Міністерство освіти і науки України
Міжнародний гуманітарний університет
Кафедра інформаційних технологій

С.Б. Приходько

ЯКІСТЬ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Опорний конспект лекцій
для здобувачів вищої освіти,
які навчаються за спеціальностями
галузі «Інформаційні технології»

Одеса 2025

Затверджено Вченою Радою Міжнародного гуманітарного університету.
Протокол № 5 від 20 січня 2025 р.

С.Б. Приходько Якість та тестування програмного забезпечення. Опорний конспект лекцій для здобувачів вищої освіти, які навчаються за спеціальностями галузі «Інформаційні технології» [Електронне видання]. Кафедра інформаційних технологій Міжнародного гуманітарного університету. Одеса, 2025. 35 с.

Дисципліна «Якість та тестування програмного забезпечення» спрямована на формування у здобувачів вищої освіти знань і навичок забезпечення якості програмних систем на різних етапах їх життєвого циклу. Курс охоплює основні принципи забезпечення якості програмного забезпечення, моделі та характеристики якості програмних систем, процеси управління якістю, методи контролю якості та тестування програмного забезпечення. У межах дисципліни розглядаються рівні тестування програмного забезпечення, методи тестування типу «чорної скриньки» та «білої скриньки», метрики якості програмних систем, а також сучасні інструментальні засоби автоматизації тестування. Вивчення дисципліни сприяє формуванню здатності аналізувати якість програмного забезпечення, планувати та організовувати процес тестування, виявляти дефекти програмних систем та застосовувати сучасні інструменти контролю якості програмного забезпечення. Особлива увага приділяється інтеграції процесів тестування у життєвий цикл розроблення програмного забезпечення та використанню сучасних інструментальних засобів автоматизації тестування.

ЗМІСТ

ТЕМА 1. ОСНОВИ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ....4
Лекція 1. Поняття якості програмного забезпечення....4
Лекція 2. Моделі якості програмного забезпечення....6
Лекція 3. Міжнародні стандарти у сфері якості програмного забезпечення....9
ТЕМА 2. ПРОЦЕСИ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ....12
Лекція 4. Процеси забезпечення якості програмного забезпечення....12
Лекція 5. Аудит, рецензування та статичний аналіз програмного забезпечення....13
Лекція 6. Управління конфігураціями та змінами програмного забезпечення....15
ТЕМА 3. ОСНОВИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ....18
Лекція 7. Основи тестування програмного забезпечення....18
Лекція 8. Рівні тестування та організація процесу тестування....19
Лекція 9. Життєвий цикл тестування та документування....21
ТЕМА 4. МЕТОДИ ТА ТЕХНІКИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ....24
Лекція 10. Методи тестування програмного забезпечення....24
Лекція 11. Методи проєктування тестів та аналіз результатів....25
Лекція 12. Методи «білої скриньки» та покриття коду....27
ТЕМА 5. АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ....29
Лекція 13. Автоматизоване тестування програмного забезпечення....29
Лекція 14. Системи управління тестуванням та відстеження дефектів....30
Лекція 15. Оцінювання якості та економічна ефективність програмного забезпечення....32
РЕКОМЕНДОВАНА ЛІТЕРАТУРА....34

ТЕМА 1. ОСНОВИ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Лекція 1. Поняття якості програмного забезпечення

Якість програмного забезпечення є фундаментальною характеристикою будь-якої програмної системи, що визначає ступінь її відповідності встановленим вимогам, очікуванням користувачів та умовам експлуатації. У сучасних умовах розвитку інформаційних технологій, коли програмні продукти інтегруються у критично важливі сфери діяльності – фінанси, транспорт, медицину, енергетику – забезпечення належного рівня якості стає не лише технічним завданням, а й стратегічною необхідністю. Якість безпосередньо впливає на надійність, безпеку, ефективність та конкурентоспроможність програмного продукту.

Поняття якості програмного забезпечення має комплексний характер і охоплює сукупність властивостей програмної системи, які визначають її здатність задовольняти встановлені та передбачувані потреби користувачів. Відповідно до міжнародних стандартів, зокрема ISO/IEC 25010, якість розглядається як багатовимірна характеристика, що включає низку взаємопов'язаних атрибутів. Такий підхід дає змогу формалізувати вимоги до програмного забезпечення, забезпечити їх вимірюваність та здійснювати об'єктивне оцінювання.

У процесі розробки програмних систем якість відіграє ключову роль на всіх етапах життєвого циклу програмного забезпечення. На етапі формування вимог якість визначає повноту, точність та несуперечливість специфікацій. На етапі проєктування вона впливає на архітектурні рішення, вибір технологій та структурних компонентів системи. Під час реалізації програмного продукту якість проявляється у правильності алгоритмів, ефективності використання ресурсів та дотриманні стандартів кодування. На етапі тестування здійснюється перевірка відповідності програмного забезпечення заданим вимогам і виявлення дефектів. Нарешті, у процесі експлуатації якість визначає стабільність роботи системи, зручність її використання та можливість подальшого розвитку.

Одним із центральних аспектів розуміння якості програмного забезпечення є її поділ на внутрішню та зовнішню. Внутрішня якість характеризує властивості програмного коду та архітектури, які не завжди безпосередньо спостерігаються користувачем, але суттєво впливають на супроводжуваність, масштабованість та надійність системи. Зовнішня якість, у свою чергу, відображає поведінку програмного продукту під час виконання і є безпосередньо помітною для кінцевого користувача. Окрім того, виділяють якість у використанні, яка визначає ефективність, результативність та задоволеність користувача під час взаємодії із системою.

Основні характеристики якості програмного забезпечення формують структуру оцінювання та аналізу програмних систем. До них належать функціональна придатність, надійність, продуктивність, зручність використання, супроводжуваність та переносимість. Кожна з цих характеристик має власний зміст, критерії оцінювання та вплив на загальну якість програмного продукту.

Функціональна придатність визначає ступінь відповідності програмного забезпечення функціональним вимогам. Вона характеризує, наскільки повно та коректно система реалізує передбачені функції. Це включає правильність обчислень, повноту реалізації функціональних можливостей та відповідність очікуванням користувача. Низький рівень

функціональної придатності призводить до помилок у роботі системи, втрати даних або неможливості виконання ключових операцій.

Надійність є однією з найважливіших характеристик якості програмного забезпечення. Вона визначає здатність системи виконувати свої функції без відмов протягом заданого періоду часу за встановлених умов експлуатації. Надійність охоплює такі аспекти, як стійкість до помилок, здатність до відновлення після збоїв та стабільність функціонування. У системах, що працюють у критичних умовах, наприклад у медичних або авіаційних застосуваннях, вимоги до надійності є особливо високими.

Продуктивність характеризує ефективність використання ресурсів системи, таких як процесорний час, пам’ять та пропускна здатність мережі. Вона визначає швидкість виконання операцій, час відгуку системи та здатність обробляти великі обсяги даних. Висока продуктивність є важливою для систем реального часу, вебсервісів та високонавантажених інформаційних систем.

Зручність використання або юзабіліті визначає, наскільки легко користувач може взаємодіяти з програмною системою. Вона охоплює такі аспекти, як інтуїтивність інтерфейсу, простота навчання, ефективність виконання завдань та рівень задоволеності користувача. Низька зручність використання може призвести до помилок користувача, зниження продуктивності праці та відмови від використання програмного продукту.

Супроводжуваність відображає здатність програмного забезпечення до модифікації, виправлення помилок та адаптації до нових умов. Вона залежить від структури коду, рівня його документованості, використання стандартів програмування та архітектурних рішень. Висока супроводжуваність дає змогу зменшити витрати на підтримку та розвиток програмного продукту.

Переносимість характеризує здатність програмного забезпечення функціонувати у різних середовищах, на різних платформах або з різними конфігураціями апаратного забезпечення. Вона є важливою для сучасних програмних систем, які повинні працювати на різноманітних пристроях, включаючи персональні комп’ютери, мобільні пристрої та вбудовані системи.

Атрибути якості програмних систем не є незалежними один від одного, а перебувають у складній взаємодії. Наприклад, підвищення продуктивності може вимагати оптимізації коду, що ускладнює його супроводжуваність. Аналогічно, забезпечення високої надійності може призводити до збільшення витрат ресурсів або зниження продуктивності. Тому під час розробки програмного забезпечення важливо знаходити компроміс між різними характеристиками якості залежно від специфіки проєкту.

Роль якості у забезпеченні надійності та ефективності програмних систем є визначальною. Високоякісне програмне забезпечення забезпечує стабільну роботу системи, мінімізує ризик відмов та знижує витрати на обслуговування. Воно також сприяє підвищенню довіри користувачів, покращенню репутації розробника та забезпеченню конкурентних переваг на ринку.

З точки зору інженерії програмного забезпечення, забезпечення якості є системним процесом, що включає планування якості, її забезпечення та контроль. Планування якості передбачає визначення стандартів, метрик та критеріїв оцінювання. Забезпечення якості охоплює діяльність, спрямовану на запобігання виникненню дефектів, зокрема використання методів аналізу вимог, проєктування та кодування. Контроль якості передбачає виявлення дефектів шляхом тестування та аналізу результатів виконання програмного забезпечення.

Особливе значення у забезпеченні якості має тестування програмного забезпечення, яке є одним із основних методів контролю якості. Воно дає змогу виявити помилки, оцінити відповідність системи вимогам та перевірити її поведінку у різних умовах. Проте тестування не може гарантувати відсутність дефектів, а лише підтверджує їх наявність або відсутність у перевірених сценаріях. Тому забезпечення якості повинно здійснюватися на всіх етапах розробки, а не лише на етапі тестування.

У сучасних умовах розробки програмного забезпечення, зокрема при використанні гнучких методологій, якість інтегрується у процес розробки як невід'ємна складова. Команди розробників застосовують практики безперервної інтеграції, автоматизованого тестування та контролю якості коду, що дає змогу оперативно виявляти та усувати дефекти. Такий підхід сприяє підвищенню якості програмного продукту та скороченню часу його виведення на ринок.

Таким чином, якість програмного забезпечення є багатовимірною характеристикою, що визначає успішність програмного продукту на всіх етапах його життєвого циклу. Вона охоплює широкий спектр атрибутів, кожен з яких має важливе значення для забезпечення ефективної та надійної роботи системи. Розуміння сутності якості, її характеристик та ролі у процесі розробки є необхідною умовою для підготовки кваліфікованих фахівців у галузі інженерії програмного забезпечення.

Лекція 2. Моделі якості програмного забезпечення

Якість програмного забезпечення як комплексна характеристика потребує формалізованих підходів до її опису, аналізу та оцінювання. Для цього в інженерії програмного забезпечення використовуються моделі якості, які задають структуру характеристик, підхарактеристик та критеріїв оцінювання. Модель якості є концептуальним інструментом, що дає змогу систематизувати вимоги до програмного продукту, забезпечити їх узгодженість та здійснювати об'єктивне вимірювання рівня якості.

Моделі якості програмного забезпечення формують основу для побудови систем управління якістю, розробки стандартів, проведення тестування та оцінювання результатів. Вони визначають, які саме властивості програмного продукту підлягають оцінюванню, як ці властивості взаємопов'язані між собою та які метрики можуть бути використані для їх кількісного визначення. Використання моделей якості є необхідним для забезпечення прозорості процесу оцінювання та можливості порівняння різних програмних систем.

