Pere Tumbas
Predrag Matković
Dušan Bobera

Project Risk Mitigation by Test Cases

Article Info:

Management Information Systems,
Vol. 3 (2008), No. 2,
pp 025-028

Received 12 Jun 2008
Accepted 24 October 2008

UDC 004.413.4:[005.52:005.334

Summary

Software product testing represents a key point at which software anomalies are discovered and thus induces their removal. One can safely say that discovering software product anomalies is an easier task that removing them. The later the moment in the software development process at which the anomalies are discovered, the more complex the removal of these anomalies is. The side effects of the discovered errors are unpredictable and it is necessary to find them as early in the software development process as possible. Use Case Guided Software Testing is the topic of this paper. The paper is concerned with the theoretical aspects of setting Test Cases based on Use Cases.

Key words

Test Case, UML, Use Case, RUP

Introduction

Historically, the characteristics of progress throughout the development models has been directly linked to reducing the time gap between the making and the discovery of errors in the software, lowering the risks in the software project that way. The key drawback of software development based on the waterfall model is exactly this time gap among origin error and detection error. Full completion of one development phase as a condition for another to begin, forces a conclusion that testing must be done on the product in whole. Most errors made in previous phases are often discovered only in the testing phase and the removal of these errors in order to satisfy user needs presents a significant blow to the project's budget and schedule. This type of testing is necessary due to the nature of the waterfall model itself. The paralleled time and content dimensions of the project is the key disadvantage of the waterfall model.

Based on the criticism of the waterfall model, a software development process based on iterative and incremental principles was developed. Instead of the previous division of the solution into separate parts, there came a shift into division the problem into separate parts. The time dimension was also separated from the content dimension, and that way the development process was divided into several iterations performed again based on a set of smaller waterfalls. The key effect of this risk prevention was achieved by significantly shortening the time gap between error inception and error discovery. The user was also a lot more actively included in the software development process, which further reduced the risks.

Unified Process, as a framework for object oriented software development process is based precisely on these iterative and incremental principles. Division the problem into parts in order to achieve iterativity in the UP was done by the use of Use Cases, key artifacts which guide the whole development process. The conclusion is thus that software testing in the Unified Process is also guided by Use Cases.

Use Case Guided Software Testing is the topic of this paper. The paper is concerned with the theoretical aspects of setting Test Cases based on Use Cases. The paper first defines the connection between user needs, system features, software requirements and software functionality testing. Conclusions and recommendations for successful testing guided by Use Cases are outlined at the end of the paper. Presentation of software requirements and their processing as well as setting the Test Cases in this paper was done using Unique Modeling Language (UML).

1. Position of the Testing and Project Management Disciplines in Rational Unified Process

Rational Unified Process defines the complete software development as a development process guided by Use Cases Testing as an RUP Discipline entirely fits this development model. Test Cases with clearly defined testing rules are defined based on Use Cases. For a broader problem definition

analyzed in this paper, the following Figures will show the position of creating Test Cases according to RUP. Figure 1 depicts the architecture of the Rational Unified Process. A close look at this Figure shows that testing represents one of the basic disciplines (the first six disciplines), and that it stretches over most of the duration of the project. This supports the hypotheses mentioned in the introduction:

a) Continuous testing shortens the time gap between the inception and the discovery of the anomaly in the project,
b) The user is constantly involved in the development project, as well as a significant part of testing.

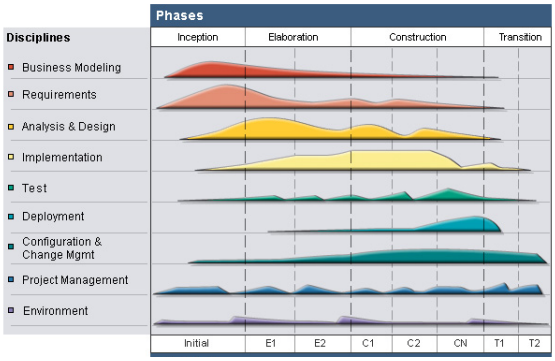

Figure 1 Rational Unified Process Architecture
(Source: Rational Unified Process for Large Projects, IBM, 2005.)

All this supports the key intention of the Rational Unified Process to early prevent large risks, therefore significantly increasing the probability of success for the development project. Rational Unified Process shows the content of each of the development disciplines in a clear and visual way by using workflows. The testing discipline workflow according to RUP is shown in Figure 2.

The workflow of the testing discipline shows activities performed during testing. The complexity of the testing discipline is clearly visible in the figure. This is supported by numerous research stating the conclusion that the testing discipline makes up for thirty to fifty percent of the total project costs. Even with the high cost, users are constantly critical of software testing. Their criticism is constantly directed at the software not having been tested sufficiently before delivery. Since none of the claims (testing costs and insufficiency) can be refuted, the complexity of testing is only thereby emphasized.

