УДК 004.415.532

О.А. РЕМІННИЙ

СТАТИЧНІ МЕТРИКИ СИСТЕМИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ КОРИСТУВАЦЬКИХ ІНТЕРФЕЙСІВ

Вінницький національний технічний університет
вул. Хмельницьке шосе 95, м. Вінниця, Україна

Анотація. В роботі проведено дослідження основних цілей написання автоматизованих тестів. Описано методологію збору статичних та динамічних метрик коду рішення автоматизованого тестування користувацьких інтерфейсів у вигляді ітераційних графіків, спроектованих на часову вісь. Визначено підхід для обчислення продуктивності розробників рішення автоматизованого тестування. Показано результати обчислень даних метрик на реальному проекті розробки рішення автоматизованого тестування.

Аннотация. В работе проведено исследование основных целей написания автоматизированных тестов. Описана методология сбора статических и динамических метрик кода решения автоматизированного тестирования пользовательских интерфейсов в виде итерационных графиков, спроектированных на временную ось. Определен подход для вычисления производительности разработчиков решения автоматизированного тестирования. Показано результаты вычислений данных метрик на реальном проекте разработки решения автоматизированного тестирования.

Abstract. Study conducted in the work explains the main goals for automated tests development. Described methodology for static and dynamic code metrics collection for GUI automation test solutions. Defined approach to calculate developers' productivity when developing such solutions. Showed real project case study with the related calculations.

Ключові слова: Автоматизоване тестування, користувацький інтерфейс, метрики коду

ВСТУП

Одна з найважливіших активностей розробки програмного застосунку є збір метрик. Це є основним критерієм досяжності певної цілі розробки, поставленої менеджментом проекту. Метрики дозволять аналізувати швидкість, продуктивність роботи команди, передбачати майбутні ключові дати готовності продукту і таке інше. В даній статі описано збір метрик коду рішення автоматизованого тестування користувацьких інтерфейсів, побудованого за допомогою багатошарової архітектури та патернів автоматизованого тестування.

Цілі проекту. Метрики не можуть існувати самі по собі, вони завжди прив'язані до певних цілей проекту. Розглянемо збір метрик відносно різних напрямків цілей автоматизованого тестування. Відповідно до дослідження було проведено набір з 10 інтерв'ю різних проектів з впровадженим автоматизованим тестуванням. Ось які три основні цілі можна виділити для замовників, що починали розробку автоматизованих тестів в себе на проекті:

Збереження часу та грошей. Напевно найскладнішим рішенням впровадження автоматизованого тестування в компанії є питання чи буде воно успішним і яким буде його результат з точки збереження часу та коштів відносно зменшення часу на ручне тестування.

З плином часу програмний продукт стає більш складним у зв'язку із збільшенням кількості рядків коду через додавання нових функцій, виправлення існуючих помилок, і т.д. Також задачі потрібно робити в більш короткі терміни та меншою кількістю людей. Складність з плином часу буде мати тенденцію до зниження тестового покриття і в кінцевому рахунку впливає на якість продукту. Інші фактори також можуть впливати на загальну вартість продукту і час нових релізів програмного забезпечення. Метрики можуть дати уявлення про стан зусиль, витрачених на автоматизоване тестування.

Загалом при використанні систематичного підходу до автоматизації, можна отримати залежність від автоматизації продуктів, описану на рис. 1 [1, 2].

