№ 663. – С.78–83. 11. Драган Я., Медиковський М., Овсяк В., Сікора Л., Яворський Б. Системний аналіз концепції та принципів побудови математичної моделі досліджуваного об'єкта в фізико-технічних науках та оцінювання її якості // Вісник Нац. ун-ту "Львівська політехніка" Комп'ютерні науки та інформаційні технології. – 2010. – № 686. – С.170–179. 12. Биркгоф Г. Математика и психология. – М.: Сов. радио, 1977. – 96 с. 13. Детловс В. Эквивалентность нормальных алгоритмов и рекурсивных функций // Тр. матем. ин-та АН СССР им. В.А. Стеклова, 1958, 52. – С.75–139. 14. Адамар Ж. Исследование процесса изобретения в области математики / пер. с франц. М.А. Шаталовой и О.П. Шаталова, под ред. И.Б. Погребысского. – М.: Сов. радио, 1970. – 152 с. 15. Ovsiak V.K. Computational models and algebra of algorithms // Вісник Нац. ун-ту "Львівська політехніка" Інформаційні системи та мережі. – 2008. – № 621. – С.3–18.

УДК 004.9

Т. Свірідова, У. Марікуца
Національний університет "Львівська політехніка",
кафедра систем автоматизованого проектування

ДОСЛІДНИЦЬКЕ ТЕСТУВАННЯ ЯК ЗАСІБ ПІДВИЩЕННЯ ЯКОСТІ АПЛІКАЦІЇ

Запропоновано використання дослідницького тестування як спосіб покращення якості аплікації, що розробляється. Розглянуто дослідницьке тестування як частину регресійного тестування.

Ключові слова: дослідницьке тестування, регресійне тестування, якість.

This paper is devoted to the investigation of quality factors which can influence on developed software. Exploratory testing is described and analysed in scope of regression testing.

Key words: exploratory testing, regression testing, quality.

Вступ

Однією з умов успішності розробленого програмного продукту є його якість. Якість програмного продукту – це сукупність його рис і характеристик, які впливають на його здатність задовольняти задані потреби користувачів.

Як відомо, коли розробник додає нову функціональність або виправляє дефект у системі, існує висока ймовірність того, що внесені зміни можуть повністю дестабілізувати систему. Якщо під час кожної зміни коду тестувати тільки новий функціонал, то не можна бути повністю впевненим, що всі решта складові системи працюють відповідно до вимог або очікувань клієнта. Власне для переконання, що аплікація цілісна та працює повністю правильно, розроблено такий підхід для тестування, як регресія. Однак, як і будь-який інший підхід, регресія має низку недоліків, одними з яких є так званий "ефект пестициду", одноманітність роботи, прогін тих самих не завжди актуальних, що, як результат, призводить до зниження якості кінцевої версії аплікації. Для покращення процесу регресійного тестування запропоновано використати дослідницьке тестування як його частину.

Основна частина

Дослідницьке тестування – це розроблення та виконання тестів у той самий час, що є повною протилежністю підходу з тест-кейсами. Дослідні тести, на відміну від класичних тестів, не визначені заздалегідь і не виконуються точно відповідно до плану.

Тестер може взаємодіяти з програмним продуктом у будь-який спосіб і використовувати інформацію, яку надає програмний продукт, щоб реагувати, змінювати курс, і загалом досліджувати його функціональність. Цей процес нагадує ad hoc тестування, проте в руках досвідченого тестера ця техніка може бути потужною. Великою перевагою дослідницького тестування є те, що можна залучити усю силу людського розуму для пошуку дефектів і перевірки функціональності.

Процес дослідницького тестування також не відбувається без документування. Тест-кейси, результати тестів і документація із тестування генеруються у процесі дослідницького тестування замість того, щоб документуватись заздалегідь у вигляді плану тесту. Такі інструменти, як захоплення екрана і запис натиснутих клавіш, є дуже корисними для запису результатів дослідницького тестування.

Дослідницьке тестування придатне для розроблення веб-аплікацій, які використовують agile-методи. Цикли розроблення є короткими, що залишає мало часу на написання сценаріїв і їх підтримку. Функції швидко еволюціонують, тому використання тест-кейсів скорочується. Їх написання і підтримка забирає багато часу.