Besides testing insufficiency, empirically, the following claims about the testing process are implied: Testing is mostly done without a clearly defined methodology; Testing tools are rarely used.

The validity of these claims jeopardizes the development project, and the more security demanding the projects are, the more jeopardy there is for core business operations that are supported by the software product tested that way. Producing an air traffic control system, a software that controls medical equipment, all kinds of financial software products are just some projects that demand a serious approach to the testing process. That is why RUP devotes great significance to the testing process, trying to circumvent the above claims.

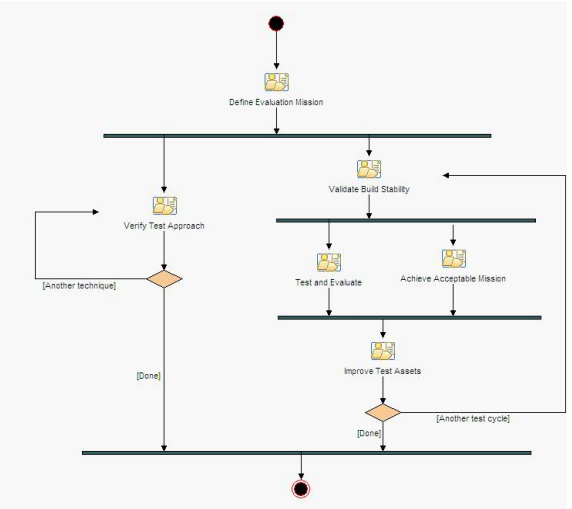

Figure 2 Test Discipline Workflow according to RUP
(Source: Rational Unified Process for Large Projects, IBM, 2005.)

2. Defining Details and Traceability of the Test Case

Activities shown in Figure 2 are testing activities at the highest level of abstraction. Specific activities appear in the form of a multitude of sub-activities performed by specific roles, members involved in the testing process. Different artifacts appear as inputs and outputs for specific activities.

Defining the details of a Test Case represents one of the activities performed in the testing discipline within the top-level activity Test and Evaluate. The Test Analyst is in charge of performing this activity. This activity defines the testing algorithms for each of the tested functionalities separately. The following artifacts appear as inputs for this activity[1]: Test Strategy,

[1] Source: Rational Unified Process for Large Projects, IBM, 2005.

Test-Ideas List, Test Data, Use Case, Supplementary specification, Change request and Design Use Case Realization.

Test Strategy, Test-Ideas List and Test Data are artifacts created in the testing discipline. The rest of the artifacts are created in the requirements and the analysis and design disciplines. As mentioned before, the RUP defines complete development as a development process guided by Use Cases. Use Cases are software requirements structured based on the identified user needs, the analysis of the current state of the business system, and the analysis of the existing information system. On one hand, they are a contract between the investor and the contractor, since they define the scope of the future project, but on the other, they are the guidelines for all the participants in the process. Therefore, Use Cases tell users what to expect from the future software product, tell development engineers what to develop, tell people in charge of documenting things what to document, and tell testers what to test. Since Use Cases assume the functionalities of the future system defined by user demands, it is entirely logical that they appear as an input in defining the details of Test Cases.

Use Cases are presented through Use Case diagrams, one of the thirteen official UML diagrams. Use Case diagrams are a broad and not too detailed („mile wide inch deep") representation of system functionality. Actually, one can say that Use Case diagrams define the scope of the system. These diagrams are supposed to answer the question of what the system does. The syntax used in UML notation for this purpose is very modest and easy to understand. However, that does not mean these diagrams are easy to concoct. Quite the contrary, making the Use Case diagrams is extremely complex and takes a lot of time and discussion until a satisfying solution can be made. Elements used to design them are Actors, Use Cases and relations which link the elements together. Every Use Case represents a separate, fully executable functionality of the system. Exceptions to this rule are Use Cases linked by content relations. This trait allows for creating Test Cases based on Use Cases.

