International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:5, No:8, 2011

Manual testing of web software systems supported by direct guidance of the tester based on design model

Karel Frajták, Miroslav Bureš, and Ivan Jelínek

***Abstract*—**Software testing is important stage of software development cycle. Current testing process involves tester and electronic documents with test case scenarios. In this paper we focus on new approach to testing process using automated test case generation and tester guidance through the system based on the model of the system. Test case generation and model-based testing is not possible without proper system model. We aim on providing better feedback from the testing process thus eliminating the unnecessary paperwork.

***Keywords*—**model based testing, test automation, test generating, tester support.

I. Introduction

IN development of web application, success or profit is often significantly influenced on error rate of the application. Random errors occurring in non-deterministic time intervals make the target audience of the application to dislike it. So the success of the system depends on how reliable it is and how fast can be the errors eliminated and fixed. If the errors are detected in the early stages of the development process the cost of fixing them is low.

Errors can be detected using intensive testing of the whole system. If the test coverage of the system is very high, the probability of hidden error is very low. 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.

In our proposal we will focus on two areas - testing of the parts of the system that require more attention than the parts testable with unit tests and on automated generation of such test cases. In most cases tester follows test case instructions written in the electronic document. Tester carries out every single step he had been instructed to and writes down his feedback. In our solution we are going to eliminate this paperwork. During the testing, tester will be instructed what to test by an interactive guideline application, instead of performing manual test cases defined in a text form. This application will lead the tester through the test case; it will make him to fulfill the defined conditions. Model of the application is the key to finding the point-cuts for the guide and for defining test case conditions.

K. Frajták is with Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University, Prague, Czech Republic (e-mail: frajtak@fel.cvut.cz).

M. Bureš is with Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University, Prague, Czech Republic (e-mail: buresm@fel.cvut.cz).

I. Jelínek is with Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University, Prague, Czech Republic (e-mail: jelinek@fel.cvut.cz).

II. Model for automated test case generation and related work

For our intention, a model describing the system is required. For our purpose, this model will describe the

- 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 labeled transition systems in [1]. The model based testing starts with a model that is presumed correct and valid. When designers do not make models, or system uses legacy or third party components, test-based modeling aims at generating models from observations made during testing using kind of black-box reverse engineering.

Very common approach is to use UML model for test case generation, for example [2]. Another example of possible approach is described in [3]. Here, test scenarios are synthetized using UML activity diagrams. UML activity

diagrams are used by developers to describe all possible flows of controls commonly known as scenarios of use cases.

Although UML diagrams can express many aspects of the application – class diagram describes system classes and their relations, sequence diagram is used for describing interaction between system components, component diagram is suitable for describing deployment of the system components on system machines or nodes – some aspects of the application require more subtle description. Expressiveness of UML diagrams is limited and for example it is not possible to use UML 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.

Another possibility is to use web modeling language WebML [4] to model the application. 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).

Use cases are used to document system requirements, UML state machine diagrams describe the behavior of a system and 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 [5].

Mathematical model defined in [6] is trying to capture portion of real specification languages provided by high-level tools for web application specification, such as WebML. The model captures the interaction of an external user with the website which is defined as "run". For every run web page generates a choice of inputs for the user querying the database or the application state, the user chooses or inputs at most one tuple among the options provider and then a state transition occurs. Actions are taken and the next web page is determined according to the specification.

According to [6] a data-driven web application 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.

A. Model driven approach

Model driven engineering (MDE) advocates usage of models and transformations to support all tasks of the software development from analysis to testing. Modern MDE technologies use various models to represent different perspectives of the system at a different level of abstraction. The paper [7] presents a model transformation framework for forward engineering stream that goes from computation independent model (CIM) to application code and the testing stream going from computation independent test (CIT) specification to executable test script. In [7] authors are describing vertical transformation for composing the two streams and horizontal mapping for reflecting changes made in the modeling framework.

Authors concentrates on the chain of transformations for producing tests and it defines metamodels for representing test cases for web applications at different levels of abstraction and on supporting of automatic alignment of the platform independent test (PIT) specification after changes made to PIM. Different modeling languages are used in different levels – Business Process Modeling Notation (BPMN) for CIM and WebML for Platform Independent Model (PIM). The WebML model enriches the BPMN process scheme with operational details

In our proposal we will reuse the differentiation of abstraction levels and targeting platform specific/independent model/test for the modeled application. Synchronization mechanism between application model and test keeps ensures that changes made to application model will be reflected in generated test cases.

B. Covering decisions and conditions

In [8] there is proposed new criterion for software testing. The requirements for testing logical structure of program are specified using control flow criteria. The aim of these criteria is testing decisions (a program point at which the control flow can divide into various paths) and conditions (atomic predicates which form component parts of decisions) in program. A decision coverage criterion states that every decision in the program has taken all possible outcomes at least once. Multiple condition coverage criterion requires 2n test case for decisions consisting of n conditions. Modified Condition/Decision Coverage (MC/DC) criterion reduces the number of necessary test cases. It requires testing of every independent condition in decision.

