуації і результуючим збитком.

11.5.1 Ризики проекту

Ризики проекту – це ризики, які відносяться до можливостей проекту досягати своєї мети, наприклад:

<u>організаційні фактори</u>:

1) недостача навичок і штатів;
2) проблеми з навчанням персоналу;
3) політичні проблеми, наприклад
   – проблеми з тестувальниками, що повідомляють свої потреби і результати тестування;
   – неможливість використовувати інформацію, отриману в результаті тестування і переглядів (наприклад, розробка й порядок тестування, що не поліпшується);
4) невідповідне відношення до очікуваних результатів тестування (наприклад, не враховується важливість виявлених у ході тестування помилок);

технічні проблеми:

1) проблеми у визначенні правильних вимог;
2) необхідна величина може бути обумовлена існуючими обмеженнями;
3) якість проектування, кодування і тестування;

проблеми постачальника:

1) проблеми третьої сторони;
2) контрактні проблеми.

При аналізі, управлінні і зменшенні цих ризиків тест-менеджер слідує встановленим принципам керівництва проектом. “Standard for Software Test Documentation” (IEEE 829) містить загальну структуру плану тестування, що вимагає урахування ризиків і непередбачених обставин.

11.5.2 Ризики продукту

Потенційні області помилок (несприятливі майбутні події або небезпеки) у програмній системі відомі як ризики продукту, оскільки вони є ризиками, пов’язаними з якістю програми:

- схильне до відмов поставлене програмне забезпечення;
- потенційний ризик того, що програмне/апаратне забезпечення може завдати шкоди окремій людині або компанії;
- слабкі програмні характеристики (наприклад, функціональність, надійність, зручність використання та раціональність);
- програмне забезпечення, що не виконує призначених функцій.

Ризики використовуються, щоб вирішити, де почати тестування і де потрібно тестувати більше; тестування використовується, щоб знизити ризик появи несприятливого ефекту або зменшити його вплив.

Ризики продукту – спеціальний тип ризику успіху проекту. Тестування, як діяльність по управлінню ризиками, забезпечує зворотний зв’язок про залишковий ризик, вимірюючи ефективність усунення критичних помилок і планування непередбачених ситуацій.

Підхід до тестування на основі ризику забезпечує дієві можливості зменшити рівні ризику продукту, які починаються на початкових стадіях проекту. Виконується ідентифікація ризиків продукту і їхнє використання в керівництві плануванням і управлінням тестуванням, розробці специфікації, підготовці і виконанні тестів. У підході на основі ризику ідентифіковані ризики можуть використовуватися:

- для визначення використовуваних методів тестування;
- для визначення обсягу тестування, що буде виконано;
- для приділення першорядної уваги тестуванню, намагаючись знайти критичні помилки якомога раніше;
- для визначення, чи можуть бути використані будь-які не пов’язані з тестуванням дії, щоб зменшити ризик (наприклад, забезпечення навчання недосвідчених проектувальників).

Тестування на основі ризику залучає колективне знання і розуміння зацікавлених сторін проекту, щоб визначити ризики і необхідні рівні тестування.

Щоб гарантувати, що шанс виникнення відмови продукту мінімальний, дії по управлінню ризиком забезпечують:

- оцінювання (і переоцінку на систематичній основі), що може йти не так, як треба (ризики);
- визначення того, які ризики є досить важливими для того, щоб ними займатися;
- виконання дій щодо цих ризиків.

Крім того, тестування може підтримувати ідентифікацію нових ризиків, може допомогти визначити, які ризики повинні бути зменшені, і може знизити невизначеність щодо ризиків.

11.6 Контроль подій

Оскільки одна із цілей тестування полягає в тому, щоб знайти помилки, невідповідності між фактичними і очікуваними результатами повинні бути зареєстровані як події. Події повинні бути простежені від відкриття й класифікації до виправлення і підтвердження рішення. Щоб управляти всіма подіями, організація повинна встановити процес і правила для їхньої класифікації.

Події можуть відбуватися протягом розробки, перегляду, тестування або використання програмного продукту. Вони можуть траплятися через проблеми в коді або робочій системі, або в будь-якому типі документації, включаючи вимоги, проектні документи, документи з тестування і користувальницьку інформацію типу довідки або настанови з інсталяції.