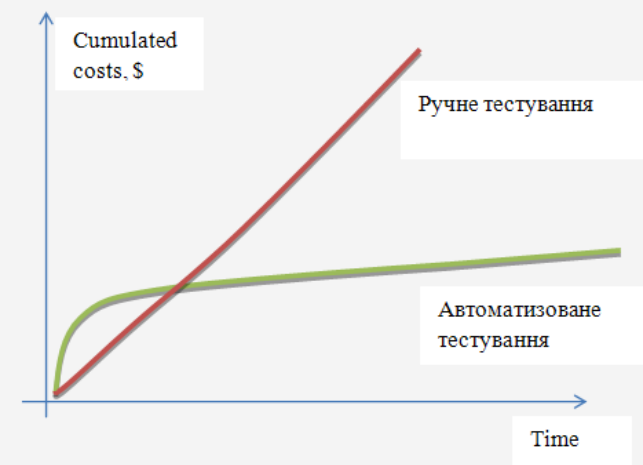

Рис. 1. Витрати на тестування (ручне або автоматизоване) з плином часу

Корпоративний стандарт. В компаніях, які мають широкий спектр продуктів, питання автоматизації їх тестування може виноситись на корпоративний рівень. Прикладом корпоративної цілі автоматизованого тестування є дане твердження «продукт буде випущено для загального доступу лише у випадку якщо 80% регресійних тестів буде автоматизовано».

Автоматизація заради технологічності. Також важливим фактором впровадження автоматизації тестування є мода[3]. Словосполучення «Автоматизоване тестування внутрішніх продуктів» може послужити як приємним фактором для замовника, так і гарним показником для його продуктів серед конкурентів. Якість самих автоматизованих тестів в таких випадках може відходити на задній план.

Відповідно до поставлених варіацій цілей автоматизованого тестування підходять різні практики імплементації автоматизованого тестування, що в основному впливає на якість самих автоматизованих тестів.

На рисунку 2 ілюструється життєвий цикл проекту який було випущено в реліз та який з часом продовжують розробляти [4].

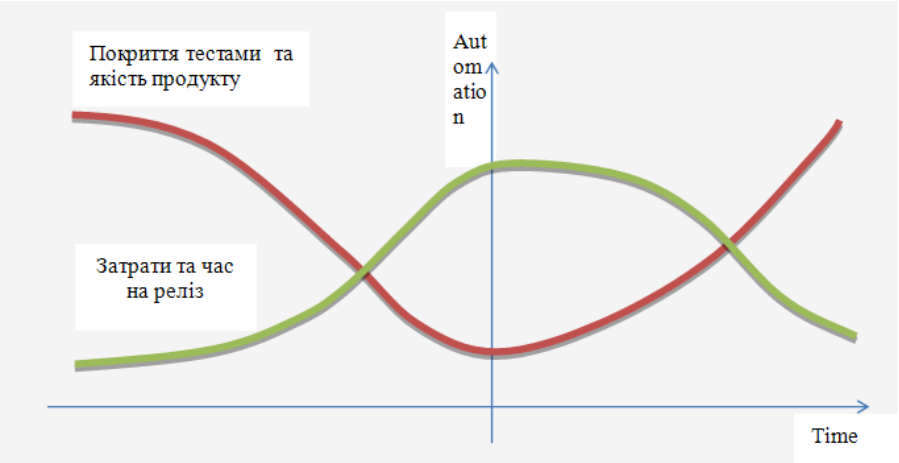

Рис. 2. Витрати на тестування до та після введення автоматизації

Загалом графіки трендів є ідеалізовані, так як на конкретних проектах може спостерігатись абсолютно різна ситуація пов’язана в основному з наступними факторами:

- Інструментація (підготовленість до автоматизованого тестування) програми, для якої розробляються автоматизовані тести[1];
- Рівень команди, що займається автоматизованим тестуванням;
- Програма емуляції дій користувача (automation tool).

Метою роботи є покращення процесу автоматизованого тестування за рахунок отримання більшої інформативності від самого коду автоматизованих тестів. Відповідно в рамках даної статті *задачею* є визначення методології збору метрик коду статичного характеру, які б відповідали прогресу розробки програмного продукту на рівні коду і могли б служити в подальшому для порівняння при автоматизації наступних програмних продуктів.

СТАТИЧНІ МЕТРИКИ КОДУ

Однак для самих автоматизованих тестів можна побудувати певний набір правил, за яким в подальшому можна обчислювати характеристики якості коду розробки автоматизованих тестів.

Для відслідковування динаміки зміни в рішенні автоматизованого тестування, збір метрик потрібно проводити періодично, наприклад раз в тиждень або раз на ітерацію.