Існує декілька відомих моделей якості програмного забезпечення, серед яких класичними є модель МакКола, модель Боема, модель FURPS та сучасні стандартизовані моделі сімейства ISO/IEC 25000. Модель МакКола була однією з перших спроб систематизації характеристик якості та включала такі фактори, як коректність, надійність, ефективність, зручність використання, супроводжуваність та переносимість. Вона орієнтована на користувацькі та технічні аспекти якості.

Модель Боема розширює підхід МакКола, вводячи ієрархічну структуру характеристик якості, де верхній рівень представлений загальними цілями, такими як корисність, підтримуваність та переносимість, а нижчі рівні деталізують конкретні властивості системи. Вона також враховує економічні аспекти якості, зокрема витрати на розробку та супровід.

Модель FURPS, запропонована компанією Hewlett-Packard, класифікує характеристики якості на функціональні (Functionality), зручність використання (Usability), надійність (Reliability), продуктивність (Performance) та супроводжуваність (Supportability). Ця модель широко використовується у практиці розробки програмного забезпечення завдяки своїй простоті та орієнтації на ключові аспекти якості.

Найбільш сучасним та стандартизованим підходом є модель якості, визначена у стандарті ISO/IEC 25010, який входить до серії стандартів SQuaRE (Software Quality Requirements and Evaluation). Ця модель є розвитком попереднього стандарту ISO/IEC 9126 і враховує сучасні вимоги до програмного забезпечення, включаючи аспекти безпеки, сумісності та якості у використанні.

Модель ISO/IEC 25010 визначає дві основні складові якості: якість продукту та якість у використанні. Якість продукту описує внутрішні та зовнішні характеристики програмного забезпечення, тоді як якість у використанні відображає ефективність та задоволеність користувача під час взаємодії із системою.

Якість продукту відповідно до ISO/IEC 25010 включає вісім основних характеристик: функціональна придатність, продуктивність, сумісність, зручність використання, надійність, безпека, супроводжуваність та переносимість. Кожна з цих характеристик деталізується через підхарактеристики, що дає змогу більш точно визначати вимоги та проводити оцінювання.

Функціональна придатність включає повноту, коректність та відповідність функцій. Продуктивність охоплює час відгуку, пропускну здатність та використання ресурсів. Сумісність визначає здатність системи взаємодіяти з іншими системами та співіснувати у спільному середовищі. Зручність використання включає зрозумілість, навчальність, керованість та привабливість інтерфейсу.

Надійність охоплює безвідмовність, відновлюваність та стійкість до помилок. Безпека визначає захищеність даних та доступу до системи. Супроводжуваність включає модульність, аналізованість, змінюваність та тестованість. Переносимість визначає здатність системи працювати у різних середовищах та умовах.

Якість у використанні включає ефективність, результативність, задоволеність користувача, відсутність ризиків та контекстну придатність. Цей аспект є особливо важливим для оцінювання програмних систем з точки зору кінцевого користувача.

Для кількісного оцінювання характеристик якості використовуються метрики програмного забезпечення. Метрика є числовим показником, який відображає певну властивість програмного продукту або процесу його розробки. Метрики дають змогу здійснювати об’єктивне оцінювання, порівняння та контроль якості програмного забезпечення.

Метрики програмного забезпечення поділяються на кілька основних категорій: метрики продукту, метрики процесу та метрики проєкту. Метрики продукту характеризують властивості самого програмного забезпечення, наприклад складність коду, кількість дефектів, обсяг коду. Метрики процесу відображають ефективність процесів розробки, такі як час виконання завдань або кількість змін. Метрики проєкту пов’язані з управлінням ресурсами, строками та витратами.

Однією з найбільш відомих метрик є кількість рядків коду (LOC), яка використовується для оцінювання розміру програмного продукту. Проте ця метрика має обмежену інформативність, оскільки не враховує складність та якість коду. Для більш точного оцінювання складності використовуються метрики цикломатичної складності,

запропоновані Томасом МакКейбом. Вони визначають кількість незалежних шляхів виконання програми і дають змогу оцінити складність тестування.

Іншою важливою групою метрик є метрики якості коду, такі як коефіцієнт дублювання коду, глибина вкладеності, зв'язність та згуртованість модулів. Вони дають змогу оцінити структуру програмного забезпечення та його супроводжуваність. Метрики дефектів, зокрема щільність дефектів (кількість дефектів на тисячу рядків коду), використовуються для оцінювання надійності програмного продукту.

Для оцінювання продуктивності застосовуються метрики часу відгуку, пропускної здатності та використання ресурсів. Для зручності використання можуть використовуватися показники часу навчання користувача, кількості помилок користувача або рівня задоволеності. Таким чином, кожна характеристика якості може бути представлена через відповідні метрики.

Вимірювання характеристик якості програмного забезпечення є складним процесом, що включає визначення об'єктів вимірювання, вибір метрик, збір даних та їх аналіз. Важливим аспектом є забезпечення валідності та надійності метрик, тобто їх здатності адекватно відображати відповідні властивості програмного продукту.

Процес вимірювання якості включає кілька етапів. На першому етапі визначаються цілі вимірювання, які повинні відповідати вимогам проєкту та очікуванням зацікавлених сторін. Далі обираються метрики, які найкраще відображають необхідні характеристики якості. На наступному етапі здійснюється збір даних, який може проводитися автоматизовано за допомогою спеціалізованих інструментів або вручну.

Після збору даних виконується їх аналіз та інтерпретація. Отримані результати порівнюються з встановленими критеріями або нормативами, що дає змогу зробити висновки про рівень якості програмного забезпечення. На основі цього можуть прийматися рішення щодо необхідності внесення змін, оптимізації або доопрацювання системи.

Особливу роль у вимірюванні якості відіграють автоматизовані засоби аналізу, такі як системи статичного аналізу коду, інструменти тестування та моніторингу продуктивності. Вони дають змогу отримувати об'єктивні дані у реальному часі та оперативно реагувати на виявлені проблеми.

Важливо зазначити, що жодна окрема метрика не може повністю відобразити якість програмного забезпечення. Тому для оцінювання використовуються системи метрик, які охоплюють різні аспекти якості. При цьому необхідно враховувати контекст використання програмного продукту, його призначення та вимоги користувачів.

Таким чином, моделі якості програмного забезпечення, зокрема ISO/IEC 25010, забезпечують структурований підхід до визначення та оцінювання характеристик якості. Метрики програмного забезпечення є інструментом кількісного вимірювання цих характеристик, що дає змогу здійснювати об'єктивний аналіз та контроль якості. Вимірювання характеристик якості є невід'ємною складовою процесу розробки програмних систем і забезпечує підвищення їх надійності, ефективності та відповідності вимогам.

Лекція 3. Міжнародні стандарти у сфері якості програмного забезпечення

У сучасній інженерії програмного забезпечення забезпечення якості базується не лише на внутрішніх практиках розробника, а й на застосуванні міжнародних стандартів, які визначають узгоджені підходи до оцінювання, контролю та управління якістю програмних продуктів. Використання стандартів дає змогу забезпечити сумісність процесів розробки, підвищити довіру до програмного забезпечення, а також гарантувати відповідність результатів розробки встановленим вимогам і очікуванням зацікавлених сторін.

Міжнародні стандарти у сфері якості програмного забезпечення розробляються провідними організаціями, зокрема ISO (International Organization for Standardization) та IEC (International Electrotechnical Commission). Спільні стандарти цих організацій, позначені як ISO/IEC, охоплюють широкий спектр питань, пов'язаних із життєвим циклом програмного забезпечення, управлінням якістю, тестуванням, оцінюванням та супроводом програмних систем.

Одним із ключових стандартів є ISO/IEC 25010, який визначає модель якості програмного продукту та якість у використанні. Він входить до серії стандартів SQuaRE (Software Quality Requirements and Evaluation), яка охоплює процеси формування вимог до якості, їх вимірювання та оцінювання. Цей стандарт забезпечує основу для визначення характеристик якості та відповідних метрик, що застосовуються у процесі розробки та тестування програмного забезпечення.

Важливе місце у системі стандартів займає ISO/IEC 12207, який визначає процеси життєвого циклу програмного забезпечення. Цей стандарт описує повний набір процесів, що охоплюють розробку, експлуатацію, супровід та виведення програмного продукту з експлуатації. Він включає основні процеси (розробка, постачання, експлуатація), допоміжні процеси (забезпечення якості, верифікація, валідація, аудит) та організаційні процеси (управління, удосконалення процесів, навчання персоналу).

Стандарт ISO/IEC 12207 відіграє ключову роль у формуванні структурованого підходу до управління якістю, оскільки він інтегрує процеси забезпечення якості у всі етапи життєвого циклу програмного забезпечення. Це означає, що контроль якості не є окремою діяльністю, а здійснюється постійно, починаючи з формування вимог і завершуючи супроводом системи.

Іншим важливим стандартом є ISO/IEC 15504, відомий також як SPICE (Software Process Improvement and Capability Determination). Він призначений для оцінювання зрілості процесів розробки програмного забезпечення та визначення їх здатності досягати поставлених цілей. Цей стандарт дає змогу організаціям оцінити рівень розвитку своїх процесів і визначити напрямки їх удосконалення.

Окрему групу становлять стандарти серії ISO 9000, які регламентують системи управління якістю. Зокрема, ISO 9001 визначає вимоги до системи управління якістю організації, включаючи процеси планування, забезпечення, контролю та вдосконалення якості. Хоча цей стандарт є загальним і не обмежується сферою програмного забезпечення, він широко застосовується у IT-компаніях для побудови систем управління якістю.

У контексті тестування програмного забезпечення важливу роль відіграє стандарт ISO/IEC/IEEE 29119, який визначає процеси, методи та документацію тестування. Він охоплює планування тестування, проєктування тестів, їх виконання та оцінювання результатів. Цей стандарт сприяє уніфікації підходів до тестування та підвищенню його ефективності.

Таким чином, стандарти ISO/IEC формують комплексну систему, що охоплює всі аспекти якості програмного забезпечення: від визначення вимог до оцінювання результатів і вдосконалення процесів. Їх застосування дає змогу забезпечити системність, передбачуваність та контрольованість процесів розробки.

Організація процесів управління якістю програмного забезпечення у межах життєвого циклу передбачає інтеграцію діяльності з якості у всі етапи створення програмного продукту. Життєвий цикл програмного забезпечення включає такі основні етапи: формування вимог, проєктування, реалізація, тестування, впровадження та супровід. На кожному з цих етапів виконуються специфічні дії, спрямовані на забезпечення якості.

На етапі формування вимог основна увага приділяється їх повноті, узгодженості та перевірюваності. Використовуються методи аналізу вимог, їх формалізації та валідації. Помилки на цьому етапі можуть призвести до значних витрат у майбутньому, тому забезпечення якості вимог є критично важливим.

На етапі проєктування здійснюється розробка архітектури програмної системи, визначення її компонентів та взаємозв'язків. Забезпечення якості на цьому етапі включає перевірку відповідності архітектурних рішень вимогам, аналіз ризиків та оцінювання альтернативних варіантів реалізації.

