УДК 001.413:338.5

A.S. Pavlenko

QUALITY DEFECTS DETECTION IN UNIT TESTS

National aviation university

Software engineering department

Lecturer - Nechay A.S., candidate of engineerings sciences, senior teacher

Подано підхід визначення дефектів якості у модульних тестах за допомогою метрик для полегшення супроводження програмного забезпечення; розглянуто спосіб розробки метрик на основні критеріїв коректних модульних тестів. Результати підтверджено на основі контрольованого експерименту.

Представлено подход определения дефектов качества в модульных тестах с помощью метрик для облегчения поддержки программного обеспечения; рассматривается способ разработки метрик с помощью *определения* критериев корректных модульных тестов. Результаты демонстрируются на основе контролируемого эксперимента.

An approach is presented for quality defects detection in unit tests with the help of metrics for software maintenance; method of developing metrics with the help of the unit tests quality characteristics was discussed. Results are demonstrated on the basis of controlled experiment.

Keywords: software engineering, unit tests, metrics, test smells, defects in unit tests, assertion roulette, interacting tests.

Introduction

The success of software organizations depends on their ability to facilitate continuous improvement of their products in order to reduce cost, effort, and time-to-market, but also to restrain the ever increasing complexity and size of software systems. This increases the need for software organizations to develop or rework existing systems with high quality within short periods of time using automated techniques to support developers, testers, and maintainers during their work.

During software maintenance quality defects appear, leading to quality degradation. Studies size the contribution of test code between 33 percent and 50 percent of the overall system [1], [2]. Unit test is a test code that uses to evaluate the behavioral characteristics of a program against the expected product-specific behavior. Unit test code also can be affected by quality defects. Quality defect of unit tests is nonconformance of the unit tests characteristics to unit test design rules. Unit tests have their own criteria's and their violations lead to lack of maintainability.

Our research is concerned with development of technique for the detection of quality defects in unit tests for the maintenance of large-scale software systems. This approach is based on metrics. We choose two quality defects described by Meszaros [4] – Assertion Roulette and Interaction Tests. After that we define a set of metrics for these defects in terms of unit test concepts. To evaluate the technique, we apply empirical analysis methods to measure the accuracy of the quality defect detection and it usefulness.

In this work, we focus on the detection of the quality defects in test environment for OO software systems. Microsoft Visual Studio Unit Test Framework was used as the test environment. The instruments developed to support decisions of both managers and software engineers. This support includes information about where quality defects should be engaged to reach a specific configuration of quality goals.

1. Technique of quality defect detection in unit tests

We propose Assertion Roulette and Interacting Tests defects to support their detection. Quality characteristics of these defects were used for their measuring [3] and described below.

1.1. Test core concepts for quality defect detection

To avoid different interpretation it was provided a formal foundation consisting of 10 definitions. The elementary sets presented in Table 2.1 form the basis for these definitions

Table 2.1.

Symbol	Entity
tm	test method, container for a single test
PIM (m)	the set of polymorphically invoked methods of method *m*
NPI (m_1, m_2)	the set of polimorphically method invocation from m_1 to m_2
SIM (m)	the set of statistically invoked methods of method *m*
NSI (m_1, m_2)	the set of static method invocation from m_1 to m_2
TEST	test code, the set of class that either are test cases or access methods or attributes of test class cases

1.2. Assertion Roulette

Description: it is hard to tell which of several assertions within the same test method caused a test failure. A test fails. Upon examining the output of the Test Runner, we cannot determine exactly which assertion failed [4].

*Impact on Maintainability Criteria: w*hen a test fails during an automated Integration Build, it may be hard to tell exactly which assertion failed. If the problem cannot be reproduced on a developer's machine (as may be the case if the problem is caused by environmental issues) fixing the problem may be difficult and time-consuming. It has impact on traceability and understandability.

We've developed two metrics to characterize Assertion Roulette from different aspects:

- Number of assertion statements (NAS) metric counts the number of assertion statements (AS) in the test method. AS = {e, a, m}, where e – expected results, a – actual results, m – assertion message.