Збір метрик неможливий без використання певних допоміжних утиліт. Для даного дослідження збір метрик відбувався на протязі виконання двох проектів автоматизованого тестування з технологією написання тестових скриптів C# .NET. Відповідно для аналізу даних програмних за стосунків використовувалась комерційна утиліта NDepend[5]. Особливістю цієї утиліти є те що за допомогою неї можна представити код програми як певну систему даних, над якою можна виконувати реляційні запити за допомогою технології CQlinq[6].

Кількість стрічок коду (Lines of Code - LOC). Кількість стрічок коду є інформативним показником загального прогресу в коді програми[7]. Кількість стрічок коду зазвичай має зростати з перебігом часу, окрім як у випадках проведення рефакторингу.

Відповідний запит в утиліті NDepend:

```
from assembly in Assemblies
select new {assembly, assembly.NbLinesOfCode}
```

Кількість бізнес методів (Business methods count). Бізнес метод є атомарною одиницею розробки автоматизованих тестів[8,9,10]. Кількість бізнес методів та кількість розроблених тестів найкраще ілюструє прогрес розробки проекту автоматизації.

Відповідний запит в утиліті NDepend (Всі бізнес методи мають відповідний атрибут (BusinessMethodAttribute)):

```
from businessMethod in Methods
where businessMethod.HasAttribute ("Core.Attributes.BusinessMethodAttribute")
orderby businessMethod.NbMethodsCallingMe descending
select new { businessMethod, ForSumCalc = 1 }
```

Кількість розроблених тест кейсів (Test Cases count). Основною характеристикою показників автоматизації на проекті для менеджменту є не показники коду, а показники саме автоматизації конкретних тестів. Дані метрики не можна зібрати з коду, ця інформація була зібрана з систем відстеження задач (Task tracking systems).

МЕТРИКИ ПРОДУКТИВНОСТІ

Відповідно до визначених метрик можна зібрати показники продуктивності окремих розробників в команді, а також знайти співвідношення між розробкою нових бізнес методів та самих тест кейсів.

Відповідно час, зазвітований розробниками, розподілявся на кількість розробників, плюс до уваги брались показники кількості розроблених тестів та бізнес методів.

Визначимо TCpD – кількість тест кейсів розроблених одним розробником (Рис. 3) протягом певного періоду (час розробки одного тест кейсу в людино-днях):

$$TCpD = \frac{\sum_1^n C \cdot t}{n \cdot N_{TC}}, \quad (1)$$

де *TCpD* – кількість тест кейсів розроблених одним розробником; *n* – кількість розробників; *t* – час, зазвітований кожним розробником на розробку тестів протягом періоду; N_{TC} – кількість тест кейсів, розробка яких закінчилась протягом даного періоду; *C* – коефіцієнт додаткової втрати часу (на рефакторинг коду, мітинги, перегляд коду і т.п.). Коефіцієнт додаткової втрати часу визначався дослідним шляхом, для даних проектів C = 1.5.

Аналогічно можна визначити швидкість розробки бізнес методів одним розробником (Рис. 4) протягом періоду (*BMpD*), з єдиною зміною в (1):

$$BMpD = \frac{\sum_1^n C \cdot t}{n \cdot N_{BM}}, \quad (2)$$

де N_{BM} – кількість нових розроблених бізнес методів протягом даного періоду.

В результаті були отримані наступні співвідношення (як показник періоду брався тиждень – одна ітерація процесу розробки):