Звіти про події мають на меті:

- забезпечення розроблювачів і інші сторони зворотним зв’язком про проблему ідентифікації, ізоляції і корекції за необхідності;
- забезпечення керівників тестування засобами відстеження якості системи при тестуванні і прогресу тестування;
- забезпечення ідей для вдосконалення процесу тестування.
- Подробиці звіту про подію можуть включати таку інформацію:
- дата випуску, організація-розроблювач і автор;
- очікувані і фактичні результати;
- ідентифікація елемента тесту (елемента конфігурації) і середовища;
- процес життєвого циклу системи, у якому спостерігалася подія;
- опис події, щоб відтворити її і вирішити, включаючи log-файли, базу даних дампів пам’яті або скріншотів;
- область або ступінь впливу на інтереси зацікавлених сторін;
- серйозність впливу на систему;

- невідкладність/пріоритет усунення;
- стан події (наприклад, відкрита, затримана, дублікат, що очікує виправлення, виправлена і очікує перетестування, закрита);
- висновки, рекомендації і схвалення;
- глобальні проблеми, такі як інші області, які можна зачепити зміною, що випливає з події;
- хронологія змін, типу послідовності дій, здійснених членами групи проекту щодо події, щоб ізолювати її, виправити і підтвердити усунення проблеми;
- посилання, включаючи ідентичність специфікації тестового набору даних, що виявив проблему.

Структура звіту про події також наведена в “Standard for Software Test Documentation” (IEEE 829).

11.7 Контрольні запитання і завдання

1. Як забезпечується незалежність при організації тестування?
2. Які вигоди від незалежності тестування та в чому її недоліки?
3. Які типові задачі виконує тест-менеджер?
4. Які типові задачі виконує тестувальник?
5. Назвіть основні етапи робіт з планування тестування.
6. Які критерії завершеності тестування?
7. Як оцінюються витрати на тестування?
8. Назвіть типові стратегії тестування.
9. Які метрики використовуються для моніторингу тестування?
10. Яку інформацію повинен містити звіт про тестування?
11. Назвіть основні ризики тестування.
12. Для чого потрібен контроль подій?

Лекція 12
ІНСТРУМЕНТАЛЬНІ ЗАСОБИ ПІДТРИМКИ ТЕСТУВАННЯ

12.1 Типи інструментальних засобів

12.1.1 Класифікація інструментальних засобів тестування

Є безліч інструментальних засобів, які підтримують різні аспекти випробування. Інструментальні засоби можна класифікувати за діями тестування, які вони підтримують.

Деякі інструментальні засоби чітко підтримують один вид діяльності; інші можуть підтримати більше одного виду діяльності, але класифіковані за діяльністю, з якою вони найбільш тісно пов'язані. Деякі комерційні інструментальні засоби пропонують підтримку тільки одного типу діяльності; інші комерційні виробники ПО пропонують набори програм або сімейства інструментальних засобів, які забезпечують підтримку багатьох або всіх цих дій.

Інструментальні засоби тестування можуть підвищити ефективність тестових дій, автоматизуючи задачі, які повторюються. Інструментальні засоби тестування можуть також підвищити надійність тестування, наприклад, автоматизація порівнянь великих масивів даних або моделювання поведінки.

Деякі типи інструментальних засобів тестування можуть зачепити фактичний результат тестування. Наприклад, результат вимірювання часу може відрізнятися залежно від того, як він вимірюється різними інструментальними засобами, або можна одержати різний ступінь покриття коду, залежно від того, який інструмент покриття використовується. Наслідки подібних інструментальних засобів називають ефектом зондування.

Деякі інструментальні засоби пропонують підтримку, яка більше відповідає розроблювачам (наприклад, у ході тестування компонентів або інтеграційного тестування).

12.1.2 Засоби підтримки управління тестуванням

Інструментальні засоби управління застосовують до всіх дій з тестування у життєвому циклі програми.