Під час реалізації програмного забезпечення якість забезпечується через дотримання стандартів кодування, використання методів статичного аналізу, проведення рецензування коду та застосування практик безперервної інтеграції. Це дає змогу виявляти дефекти на ранніх етапах та зменшувати їх вплив на кінцевий продукт.

Етап тестування є ключовим для контролю якості програмного забезпечення. Він включає різні види тестування, такі як модульне, інтеграційне, системне та приймальне тестування. Метою тестування є виявлення дефектів, перевірка відповідності системи вимогам та оцінювання її поведінки у різних умовах.

На етапі впровадження здійснюється розгортання програмного забезпечення у робочому середовищі. Тут важливими є перевірка сумісності, налаштування системи та навчання користувачів. Забезпечення якості включає контроль правильності встановлення та функціонування системи у реальних умовах.

Етап супроводу передбачає виправлення помилок, оновлення функціональності та адаптацію системи до змін у середовищі. Якість на цьому етапі визначається швидкістю реагування на проблеми, ефективністю внесення змін та стабільністю роботи системи після оновлень.

Управління якістю програмного забезпечення включає три основні складові: планування якості, забезпечення якості та контроль якості. Планування якості передбачає визначення стандартів, процедур та метрик, які будуть використовуватися у процесі розробки. Забезпечення якості спрямоване на запобігання дефектам шляхом впровадження ефективних процесів і методів роботи. Контроль якості включає виявлення дефектів та оцінювання відповідності продукту встановленим вимогам.

Важливим аспектом є використання підходу безперервного вдосконалення процесів. На основі аналізу результатів розробки та тестування здійснюється коригування процесів, впровадження нових методів та підвищення рівня зрілості організації. Це дає змогу поступово підвищувати якість програмного забезпечення та ефективність його розробки.

У сучасних умовах значну роль відіграють автоматизовані засоби управління якістю, такі як системи контролю версій, інструменти безперервної інтеграції та автоматизованого

тестування. Вони забезпечують оперативний контроль стану програмного продукту, зменшують ризик виникнення помилок та підвищують ефективність командної роботи.

Таким чином, міжнародні стандарти у сфері якості програмного забезпечення формують основу для побудови ефективних процесів управління якістю. Вони забезпечують узгодженість підходів, підвищують прозорість процесів та сприяють досягненню високого рівня якості програмних продуктів. Організація процесів управління якістю у межах життєвого циклу дає змогу інтегрувати діяльність з якості у всі етапи розробки та забезпечити створення надійних, ефективних і конкурентоспроможних програмних систем.

ТЕМА 2. ПРОЦЕСИ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Лекція 4. Процеси забезпечення якості програмного забезпечення

Забезпечення якості програмного забезпечення є системною діяльністю, що охоплює всі етапи життєвого циклу програмної системи та спрямована на досягнення відповідності продукту встановленим вимогам. У межах проєктів розроблення програмного забезпечення ці процеси інтегруються у загальну структуру управління проєктом і виконуються паралельно з процесами аналізу вимог, проєктування, реалізації, тестування та супроводу. Забезпечення якості не обмежується лише виявленням дефектів, а передусім спрямоване на їх попередження шляхом організації правильних процесів розробки.

Процеси забезпечення якості у проєктах розроблення програмного забезпечення формуються на основі стандартів, внутрішніх регламентів організації та вимог конкретного проєкту. Вони охоплюють планування, впровадження процедур контролю, виконання перевірок, аналіз результатів та вдосконалення процесів. Усі ці дії мають бути узгоджені між собою та підпорядковані єдиній меті – забезпеченню стабільної якості програмного продукту.

Однією з ключових складових є планування забезпечення якості програмного забезпечення. На цьому етапі визначаються цілі якості, критерії їх досягнення, методи контролю та інструменти, які будуть використовуватися у процесі розробки. План забезпечення якості є документом, що описує організацію робіт, відповідальність учасників, порядок виконання перевірок та способи фіксації результатів.

Під час планування необхідно враховувати специфіку проєкту, його масштаб, критичність, обмеження за ресурсами та термінами. Для складних або критично важливих систем встановлюються більш жорсткі вимоги до якості, що передбачає використання розширених процедур перевірки, багаторівневого тестування та формалізованих методів аналізу.

Важливим елементом планування є визначення метрик якості, які будуть використовуватися для оцінювання стану програмного продукту. Це можуть бути показники кількості дефектів, часу їх усунення, покриття тестами, складності коду або продуктивності системи. Наявність чітко визначених метрик дає змогу здійснювати об’єктивний контроль якості та приймати обґрунтовані рішення.

У межах плану також визначаються процедури верифікації та валідації. Верифікація спрямована на перевірку відповідності продукту технічним специфікаціям, тоді як валідація підтверджує, що програмна система відповідає потребам користувача. Ці процедури виконуються на різних етапах життєвого циклу і є важливими для забезпечення відповідності продукту як технічним, так і функціональним вимогам.

Контроль якості програмного забезпечення здійснюється на всіх етапах життєвого циклу. На етапі формування вимог контроль передбачає перевірку їх повноти, узгодженості та можливості перевірки. Використовуються методи рецензування документів, аналізу вимог та їх формалізації. Якість вимог безпосередньо впливає на якість кінцевого продукту, тому цей етап є критично важливим.

На етапі проєктування контроль якості спрямований на оцінювання архітектурних рішень, перевірку логіки взаємодії компонентів та відповідності обраної структури вимогам.

Проводяться технічні огляди, аналіз ризиків та перевірка альтернативних варіантів реалізації. Виявлення помилок на цьому етапі дає змогу значно зменшити витрати на їх виправлення.

Під час реалізації програмного забезпечення контроль якості включає рецензування коду, використання статичного аналізу та дотримання стандартів програмування. Рецензування коду дає змогу виявити логічні помилки, покращити структуру програми та підвищити її зрозумілість. Статичний аналіз дозволяє автоматично знаходити потенційні дефекти, такі як некоректне використання змінних або порушення правил безпеки.

На етапі тестування контроль якості набуває найбільшої інтенсивності. Виконуються різні види тестування, включаючи модульне, інтеграційне, системне та приймальне. Метою є перевірка правильності роботи програмного забезпечення, виявлення дефектів та оцінювання його поведінки у різних умовах. Результати тестування документуються та аналізуються для визначення рівня якості продукту.

На етапі впровадження контроль якості передбачає перевірку правильності розгортання системи, її сумісності з середовищем та відповідності очікуванням користувачів. Проводяться тестування у реальних умовах експлуатації, що дає змогу виявити проблеми, які не були помічені раніше.

У процесі супроводу контроль якості спрямований на забезпечення стабільності системи після внесення змін. Кожне оновлення програмного забезпечення повинно супроводжуватися перевіркою його впливу на існуючу функціональність. Використовуються регресійні тести, які дозволяють переконатися, що нові зміни не призвели до появи нових дефектів.

Ефективність процесів забезпечення якості значною мірою залежить від рівня їх автоматизації. Сучасні інструменти дозволяють автоматизувати збір метрик, виконання тестів, аналіз коду та контроль змін. Це дає змогу зменшити вплив людського фактора, підвищити швидкість перевірок та забезпечити стабільність процесів.

Важливим елементом є організація зворотного зв’язку у процесі забезпечення якості. Інформація про виявлені дефекти, результати тестування та показники метрик повинна використовуватися для вдосконалення процесів розробки. Це дає змогу поступово підвищувати рівень якості програмного забезпечення та ефективність роботи команди.

У проєктах розроблення програмного забезпечення забезпечення якості є відповідальністю не лише окремих спеціалістів, а всієї команди. Розробники, тестувальники, аналітики та менеджери повинні взаємодіяти для досягнення спільної мети. Такий підхід сприяє формуванню культури якості, у якій кожен учасник процесу усвідомлює свою роль у забезпеченні надійності та ефективності програмного продукту.

Таким чином, процеси забезпечення якості програмного забезпечення є невід’ємною складовою розробки програмних систем. Планування якості визначає напрям діяльності, контроль якості забезпечує виявлення дефектів, а інтеграція цих процесів у всі етапи життєвого циклу дає змогу створювати програмні продукти, що відповідають встановленим вимогам та очікуванням користувачів.

Лекція 5. Аудит, рецензування та статичний аналіз програмного забезпечення

Аудит програмного забезпечення є формалізованою процедурою незалежного оцінювання програмного продукту, процесів його розробки та супроводу з метою визначення відповідності встановленим вимогам, стандартам і регламентам. У межах інженерії

програмного забезпечення аудит використовується для підтвердження якості, виявлення невідповідностей та оцінювання рівня зрілості процесів. Він може проводитися як внутрішніми підрозділами організації, так і зовнішніми експертами, що забезпечує об'єктивність оцінювання.

Аудит програмного забезпечення охоплює перевірку технічної документації, програмного коду, процесів розробки, методів тестування та результатів виконання програмних систем. Основною метою є встановлення факту відповідності продукту вимогам, які були визначені на етапі формування специфікацій. Важливою складовою є також перевірка дотримання стандартів розробки, внутрішніх політик організації та вимог безпеки.

Процес аудиту передбачає кілька послідовних етапів. На початковому етапі визначається обсяг аудиту, його цілі та критерії оцінювання. Далі здійснюється збір інформації, який включає аналіз документації, інтерв'ю з учасниками проєкту та вивчення програмного коду. Наступним етапом є безпосереднє оцінювання відповідності, під час якого виявляються відхилення від встановлених вимог. Завершується аудит формуванням звіту, що містить результати перевірки, виявлені проблеми та рекомендації щодо їх усунення.

Аналіз відповідності програмних систем встановленим вимогам є ключовим завданням аудиту. Він передбачає зіставлення фактичних характеристик програмного продукту з вимогами, визначеними у технічному завданні, специфікаціях або стандартах. Такий аналіз може виконуватися як у процесі розробки, так і після завершення створення програмного продукту.

Важливою складовою є трасування вимог, тобто встановлення зв'язку між вимогами та відповідними елементами програмного забезпечення. Це дає змогу визначити, чи всі вимоги були реалізовані, а також оцінити повноту та коректність реалізації. У випадку виявлення невідповідностей здійснюється їх класифікація за рівнем критичності та визначаються заходи щодо усунення.

Рецензування програмного коду та технічної документації є ефективним методом забезпечення якості, що дозволяє виявляти дефекти на ранніх етапах розробки. Рецензування передбачає систематичний перегляд матеріалів іншими учасниками команди з метою виявлення помилок, покращення структури та підвищення якості.

Рецензування програмного коду включає перевірку правильності реалізації алгоритмів, відповідності стандартам програмування, ефективності використання ресурсів та зрозумілості коду. Особлива увага приділяється структурі програми, використанню змінних, обробці помилок та забезпеченню безпеки. Рецензування дозволяє не лише виявити дефекти, а й підвищити загальний рівень розробки за рахунок обміну досвідом між учасниками команди.

Існує кілька форм рецензування, серед яких неформальні перевірки, технічні огляди та інспекції. Неформальні перевірки виконуються без чітко визначеної процедури і зазвичай мають швидкий характер. Технічні огляди передбачають обговорення рішень у групі спеціалістів. Інспекції є найбільш формалізованим видом рецензування і виконуються за встановленою процедурою з фіксацією результатів.