NAS (tm) =$|\cup_{\text{tm} \in \text{TEST}} \text{AS}|$, tm ∈ $TEST$ (3)

- Presence of assertion message (PAM) metric counts the percentage of number of assertion messages per number of assertion statements from formula (3) and described in formula (4).

AMeS = {AS | m ∉ AS}

PAM (tm) = $\frac{|\cup_{\text{tm} \in \text{TEST}} \text{AMeS}|}{NAS(tm)}$, tm ∈ $TEST$ (4)

1.3. Interacting Tests

Description: when a test depends on other tests. There is not enough information to understand the tested functionality [4].

Impact on Maintainability Criteria: This quality defect makes hard to see the relationship between the tests. Hence test doesn't do the role of tests as documentations and as consequence has impact on it readability. Also it has direct impact on test maintainability and isolation criteria in case when somebody modified or deleted another test without realizing that this test will impact on the test run. Chances for this defect increase when more tests are dependent from each other.

Several metrics were defined to characterize Interacting Tests from different aspects:

- Test methods invocation (Tinp) metric counts the number of invocations other test methods:

Tinp (tm) = $|\cup_{\text{tm}' \in \text{TEST}}$ NSI (tm, tm′)| + $|\cup_{\text{tm}' \in \text{TEST}}$ NPI (tm, tm′)|,
tm ∈ TEST

- Test method using (Tout) metric counts the number of invocation test method by other test methods:

Tout (tm) = $||PIM(tm)| \cup |SIM(tm)||$, tm ∈ TEST (6).

To summarize results we explain metrics for Assertion Roulette and Interacting Results defects. For former quality defect we can identify the test cases with a large number of assertion methods with help of AS metrics and determine assertion methods without message by using PAM.

Test methods with high values of Tinp and TOut make many invocations between test cases to satisfy the definition of Interacted Tests defect. We describe these metrics by using set theory. It helps us to determine metrics not only with help of informal description, but also with mathematical expression for more precision and understandability. These metrics will be used to automate quality defect detection in unit tests.

2. Evaluation of defect detection tool

2.1. Experiment design

Since the quality defects detection in unit tests is still an informational retrieval problem – only a few detection methods and techniques are known for a few quality defects. The following studies were conducted to explore the usefulness and effectiveness of this technique.

Rationales for investigating these casual effects are:

- Quality defect detection in unit tests might be perceived as distracting or interfering with day-to-day work; hence, the acceptance by and usefulness to software developers need to be studied.
- Precision should quantify the number of detected quality defect in unit tests that are truly quality defect; hence we should answer "Are detected defects truly quality defects"?

Research Goal: Evaluate the acceptance of quality defect detection in unit tests tool by software engineers and accuracy of defect detection.

Hypotheses:

H₁ Technology is accepted by software engineers and they intend to use it.

H₂ Technology is perceived to increase the work performance of software engineers

H₃ Technology is perceived to decrease the effort for software engineers.

H₄Technology has positive effect on a software engineers attitude toward using it.

H₅ Technology is perceived to be easy to use by software engineers

H₆ Software engineers have no negative emotions regarding the use of technology

Study Form: Controlled experiment

Background: Laboratory experiment with software developers from Itera Consulting Company

Materials: The system for quality defect detection in unit test.

Subjects: 10 software developers.

Task: Understand and detect quality defects in unit tests. Manually verify whether a test case indicated by the proposed detector indeed exhibits quality defect characteristics.

Examined cause: Using quality defect detection vs. not using it

Independent Factors: Methods vs. system

Dependent Factors: UTAUT factors (acceptance and use), detected quality defect in unit tests.

Confounding Factors: Experience factors

*Experimental Subjects***:** the experimental subjects were all developers with at least one year experience.

Requirement to the experiment was that the tool must perfectly integrate into the system's development process. We ensured that the developers were motivated; they got sweets on their lunch.

Experimental materials: during the experiment, the next main materials were provided to the subjects: a) the quality defects detection in unit tests tool for defect detection, b) training materials and literature on quality defect in unit tests, and c) object-oriented software system that was to be processed by them.

*Analyzed systems***:** quality defect detection tool was used by all subjects. They detected Assertion Message and Interacting Tests quality defects.

Execution of the Experiment: The quality defect detection in unit tests experiment was conducted in one office with the same equipment for each subject.