Інструментальні засоби управління тестуванням

Характеристики інструментальних засобів управління тестуванням включають:

- підтримку управління тестуванням і виконуваними діями з тестування;
- взаємодію з інструментальними засобами виконання тестів, засобами відстеження дефектів і засобами управління вимогами;
- незалежний контроль версій або інтерфейс із зовнішнім інструментом управління конфігурацією;
- підтримку трасованості тестів, результатів тестування і вихідних документів, таких як специфікації вимог;
- реєстрацію результатів тестування і генерацію звіту про досягнуті результати;

– кількісний аналіз (метрики) тестів (наприклад, виконуваний тест і пройдений тест) і об’єкта тестування (наприклад, події, що відбулися), щоб дати інформацію про об’єкт тестування, контролювати і поліпшувати процес тестування.

Інструментальні засоби управління вимогами зберігають вимоги, перевіряють узгодженість і невизначені (відсутні) вимоги, розташовують вимоги за пріоритетами і надають можливість індивідуальним тестам бути простежуваними щодо вимог, функцій і/або особливостей. У звітах про прогрес тестування міститься інформація про трасованість, покриття вимог, функцій і/або особливостей набором тестів.

Інструментальні засоби контролю подій зберігають і управляють звітами про події, тобто дефекти, відмови або виявлені проблеми і аномалії, і підтримують управління звітами про події способами, які включають:

– полегшення встановлення їхніх пріоритетів;
– призначення дій людям (наприклад, виправлення або тестування відповідності);
– приписування стану (наприклад відхилений, готовий до тестування або відкладений до наступного релізу).

Ці інструментальні засоби дають можливість моніторингу подій, часто забезпечують підтримку статистичному аналізу і забезпечують звіти про події. Вони також відомі як засоби відстеження дефектів.

Інструментальні засоби управління конфігурацією є не строго засобами тестування, але зазвичай необхідні, щоб стежити за різними версіями і релізами програмних засобів і тестів.

Інструментальні засоби керування конфігурацією:

– зберігають інформацію про версії та builds програми і інструментів тестування;
– допускають трасованість між інструментами тестування і варіантами програмного продукту;
– особливо корисні при розробці для більш ніж однієї конфігурації апаратних засобів/програмного середовища (наприклад, для різних версій операційної системи, різних бібліотек або компіляторів, різних браузерів або різних комп’ютерів).

12.1.3 Засоби підтримки статичного тестування

Інструментальні засоби перегляду (також відомі як інструментальні засоби підтримки процесу перегляду) можуть зберігати інформацію про процеси перегляду, зберігати та повідомляти коментарі перегляду, звіт про дефекти і зусилля, управляти посиланнями на правила перегляду і/або контрольний список питань, стежити за трасованістю між документами і вихідним кодом. Вони можуть також допомогти в організації online переглядів, які є корисними, якщо група географічно розосереджена.

Інструментальні засоби статичного аналізу підтримують розроблювачів, тестувальників і персонал забезпечення якості у виявленні помилок перед динамічним тестуванням. Їхні головні цілі включають:

- дотримання стандартів програмування;
- аналіз структур і залежностей (наприклад, зв’язані web-сторінки);
- допомога в розумінні коду.

Інструментальні засоби статичного аналізу можуть обчислювати метрики коду (наприклад, складність), які можуть дати цінну інформацію, наприклад, для планування або аналізу ризиків.

Інструментальні засоби моделювання здатні перевірити правильність моделей програмного забезпечення. Наприклад, перевірка моделі бази даних може знайти помилки і неузгодженості в моделі даних; інші інструментальні засоби моделювання можуть знайти помилки в моделі станів або об’єктній моделі. Ці інструментальні засоби часто допомагають генерувати деякі тестові набори даних, засновані на моделі.

Головна вигода інструментальних засобів статичного аналізу і інструментальних засобів моделювання – зниження вартості виявлення більшої кількості помилок на ранніх стадіях розробки. У результаті, процес розробки може прискоритися і покращитися за рахунок зменшення переробок.

12.1.4 Інструментальні засоби підтримки тестових специфікацій

Засоби проектування тестів генерують тестові дані або виконувані тести з вимог, із графічного інтерфейсу користувача, з моделей проектування (станів, даних або об’єктної) або з коду. Цей тип інструментів може також генерувати очікувані результати (тобто може використовувати передбачення тестів). Згенеровані тести з моделі станів або об’єктної моделі корисні, щоб перевірити реалізацію моделі в програмному засобі, але рідко достатні для того, щоб перевірити всі аспекти програмного продукту або системи. Вони можуть заощаджувати цінний час і забезпечувати підвищену ретельність тестування через повноту тестів, які інструмент може генерувати.

