Automated generation of direct guidance of the tester based on design model of the tested application

Karel Frajták

Department of Computer Science and Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague, Karlovo náam. 13, 121 35 Prague 2, CZ

frajtak@fel.cvut.cz

Abstract. *Low error rate and reliability of a web application are important aspects to its success. Errors discourage the target users of the application. According to the testing theory it is quite impossible to deliver 100% reliable application. Efficiency of testing is very important, by choosing the right tool the time between discovering the error and its elimination can be reduced, which has positive impact on project economics and quality of the developed application. We are proposing new approach to application testing based on the model of the system with the help of an application for tester's guidance through the testing process. In comparison with classical manual testing using written or automatically generated test case scenarios our approach makes the testing process more trustworthy by verifying work of the tester. It gives better insight into testing process by collecting the testers reports and monitoring metrics.*

Keywords

model–based testing, test automation, test case scenario generation, web–based applications, software development life cycle, quality assurance, test coverage

1. Introduction

Success and profit of a web application is often significantly influenced by error rate of the application. Target audience will stop using the application when random errors occur randomly in non–deterministic time intervals. The success of the web application depends on how reliable it is and how quickly the errors are eliminated and fixed.

Errors discovered in later development stages low are most significant problem. In classic waterfall model of development process (analysis — design – implementation — verification — maintenance, as depicted in Figure 1 as R–D — C–U — I–S — E–R — P–R) testing (verification) stage is included as the last but one stage. It is possible that an error made in design stage is detected much later in testing stage or in maintenance stage.

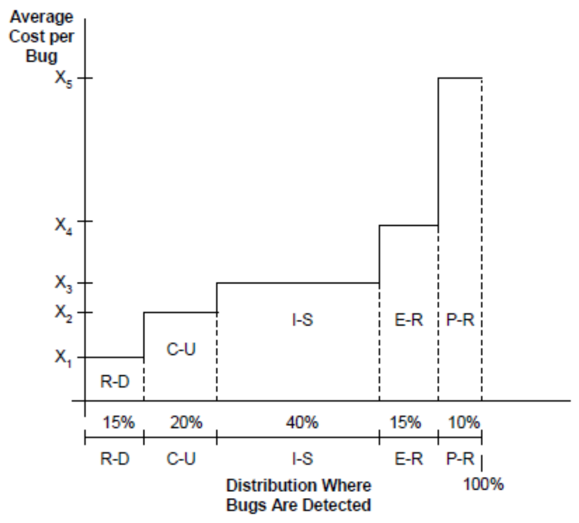

Fig. 1. Average cost per bug shown by where bugs are detected [6]

There is direct correlation between the time of discovering an error and costs to fix it (see Figure 1).The later the error is discovered, the higher is the price to fix the error. Fixing error in a released application in production environment is very expensive and occurring errors discourage users from further use. This is the reason to find effective way to discover errors in developed system as early as possible.

With an appropriate and effective methodology for error detection and elimination in the design stage the leakage of hidden errors into the later stages of the software development life cycle is can be minimized. Moreover, fixing errors in early stages is much more cost effective in comparison with later stages.

System errors or other violations against the system requirements can be detected in testing phase using intensive testing of the whole system. Different approaches to software testing can be chosen to be used in this phase.

The unit tests are executed repeatedly to discover the errors made during development or when a new feature is introduced. Only single subsystem or individual module is tested in unit testing phase. The more subsystems are involved in unit tests, the more complicated is the initialization of the test case — the subsystems that are not directly under test must be mocked or proxies have to be created to simulate their functionality.

Integration testing is the next level of testing — individual modules are combined and tested as a group. With increased complexity of the system testing automated testing process won't cover all the boundary cases — complex test case scenarios or difficult end–to–end through of the testing process. These types of test case scenarios are typically prepared manually by test architect. These manually created test case scenarios are given to testers who follow each step defined in the scenarios and report back the result — the success or failure of the scenario.