Рецензування технічної документації спрямоване на перевірку її повноти, узгодженості та відповідності вимогам. Це включає аналіз специфікацій, описів архітектури, інструкцій користувача та іншої документації. Якість документації має важливе значення для супроводжуваності програмного забезпечення та ефективної взаємодії між учасниками проєкту.

Застосування методів статичного аналізу програмного забезпечення є важливим інструментом контролю якості, що дозволяє виявляти дефекти без виконання програмного коду. Статичний аналіз базується на дослідженні структури програми, її синтаксису та семантики з використанням автоматизованих інструментів.

Методи статичного аналізу дають змогу виявляти широкий спектр проблем, зокрема синтаксичні помилки, потенційні дефекти, порушення стандартів програмування, небезпечні конструкції та вразливості. Вони також використовуються для оцінювання складності коду, рівня його зв'язності та можливості подальшого супроводу.

Однією з переваг статичного аналізу є можливість його застосування на ранніх етапах розробки, що дозволяє зменшити витрати на виправлення помилок. Крім того, автоматизація процесу аналізу забезпечує високу швидкість перевірки та об'єктивність результатів.

Разом із тим, статичний аналіз має певні обмеження. Він не дозволяє виявити всі можливі дефекти, зокрема ті, що пов'язані з динамічною поведінкою програми або взаємодією з зовнішнім середовищем. Тому його доцільно використовувати у поєднанні з іншими методами забезпечення якості, зокрема тестуванням та рецензуванням.

У сучасних умовах розробки програмного забезпечення статичний аналіз інтегрується у процес безперервної інтеграції. Це дає змогу автоматично перевіряти код при кожній зміні, оперативно виявляти проблеми та забезпечувати стабільність програмного продукту. Використання таких підходів сприяє підвищенню якості та зменшенню кількості дефектів у кінцевому продукті.

Важливим елементом є поєднання аудиту, рецензування та статичного аналізу у єдину систему забезпечення якості. Аудит забезпечує незалежне оцінювання відповідності, рецензування дозволяє виявляти дефекти на ранніх етапах, а статичний аналіз забезпечує автоматизований контроль якості коду. Разом ці методи формують комплексний підхід до забезпечення якості програмного забезпечення.

Таким чином, аудит програмного забезпечення, аналіз відповідності вимогам, рецензування та статичний аналіз є важливими складовими системи забезпечення якості. Їх застосування дає змогу підвищити надійність, ефективність та супроводжуваність програмних систем, а також забезпечити відповідність встановленим вимогам і стандартам.

Лекція 6. Управління конфігураціями та змінами програмного забезпечення

Управління конфігураціями програмного забезпечення є одним із ключових процесів інженерії програмного забезпечення, що забезпечує контроль за складом, станом і змінами програмного продукту протягом усього життєвого циклу. Воно спрямоване на підтримання цілісності програмної системи, узгодженості її компонентів та можливості відтворення будь-якої версії продукту у визначений момент часу. У складних проєктах, де розробка виконується командою спеціалістів і супроводжується численними змінами, роль управління конфігураціями є визначальною.

Конфігурація програмного забезпечення є сукупністю всіх елементів, що входять до складу програмної системи, включаючи вихідний код, виконувані файли, бібліотеки, документацію, конфігураційні файли та інші артефакти. Кожен із цих елементів підлягає обліку, контролю та управлінню. Важливою умовою є ідентифікація конфігураційних одиниць, тобто визначення тих елементів, зміни в яких мають відслідковуватися та контролюватися.

Процес управління конфігураціями включає кілька основних дій: ідентифікацію конфігураційних елементів, контроль змін, облік стану конфігурації та аудит конфігурації. Ідентифікація передбачає визначення структури програмного продукту та виділення його складових. Контроль змін забезпечує впорядковане внесення змін до системи. Облік стану дозволяє відслідковувати поточний стан кожного елемента. Аудит конфігурації спрямований на перевірку відповідності фактичного стану системи задокументованому.

Управління конфігураціями тісно пов’язане з управлінням змінами, яке визначає порядок внесення, аналізу та затвердження змін у програмному проєкті. Зміни можуть виникати з різних причин: уточнення вимог, виправлення дефектів, оптимізація продуктивності, адаптація до нових умов або впровадження нових функцій. Без належного управління змінами такі процеси можуть призвести до втрати контролю над проєктом, появи несумісностей та зниження якості програмного забезпечення.

Управління змінами передбачає формалізований підхід до обробки кожної зміни. Спочатку формується запит на зміну, який містить опис проблеми або пропозиції, обґрунтування необхідності змін та оцінку їх впливу. Далі виконується аналіз цього запиту, що включає оцінювання впливу на функціональність, терміни, ресурси та ризики. Після цього приймається рішення щодо доцільності внесення змін, і у разі позитивного рішення визначається порядок їх реалізації.

Важливим елементом є відстеження змін, яке дозволяє контролювати процес їх виконання та забезпечувати прозорість. Усі зміни повинні бути задокументовані, що дає змогу аналізувати історію розвитку програмного продукту та відновлювати попередні стани у разі необхідності. Це особливо важливо у випадках виникнення помилок або необхідності повернення до стабільної версії системи.

У процесі управління змінами важливу роль відіграє взаємодія між учасниками проєкту. Розробники, тестувальники, аналітики та менеджери повинні узгоджувати свої дії, щоб забезпечити правильність реалізації змін та мінімізувати ризики. Використання формалізованих процедур дозволяє уникнути хаотичного внесення змін і забезпечити стабільність системи.

Застосування систем контролю версій є основним інструментом реалізації управління конфігураціями та змінами. Система контролю версій дозволяє зберігати історію змін програмного коду, відслідковувати внесені модифікації, координувати роботу команди та забезпечувати можливість повернення до попередніх версій.

Системи контролю версій поділяються на централізовані та розподілені. Централізовані системи передбачають наявність єдиного сховища, до якого підключаються всі учасники проєкту. Розподілені системи дозволяють кожному учаснику мати локальну копію повного репозиторію, що забезпечує більшу гнучкість та незалежність у роботі.

Основними функціями систем контролю версій є фіксація змін, управління гілками розробки, об’єднання змін та вирішення конфліктів. Фіксація змін дозволяє зберігати стан програмного коду у певний момент часу. Гілки розробки дають змогу працювати над різними функціональними можливостями незалежно один від одного. Об’єднання змін забезпечує інтеграцію результатів роботи різних учасників. Вирішення конфліктів є необхідним у випадках, коли кілька розробників вносять зміни до одних і тих самих частин коду.

У практиці розробки програмного забезпечення широко використовуються сучасні системи контролю версій, такі як Git, Subversion та Mercurial. Найбільш поширеною є

система Git, яка підтримує розподілену модель та надає широкі можливості для організації командної роботи.

Використання систем контролю версій дозволяє реалізувати ефективні підходи до організації розробки, зокрема гілкові моделі. Однією з поширених є модель Git Flow, яка передбачає використання окремих гілок для розробки, тестування та стабільних версій. Інший підхід – використання короткоживучих гілок для окремих задач, що інтегруються у основну гілку після завершення роботи.

Інтеграція систем контролю версій із засобами безперервної інтеграції дозволяє автоматизувати процес перевірки змін. При кожному оновленні коду запускаються автоматичні тести, виконується аналіз якості та перевіряється відповідність встановленим вимогам. Це забезпечує швидке виявлення дефектів та підтримання стабільності програмного продукту.

Важливим є також управління версіями програмного забезпечення, яке передбачає присвоєння унікальних ідентифікаторів різним станам системи. Версії можуть позначатися за допомогою числових або комбінованих схем, що відображають рівень змін у продукті. Це дозволяє користувачам та розробникам чітко ідентифікувати стан системи та відслідковувати її розвиток.

Таким чином, управління конфігураціями та змінами програмного забезпечення є невід’ємною складовою сучасної розробки програмних систем. Воно забезпечує контроль за складом продукту, впорядкованість змін та можливість відтворення різних версій. Застосування систем контролю версій дозволяє ефективно організувати командну роботу, підвищити якість програмного забезпечення та зменшити ризики, пов’язані з розробкою.

ТЕМА 3. ОСНОВИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Лекція 7. Основи тестування програмного забезпечення

Тестування програмного забезпечення є систематичним процесом перевірки та оцінювання програмного продукту з метою виявлення дефектів, підтвердження відповідності встановленим вимогам і оцінювання його якості. У межах інженерії програмного забезпечення тестування виконує критично важливу роль, оскільки дозволяє зменшити ризики використання некоректного або нестабільного програмного продукту. При цьому тестування не доводить повну відсутність помилок, а лише підтверджує їх наявність у перевірених умовах.

Основною метою тестування є забезпечення впевненості у тому, що програмне забезпечення працює відповідно до заданих вимог і очікувань користувача. Для досягнення цієї мети використовуються різні методи, підходи та рівні тестування, які охоплюють як окремі компоненти системи, так і систему в цілому. Тестування є невід'ємною частиною процесу забезпечення якості і повинно виконуватися протягом усього життєвого циклу програмного забезпечення.

У контексті тестування важливими є поняття верифікації та валідації. Верифікація передбачає перевірку відповідності програмного продукту технічним специфікаціям і вимогам, визначеним на етапі проєктування. Вона відповідає на питання, чи правильно реалізовано систему відповідно до заданих вимог. Верифікація може включати аналіз документації, рецензування коду, статичний аналіз та виконання тестів.

Валідація, у свою чергу, спрямована на перевірку того, чи відповідає програмне забезпечення реальним потребам користувачів. Вона відповідає на питання, чи створено правильний продукт. Валідація виконується шляхом тестування системи у умовах, максимально наближених до реальних, та аналізу результатів її роботи з точки зору кінцевого користувача.

Різниця між верифікацією та валідацією полягає у спрямованості перевірки. Верифікація орієнтована на внутрішню відповідність вимогам, тоді як валідація – на зовнішню відповідність потребам. Обидва ці процеси є взаємодоповнюючими і необхідними для забезпечення якості програмного забезпечення.

Місце тестування у життєвому циклі програмного забезпечення визначається моделлю розробки, яка використовується у проєкті. У класичних моделях тестування розглядається як окремий етап, що виконується після завершення реалізації. Проте у сучасних підходах тестування інтегрується у всі етапи життєвого циклу і виконується паралельно з іншими процесами.

На етапі формування вимог тестування проявляється у вигляді аналізу вимог та перевірки їх якості. На етапі проєктування виконуються перевірки архітектурних рішень і розробляються тестові сценарії. Під час реалізації програмного забезпечення проводиться модульне тестування окремих компонентів. На етапі інтеграції виконується перевірка взаємодії між компонентами. Під час системного тестування оцінюється робота всієї системи. На етапі приймального тестування перевіряється відповідність продукту вимогам користувача.

Тестування вимог програмного забезпечення є важливою складовою процесу забезпечення якості. Воно передбачає перевірку вимог на повноту, узгодженість,

однозначність та можливість перевірки. Нечіткі або суперечливі вимоги можуть призвести до неправильного розуміння задачі та створення програмного продукту, що не відповідає очікуванням.