Інші інструментальні засоби в цій категорії можуть допомогти в підтримці генерації тестів, забезпечуючи структурні шаблони, які генерують тести або заглушки, і в такий спосіб прискорюють процес тестування проекту.

Інструментальні засоби підготовки тестових даних управляють базами даних, файлами або передачею даних, щоб установити тестові дані, які використовуються в ході виконання тестів. Перевагою цих інструментальних засобів є забезпечення того, що реальні дані, передані середовищу тестування, зроблені анонімними, для захисту даних.

12.1.5 Засоби підтримки виконання тестів і реєстрації

Інструментальні засоби виконання тестів дають можливість тестам виконуватися автоматично або напівавтоматично, використовуючи збережені уведення і очікувані результати, за допомогою мови сценаріїв. Мова сценаріїв дозволяє управляти тестами з обмеженим зусиллям, наприклад, повторити тестування з різними даними або перевіряти різні частини системи з подібними кроками. Зазвичай ці інструментальні засоби включають функції динамічного порівняння і забезпечують реєстрацію результатів для кожного тестового прогону.

Інструментальні засоби виконання тестів можуть також використовуватися для запису тестів, коли вони можуть посилатися на засоби відтворення збору даних. Фіксація іспитових уведень протягом дослідницького тестування або непідготовленого тестування може бути корисна, щоб відтворити і/або документувати тест, наприклад, якщо відбувається відмова.

Інструментальні засоби структури тестування модулів можуть полегшити тестування компонентів або частини системи, моделюючи середовище, у якій об'єкт тестування буде працювати. Це може бути зроблене або тому що інші компоненти середовища ще не доступні і замінені заглушками і/або драйверами, або просто забезпечувати передбачуване і кероване середовище, у якому будь-які помилки можуть бути локалізовані в об'єкті тестування.

Може бути створена структура, де частина коду, об'єкта, методу або функції, модуля або компонента може бути виконана, викликаючи об'єкт, що буде тестуватися і/або використовуючи зворотний зв'язок із цим об'єктом. Це можна зробити, використовуючи штучні засоби уведення даних для об'єкта тестування, і/або використовуючи заглушки для виведення даних від об'єкта замість реального виведення.

Інструментальні засоби тестування можуть також використовуватися, щоб забезпечити структуру виконання в мікропрограмних засобах, де мови, операційні системи або апаратні засоби повинні тестуватися спільно.

Їх можна назвати інструментальними засобами структури тестування модулів, коли вони фокусуються на рівні тестування компонентів. Цей тип інструмента допомагає у виконанні тестування компонентів паралельно зі складанням коду.

Тестові компаратори визначають розходження між файлами, базами даних або результатами тестування. Засоби виконання тестів зазвичай включають динамічні компаратори, але порівняння після виконання може бути зроблено окремим інструментом порівняння. Тестовий компаратор може використовувати тестовий провісник, особливо якщо це автоматизовано.

Інструментальні засоби вимірювання покриття можуть впливати або не впливати на результат залежно від використовуваних методів вимірювання, об'єкта вимірювання та мови кодування. Інструментальні засоби покриття коду вимірюють відсоток від певних типів структури коду, які були здійснені (наприклад, оператори, гілки або рішення, і модуль або виклики функції). Ці інструментальні засоби показують, як повністю виміряний тип структури був покритий тестами.

Інструментальні засоби захисту перевіряють комп'ютерні віруси і атаки відмов в обслуговуванні. Система мережного захисту, наприклад – не строго інструмент тестування, але може використовуватися в тестуванні захисту. Інструментальні засоби захисту шукають певну уразливість системи.

12.1.6 Засоби підтримки виконання та моніторингу

Інструментальні засоби динамічного аналізу знаходять помилки, які є очевидними тільки коли програмний засіб виконується, типу часових

залежностей або витоків пам’яті. Вони зазвичай використовуються в тестуванні модулів і тестуванні інтеграції модулів, а також при тестуванні мікропрограмних засобів.