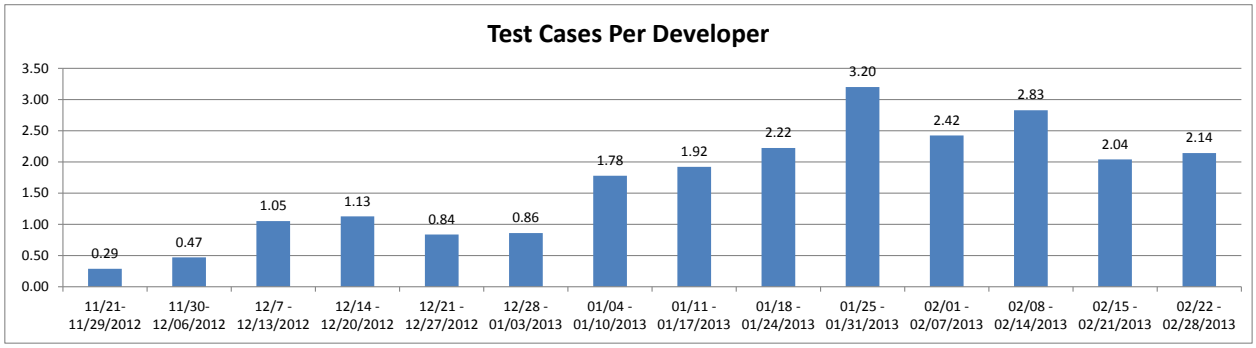

Рис. 3. Кількість тестових сценаріїв, розроблених одним розробником протягом тижня

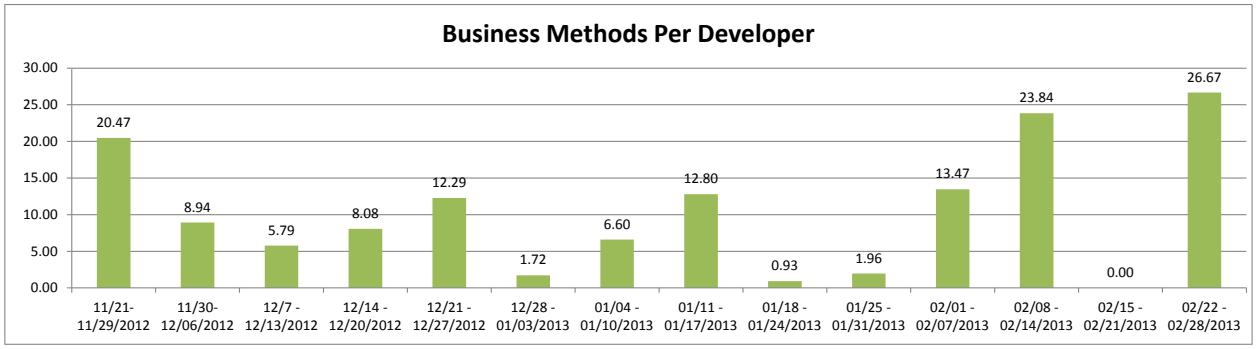

Рис. 4. Кількість бізнес методів, розроблених одним розробником протягом тижня

Відповідно можна побачити, що залежність кількості розроблених автоматизованих тестів обернено пропорційна до кількості розроблених бізнес методів протягом того ж періоду. Це пояснюється тим фактом, що розробка та налагодження бізнес методів займає час, а коли дані методи вже розроблено та налагоджено, їх пере використання в інших тестових сценаріях є швидким процесом.

Метрики перевикористання коду

Найбільш цікавою метрикою з точки зори розробки тестів є повторного використання існуючого коду для усунення дублювання функціоналу (Рис. 5).

Відповідний запит в утиліті NDepend:

```
from m in Methods where m.HasAttribute ("Core.Attributes.BusinessMethodAttribute")
orderby m.NbMethodsCallingMe descending
select new { m, m.NbMethodsCallingMe }
```