Недоліком дослідницького тестування є те, що тестер ризикує витратити велику частину часу, блукаючи по програмному продукту в пошуках речей, які можна тестувати, і намагаючись знайти дефекти. Відсутність підготовки, структури і керівництва під час тестування може призвести до багатьох непродуктивних годин роботи і перетестування тієї самої функціональності, особливо коли задіяні кілька тестерів або команда тестерів.

Для цього і потрібно керівництво. Керівництво задає напрям тестування, що дає змогу зробити дослідження методичнішим. В дослідницькому тестуванні існує три підходи, які допомагають у процесі прийняття рішень:

1. Дослідницьке тестування у прийнятті невеликих рішень, що допомагає прийняти локальні рішення під час виконання тесту.
2. Дослідницьке тестування у прийнятті великих рішень, що допомагає тестеру спроектувати загальний план тестування і стратегію.
3. Дослідницьке тестування в комбінації з тестуванням на основі сценаріїв, що поєднує елементи дослідницького з ручним тестуванням на основі сценаріїв.

У роботі тестера під час ручного тестування багато в чому є варіанти. Тестер повинен вибирати, які вхідні дані вводити, які сторінки відвідати, які пункти меню вибрати, які саме значення вводити в кожне поле для вхідних даних, які він знаходить. Є сотні таких рішень, які потрібно приймати у ході тестування.

Дослідницьке тестування може допомогти прийняти ці рішення. Коли тестер використовує стратегію дослідницького тестування у вирішенні таких питань, це називається дослідницьким тестуванням у прийнятті невеликих рішень, оскільки область цих рішень невелика. Тестер розглядає певну веб-сторінку або діалогове вікно і потребує порад, що робити у цій ситуації. Це локалізований процес прийняття рішень, який тестер виконує десятки раз протягом одного тест-кейсу і сотні разів за день.

Ці невеликі вибори, які тестер робить під час тестування, можна розділити на п’ять специфічних властивостей програмного продукту:

1. Вхідні дані.
2. Стан.
3. Шлях коду.
4. Користувацькі дані.
5. Середовище виконання.

Навіть кожна з цих властивостей зокрема ускладнює проблеми тестування, зважаючи на обмежені ресурси. Тоді загалом процес тестування видається нездійсненним. У цьому допомагає великою мірою керівництво щодо того, як підійти до вирішення проблеми – сукупність тактик.

Одним із завдань тестера є відповісти на такі питання:

1. Чи програма працюватиме так, як заплановано?

2. Чи програма виконуватиме функції, для яких користувач її купує?
3. Чи програма виконуватиме ці функції достатньо швидко, достатньо безпечно, достатньо потужно тощо?

Тестер виконує це завдання, встановлюючи програму у певне операційне середовище і вводячи певні вхідні дані. Саме тут тестер стикається з проблемою нескінченності. Ця проблема полягає у тому, що є дуже багато вхідних даних, які можна ввести, багато середовищ, в яких можна протестувати програму, а ресурси обмежені. Ось чому тестування стосується варіації речей. Тестер повинен визначити усі речі, які можуть варіюватись протягом тестування, і переконатись, що рішення, які він прийняв, розумні. Тестер вибирає певні варіації, а решту відкидає.

Такий підхід звужує тестування до завдання вибору підмножини вхідних даних, середовищ тощо, застосування їх і припущення, що це оптимальний варіант. Тестер, який глибше вивчить вхідні дані, середовища і різні речі, які можуть варіюватися, буде краще підготований для дослідження програми.