Засоби виконання тестування контролюють і повідомляють про те, як система поводиться в різноманітних модельованих умовах використання. Вони моделюють навантаження на програму, базу даних або системне середовище, типу мережі або сервера. Інструментальні засоби часто називаються на честь того аспекту виконання, який вони вимірюють, типу завантаження або стресу, вони також відомі як засоби навантажувального тестування і засобу стресового тестування. Вони часто засновані на автоматизованому повторному виконанні тестів, керованому параметрами.

Інструментальні засоби моніторингу не є строго засобами тестування, але подають інформацію, що може використовуватися з метою тестування і яку не можна одержати іншими засобами.

Інструментальні засоби моніторингу безупинно аналізують, перевіряють і повідомляють про використання певних системних ресурсів, видають попередження про можливі проблеми. Вони зберігають інформацію про версію і складання (build) програмного продукту і засобу тестування, допускають трасованість.

12.1.7 Інструментальні засоби підтримки певних прикладних областей

Індивідуальні приклади типів інструментальних засобів наведеної вище класифікації можуть бути спеціалізовані для використання в певному типі додатків. Наприклад, існують інструментальні засоби тестування спеціально для web-додатків, інструментальні засоби статичного аналізу для певних платформ розробки, і інструментальні засоби динамічного аналізу спеціально для перевірки аспектів безпеки.

Комерційні набори інструментальних програм можуть бути призначені для певних прикладних областей (наприклад, вбудовані системи).

12.1.8 Інструментальні засоби підтримки, що використовують інші інструментальні засоби

Інструментальні засоби тестування, перераховані тут – не єдині типи інструментальних засобів, використовуваних тестувальниками – вони можуть також використовувати електронні таблиці, SQL, засоби керування ресурсами або засоби налагодження.

12.2 Ефективне використання інструментальних засобів: потенційні вигоди та ризики

12.2.1 Потенційні вигоди і ризики засобів підтримки тестування (для всіх інструментальних засобів)

Просто купівля або оренда інструментального засобу ще не гарантують успіх. Кожний тип інструмента може вимагати додаткових зусиль для досягнення реальних і тривалих вигід. Існують потенційні вигоди і переваги використання інструментальних засобів тестування, але є також ризики.

Потенційні вигоди від використання інструментальних засобів включають:

- зменшення повторюваної роботи (наприклад, регресійне тестування, повторне уведення одних тих же тестових даних, перевірка відповідності стандартам програмування);
- більшу узгодженість і відтворюваність (наприклад, тестування, виконане інструментом, і тестування, що виходить із вимог);
- об’єктивну оцінку (наприклад, статичні вимірювання, покриття);
- полегшення доступу до інформації про тести або тестування (наприклад статистика і графіки прогресу тестування, подій і виконання).

Ризики використання інструментальних засобів включають:

- нереалістичні очікування від інструмента (включаючи функціональні можливості і зручність використання);
- недооцінку часу, вартості і зусиль для початкового введення інструмента (включаючи навчання і зовнішню експертизу);
- недооцінку часу і зусиль, необхідних для досягнення істотної і довгострокової вигоди від інструмента (включаючи необхідність у змінах у процесі випробування і безперервному вдосконаленні способу використання інструмента);
- недооцінку зусиль, необхідних для підтримки фонду тестів, згенерованих інструментом;
- перебільшену довіру до інструментального засобу (використання його для проектування тестів у випадках, коли ручне тестування було б краще).

12.2.2 Деякі міркування для окремих типів інструментальних засобів

Інструментальні засоби виконання тестів програють сценарії, розроблені, щоб виконати тести, які зберігаються в електронному вигляді. Цей тип інструмента часто вимагає істотних зусиль для досягнення істотних вигід.

Фіксація тестів шляхом запису дій тестувальника здається привабливою, але цей підхід не підходить для великої кількості автоматизованих тестів. Зафіксований сценарій є лінійним поданням з певними даними і діями. Цей тип сценарію може бути непостійним, якщо відбуваються несподівані події.

Керований даними підхід виділяє тестові входи (дані), зазвичай в електронну таблицю, і використовує більш універсальний сценарій, що може читати тестові дані і виконувати той самий тест із різними даними.