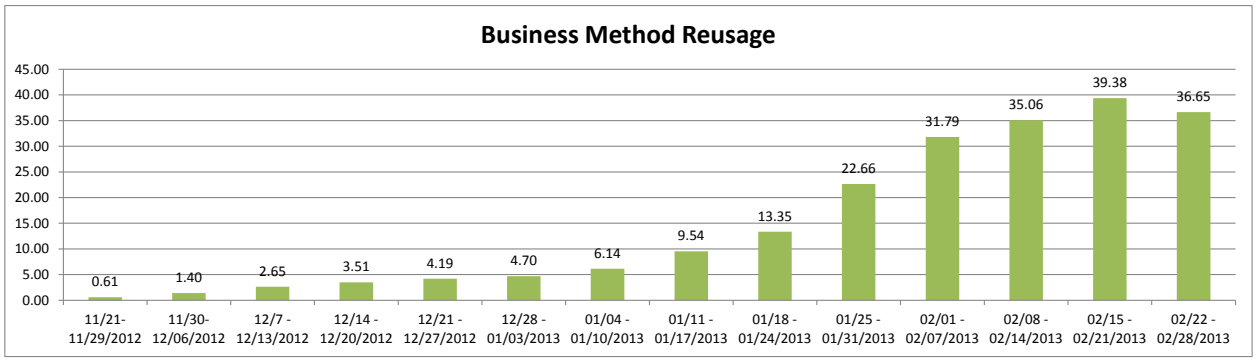

Рис. 5. Повторне використання бізнес методів у рішенні автоматизованого тестування

Аналіз даного графіку показує що в результаті розробки даного програмного рішення автоматизованого тестування кінцевий код тестів в середньому перевикористовував кожен бізнес метод більше 36 раз. Це означає що у випадку зміни користувацького інтерфейсу, зміна в одному бізнес методі автоматично буде розповсюджена в 36 різних місць, де викликається даний бізнес метод. За рахунок таких характеристик перевикористання і уніфікації бізнес методів досягається високий рівень підтримуваності програмного додатку.

ВИСНОВКИ

Основною ціллю збору метрик на проекті є відстеження приближення до певної цілі. Відповідно основною метрикою для менеджменту є кількість автоматизованих командою тест кейсів. З точки зору продуктивності також треба враховувати швидкість розробки одного тестового сценарію/ одного тестового методу.

В статті вперше запропонована методологія відслідковування кількості розроблених бізнес методів та відстеження їх перевикористання, що забезпечується використанням багатошарової архітектури тестового коду та патернів побудови тестових бібліотек.

СПИСОК ЛІТЕРАТУРИ

1. Elfriede Dustin, Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality/ Elfriede Dustin, Thom Garrett, Bernie Gauf. Addison-Wesley Professional – 2009. - 368 p.
2. InfoStretch Corporation, Business Driven Quality Metrics [Електронний ресурс] // Режим доступу до файлу: http://www.slideshare.net/Infostretchmarketing/info-stretch-bizdrivenqualitymetricsfinal
3. Ramesh Ramani, Automate. Accelerate. Proliferate [Електронний ресурс] // Режим доступу до файлу: http://www.tavant.com/blog/?p=22
4. Thom Garrett, Implementing Automated Software Testing - Continuously Track Progress and Adjust Accordingly [Електронний ресурс] // Режим доступу до файлу: http://www.methodsandtools.com/archive/archive.php?id=94
5. NDepend - a Visual Studio tool to manage complex .NET code and achieve high Code Quality [Електронний ресурс] // Режим доступу до файлу: http://ndepend.com/
6. Code Rule and Code Query over LINQ (CQLinq) [Електронний ресурс] // Режим доступу до файлу: http://ndepend.com/Features.aspx#CQL
7. Source lines of code [Електронний ресурс] // Режим доступу до файлу: http://en.wikipedia.org/wiki/Source_lines_of_code
8. Gerard Meszaros, xUnit Test Patterns: Refactoring Test Code / Gerard Meszaros – 2007. – 833p.
9. Misha Rybalov, Design Patterns for Customer Testing [Електронний ресурс] // Режим доступу до файлу: http://www.autotestguy.com/archives/Design%20Patterns%20for%20Customer%20 Testing.pdf
10. Липаев В. В. Тестирование компонентов и комплексов программ / В. В. Липаев. – Синтег, 2010. – 400 стр.

Надійшла до редакції 18.11.2013р.

РЕМІННИЙ ОЛЕКСАНДР АНДРІЙОВИЧ – аспірант кафедри автоматики та інформаційно-вимірювальної техніки, Вінницький національний технічний університет, м. Вінниця, Україна.