Таблиця 1

Ознаки турів для використання під час регресійного тестування

Ознаки / Тури	Наявний час	Імітація поведінки користувача	Тестування основного функціоналу	Тестування частин з можливими дефектами	Накопичені знання і досвід	Пошук нових дефектів	Швидкість	Застосування зі сценаріями
Тур з путівником	+	+	+		+			
Грошовий тур	+	+	+					+
Тур з орієнтирами			+			+		+
Тур інтелектуала			+					+
Тур поштової системи			+		+			
Тур після робочих годин			+					
Тур збирача сміття	+		+		+		+	
Тур по поганим околицям	+			+	+			
Музейний тур				+				
Тур попередньої версії			+		+			
Тур актора другого плану		+						+
Тур задньої алеї					+	+		+
Нічний тур								+
Тур колекціонування			+		+	+		+
Тур самотнього бізнесмена	+		+		+	+		
Тур супермоделі							+	+
Тур копій						+		
Тур по маловідомим місцям					+			
Зірваний тур		+				+		+
Тур ледаря		+				+		
Диверсант		+				+		+
Антисоціальний тур			+			+		
Нав'язливо-примусовий тур		+				+		+
Тур непроханого гостя								+

Одним з дослідницьких підходів, який ідеально підходить для регресії, є тестування турами. Будь-яке планування тестування повинно починатись з декомпозиції програмного продукту на менші частинки, які є керованішими. Тут корисним стає таке поняття, як тестування функцій, де зусилля тестування розподілено між функціями, які утворюють програму. Це полегшує відстеження прогресу тестування і призначення ресурсів тестування, але також вводить велику частку ризику.

Функції рідко є незалежними одна від одної. Вони часто розділяють між собою ресурси програми, обробляють спільні вхідні дані і працюють на тій самій внутрішній структурі даних. Отже, незалежно функції можуть перешкоджати пошуку дефектів, які виникають, лише коли функції взаємодіють одна з одною.

Туристична метафора наполягає на відсутності такої декомпозиції. Насправді вона пропонує декомпозицію, яка ґрунтується на цілі більше, ніж на якійсь властивій структурі програми під тестами. Як турист має намір під час відпустки побачити якомога більше в найкоротший період часу, так само тестер організовуватиме свої тури. Справжній турист вибере поєднання пам'яток, які хоче побачити, і місць для відвідування, так само тестер вибиратиме, щоб поєднати і підібрати функції програми з наміром зробити щось специфічне. Цей намір часто вимагає комбінації якоїсь кількості функцій у способи, яких не було б при строгій моделі функцій у тестуванні.

Путівник туриста часто ділить пункт призначення на райони. Це бізнес-район, район розваг, район театрів тощо. Туристу така сегментація часто показує фізичні обмеження. Для тестера програми така сегментація є строго логічним поділом функцій програми, оскільки відстань – нереальна проблема для тестера. Натомість тестер повинен досліджувати шляхи програми, яка виконується за різних функцій у різній послідовності.

Проаналізувавши особливості тестування турами, можна виділити ознаки, за якими тури можуть бути корисні для регресійного тестування. Результати аналізу відображено у табл. 1.

Висновки

Застосовуючи дослідницьке тестування під час регресії, можна усунути певні недоліки регресійного тестування:

1. Процес регресії стає ефективнішим, оскільки розширюється покриття тестами програми, наявний час, імітація поведінки користувача, тестування основного функціонала, тестування частин з можливими дефектами, накопичені знання і досвід, пошук нових дефектів, швидкість.
2. З'являються нові ідеї тестів та підвищується ймовірність знаходження нових дефектів.
3. Тестування турами робить сам процес різноманітнішим і цікавим для тестера.
4. Не виникає проблеми вибору тестів з існуючих.
5. Не потрібно постійно підтримувати тести в актуальному стані, що значно економить час тестувальника.
6. Ймовірність позбутись так званого "ефекту пестициду" надзвичайно велика.
7. Тестувальник постійно має свіжий погляд на систему.

До недоліків можна зарахувати труднощі щодо документування результатів тестування та повторного відтворення дефектів для подальшої верифікації.

1. Kaner C. A Tutorial in Exploratory Testing, p.36, Apr. 2008. 2. Bach J. Exploratory Testing Explained // The Test Practitioner. 2002. 3. Bach J. Where Does Exploratory Testing Fit? 3. Kohl J. Demystifying Exploratory Testing // Better Software. Apr. 2010. 4. Whittaker J. Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design // Pearson Education. 2009. P.16-75. 5. http://uk.wikipedia.org/wiki/Регресивне_тестування 6. Котляров В.П. Основы тестирования программного обеспечения. – 2005.