Одним із ключових понять у цьому контексті є тестованість вимог. Тестованість визначає, наскільки легко можна перевірити виконання вимоги за допомогою тестів. Вимога вважається тестованою, якщо для неї можна сформулювати чіткі критерії прийняття, які дозволяють однозначно визначити, чи виконана вона.

Для забезпечення тестованості вимог необхідно дотримуватися певних принципів. Вимоги повинні бути конкретними, вимірюваними, однозначними та перевірюваними. Наприклад, вимога типу система повинна працювати швидко є неконкретною, оскільки не містить чітких критеріїв оцінювання. Натомість вимога система повинна обробляти запит не більше ніж за 2 секунди є тестованою, оскільки дозволяє виконати перевірку.

Аналіз тестованості вимог включає оцінювання їх структури, формулювання та можливості побудови тестових сценаріїв. Для цього можуть використовуватися різні методи, такі як рецензування вимог, створення тестових випадків на їх основі та перевірка їх відповідності критеріям якості.

Важливим інструментом є трасування вимог, яке дозволяє встановити зв’язок між вимогами, тестами та реалізацією. Це дає змогу контролювати покриття вимог тестами і забезпечує впевненість у тому, що всі вимоги були перевірені. У разі змін у вимогах трасування дозволяє швидко визначити, які частини системи та тести потребують оновлення.

Тестування вимог також дозволяє виявити помилки на ранніх етапах розробки, що значно зменшує витрати на їх виправлення. Відомо, що вартість виправлення дефектів зростає зі збільшенням етапу життєвого циклу, на якому вони були виявлені. Тому раннє тестування є економічно доцільним і сприяє підвищенню якості програмного забезпечення.

У сучасних підходах до розробки програмного забезпечення широко застосовується практика тестування, орієнтованого на вимоги. Вона передбачає розробку тестів на основі вимог ще до початку реалізації програмного продукту. Це дозволяє чітко визначити критерії прийняття і забезпечити відповідність реалізації очікуванням.

Таким чином, тестування програмного забезпечення є комплексним процесом, що охоплює перевірку продукту на різних етапах його створення. Верифікація та валідація забезпечують перевірку відповідності як технічним вимогам, так і потребам користувачів. Місце тестування у життєвому циклі визначається інтеграцією у всі етапи розробки. Тестування вимог і аналіз їх тестованості дозволяють забезпечити якість програмного продукту на ранніх етапах і створюють основу для ефективного тестування у подальшому.

Лекція 8. Рівні тестування та організація процесу тестування

Тестування програмного забезпечення є багаторівневим процесом, у межах якого перевірка виконується на різних рівнях деталізації програмної системи. Такий підхід дозволяє послідовно виявляти дефекти, починаючи з окремих елементів і завершуючи перевіркою всієї системи в цілому. Кожен рівень тестування має власну мету, методи та область застосування, а їх поєднання забезпечує повноту перевірки програмного продукту.

Основними рівнями тестування є модульне, інтеграційне, системне та приймальне тестування. Ці рівні відповідають етапам формування програмної системи і логічно

доповнюють один одного. Правильна організація тестування передбачає їх послідовне або паралельне застосування залежно від моделі розробки.

Модульне тестування є початковим рівнем, на якому перевіряються окремі програмні компоненти, такі як функції, методи або класи. Метою цього рівня є підтвердження правильності реалізації логіки кожного модуля незалежно від інших частин системи. Модульне тестування дозволяє виявити помилки на ранніх етапах розробки, що значно зменшує витрати на їх виправлення.

Характерною особливістю модульного тестування є ізольованість об'єкта тестування. Для цього використовуються допоміжні засоби, які імітують взаємодію з іншими компонентами системи. Це дозволяє перевірити поведінку модуля у контрольованих умовах і точно визначити джерело помилки. Модульні тести зазвичай виконуються розробниками і часто автоматизуються.

Інтеграційне тестування виконується після модульного і спрямоване на перевірку взаємодії між компонентами системи. На цьому рівні перевіряється правильність обміну даними, узгодженість інтерфейсів та коректність роботи інтегрованих модулів. Основною метою є виявлення дефектів, що виникають у процесі взаємодії компонентів.

Існує кілька підходів до інтеграційного тестування, серед яких послідовне об'єднання модулів, тестування зверху вниз, знизу вгору або комбіновані підходи. Вибір підходу залежить від архітектури системи, складності проєкту та доступності компонентів. Інтеграційне тестування дозволяє виявити проблеми, які не можуть бути виявлені на рівні окремих модулів.

Системне тестування є рівнем, на якому перевіряється вся програмна система як єдине ціле. Воно виконується у середовищі, максимально наближеному до реальних умов експлуатації. Метою є оцінювання відповідності системи функціональним і нефункціональним вимогам, включаючи продуктивність, надійність та зручність використання.

На цьому рівні виконуються різні види тестування, зокрема функціональне, навантажувальне, стресове та інші. Системне тестування дозволяє оцінити поведінку програмного забезпечення у комплексі та виявити дефекти, що проявляються лише при взаємодії всіх компонентів.

Приймальне тестування є завершальним рівнем і виконується з метою підтвердження готовності програмного продукту до використання. Воно проводиться замовником або кінцевими користувачами і базується на вимогах, визначених у технічному завданні. Основною метою є перевірка того, чи відповідає програмне забезпечення очікуванням користувача.

Приймальне тестування може включати перевірку функціональності, зручності використання та відповідності бізнес-вимогам. Результати цього тестування є підставою для прийняття рішення про впровадження програмного продукту. У разі виявлення невідповідностей система може бути повернена на доопрацювання.

Рівні тестування не існують ізольовано, а утворюють єдину систему перевірки якості. Помилки, які не були виявлені на ранніх рівнях, можуть проявитися на пізніших, що ускладнює їх виправлення. Тому важливо забезпечити повноцінне виконання тестування на кожному рівні.

Організація процесу тестування у програмному проєкті є важливою складовою управління якістю. Вона включає планування тестування, визначення відповідальних осіб, вибір методів і інструментів, а також контроль виконання тестових робіт. Ефективна

організація дозволяє забезпечити своєчасне виявлення дефектів і підвищити якість програмного продукту.

Планування тестування передбачає визначення цілей тестування, обсягу робіт, ресурсів і термінів виконання. На цьому етапі формується стратегія тестування, яка визначає підходи до перевірки програмного забезпечення, рівні тестування та порядок їх виконання. Також визначаються критерії завершення тестування.

Важливим елементом є розробка тестових сценаріїв і тестових випадків, які описують умови виконання тестів, вхідні дані та очікувані результати. Якість тестових сценаріїв безпосередньо впливає на ефективність тестування, оскільки від цього залежить повнота перевірки програмного продукту.

Організація тестування передбачає також управління дефектами. Виявлені дефекти фіксуються у спеціалізованих системах, де відслідковується їх статус, пріоритет і відповідальні за їх усунення. Це дозволяє контролювати процес виправлення помилок і забезпечувати прозорість роботи команди.

У сучасних проєктах широко застосовується автоматизація тестування, яка дозволяє зменшити час виконання тестів і підвищити їх точність. Автоматизовані тести можуть виконуватися регулярно, що забезпечує постійний контроль якості програмного забезпечення. Особливо ефективною є автоматизація модульного та інтеграційного тестування.

Інтеграція тестування у процес безперервної інтеграції дозволяє автоматично перевіряти зміни у коді та оперативно виявляти дефекти. Це забезпечує стабільність програмного продукту та дозволяє швидко реагувати на проблеми.

Ефективність процесу тестування залежить від взаємодії між учасниками проєкту. Розробники, тестувальники та аналітики повинні координувати свої дії, щоб забезпечити повноту перевірки та правильність інтерпретації результатів. Важливо також забезпечити наявність чіткої документації, яка описує процес тестування та його результати.

Таким чином, рівні тестування програмного забезпечення забезпечують поетапну перевірку програмного продукту від окремих компонентів до системи в цілому. Організація процесу тестування визначає ефективність цієї діяльності і впливає на якість кінцевого результату. Комплексний підхід до тестування дозволяє забезпечити відповідність програмного забезпечення встановленим вимогам і очікуванням користувачів.

Лекція 9. Життєвий цикл тестування та документування

Життєвий цикл тестування програмного забезпечення є впорядкованою послідовністю дій, що охоплює планування, підготовку, виконання та аналіз результатів тестування. Він визначає структуру роботи тестувальників у межах проєкту та забезпечує системність перевірки програмного продукту. Життєвий цикл тестування тісно пов’язаний із життєвим циклом розробки програмного забезпечення і виконується паралельно з ним.

Життєвий цикл тестування включає кілька основних етапів: аналіз вимог, планування тестування, розробка тестових матеріалів, виконання тестування, аналіз результатів та завершення тестування. Кожен із цих етапів має власну мету та набір дій, які забезпечують ефективність процесу тестування.

На етапі аналізу вимог виконується оцінювання їх якості та визначення можливості тестування. Вимоги аналізуються на предмет повноти, узгодженості та наявності критеріїв

перевірки. На основі цього формується розуміння обсягу тестування та визначаються основні напрямки перевірки програмного забезпечення.

Планування тестування є одним із ключових етапів життєвого циклу. На цьому етапі визначаються цілі тестування, обсяг робіт, ресурси, строки виконання та відповідальні особи. Формується стратегія тестування, яка описує підходи до перевірки, рівні тестування, методи та інструменти. Результатом є план тестування, який використовується як основний документ для організації тестових робіт.

У процесі планування також визначаються критерії початку та завершення тестування, що дозволяє контролювати хід виконання робіт. Важливим є визначення ризиків, пов'язаних із тестуванням, та розробка заходів для їх мінімізації. Це дає змогу забезпечити стабільність процесу та уникнути затримок у виконанні проєкту.

Після планування виконується підготовка тестових сценаріїв та тестових випадків. Тестовий сценарій описує загальну послідовність дій користувача або системи, що підлягає перевірці. Тестовий випадок є більш деталізованим і містить конкретні вхідні дані, умови виконання, кроки тестування та очікувані результати.

Підготовка тестових випадків є важливим етапом, оскільки від їх якості залежить ефективність тестування. Тестові випадки повинні бути чіткими, однозначними та відтворюваними. Вони повинні покривати всі основні сценарії використання системи, включаючи як стандартні, так і граничні умови. Особлива увага приділяється перевірці критичних функцій, від яких залежить працездатність системи.

Для забезпечення повноти тестування використовуються різні техніки розробки тестових випадків, такі як розбиття на класи еквівалентності, аналіз граничних значень, комбінаторні методи та інші. Це дозволяє оптимізувати кількість тестів і забезпечити достатнє покриття функціональності системи.

Після підготовки тестових матеріалів виконується безпосереднє тестування. На цьому етапі тестові випадки виконуються у визначеному середовищі, результати фіксуються та порівнюються з очікуваними. У разі виявлення відхилень формуються записи про дефекти, які передаються на виправлення.

Документування результатів тестування є важливою складовою процесу, що забезпечує фіксацію отриманих результатів і можливість їх подальшого аналізу. Результати тестування оформлюються у вигляді звітів, які містять інформацію про виконані тести, виявлені дефекти, рівень покриття та загальний стан якості програмного продукту.