Тестувальники, які не знайомі з мовою сценаріїв, можуть увести тестові дані для цих заздалегідь визначених сценаріїв.

У керованому ключовим словом підході електронна таблиця містить ключові слова, що описують дії, які будуть початі (також називані словами дії) і тестові дані. Тестувальники (навіть якщо вони не знайомі з мовою сценаріїв) можуть тоді визначити тести, використовуючи ключові слова, які можуть бути пристосовані до програми, яка тестується.

Технічна експертиза в мові сценаріїв необхідна для всіх підходів (виконується або тестувальниками або фахівцями з автоматизації тестування).

Яка б методика створення сценарію не використовувалася, очікувані результати для кожного тесту необхідно зберігати для наступного порівняння.

Інструментальні засоби оцінки характеристик тестування мають потребу в експертизі, щоб допомогти проектувати тести та інтерпретувати результати.

Інструментальні засоби статичного аналізу, застосовувані до вихідного коду, можуть пропонувати стандарти кодування, але якщо вони застосовуються до існуючого коду, це може привести до генерування великої кількості повідомлень. Попереджуючі повідомлення не зупиняють виконання коду, відтрансльованого у програму, що виконується, але потрібно постаратися зробити так, щоб обслуговування коду було простішим у майбутньому. Поступове виконання з початковими фільтрами, щоб виключити деякі повідомлення, було б ефективним підходом.

Інструментальні засоби керування тестуванням повинні зв’язуватися за допомогою інтерфейсу з іншими інструментальними засобами або електронними таблицями, щоб видати інформацію в кращому форматі для поточних потреб організації. Повідомлення повинні розроблятися і контролюватися так, щоб вони забезпечили вигоду.

12.3 Впровадження інструментальних засобів в організації

Основні міркування з вибору інструмента для організації включають:

- оцінку зрілості організації, сильних і слабких сторін і ідентифікацію можливостей поліпшення процесу тестування, підтримуваного інструментальними засобами;
- оцінку ясних вимог і об’єктивних критеріїв;
- підтвердження концепції тестування необхідної функціональності і визначення, чи досягає програма своєї мети;
- оцінку постачальника (включаючи навчання, підтримку і комерційні аспекти);
- ідентифікацію внутрішніх вимог для тренування та наставництва у використанні інструмента.

Впровадження обраного інструмента в організації починається з пілотного проекту, що спрямований на досягнення таких цілей:

- більш детальне вивчення інструмента;

- оцінка того, як інструмент відповідає існуючим процесам і діям, і визначення того, що повинно змінитися;
- вибір стандартних способів використання, управління, зберігання і супроводу інструмента і фонду тестів (наприклад, угода про імена для файлів і тестів, створення бібліотек і визначення модульності наборів тестів);
- оцінка того, чи будуть досягнуті вигоди за розумну вартість.

Фактори, що визначають успіх розгортання інструмента в межах організації, включають:

- покрокове розгортання інструмента в організації;
- пристосовування і поліпшення процесів, щоб відповідати використанню інструмента;
- забезпечення навчання і тренування / наставництва для нових користувачів;
- визначення рекомендацій з використання;
- здійснення способу здобувати уроки з використання інструмента;
- використання засобів моніторингу та вигід.

12.4 Контрольні запитання і завдання

1. Наведіть класифікацію інструментальних засобів тестування.
2. Які функції виконують засоби підтримки управління тестуванням?
3. Які функції виконують засоби підтримки статичного тестування?
4. Які інструментальні засоби використовуються для підтримки виконання тестів і реєстрації?
5. Які функції покладено на засоби виконання тестування та моніторингу?
6. Які потенційні вигоди і ризики має використання інструментальних засобів підтримки тестування?

Електронне навчальне видання

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни

«ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ ЯКОСТІ ПРОГРАМНИХ ЗАСОБІВ»

для студентів спеціальності
152 «Метрологія та інформаційно-вимірювальна техніка»
спеціалізацій «Метрологія та вимірювальна техніка»,
«Метрологічне забезпечення випробувань та якості продукції»,
«Якість, стандартизація та сертифікація»

Упорядник ЗАПОРОЖЕЦЬ Олег Васильович

Відповідальний випусковий І.В. Руженцев

Авторська редакція