According to [3] test coverage is defined as ratio between number of units covered by test and the total number of units, for example number of methods covered by test and total number of methods ratio. The higher the test coverage of the system is the lower is the probability of hidden error in the system. However it impossible to manually create and execute tests for the whole system and reach 100% test coverage. Although the tests can be automatically generated and executed, there are still some parts of the system that cannot be tested using this approach. It is problematic to create automatic tests or test case scenarios for end–to–end testing or for user interface testing.

A test case scenario instruct tester and tells him what to do in every step of test case scenario. Every test case scenario consists of one or more test case scenario steps — each step contains instructions describing actions that must be taken and conditions that must be met to move to another step. Preconditions (conditions that must be met for the scenario to be started) and postconditions (conditions met after scenario has ended) can be defined in the scenario. Test scenario fails when any of the defined conditions is not met. The user could have filled in wrong value or new error was discovered.

2. Our goal

We are focusing on two areas — testing of the components and subsystems of the system that are not easily testable with unit tests. And automated generation of such test case scenarios with tester direct guidance in mind. In most cases tester follows test case instructions written in the electronic document or in document created by test design tool. Tester carries out every single step he had been instructed to and writes down the results of the test or bug report.

In our solution we are going to eliminate this paperwork. During the testing tester will be instructed what to test by an interactive guideline system instead of performing manual test cases defined in a textual form. This application will guide the tester through the test case; it will make him to meet the defined conditions. Model of the application is the key to locate the pointcuts[1] for the guide and for defining test case conditions.

We believe that use cases are essential part of the system usage "how–to" and that they should not be only used in analysis part. Shipping the test case scenarios with the product is not usual approach and this relation is lost. The end–users have to read the manual to handle complex usage scenarios.

3. Related work

Our goal is the automated test case scenario generation for on–line guidance of the tester (not just generating of text description of the tests). A model describing the system is required. In this model following features will be described

- Domain entities, their properties and relations between the entities – test case constraints or conditions will be defined using properties of the domain model entities, for example Customer, Order, etc.
- Entities not in domain – i.e. entities that are not part of the problem domain but can be used to define test case constraint/condition, for example system configuration entities
- Domain methods – model will contain signatures of domain methods that can be used in use cases
- Metadata – definition of non–model requirements
- Use cases – description of system processes using the domain entities, not–in–domain entities, domain methods, use case model can be extended using constraints/conditions

Model of the system under test is expressed as labelled transition systems in [7]. The model based testing starts with a valid model. Without a model or when system uses legacy or third party components, test–based modelling generates models from observations made during testing using kind of black–box reverse engineering.

UML diagrams are commonly used for test case generation, for example [5]. Another approach is described in [4] where test scenarios are synthesized using UML activity diagrams. UML activity diagrams are used by developers to describe scenarios of use cases. A test–ready meta model is generated for every use case in [8], use case specify the functionality of the system but they require some additional features to become useful.

UML diagrams can express many aspects of the application, for example class diagram describes system classes

[1] In aspect–oriented computer programming, a pointcut is a set of join points. Whenever the program execution reaches one of the join points described in the pointcut, a piece of code associated with the pointcut (called advice) is executed.

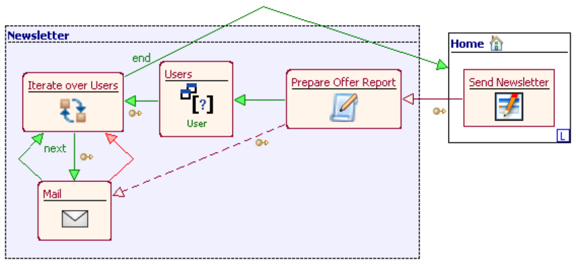

Fig. 2. Example of WebML diagram: Newsletter sending operation chain [10]

and their relations, sequence diagram is used for describing interaction between system components, but some aspects of the application require more detailed description. Expressiveness of UML diagrams is limited to use the diagrams to model complexity of web application – the relationships between web pages, their input and action elements – i.e. to design navigation schema of the application.