The definition of MC/DC criterion is the following: Every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken on all possible outcomes at least once, every decision in the program has taken all possible outcomes at least once, and each condition in a decision has been shown to independently affect the decision's outcome. A condition is shown to independently affect a decision's outcome by varying just that condition while holding fixed all other possible conditions.

MC/DC has some shortcomings, for example the independency of the conditions in decision. Reinforced Condition/Decision Coverage (RC/DC) criterion proposed in [8] is focusing on eliminating of the shortcomings of MC/DC criterion. Further details can be found in [8].

Our proposal focuses on testing those parts of application that cannot be easily unit tested and involve tester interaction. We will focus on covering of the code handling user interaction with test cases. So MC/DC can be used for our needs modifying its definition - every point of handling user interaction (a method handling button click event, menu click event etc.) and exit in this handler has been invoked at least once.

We will use this approach for generating test cases automatically from the application model. We would like to generate limited number of test case scenarios with the highest test coverage of the system under test. It can be also used for generating test cases using reverse engineering from existing application – test cases will be generated focusing on covering code handling user interaction.

III. Basic principles and model architecture

For our purposes described above, we decided to reuse and adopt use certain parts of mathematical model from [6]. Model of the web application will be extended to include specification of metadata. While [6] focuses on web application schema verification and defines extensive mathematical model, we focus on the tester involved in testing of web application. Our target is to define test case runs (runs defined in [6] and extended with new properties and relations) and use them for defining test case scenarios. Test case run properties and relations will be used for defining metadata and run conditions to be satisfied for a successful test case run. Tester will then follow the requirements defined in each test case run and his task will be to satisfy all the requirements and conditions.

After summarizing our model requirements and analyzing related work, our model will be extension of the modified model defined in [6] and

- it will describe data-driven web application (based on model in [6])
- it will be used for defining test case runs (based on runs in [6])
- it will be used for automated generating of test case runs
- it will extend the model with the metadata properties and relations for defining test case runs
- include metadata properties for defining non-model requirements
- the application will be described using WebML notation

Our model is describing the web application that includes the domain and non-domain entities and their properties, relations between entities and metadata expression non-model requirements. The key concept for generating test case scenarios is the use cases. Use case will be also described using our model; therefore our model will describe the domain methods too.

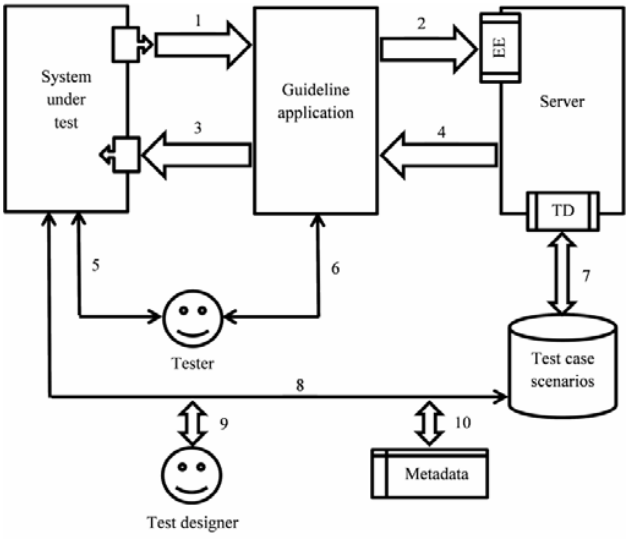
In our proposal a web application is a tuple ⟨**D**, **S**, **I**, **A**, **W**, **M**, **T**⟩, where:

- **D**, **S**, **I**, **A** are relational schemas called database, state, input and action schemas
- **W** is a finite set of web schemas
- **M** is a finite set of business metadata,
- **T** is a finite set of technical (non-model) metadata.

Each step in test case scenario is an interaction between the user/tester and the system/SUT including the database state, user's action on a web page, inputs to the system and states of the system. This can is be described by proposed test case scenario "run" and its configuration defined as tuple ⟨V_i, S_i, I_i, A_i, R_i, TD_i⟩, where

- $V_i, \in$ **W** is the web page schema in the *i*-th run,
- S_i is an instance of **S** describing the state of the application,
- I_i is an instance of **I** describing the input provided by the user,
- A_i is an instance of $\mathbf{A}_{Vi}$ describing the action taken by the user,
- R_i is a set constraints for data used in the test case scenario step,
- TD_i is set of testing data that must satisfy all constraints in R_i.

Above-mentioned model which basic concept we have



outlined will be cornerstone for our system for guiding tester through test case scenarios. Architecture of the system is described in following text.

