РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ВИДИ ЙОГО ТЕСТУВАННЯ

Григоренко Н. А., студент 4 курсу, Навчально-науковий інститут енергетичної, інформаційної та транспортної інфраструктури

Сенчук Т. С., старший викладач кафедри «Комп'ютерних наук та інформаційних технологій»

Харківський Національний Університет Міського Господарства імені О. М. Бекетова

Протягом тривалого часу розробка програмного забезпечення приділяла основну увагу великомасштабним науковим програмам, а також програмам міністерства оборони, пов'язаним із системами корпоративних баз даних, які проектувалися на базі універсальної ЕОМ. Тестові сценарії записувалися на папір. З їхньою допомогою перевірялися цільові потоки управління, обчислення складних алгоритмів та маніпулювання даними. Остаточний набір тестових процедур міг ефективно протестувати всю систему. Тестування зазвичай починалося лише після завершення плану-графіка проекту та виконувалося тим же персоналом.

Тестування програмного забезпечення (software testing)- це процес аналізу чи експлуатації програмного забезпечення з виявлення дефектів. Незважаючи на всю простоту цього визначення, він містить пункти, які вимагають подальших пояснень. Слово процес (process) використовується для того, щоб підкреслити, що тестування є планова, упорядкована діяльність. Цей момент дуже важливий, якщо ми зацікавлені у швидкій розробці, бо добре продуманий, систематичний підхід швидше призводить до виявлення програмних помилок, ніж погано сплановане тестування, до того ж проведене поспіхом.

Відповідно до цього визначення, тестування передбачає "аналіз" чи "експлуатацію" програмного продукту. Тестова діяльність, що з аналізом результатів розробки програмного забезпечення, називається статичним тестуванням (static testing). Статичне тестування передбачає перевірку програмних кодів, наскрізний контроль і перевірку програми без запуску на машині, тобто перевірка за столом (desk checks). На відміну від цього, тестова діяльність, що передбачає експлуатацію програмного продукту, має назву динамічного тестування (dynamic testing). Статичне та динамічне тестування доповнюють один одного, і кожен із цих типів тестування реалізує власний підхід до виявлення помилок.

Останній пункт визначення, що вимагає додаткових пояснень, - це поняття дефекту (bug). Говорячи простими словами, програмна помилка - не що інше, як вада в розробці програмного продукту, що викликає невідповідність очікуваних результатів виконання програмного продукту та фактично отриманих результатів. Дефект може виникнути на стадії кодування, на стадії

формулювання вимог або на стадії проектування, або його причина може критися в некоректній конфігурації або даних. Дефектом може бути щось інше, що не відповідає очікуванням замовника і що може бути, а може і не бути визначено в специфікації програмного продукту.

Перша дія у плануванні випробувань передбачає розробку стратегії тестування на високому рівні. У загальному випадку стратегія тестування повинна визначати обсяги тестових робіт, типи методик тестування, які повинні застосовуватися для виявлення дефектів, процедури, що повідомляють про виявлення та усувають дефекти, критерії входу та виходу з випробувань, що керують різними видами тестування. Реалізуючи принцип тісного інтегрування розробки та тестування для оптимізації графіка розробки, стратегія тестування повинна відображати різні види тестової діяльності на життєвий цикл розробки. При формулюванні загальної стратегії має бути передбачено як статичне так і динамічне тестування.

Якщо для підтримки різних видів тестової діяльності використовується автоматизація, то стратегія автоматизації повинна розглядатися як складова загальної стратегії тестування. Автоматизація вимагає виконання незалежних паралельних робіт, які мають ретельно плануватись і виконуватись лише в тих випадках, коли це не призводить до зниження ефективності.

Існують такі підходи до формулювання стратегії тестування:

1. Визначити обсяги тестових робіт шляхом аналізу документів, які містять вимоги до програмного продукту (технічні умови), щоб з'ясувати, що слід тестувати. Розглянути види тестування, які не випливають безпосередньо з документів з вимогами, такі як тестування можливості встановлення та нарощування можливостей програмного продукту, зручність та простота обслуговування продукту, а також здатності до взаємодії з іншими видами апаратних засобів із середовища замовника.