Документація тестування може включати такі елементи, як план тестування, тестові сценарії, тестові випадки, звіти про виконання тестів, звіти про дефекти та підсумкові звіти. Наявність повної та структурованої документації дозволяє забезпечити прозорість процесу тестування та полегшує взаємодію між учасниками проєкту.

Важливим є також аналіз результатів тестування, який дозволяє оцінити рівень якості програмного забезпечення та визначити необхідність подальших дій. Аналіз включає оцінювання кількості та критичності дефектів, визначення тенденцій їх виникнення та оцінювання ефективності тестування. На основі цього приймаються рішення щодо готовності продукту до наступного етапу або впровадження.

Завершення тестування передбачає підбиття підсумків, підготовку фінального звіту та архівування тестових матеріалів. На цьому етапі оцінюється досягнення цілей тестування, визначається відповідність продукту встановленим вимогам і формується висновок щодо його готовності до використання.

У сучасних умовах життєвий цикл тестування часто інтегрується у процеси безперервної інтеграції та безперервного постачання. Це дозволяє автоматизувати виконання тестів, забезпечити оперативне виявлення дефектів та підвищити ефективність розробки. Автоматизація особливо ефективна для повторюваних тестів, що виконуються при кожній зміні програмного коду.

Ефективність життєвого циклу тестування залежить від чіткої організації процесів, використання відповідних інструментів та взаємодії між учасниками проєкту. Важливо забезпечити узгодженість між вимогами, тестовими матеріалами та результатами тестування, що дозволяє досягти високого рівня якості програмного забезпечення.

Таким чином, життєвий цикл тестування програмного забезпечення визначає структуру та послідовність виконання тестових робіт. Планування тестування забезпечує визначення цілей і ресурсів, підготовка тестових сценаріїв і тестових випадків створює основу для перевірки, а документування результатів дозволяє оцінити якість програмного продукту та прийняти обґрунтовані рішення щодо його подальшого використання.

ТЕМА 4. МЕТОДИ ТА ТЕХНІКИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Лекція 10. Методи тестування програмного забезпечення

Методи тестування програмного забезпечення визначають підходи до перевірки правильності функціонування програмних систем і є основою для побудови ефективного процесу тестування. Класифікація методів тестування дозволяє систематизувати різні способи перевірки програмного продукту залежно від способу аналізу, рівня доступу до внутрішньої структури та характеру виконання тестів. Правильний вибір методів тестування забезпечує повноту перевірки та підвищує ймовірність виявлення дефектів.

Існує кілька підходів до класифікації методів тестування. Одним із основних є поділ на статичне та динамічне тестування. Цей поділ базується на тому, чи виконується програмний код у процесі перевірки.

Статичне тестування передбачає аналіз програмного забезпечення без його виконання. Воно включає перевірку програмного коду, технічної документації, специфікацій та інших артефактів розробки. Основною метою є виявлення помилок на ранніх етапах, коли їх виправлення є менш затратним. До методів статичного тестування належать рецензування, інспекції, аналіз документації та використання засобів статичного аналізу коду.

Перевагою статичного тестування є можливість раннього виявлення дефектів, зокрема логічних помилок, порушень стандартів програмування та потенційних проблем безпеки. Воно не потребує виконання програми, що дозволяє застосовувати його ще до завершення реалізації. Разом із тим, статичне тестування не дозволяє оцінити поведінку системи у процесі виконання, тому повинно доповнюватися іншими методами.

Динамічне тестування виконується шляхом запуску програмного забезпечення та аналізу його поведінки у різних умовах. Воно дозволяє перевірити правильність виконання функцій, взаємодію компонентів та реакцію системи на різні вхідні дані. Динамічне тестування є основним методом перевірки працездатності програмного продукту.

У межах динамічного тестування використовуються різні підходи, серед яких важливе місце займає тестування типу чорної скриньки. Цей підхід передбачає перевірку програмного забезпечення без урахування його внутрішньої структури. Тестувальник аналізує лише вхідні дані та результати роботи системи, не враховуючи реалізацію алгоритмів.

Тестування типу чорної скриньки базується на вимогах до програмного забезпечення і спрямоване на перевірку відповідності системи очікуваним результатам. Воно дозволяє оцінити функціональність програмного продукту з точки зору користувача та виявити невідповідності між очікуваною і фактичною поведінкою системи.

Серед основних методів тестування типу чорної скриньки виділяють розбиття на класи еквівалентності. Цей метод передбачає поділ множини вхідних даних на групи, для яких очікується однакова поведінка системи. Замість перевірки всіх можливих значень достатньо протестувати по одному представнику з кожного класу. Це дозволяє значно скоротити кількість тестів без втрати ефективності.

Іншим важливим методом є аналіз граничних значень. Він базується на тому, що помилки часто виникають на межах допустимих значень вхідних даних. Тому перевірка

значень, що знаходяться на межах або поблизу них, дозволяє виявити дефекти, які можуть залишитися непоміченими при використанні інших методів.

Метод таблиць рішень використовується для перевірки складних логічних умов. Він передбачає побудову таблиці, у якій відображаються всі можливі комбінації вхідних умов і відповідні їм результати. Це дозволяє забезпечити повне покриття логіки системи та виявити помилки у реалізації умов.

Метод причинно-наслідкових зв'язків дозволяє встановити залежність між вхідними умовами та результатами роботи системи. Він використовується для побудови тестових випадків, які враховують різні комбінації факторів, що впливають на поведінку програмного забезпечення.

Метод тестування переходів станів застосовується для систем, поведінка яких залежить від попереднього стану. Він передбачає перевірку переходів між станами та правильності реакції системи на різні події. Цей метод є ефективним для тестування систем керування, протоколів та інших складних систем.

Також використовується метод тестування сценаріїв використання, який базується на моделюванні реальних дій користувача. Тестові сценарії відображають типові або критичні ситуації використання системи і дозволяють перевірити її поведінку у реальних умовах.

Методи тестування типу чорної скриньки мають ряд переваг. Вони не потребують знання внутрішньої структури програми, орієнтовані на вимоги та дозволяють оцінити систему з точки зору користувача. Разом із тим, вони не забезпечують повного покриття коду і можуть пропустити деякі дефекти, пов'язані з внутрішньою логікою програми.

У практиці розробки програмного забезпечення методи тестування часто комбінуються. Поєднання статичного та динамічного тестування, а також використання різних підходів дозволяє забезпечити більш повну перевірку програмного продукту. Вибір конкретних методів залежить від типу системи, її складності та вимог до якості.

Таким чином, класифікація методів тестування програмного забезпечення дозволяє систематизувати підходи до перевірки та вибрати найбільш ефективні методи для конкретного проєкту. Статичне та динамічне тестування доповнюють одне одного, а тестування типу чорної скриньки забезпечує перевірку функціональності програмного забезпечення відповідно до встановлених вимог.

Лекція 11. Методи проєктування тестів та аналіз результатів

Проєктування тестів є ключовим етапом процесу тестування програмного забезпечення, оскільки саме на цьому етапі визначається, які перевірки будуть виконані, які дані будуть використані та які результати вважаються правильними. Ефективність тестування значною мірою залежить від правильного вибору методів проєктування тестів, які дозволяють забезпечити достатнє покриття функціональності при обмеженій кількості тестових випадків.

Методи проєктування тестів спрямовані на систематичне формування тестових випадків на основі вимог до програмного забезпечення. Вони дозволяють уникнути хаотичного підходу до тестування і забезпечити логічно обґрунтований вибір вхідних даних. Найбільш поширеними є метод еквівалентних класів, метод граничних значень, тестування умов та тестування переходів станів.

Метод еквівалентних класів базується на припущенні, що всі значення в межах певного класу вхідних даних обробляються системою однаково. У межах цього методу всі можливі вхідні дані розбиваються на групи, які називаються класами еквівалентності. Для кожного класу достатньо вибрати один або кілька представників, щоб перевірити поведінку системи.

Класи еквівалентності поділяються на допустимі та недопустимі. Допустимі класи відповідають правильним вхідним даним, які система повинна обробляти без помилок. Недопустимі класи містять дані, що порушують встановлені обмеження, і використовуються для перевірки обробки помилкових ситуацій. Такий підхід дозволяє суттєво зменшити кількість тестових випадків без втрати якості перевірки.

Метод граничних значень є доповненням до методу еквівалентних класів і спрямований на перевірку значень, що знаходяться на межах допустимих інтервалів. Практика показує, що найбільша кількість помилок виникає саме на межах діапазонів, тому перевірка цих значень є особливо важливою.

Для застосування методу граничних значень визначаються мінімальні та максимальні допустимі значення, а також значення, що знаходяться безпосередньо за межами цих діапазонів. Наприклад, якщо допустимий діапазон значень становить від 1 до 100, доцільно перевірити значення 1, 100, а також значення 0 і 101. Це дозволяє виявити помилки, пов’язані з некоректною обробкою граничних умов.

Тестування умов використовується для перевірки логічних виразів, що визначають поведінку програмного забезпечення. Воно спрямоване на забезпечення правильності виконання умовних операторів та логічних конструкцій. У межах цього підходу перевіряються різні комбінації значень умов, що дозволяє виявити помилки у логіці програми.

Особливу увагу приділяють перевірці складних умов, які містять кілька логічних змінних. Для цього можуть використовуватися таблиці істинності або спеціальні методи, що забезпечують покриття всіх можливих комбінацій умов. Це дозволяє гарантувати правильність роботи програмного забезпечення у різних ситуаціях.

Тестування переходів станів застосовується для систем, поведінка яких залежить від попереднього стану. Такі системи описуються у вигляді моделей станів, де визначено можливі стани та переходи між ними. Метод передбачає перевірку правильності переходів між станами та реакції системи на події.

У процесі тестування переходів станів визначаються всі можливі стани системи, події, що викликають переходи, та очікувані результати. На основі цього формуються тестові випадки, які перевіряють як коректні, так і некоректні переходи. Це дозволяє виявити помилки, пов’язані з неправильним управлінням станами.

Ефективність тестування значною мірою залежить не лише від правильного проєктування тестів, а й від аналізу результатів тестування. Аналіз результатів дозволяє оцінити якість програмного забезпечення, визначити рівень його готовності та прийняти рішення щодо подальших дій.

Аналіз результатів тестування включає оцінювання відповідності фактичних результатів очікуваним, визначення кількості та критичності дефектів, а також аналіз їх причин. Важливим є класифікація дефектів за рівнем впливу на роботу системи, що дозволяє визначити пріоритети їх усунення.

Крім того, аналіз результатів включає оцінювання покриття тестами, тобто визначення того, яка частина функціональності була перевірена. Недостатнє покриття може

свідчити про необхідність додаткових тестів. Також аналізуються тенденції виникнення дефектів, що дозволяє виявити проблемні області системи.

Важливим елементом є документування результатів тестування, що забезпечує можливість їх подальшого використання. Звіти про тестування містять інформацію про виконані тести, виявлені дефекти та загальний стан системи. Це дозволяє забезпечити прозорість процесу та обґрунтованість прийнятих рішень.