Using web modelling language WebML [1] to model the application has some advantages over UML. WebML is a graphical notation like UML extended with means of defining how to separate data model (the content of the web pages), hypertext model (structure of the system and navigation between web pages) and presentation model (user interface), see Figure 2 to see an example of diagram for the newsletter sending operation chain.

Use cases describe system requirements, UML state machine diagrams capture the behaviour of a system and could serve as a basis to automate test case generation. Automated support for the transition from use cases to state machines would provide significant, practical help for testing system requirements. Additionally, traceability could be established through, which could then be used for instance to link requirements to design decisions and test cases, and assess the impact of requirements changes as described in [9].

The model [2] describes the interaction of an external user with the website which is defined as “run”. For every run web page generates inputs for the user querying the database or the application state, the user fills in data or inputs at most one tuple among the options provided and then a state transition occurs. Actions are taken and the next web page is determined according to the specification. A data–driven web application is described as a set of web page schemas and has following components

- A database
- A set of state relations
- A set of web page schemas (web pages), of which one is designated as the “home page” and another as an “error page”
- Each schema defines how the query on the database and state is defined by the set of current input values

The model is formalized by temporal language – a variant of linear–time and branching time temporal logic – for specifying properties of web application. Authors are focusing on verification of web applications – the run is called error free if an error page is not reached and web application is called error free when it generates error free runs only.

User interaction with the web page schema is described by input and action schemas. On a single page represented by web page schema user input is captured in input schema. User then executes one of the actions available in the action schema. Resulting sequence of events is then verified.

4. Proposed solution

Formal description of a web application is the cornerstone of our proposed solution. We are targeting 100% test coverage with generated test case scenarios. Also we have defined custom metrics to calculate the coverage.

4.1. Formal model

Our system requires model of a web application as source for automated test case scenario generation. Model formally describes the system under test. Our proposed model uses following components to define the web application

- set of container schemas representing each web page used in web application
- links defined between web pages defining conditions and transitions from one page to another
- set describing input, resp. action, values for each web page representing values of input elements (text box, check box), resp. action elements (button, hyperlink), placed on a web page
- technical, resp. business, metadata that are used to describe non–model requirements

Test case scenario consists of individual step and each step is defined using following components

- web page – where the step should be executed
- action – what action should be executed
- constraint - what conditions must be met for given step
- data - which data is used in given step

When a constraint is violated user can fix the problem either by entering the correct input values or by aborting the test case scenario and create an error report and provide feedback. In case of an unexpected error (this often indicates

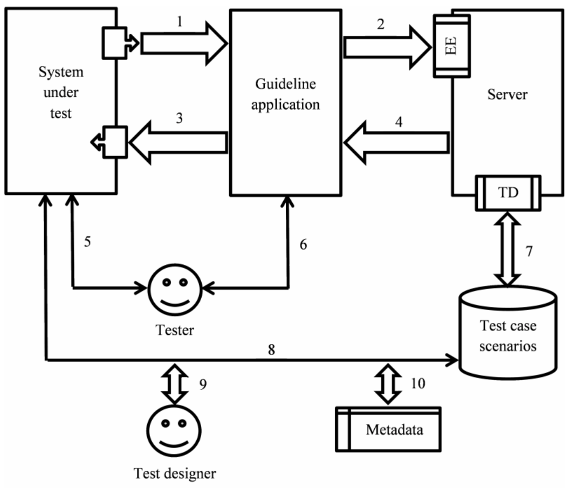

Fig. 3. System architecture

a problem inside the system) the scenario is automatically aborted and the error log is created. Problem is reported back to the developers with the context of the SUT containing all data necessary to reproduce the error so they can fix it. Tester can provide additional feedback before the error report is submitted.