2. Визначити підхід до тестування за рахунок вибору статичних та динамічних тестів, пов'язаних із кожною стадією розробки. Тут буде потрібно включити описи всіх робочих продуктів, які повинна підготувати тестова група.

3. Визначити критерії входу та виходу для кожної стадії тестування, так само як і всі точки контролю якості, для чого потрібна участь фахівців із тестування.

4. Визначити стратегію автоматизації у разі, якщо планується використання автоматизації будь-якого виду тестової діяльності. Автоматизація вимагає проведення незалежних паралельних робіт, які мають ретельно плануватися та виконуватися лише у тих випадках, коли це не призводить до зниження ефективності.

Розглянемо ще одне питання визначення обсягів тестових робіт. Оскільки тестувати абсолютно все неможливо, важливість вибору того, що потрібно протестувати, сумнівів не викликає. Якщо допустити "перебір" у тестуванні, тобто якщо тестове покриття буде надлишковим, то для налагодження програмного продукту знадобиться значний час, що поставить під загрозу термін здачі проекту. Якщо тестування виявиться недостатнім (точніше, недостатнім буде тестове покриття), збільшиться ризик пропуску того

чи іншого дефекту, усунення якого коштуватиме дуже дорого, особливо після здачі програмного продукту в експлуатацію. Знайти потрібний баланс між цими двома крайнощами допоможе досвід і спосіб вимірювання успішності тестування. Ось кілька пропозицій щодо розробки стратегії тестування, які допоможуть у пошуку оптимального тестового покриття:

- Тестувати насамперед вимоги з найвищим пріоритетом.
- Тестувати нові функціональні можливості та програмний код, який змінювався з метою виправлення чи вдосконалення старих функціональних засобів.
- Використовувати розбиття на еквівалентні класи та аналіз граничних значень для зниження трудовитрат на тестування.
- Тестувати ті ділянки, в яких найімовірніше присутність проблем.
- Зосередити свою увагу на функціях і конфігураціях, з якими найчастіше матиме справу кінцевий користувач.
- Визначення підходу до тестування.

За наявності реальних планів та розумних припущень використання автоматизованих інструментальних засобів та автоматизованих тестових випадків є чудовим способом зниження тимчасових витрат на тестування програмного продукту. Будь-яке завдання, що багаторазово виконується, є кандидатом на автоматизацію. Однак зазвичай на автоматизацію завдання йде набагато більше часу, ніж на її виконання, тому для кожного завдання, яке може бути автоматизовано, доцільно провести ретельний аналіз потенційного виграшу від автоматизації. Виконуючи аналіз можливих вигод, слід пам'ятати, що самої автоматизації характерний власний автономний життєвий цикл.

Ефективна автоматизація вимагає спеціальної підготовки персоналу, розробки, налагодження та верифікації, як будь-який інший проект розробки програмного забезпечення. Безпланова і погано виконана автоматизація означає марну витрату ресурсів, вона може призвести до порушення графіка виконуваних робіт, якщо час витрачатиметься на налагодження автоматизації, а не на тестування.

Література:

1. Буч Г. Язык UML. Руководство пользователя / Буч Г., Рамбо Д., Якобсон И. // Пер. с англ. Мухин Н. — 2-е изд. – М.: ДМК Пресс. – 496 с.
2. Лавріщева К.М. Програмна інженерія. — К. — 2008. — 319 с.
3. Pro Git. Режим доступу: https://git-scm.com/book/uk/v2 .
4. Керівництва GitHub. Режим доступу: https://guides.github.com/
5. Книберг Х. Kanban и Scrum: выжимаем максимум / Хенрик Книберг, Маттиас Скарин. — C4Media, Издательство InfoQ.com, 2010. — 78 с.
6. Книберг Х. Scrum и XP: заметки с передовой. Как мы делаем Scrum. — C4Media, Издательство InfoQ.com, 2010. — 94 с.
7. Мартін Р. Чистий код: створення і рефакторинг за допомогою Agile / пер. з англ. І. БондарТерещенко. — Харків : Вид-во «Ранок» : Фабула, 2019. — 448 с.