У сучасних проєктах аналіз результатів тестування часто автоматизується за допомогою спеціалізованих інструментів, які дозволяють збирати статистику, будувати звіти та відслідковувати динаміку якості програмного забезпечення. Це підвищує ефективність процесу та дозволяє оперативно реагувати на проблеми.

Таким чином, методи проєктування тестів забезпечують систематичний підхід до формування тестових випадків і дозволяють ефективно перевіряти програмне забезпечення. Метод еквівалентних класів і метод граничних значень дозволяють оптимізувати кількість тестів, тестування умов і переходів станів забезпечує перевірку логіки та поведінки системи, а аналіз результатів тестування дозволяє оцінити якість програмного продукту та визначити напрямки його вдосконалення.

Лекція 12. Методи «білої скриньки» та покриття коду

Методи тестування типу «білої скриньки» базуються на аналізі внутрішньої структури програмного забезпечення і передбачають використання знань про реалізацію алгоритмів, структуру коду та логіку виконання програми. На відміну від підходу «чорної скриньки», де тестування виконується без урахування внутрішньої будови системи, у даному випадку тестувальник або розробник працює безпосередньо з програмним кодом. Це дозволяє забезпечити більш глибоку перевірку та виявити дефекти, які можуть залишитися непоміченими при зовнішньому аналізі.

Основною метою методів «білої скриньки» є перевірка правильності реалізації алгоритмів, структури керування програмою та логіки виконання. Тестування спрямоване на забезпечення проходження різних шляхів виконання програми та перевірку коректності обробки умов, циклів і переходів.

Одним із ключових елементів такого тестування є аналіз структури програмного коду. Він передбачає дослідження логіки виконання програми, зокрема послідовності операторів, умовних переходів, циклів та викликів функцій. Для цього часто використовуються графи керування, які відображають можливі шляхи виконання програми. Такий підхід дозволяє визначити, які частини коду були перевірені, а які залишилися непокритими.

Аналіз структури коду дає змогу виявити недосяжні ділянки програми, надлишкові або дубльовані фрагменти, а також потенційно небезпечні конструкції. Крім того, він дозволяє оцінити складність програмного забезпечення, що є важливим для планування тестування та визначення необхідної кількості тестових випадків.

У межах тестування «білої скриньки» використовуються різні критерії покриття коду, які визначають рівень перевірки програми. Критерій покриття показує, яка частина програмного коду була виконана під час тестування. Чим вищий рівень покриття, тим більш повною є перевірка програмного забезпечення.

Одним із базових критеріїв є покриття операторів. Він передбачає, що кожен оператор програми повинен бути виконаний хоча б один раз під час тестування. Це найпростіший

критерій, який дозволяє перевірити основні частини коду, але не гарантує повного покриття логіки, оскільки не враховує всі можливі варіанти виконання умов.

Більш строгим є критерій покриття гілок. Він вимагає, щоб кожна гілка умовного переходу була виконана хоча б один раз. Це означає, що для кожної умови повинні бути перевірені як істинний, так і хибний варіанти. Такий підхід дозволяє більш повно перевірити логіку програми та виявити помилки, пов’язані з неправильними умовами.

Ще більш детальним є критерій покриття умов. Він передбачає перевірку всіх можливих значень логічних виразів, що входять до складу умов. Це особливо важливо для складних умов, які містять кілька логічних змінних. Покриття умов дозволяє виявити помилки у логічних виразах, які можуть не проявитися при використанні інших критеріїв.

У практиці тестування також можуть використовуватися комбіновані критерії, які поєднують різні підходи до покриття. Наприклад, критерій покриття умов і гілок дозволяє забезпечити більш повну перевірку логіки програми. Вибір конкретного критерію залежить від складності системи, вимог до якості та доступних ресурсів.

Важливо зазначити, що досягнення 100% покриття коду не гарантує відсутності дефектів. Покриття лише показує, які частини програми були виконані, але не гарантує правильність результатів. Тому критерії покриття повинні використовуватися разом із іншими методами тестування.

Методи «білої скриньки» часто застосовуються на рівні модульного тестування, де розробники мають доступ до програмного коду і можуть детально перевірити його логіку. Вони також використовуються для оптимізації коду, виявлення неефективних конструкцій та підвищення його якості.

У сучасних умовах для оцінювання покриття коду широко використовуються автоматизовані інструменти, які дозволяють збирати інформацію про виконання програми під час тестування. Це дає змогу швидко визначити непокриті ділянки коду та прийняти рішення щодо необхідності додаткових тестів.

Ефективне використання методів «білої скриньки» потребує глибокого розуміння програмної логіки, структури коду та принципів побудови алгоритмів. Це робить їх особливо корисними для розробників, які можуть використовувати ці методи для підвищення якості власного коду.

Таким чином, методи тестування типу «білої скриньки» забезпечують детальний аналіз внутрішньої структури програмного забезпечення і дозволяють перевірити правильність реалізації алгоритмів. Аналіз структури програмного коду та використання критеріїв покриття, таких як покриття операторів, гілок і умов, дають змогу оцінити повноту тестування та підвищити якість програмного продукту.

ТЕМА 5. АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Лекція 13. Автоматизоване тестування програмного забезпечення

Автоматизоване тестування програмного забезпечення є підходом до перевірки програмних систем, за якого виконання тестів здійснюється з використанням програмних засобів без безпосередньої участі людини під час кожного запуску. У цьому підході тестові сценарії, перевірки результатів і аналіз виконання реалізуються у вигляді програмного коду або конфігурацій, що дозволяє багаторазово відтворювати перевірки з однаковими умовами.

Основною метою автоматизованого тестування є підвищення ефективності процесу тестування, зменшення витрат часу на повторювані перевірки та забезпечення стабільності якості програмного продукту. Особливо актуальним цей підхід є у проєктах, де відбуваються часті зміни коду, і виникає потреба у регулярному виконанні великої кількості тестів.

Автоматизація тестування передбачає створення тестових скриптів або використання спеціалізованих інструментів, які виконують задані перевірки. Такі скрипти можуть включати підготовку тестових даних, виконання дій у програмній системі, отримання результатів та їх порівняння з очікуваними значеннями. Результати виконання тестів фіксуються у звітах, що дозволяє аналізувати стан програмного забезпечення.

Однією з ключових характеристик автоматизованого тестування є його повторюваність. Тестові сценарії можуть виконуватися багаторазово без зміни логіки, що дозволяє використовувати їх для перевірки стабільності системи після кожного внесення змін. Це особливо важливо у процесах безперервної інтеграції, де перевірка коду виконується автоматично при кожному оновленні.

Переваги автоматизації тестування проявляються у кількох напрямках. По-перше, значно скорочується час виконання тестів, особливо у випадках великої кількості тестових сценаріїв. Автоматизовані тести можуть виконуватися значно швидше за ручні, що дозволяє оперативно отримувати результати перевірки.

По-друге, автоматизація забезпечує високу точність виконання тестів. Відсутність людського фактора зменшує ймовірність помилок, пов’язаних із неправильним виконанням тестових сценаріїв або пропуском перевірок. Це підвищує надійність результатів тестування.

По-третє, автоматизоване тестування дозволяє ефективно виконувати регресійні перевірки. При внесенні змін у програмний код необхідно переконатися, що існуюча функціональність не була порушена. Автоматизовані тести можуть швидко перевірити це, що є важливим для підтримання стабільності системи.

Крім того, автоматизація дозволяє виконувати тестування у різних середовищах і конфігураціях без значних додаткових витрат. Це дає змогу перевірити програмне забезпечення у різних умовах і забезпечити його переносимість.

Разом із тим, автоматизація тестування має певні обмеження. По-перше, створення автоматизованих тестів потребує значних початкових витрат часу і ресурсів. Необхідно розробити тестові сценарії, налаштувати інструменти та забезпечити їх підтримку.

По-друге, не всі види тестування доцільно автоматизувати. Наприклад, перевірка зручності використання або оцінювання інтерфейсу користувача часто потребує участі людини, оскільки такі перевірки мають суб’єктивний характер. У таких випадках ручне тестування залишається більш ефективним.

Ще одним обмеженням є необхідність підтримки автоматизованих тестів. При зміні програмного коду тестові сценарії можуть втрачати актуальність і потребувати оновлення. Це створює додаткові витрати і вимагає постійного контролю.

Інструментальні засоби автоматизації тестування програмного забезпечення є програмними продуктами, що забезпечують створення, виконання та аналіз тестів. Вони дозволяють спростити процес автоматизації і підвищити його ефективність.

Для автоматизації модульного тестування широко використовуються такі інструменти, як JUnit та pytest. Вони дозволяють створювати тести для окремих компонентів програмного забезпечення і автоматично перевіряти їх правильність.

Для тестування вебзастосунків застосовуються інструменти, такі як Selenium, який дозволяє автоматизувати взаємодію з вебінтерфейсом, і Cypress, що забезпечує швидке виконання тестів у браузері.

У процесах безперервної інтеграції використовуються системи, такі як Jenkins, які дозволяють автоматично запускати тести при кожній зміні коду. Це забезпечує постійний контроль якості програмного забезпечення.

Існують також інструменти для аналізу продуктивності, автоматизації тестування інтерфейсів, управління тестами та збору звітів. Вибір конкретних засобів залежить від типу програмного забезпечення, технологій, що використовуються, та вимог проєкту.

Ефективне застосування автоматизації тестування передбачає визначення доцільності її використання. Необхідно враховувати частоту виконання тестів, складність сценаріїв та витрати на їх підтримку. Найбільшу користь автоматизація приносить у випадках повторюваних перевірок, стабільної функціональності та необхідності швидкого отримання результатів.

У сучасних умовах автоматизоване тестування є невід’ємною складовою процесу розробки програмного забезпечення. Воно інтегрується у процеси безперервної інтеграції та постачання, що дозволяє забезпечити швидкий зворотний зв’язок і підтримувати високий рівень якості програмного продукту.

Таким чином, автоматизоване тестування програмного забезпечення є ефективним інструментом забезпечення якості, який дозволяє підвищити швидкість, точність та стабільність перевірки програмних систем. Його переваги та обмеження повинні враховуватися при організації процесу тестування, а використання сучасних інструментів дозволяє реалізувати автоматизацію у різних типах проєктів.

Лекція 14. Системи управління тестуванням та відстеження дефектів

У сучасних проєктах розроблення програмного забезпечення ефективне тестування неможливе без використання спеціалізованих програмних засобів, що забезпечують організацію, контроль і аналіз процесу перевірки. До таких засобів належать системи управління тестами та системи відстеження дефектів, які дозволяють структурувати тестову діяльність, підвищити її прозорість і забезпечити контроль якості на всіх етапах життєвого циклу.

Системи управління тестами призначені для організації процесу тестування, зберігання тестових матеріалів і контролю виконання тестових робіт. Вони дозволяють централізовано зберігати тестові сценарії, тестові випадки, результати виконання тестів та

іншу інформацію, пов’язану з тестуванням програмного забезпечення. Це забезпечує узгодженість дій команди та спрощує управління великими обсягами тестових даних.