4.2. Proposed system architecture

Principle of the system is displayed in Figure 3: tester interacts with SUT (5) and the guideline system (6) at the same time. Test case scenario is loaded from the database (7) by the test dispatcher (TD) from the server and passed to guideline system (4). Each tester's step in SUT is passed to guideline system through extension points (point–cuts) added to SUT (1). Each step is the passed to the server (2) and conditions or constraints defined for given test case scenario step are evaluated in expression engine (EE). Feedback is provided to the tester in guideline system. Formal model is used to generate test case scenarios (8) and store them in database. Scenarios can be also created manually by test designer (9). Metadata describing non–model requirements are applied to generated test cases (10).

5. Conclusion

We have presented model based improvement of testing process of the web application. We focused on elimination of the unnecessary paperwork (which has significant positive economic impact on project) by introducing the formal automatic test case specification. By using our approach test managers can verify the work of the testers — the guideline application tracks the progress of the tester, records his actions and verifies all conditions defined in each test case scenario step taken by the tester.

Our solution is based on automatic test case scenario generation process. We are not going to create test scenarios in a text form, but as a structured data for the guidance of the tester. According to our experience with software testing this approach is highly promising to be more efficient. Automation of this process helps to generate test case scenarios effectively covering the system under test providing higher test coverage. Model of web application formalized by a formal model will be used as a basis for test case scenario generation process.

Acknowledgements

Research described in the paper was supervised by Ing. M. Bureš, PhD., FEE CTU in Prague and supported by the CTU under under grant No. SGS11/157/OHK3/3T/13 and by MSMT under research program No. 6840770014.

References

[1] ACERBIS, R., BONGIO, A., BRAMBILLA, M., BUTTI, S., CERI, S., FRATERNALI, P. Web Applications Design and Development with WebML and WebRatio 5.0. In *Objects, Components, Models and Patterns*. Springer Berlin/Heidelberg, 2008, p. 392 - 411.

[2] DEUTSCH, A., SUI, L., VIANU, V. Specification and verification of data-driven Web applications. In *Journal of Computer and System Sciences*. Academic Press, Inc., Orlando (USA), 2007, p. 442 - 474.

[3] *International Software Testing Qualifications Board Syllabus Foundation Level*, 2012, http:/www.istqb.org.

[4] NAYAK, A., SAMANTA, D. Synthesis of Test Scenarios Using UML Activity Diagrams. In *Software System Modelling*, Springer-Verlag New York, Inc., 2011, vol. 10, p. 63 - 89.

[5] SAWANT, V., SHAH, K. Construction of Test Cases from UML Models, *Systems and Management*, Springer Berlin/Heidelberg, 2011, p. 61 - 68, ISBN 978-3-642-20209-4.

[6] TASSEY, G. The Economic Impacts of Inadequate Infrastructure for Software Testing, *National Institute of Standards and Technology*, 2002.

[7] TRETMANS, J. Model-Based Testing and Some Steps towards Test-Based Modelling, *Formal Methods for Eternal Networked Software Systems*, Springer Berlin/Heidelberg, 2011, p. 297 - 326, ISBN 978-3-642-21454-7.

[8] WILLIAMS, C. E. Toward a Test-Ready Meta-model for Use Cases. In *Proceedings of the workshop on practical UML–based rigorous development methods*, 2011, p. 270 - 287.

[9] YUE, T., ALI, S., BRIAND, L. Automated Transition From Use Cases to UML State Machines to Support State-based Testing. In *Proceedings of the 7th European conference on Modelling foundations and applications*. Springer Berlin/Heidelber, 2011, p. 115 - 131.

[10] WebRatio wiki. How to use a generated Report as attachment, 2012.

About Authors...

Karel FRAJTÁK was born in Benešov, Czech Republic. He graduated from the CTU in 2005 with a thesis 'Query language over hierarchy of objects'. He is currently doing his PhD at the same university. He is member of Webing research group lead by doc. Ing. Ivan Jelínek.