The software developers were informed about the experiment and the execution during 15-minute presentation.

The goal was to inform the subjects about the experimental settings and the procedure, information about quality defects in unit tests.

As a guideline, the stepwise process was described and provided to each developer in order to perform the following steps for quality defect detection in unit tests:

1) Walk through the system and detect quality defects in unit tests with help of our tool

2) Choose any 5 test methods with found quality defects

3) Inspect manually chosen test methods

4) Note your results to the protocol

After the execution phase the subjects were conducted a debriefing questionnaire, where their general attitude to system was defined.

2.2. Experimental result

In general, we can abstract the following information from the correlation and regression analysis of received results:

- Detection defects in unit tests technology is perceived to help in programming, to have a low effort to use, is a good idea, is usable without help, and isn't intimidating for software engineers.
- The subject made conclusion that the technology will helping in programming and will be accepted as good idea if it has supporting of required environment. They will intend to use the system because this technology helps in designing unit tests.
- Also we define that observation for verifying of quality defect accuracy detection should attaint agreement between reviewers related to the test design criteria. Because each of software engineers have their personal opinion about rules of unit testing design and it is hard to detect how accurately our tool detect defects in unit tests.
- Questionnaire shows us that software engineers are intended to use this tool, but we should develop metrics for more quality defects. We can consider this mean of quality defect detection as effort and time saving. It will be helpfulness for both manager and software engineer.

- Questionnaire shows us that software engineers are intended to use this tool, but we should develop metrics for more quality defects. We can consider this mean of quality defect detection as effort and time saving. It will be helpfulness for both manager and software engineer.

Conclusion

My research was concerned with development of technique for the detection of quality defects in unit tests for the maintenance of large-scale software systems.

This approach is based on metrics. For research it was chosen two quality defects in unit test for verifying this approach.

To evaluate it, we apply empirical analysis methods to measure the accuracy of the quality defect detection and it usefulness.

In this work metrics for Assertion Roulette and Interacting tests quality defects were formalized with the help of set theory to avoid misconception. It makes description more precise and understandable.

The tool for quality defect detection was designed and implemented some functionality. It helps research the value of this approach by providing case study.

It was used by software engineers to quality defect detection in unit tests.

By conducting the experiment, it was states that instruments developed to support decisions of both managers and software engineers. It is good approach to determine quality defects with using metrics for unit tests design characteristic.

Developed mean of quality defect detection has a science and practical value. It adds new direction for development and adapting this approach. It gives the bases for quality defect detection in unit tests.

Practical value was evaluated by an experiment. This mean was defined as helpful in writing and supporting unit tests by software engineers and managers.

That's why to advance the research results it should be established composed metrics that consist of a combination metrics for individual quality defect in unit tests to improve predictive power. It should be investigated the interplay between quality defects and frequently changing test cases to understand how quality defects emerge and grow.

REFERENCES:

1. A. van Deursen, L. Moonen, A. van den Bergh, and G.Kok, "Refactoring test code" , Proc. Second Int'l Conf. Extreme programming and Flexible Processes in Software Eng., M. Marchesi and G. Gussi, edc., 2001.
2. R. Sangwan and P. Laplante, "Test-Driven development in Large Projects," IT Professionals, vol. 8, no. 5, pp. 25-29, Sept./Oct. 2006.
3. Manuel Breugelmans and Bart Van Rompaey, *TestQ: Exploring Structural and Maintenance Characteristics of Unit Test Suites*. 2008
4. G. Meszaros, *xUnit Test Patterns: Refactoring test code*. Addison Whesley, 2007
5. Bart Van Rompaey, *On the detection of Test Smell: A Matric-Based Approach for General Fixture and Eager Test*. IEEE Transactions on software engineering, vol.33, no.23, December 2007
6. F. Belli and R. Crisan, "Empirical Performance analysis of Computer – Supported Code-Reviwers", Proc Eight Int'l Symp. Software Reliabiliy Eng., pp 245-255, 1997.
7. Jorg Rech, *Context-sensitive Diagnosis of Quality Defects in Object-Oriented Software Sustems.* University Hildesheim, 2009

Відомості про авторів

Павленко Анастасія – студентка 5 курсу факультету комп'ютерних наук Національного авіаційного університету, кафедра «Інженерії програмного забезпечення». Наукові інтереси – інженерія програмного забезпечення, еволюція програмного забезпечення.