Every Use Case represents a functionality of the future system that can be executed in a number of ways. There is always one, so-called basic scenario, which performs the exact thing the functionality was designed for. Examples of this may include:

- Use Case: Withdrawing cash from an ATM ➥ Basic scenario: Cash successfully withdrawn from the ATM
- Use Case: Submitting an application for membership ➥ Basic scenario: Membership application successfully submitted

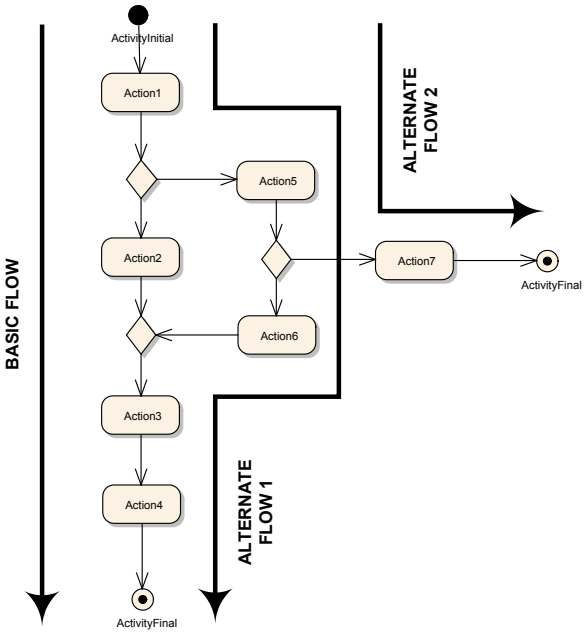

Figure 3 Basic Flow of Events and Alternate Flows of Events for a Use Case

Besides the basic scenario, there are also a lot of other scenarios which could execute the functionality of Use Cases. All these other scenarios which are able to execute the Use Case functionality are called alternative scenarios. For the testing process, it is essential to accurately perceive all the potential scenarios and include them in Test Cases.

The process of creating Test Cases is done in the following three steps[2]:

- For each Use Case all potential scenarios are defined,
- For each scenario at least one Test Case and conditions for the execution of the scenario are defined,
- For each Test Case the inputs and outputs are defined, based on which the testing will be performed.

Since UML version 2.0, activity diagrams have considerably improved, thereby creating a basis for a visual description of even the most complex functionalities presented through Use Cases. Well designed activity diagrams are artifacts, by analysis of which one can very well identify all the potential

[2] Jim Heumann, Generating Test Case from Use Case, The Rational Edge, June 2001.

scenarios for executing the Use Cases. They are at the same time a suggestion for the first step in creating Test Cases. For step two, a great help are also the activity diagrams. Since they graphically present all actions taking place when certain functionality is being executed, as well as the conditions that cause them, one can very simply take the second step in creating Test Cases. Looking at a Test Case from a „Black Box" perspective, the inputs and outputs are the only things that should be assumed. From a tester's perspective, each functionality and each of the scenarios being tested are black boxes. A tester is expected to give a detailed report showing which Test Cases were successful, and which were not. After testing, the tester drafts a report that suggests which functionalities should be corrected, and which were well implemented.

Conclusion

The aim of every approach to software development is to minimize key risks of the development project. The iterative approach has allowed for the bases for testing smaller parts of the system, from the early phases of the project, time-wise. This, in turn allowed for more quality of interaction with the user during the project. Since testing starts early in the development project, the feedback from the testing process helps to painlessly overcome the anomalies in the project.

With the testing discipline, the RUP has given a lot of suggestions on how to implement testing. When defining a specific methodological framework, it is necessary to define specific rules for testing as well. This paper gave a suggestion to define Test Cases based on Use Cases and the derived artifacts to do se. By applying a clear methodology, clear testing rules which simplify the testing process, raise the efficiency of the development process, and lower the costs of the development project, are also defined. Through the application of the RUP, testing can be seen as a separate and independent part of the project. According to RUP testing is invariably integrated with other disciplines in the project by way of the iterative approach. If Test Cases are identified well and set up in a way suggested in this paper, the tester can easily implement the testing process. Of course, the suggested method is no rule to strictly adhere to in the testing process. The great quality of the RUP is that it defines general suggestions that can be implemented by using a great number of different techniques. The important thing is to define in detail the way testing will be performed in the implementation phase.

References

IBM Team, Rational Unified Process for Large Projects, IBM, 2005.

Jim Heumann, Generating Test Case from Use Case, The Rational Edge, June 2001.

Paul Szymkowiak, Philippe Kruchten, Testing: The RUP Philosophy, The Rational Edge, February 2003.

Peter Zielczynski, Traceability from Use Cases to Test Cases, www.ibm.com/developerworks/rational/library/04/r-3217, May 2006

Pere Tumbas

University of Novi Sad
Faculty of Economics Subotica
Segedinski put 9-11
24000 Subotica
Serbia

Email: ptumbas@subotica.net

Predrag Matković

University of Novi Sad
Faculty of Economics Subotica
Segedinski put 9-11
24000 Subotica
Serbia

Email: pedja_m@ef.uns.ac.rs

Dušan Bobera

University of Novi Sad
Faculty of Economics Subotica
Segedinski put 9-11
24000 Subotica
Serbia

Email: bobera@ef.uns.ac.rs