Основними функціями систем управління тестами є планування тестування, створення та зберігання тестових випадків, організація виконання тестів, збір результатів та формування звітності. Такі системи дозволяють визначати пріоритети тестування, розподіляти завдання між учасниками команди та контролювати їх виконання. Це сприяє підвищенню ефективності роботи та забезпечує своєчасне виконання тестових робіт.

Важливим елементом є можливість відслідковування стану тестування, що дозволяє оцінити рівень покриття функціональності, кількість виконаних тестів та їх результати. Це дає змогу оперативно виявляти проблемні області та приймати рішення щодо подальших дій.

У практиці розробки програмного забезпечення широко використовуються такі системи управління тестами, як TestRail, Zephyr та qTest. Вони забезпечують широкий набір функцій для організації тестування і можуть інтегруватися з іншими інструментами розробки.

Системи відстеження дефектів призначені для реєстрації, аналізу та контролю процесу усунення помилок у програмному забезпеченні. Вони дозволяють фіксувати інформацію про дефекти, включаючи їх опис, умови виникнення, пріоритет, статус та відповідальних осіб. Це забезпечує системний підхід до управління дефектами та підвищує ефективність їх усунення.

Основною функцією таких систем є підтримка життєвого циклу дефекту, який включає виявлення, реєстрацію, аналіз, виправлення та перевірку. Кожен дефект проходить через певні стани, що дозволяє відслідковувати його обробку та забезпечувати контроль якості.

До поширених систем відстеження дефектів належать Jira, Bugzilla та Redmine. Вони дозволяють організувати роботу з дефектами, забезпечити взаємодію між учасниками проєкту та підвищити прозорість процесів.

Важливою перевагою використання таких систем є можливість інтеграції з іншими інструментами, зокрема системами контролю версій, засобами автоматизованого тестування та платформами безперервної інтеграції. Це дозволяє створити єдине середовище розробки, у якому всі процеси взаємопов’язані.

Застосування автоматизованих засобів тестування у процесі розробки програмних систем дозволяє значно підвищити ефективність перевірки та забезпечити постійний контроль якості. Автоматизовані засоби використовуються для виконання тестів, аналізу результатів, збору статистики та формування звітів.

У межах сучасних підходів до розробки програмного забезпечення автоматизовані засоби інтегруються у процеси безперервної інтеграції. Це означає, що при кожній зміні програмного коду автоматично запускаються тести, що дозволяє оперативно виявляти дефекти та забезпечувати стабільність системи.

Важливим є використання автоматизованих засобів для різних рівнів тестування. На рівні модульного тестування застосовуються фреймворки для перевірки окремих компонентів. На рівні інтеграційного та системного тестування використовуються інструменти для перевірки взаємодії компонентів і поведінки системи. Для тестування інтерфейсів застосовуються засоби автоматизації взаємодії з користувачем.

Автоматизація також дозволяє виконувати навантажувальне тестування, що дає змогу оцінити поведінку системи при високих навантаженнях. Це є важливим для забезпечення стабільності та продуктивності програмного забезпечення.

Ефективне застосування автоматизованих засобів тестування потребує правильної організації процесу. Необхідно визначити, які тести доцільно автоматизувати, розробити відповідні сценарії та забезпечити їх підтримку. Також важливо інтегрувати автоматизовані засоби з іншими інструментами розробки, що дозволяє створити єдину систему управління якістю.

Взаємодія між системами управління тестами, системами відстеження дефектів та автоматизованими засобами тестування забезпечує комплексний підхід до контролю якості програмного забезпечення. Це дозволяє ефективно організувати процес тестування, забезпечити прозорість роботи та підвищити якість кінцевого продукту.

Таким чином, системи управління тестами та системи відстеження дефектів є важливими інструментами організації процесу тестування. Вони забезпечують контроль, структурованість і ефективність тестових робіт. Застосування автоматизованих засобів тестування дозволяє підвищити швидкість і точність перевірки програмного забезпечення та забезпечити стабільність процесу розробки.

Лекція 15. Оцінювання якості та економічна ефективність програмного забезпечення

Оцінювання якості програмного забезпечення є системним процесом визначення ступеня відповідності програмного продукту встановленим вимогам, стандартам і очікуванням користувачів. Воно базується на використанні моделей якості, метрик та процедур вимірювання, що дозволяють отримати об’єктивну інформацію про стан програмної системи. Результати оцінювання використовуються для прийняття рішень щодо готовності продукту, необхідності його доопрацювання або подальшого розвитку.

Методи оцінювання якості програмного забезпечення можна поділити на кількісні та якісні. Кількісні методи базуються на використанні метрик, які відображають окремі властивості програмного продукту у числовій формі. До таких метрик належать кількість дефектів, щільність дефектів, час відгуку, використання ресурсів, рівень покриття тестами та інші показники. Вони дозволяють виконувати порівняння, аналіз динаміки та визначення тенденцій.

Якісні методи ґрунтуються на експертному оцінюванні, рецензуванні та аналізі відповідності вимогам. Вони використовуються для оцінювання характеристик, які складно виміряти числово, наприклад зрозумілість коду або зручність використання системи. Поєднання кількісних і якісних методів дозволяє отримати повне уявлення про якість програмного забезпечення.

Одним із важливих напрямків є оцінювання надійності програмного забезпечення. Надійність визначає здатність системи виконувати свої функції без відмов протягом певного часу. Для її оцінювання використовуються такі показники, як середній час безвідмовної роботи, інтенсивність відмов, коефіцієнт готовності та час відновлення після збоїв.

Аналіз надійності передбачає збір статистичних даних про роботу системи, виявлення закономірностей виникнення дефектів і прогнозування поведінки програмного забезпечення. Це дозволяє оцінити ризики та визначити необхідність додаткових заходів для підвищення стабільності системи.

Продуктивність програмного забезпечення характеризує ефективність використання ресурсів і швидкість виконання операцій. Для її оцінювання використовуються такі

показники, як час відгуку, пропускна здатність, завантаження процесора та використання пам’яті. Аналіз продуктивності є важливим для систем, що працюють у режимі реального часу або обробляють великі обсяги даних.

Оцінювання продуктивності виконується за допомогою спеціалізованих тестів, які моделюють різні умови навантаження. Це дозволяє визначити межі працездатності системи, виявити вузькі місця та оцінити її поведінку у критичних ситуаціях. Результати такого аналізу використовуються для оптимізації програмного забезпечення.

Економічні аспекти забезпечення якості програмного забезпечення є важливою складовою процесу розробки. Забезпечення якості потребує витрат ресурсів, які включають час, фінанси та зусилля команди. Водночас низька якість програмного продукту може призвести до значно більших втрат, пов’язаних із виправленням дефектів, зниженням продуктивності та втратою довіри користувачів.

Одним із ключових принципів є співвідношення витрат на забезпечення якості та витрат на усунення дефектів. Відомо, що вартість виправлення помилок зростає на пізніх етапах життєвого циклу, тому інвестування у ранні етапи забезпечення якості є економічно доцільним. Це включає аналіз вимог, рецензування та тестування на ранніх етапах розробки.

Економічна ефективність програмних систем визначається співвідношенням отриманих результатів і витрат на їх досягнення. Оцінювання ефективності включає аналіз витрат на розробку, впровадження, супровід та експлуатацію програмного забезпечення, а також оцінювання вигод, які отримує організація або користувач.

Для оцінювання економічної ефективності можуть використовуватися такі показники, як окупність інвестицій, чистий дохід, період окупності та інші. Ці показники дозволяють визначити доцільність розробки програмного продукту та оцінити його вплив на діяльність організації.

Важливим є врахування непрямих ефектів, таких як підвищення продуктивності праці, зменшення кількості помилок, покращення якості обслуговування та підвищення конкурентоспроможності. Хоча такі ефекти складно оцінити кількісно, вони мають суттєвий вплив на загальну ефективність програмного забезпечення.

Оцінювання економічної ефективності також включає аналіз ризиків, пов’язаних із реалізацією проєкту. Це дозволяє врахувати можливі втрати та прийняти обґрунтовані рішення щодо інвестування у розробку програмного забезпечення.

У сучасних умовах важливу роль відіграє оптимізація витрат на забезпечення якості. Це досягається шляхом використання автоматизованих засобів тестування, впровадження ефективних процесів розробки та підвищення кваліфікації персоналу. Такий підхід дозволяє забезпечити високий рівень якості при оптимальних витратах.

Таким чином, оцінювання якості програмного забезпечення є комплексним процесом, що включає використання різних методів і метрик. Аналіз надійності та продуктивності дозволяє оцінити технічні характеристики системи, а врахування економічних факторів забезпечує обґрунтованість рішень щодо її розробки та використання. Оцінювання економічної ефективності програмних систем дозволяє визначити доцільність інвестицій та забезпечити максимальну користь від використання програмного забезпечення.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Основна

1. Ammann, Paul, Offutt, Jeff. Introduction to Software Testing. 2nd ed. Cambridge University Press, 2017. 345 p. ISBN 9781107172012.
2. Spillner, A., Linz, T. Software Testing Foundations: A Study Guide for the Certified Tester Exam- Foundation Level- ISTQB Compliant. Dpunkt.Verlag. 2021. 339 p. ISBN: 9783969102992, 3969102995
3. Lewis, William E. Software Testing and Continuous Quality Improvement. 3rd ed. CRC Press, 2017. 688 p. ISBN: 9781439834367, 1439834369.
4. Авраменко А. С. Тестування програмного забезпечення : навчальний посібник. Черкаси : Черкаський національний університет ім. Б. Хмельницького, 2017. 208 с. ISBN 9789669201997.
5. Трофименко О. Г. Тестування та забезпечення якості програмних систем : навчально-методичний посібник [Електронне видання]. Одеса : Фенікс, 2024. 140 с.

Допоміжна

6. Myers, Glenford J., Sandler, Corey, Badgett, Tom. The Art of Software Testing. 3rd ed. John Wiley & Sons, 2011. 256 p. ISBN: 9781118133156, 1118133153.
7. Jorgensen, P. C., DeVries, B. Software Testing: A Craftsman's Approach, Fifth Edition. CRC Press. 2021. 550 p. ISBN: 9781000391527, 1000391523.
8. Gregory, J., Crispin, L. The Agile Testing Collection. Pearson Education. 2015. 1114 p. ISBN: 9780134190631, 0134190637.
9. Software Engineering for Agile Application Development. IGI Global. 2020. 330 p. ISBN: 9781799825333, 1799825337
10. Скорін Ю. І. Тестування програмного забезпечення : навчально-методичний посібник. Харків : ХНЕУ ім. С. Кузнеця, 2022. 47 с.

Інформаційні ресурси

11. International Software Testing Qualifications Board (ISTQB). URL: https://www.istqb.org/
12. Selenium – Web Browser Automation. URL: https://www.selenium.dev/
13. JUnit – Unit Testing Framework for Java. URL: https://junit.org/
14. SonarQube – Platform for Continuous Code Quality Inspection. URL: https://www.sonarsource.com/products/sonarqube/
15. Automation Testing Tutorial. Guru99. URL: https://www.guru99.com/software-testing.html

Навчальне видання

Сергій Борисович Приходько

ЯКІСТЬ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Опорний конспект лекцій для здобувачів вищої освіти,
які навчаються за спеціальностями галузі «Інформаційні технології»

Українською мовою