Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Supporting Exploratory Testing by Automated Navigation Using the Model of System Under Test

By
KAREL FRAJTÁK
A summary of the dissertation thesis submitted to
the Faculty of Electrical Engineering, Czech Technical
University in Prague,

in partial fulfillment of the requirements for the degree of Doctor.
PhD Programme: Electrical Engineering and Information Technology
Branch of Study: Information Science and Computer Engineering

August 2017

Supervisor: doc. Ing. Ivan Jelínek, CSc.
Supervison specialist: Ing. Miroslav Bureš, PhD.

1. Introduction

As a stage where the defects present in the system should be discovered and fixed, software testing represents an important part the software development lifecycle (Kaner et al., 2008). The current software testing is also considered to be a costly process. Eldh et al. (2006) estimated its costs to be even between 40% and 80% of the total costs of the development. As the First Boehm's law, based on empirical evidence from an extensive number of software development projects says, the price of defect detection and removal grows with the stage of the software project. Correcting a wrong requirement in a requirement catalogue can be a matter of a half-hour discussion with the investor. Correcting a defect caused by this wrong requirement in production stage can impose extensive costs: It is not only expensive because the application must be reworked and released again, but it can annoy users to the point when they stop using the tested system, which can result in loss of potential income acquired by the tested system.

To ensure efficient testing of software projects, various strategies can be taken. On one side of the spectre, we can position the Model-Based Testing, where high coverage test cases are generated from a suitable model of the System Under Test (SUT). On the other side of the spectrum, Exploratory Testing (ET) can be positioned. Here, test case scenarios are not prepared in advance and are defined during the testing process. Thus, Exploratory Testing is suitable for projects, where test basis is not present, or at least not complete and consistent to the extent allowing the creation of efficient test cases. In this Dissertation Thesis, we focus on a fusion of these two approaches, despite the fact, that such a combination may seem unusual.

The Exploratory Testing is software testing technique, which applies to software development projects, in which test basis (design documentation) is not available, or at least not complete and consistent enough to allow the creation of efficient test cases. In ET, we explore the SUT by testing its functionalities and together with that we document the explored parts of the SUT and exercised test cases.

The documentation part is essential for the efficiency of ET technique. Documentation of the explored paths in the SUT also allows more accurate

reports of found defects, together with the possibility of the more systematic creation of the test cases during the exploratory testing process, preventing possible duplicities in executed tests. Created test cases can be used later in the next testing phases (retesting of fixed defects or regression testing, for instance).

Currently, a large ratio of web applications is usually developed without any or sufficient underlying models. In most of the cases, this is a consequence of required low development costs and usually short delivery time in the competitive software development environment. Nevertheless, the model is still implicitly present in the SUT code and with proper techniques, it is possible to reconstruct it from the SUT. In this thesis, we explore one of the prospective possibilities how to provide an automated testing support for such a situation.

2. Goals of the Thesis

The goals of this Dissertation Thesis are the following:

1. Explore the possibility how to aid an Exploratory Testing process by a suitable automated support, which will (1) take over part of administrative overhead related to operational management of testing team, (2) will lead to more efficient exploration of SUT functions, (3) enables the testing team to detect more defects in the SUT and (4) will prevent duplication of executed tests by individual testing team members. By operational management in this context, we mean the assignment of particular testing tasks to individual testers and documentation of already explored SUT functions and exercised test data combinations.
2. Design a model of SUT, which will serve as a basis for this automated support. Explore already existing screen-flow based models and either adapt and adjust an existing model or to design an own model, if the situation requires.
3. Use the defined SUT model for a real-time generation of navigational test cases, which will help the exploratory tester to explore SUT

efficiently. Formalize a mechanism of generation of these test cases. Include also test data suggestions to this process.

4. Implement a framework, which will guide the exploratory testers in the SUT and support them by the navigational test cases in accord with the defined model and principles of generation of these test cases. The framework will consist of three principal parts: (1) browser extension, which will analyse the SUT front-end pages, (2) back-end system, which will maintain the SUT model, generate the navigational test cases and present them to the testers in a separate guidance web application, (3) administration part, which will allow to browse, edit and visualize the SUT model, as well as manage the testing team using the framework.
5. Conduct case studies, which will verify the functionality of the implemented framework and compare its efficiency with Exploratory Testing performed manually. Test the applicability of the proposed system on at least three different system under tests. As one of these systems, select an SUT, which HTML front-end structure will be dynamically generated, which could cause the problems with analysing of the front-end pages. Assess applicability of the proposed framework also for this type of SUT.