E-mail: nisson@rambler.ru

Нечай Олександр Сергійович – к.т.н., старший викладач кафедри «Інженерія програмного забезпечення» факультету комп'ютерних наук Національного авіаційного університету. Наукові інтереси – створення програмного забезпечення, архітектура програмного забезпечення, еволюція програмного забезпечення, зворотна розробка, модуляція програмного забезпечення, дослідницькі методи для емпіричної розробки програмного забезпечення.

E-mail: alexander.nechay@livenau.net

ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

УДК 004.415.5.

О.В.Тегельман

Розробка вимог до СКБД на базі стандарту ISO 9126

Національний авіаційний університет

Кафедра комп'ютерних інформаційних технологій

Науковий керівник – Харченко О.Г., д.т.н., професор

Розглянуто питання розробки вимог якості до систем керування базами даних(СКБД). Пропонується використовувати підхід, що базується на моделі якості стандарту ISO 9126. У відповідності до цього підходу, вимоги користувачів до якості СКБД, що сформовані в термінах моделі якості у використанні, проектуються на вимоги зовнішньої якості для яких ставляться у відповідність вимоги внутрішньої якості.

Рассмотрен вопрос разработки требований качества к системам управления базами данных(СКБД). Предлагается использовать подход, который базируется на модели качества стандарта ISO 9126. В соответствии с этим подходом, требования пользователей к качеству СКБД, которые сформированы в терминах модели качества в использовании, проектируются по требованиям внешнего качества для которых относятся в соответствие требования внутреннего качества.

The question of development of requirements of quality is considered to control system by the bases of data(СКБД). It is suggested to take approach, which is based on the model of quality of standard of ISO 9126. In accordance with this approach, requirements of users to quality of СКБД, that the internals formed in terms of model are in the use, designed on the requirements of external quality for which belong in accordance of requirement of internal quality.

Ключеві слова: СКБД, якість програмних систем, формування вимог якості, база знань, моделі якості

Вступ

Забезпечення якості програмних систем при їх проектуванні є надзвичайно важливим в даний час, коли на ринку програмних систем (ПС) спостерігається жорстка конкуренція. Це особливо важливо для ПС масового використання, до яких відносяться і системи керування базами даних (СКБД).

Для підтвердження заявлених характеристик якості проводяться процедури атестації та сертифікації. Однак використовувані технології створення ПС орієнтовані і в основному на задоволення функціональних вимог. І хоча на даний час вже розроблено третє сімейство стандартів якості ПС [1], їх впровадження в практику йде дуже повільно. Основними причинами такої ситуації є недостатнє використання формальних методів, та відсутність ефективних інструментальних засобів автоматизації процесів забезпечення якості.

В архітектурі процесів життєвого циклу (ЖЦ) ПС представлено два процеси, які стосуються якості, це

- Підтримуючий процес «забезпечення гарантії якості».
- Організаційний процес «управління якістю».[2]

Перший з процесів реалізується шляхом впровадження стандартів якості і відповідних процедур в практику розробки ПС, а другий процес – «управління якістю» полягає в моніторингу якості проміжного та кінцевого продукту на стадіях ЖЦ ПС. Для цього необхідно представити вимоги замовника до якості ПС і виконати процедури комунікації (трасування) цих вимог на стадії ЖЦ.

Для розробки таких процедур необхідно розробити формалізовані та стандартизовані моделі представлення вимог та запропонувати деякі формальні математичні алгоритми комунікації вимог на етапах ЖЦ.

Ці задачі можна вирішити на основі стандарту з якості [1] в якому визначено три типи моделей якості: у використанні (експлуатаційна), зовнішньої та внутрішньої. Представивши вимоги якості за цими моделями й віднісши їх до відповідних фаз ЖЦ можна реалізувати процес моніторингу якості.

Аналіз технологій формування вимог

Для формування вимог до ПС можна використовувати стандарт [3] , в якому задана стру-