IV. System architecture

Based on the model described above, guideline application

for tester will be designed and implemented.

In this application, tester will select test case scenario from the list of available scenarios and it will be loaded into the guideline application. All steps and requirements/conditions will be displayed in the application in human readable form – for example "Create an order with 5 items." The guideline application will not let the tester continue in testing the scenario until he fulfills the required condition. This application will be connected with system-under-test (SUT) through messaging subsystem. One of our tasks is to specify the point-cuts in SUT creating one end point of the messaging channel.

In the implemented system server application will maintain database of test case scenarios. We will generate test case scenarios automatically using the model describing SUT. Steps of the scenario and the constraints will be extracted from the use cases described by our model.

Server side application will serve the scenarios to the tester based on tester's privileges, SUT components availability, testing priority or others. The guideline application will not offer tester a test case scenario for testing unavailable system components or it will not force tester to test not accessible or restricted system components.

Fig.1 depicts principle of the system: tester is interacting with system under test (5) and guideline application (6) at the same time (see Fig. 1). Test case scenario is loaded from the database (7) by the test dispatcher (TD) from the server and passed to guideline application (4). Each tester's step in SUT is passed to guideline application through extension points (point-cuts) added to SUT (1). Each step is the passed to server (2) and evaluated in expression engine (EE). EE evaluates conditions/constraints defined for given test case scenario step. Feedback is sent back and provided to the tester in guideline application and to SUT (2).

Model of the SUT is used to generate test case scenarios (8) and store them in the database. Scenarios can be created manually by test designer (9). Metadata describing non-model requirements can be applied to generated test cases (10).

V. Conclusion and Future Work

We have presented an approach to support testing of web applications. In order to use automatic generation of test case scenarios and to support tester guidance through testing, model for describing the system is proposed. This model describes various parts of the system including domain and non-domain entities and use cases. We have decided to use model defined in [6] as a basis and adopt and extend this for our purpose. We will use WebML to describe the structure of the web application – composition model to describe the pages; navigation model to express how pages and content units are linked to form the hypertext; and presentation model to describe the appearance of pages. Metadata properties will extend the model to describe non-model requirements.

Model of the application will be used to generate test case scenarios that will be used in our guideline application to support testing process. The guideline application will guide tester through the loaded test case scenario. Tester will have to go through the steps of the scenario and fulfill the defined conditions/constraints. This process will provide better feedback than forcing the tester to fill in test case description documents. We have also proposed the architecture of our system and discussed the way of tester interaction with the system and SUT.

In the future work we will finish the implementation of proposed system components – guideline application, server side application and communication subsystem. We will verify our proposal on real software development projects and evaluate the feedback. This feedback will be then used to improve the model and implementation iteratively.

References

[1] J. Tretmans, "Model-Based Testing and Some Steps towards Test-Based Modelling" in Formal Methods for Eternal Networked Software Systems, vol. 6659, Springer Berlin/Heidelberg, 2011, pp. 297-326.

[2] V. Sawant, K. Shah, "Construction of Test Cases from UML Models" in *Technology Systems and Management*, vol. 145. Springer Berlin/Heidelberg, 2011, pp. 61-68.

[3] A. Nayak, S. Debasis, "Synthesis of test scenarios using UML activity diagrams" in *Software and Systems Modeling*, vol. 10, Springer Berlin/Heidelberg, 2011, pp. 63-89.

[4] M. Brambilla, S. Comai, P. Fraternali, M. Matera. "Designing Web Applications with WebML and WebRatio" in *Web Engineering: Modelling and Implementing Web Applications* (Human-Computer Interaction Series), G. Rossi, O. Pastor, D. Schwabe, L. Olsina (Eds.).. Springer, October 2007.

[5] T. Yue, S. Ali, L. Briand, "Automated Transition from Use Cases to UML State Machines to Support State-Based Testing" in *Modelling Foundations and Applications*, vol. 6698, Springer Berlin/Heidelberg, 2011, pp. 115-131.

[6] A. Deutsch, L. Sui, V. Vianu, "Specification and verification of data-driven Web applications" in *Journal of Computer and System Sciences - JCSS*, vol. 73, no. 3, Los Angeles, 2007, pp. 442-474.

[7] P. Fraternali, M. Tisi, "Multi-level Tests for Model Driven Web Applications" in *Web Engineering, Lecture Notes in Computer Science*, vol. 6189. Springer Berlin/Heidelberg, 2010, pp. 158-172.

[8] S. Vilkomir, J. Bowen, "Reinforced Condition/Decision Coverage (RC/DC): A New Criterion for Software Testing" in *ZB 2002:Formal Specification and Development in Z and B*, vol. 2272, Springer Berlin/Heidelberg, 2002, pp. 229-239.