In this Dissertation Thesis, we limit the scope of the systems under tests to web-based applications and information systems, providing an HTML-based user interface.

3. Background and State-of-the-Art

As the topic solved in this Dissertation Thesis is practically a combination of Exploratory Testing, Model Reverse Engineering and Model-Based Testing in its specific variant, in this section, we provide a brief overview of the state-of-the-art in these disciplines.

In the **Model-Based Testing** (Utting and Legeard, 2007; Lochau et al., 2017; Jorgensen, 2017; Dias et al., 2007; Sharma et al., 2014) the standard approach is the test case generation from the formal model of the system requirements and behaviour. The model is usually an abstract partial presentation of the desired behaviour of the system and is created manually. Test cases generated

from such a model are functional tests on the same level of abstraction as the model forming an abstract test suite. This suite cannot be executed directly against SUT because there are physical details of the test missing. An executable test suite is derived from an abstract test suite using a model transformation. In some model-based testing environments, the models do contain enough information to generate executable test suites directly. A mapping must exist between the elements of the abstract test suite and executable code in the software to create a test suite.

There are many general-purpose modelling languages available for describing the system high-level model used in the industry. The most widespread and used is Unified Modelling Language (UML). The UML diagrams (sequence diagrams (Bandyopadhyay and Ghosh, 2009; Kumar and Singh, 2015), state chart diagrams (Bubna and Chakrabarti, 2016), activity diagrams (Kansomkeat et al., 2010; Jena et al., 2014), or state machine diagrams (Yue et al., 2011) are transformed into more suitable application model, from which the test case scenarios are generated. Nevertheless, an accurate, consistent and detailed model is often very costly to create, and the maintenance and synchronisation of such model with the rapid development of the project are difficult.

The standard set of UML 2.0 diagrams does not directly focus on the user front-end interaction. Though UML notation was used to model the user interface it does not have the capabilities to model all the nuances of the UI. Although UI diagram was introduced with a user interface specialisation in (Ferri, 2008), there are more suitable modelling languages available to model UI of an application.

The first example of these web modelling languages is the Web Modelling Language (WebML) (Brambilla and Fraternali, 2014), which was created introducing visual notations and a methodology for designing complex data-intensive Web applications. Later on, this language then evolved into IFML (Brambilla et al., 2014) to cover a wider spectrum of the front-end interfaces and the data flow between the application front-end components. IFML was standardised in 2013. Its notation is easily extensible - new containers, components, events can be added, or custom UML stereotypes applied (Brambilla et al., 2014). Also, new components and events (swipe, camera event, location sensor event for mobile UI modelling) were added to define mobile-specific interfaces and interaction. IFML represents a prospective

modelling tool to describe application front-end and a flexible and easily extensible notation.

The Model-Based Testing represents very prospective concept to create efficient test cases in an automated way. However, its limitation is the presence of the consistent and up-to-date model of the system under test. Here, the question arises: Are we able to use this powerful concept, when the model is partial only, or when this model is generated by reverse engineering technique?

In the case when the SUT model is not available, a **Reverse Engineering Technique** (RET) is a suitable approach to recreate it from the actual state of the SUT. The model of the application can be recreated using RET by analysing the static content of the HTML pages - namely the HTML elements - and to build a directed graph with web pages as nodes and transitions/links as edges. Not every web applications have static pages only; some content is dynamic, and the content has to be treated differently in this case. Analysing of the content requires the execution of the code, and it can depend on the value of input variable(s). Prospectively, not only all the possible flows of screens and actions in the SUT but also flows representing business processes could be re-engineered. In this process, proper manual input to mark which sequence parts belong to the particular workflow is needed. Morgado and Paiva (2015) have created a tool called iMPAcT for mobile pattern testing. They are focusing on an automated testing of recurring behaviour, i.e. UI patterns. For a UI pattern to be matched, the current state of the application is analysed when an event is fired. The pattern preconditions must be verified, and all checks met. Each UI pattern has a test pattern associated with testing. A catalogue of UI patterns was created for the mobile application, which can be prospectively also used in reverse engineering of the SUT model.

The **Exploratory Testing** technique itself is a subject of current software testing research. As explained in the introduction, this technique gives software testers certain level of freedom to design and execute the tests while exploring the product (Hendrickson, 2014; Itkonen and Rautiainen, 2005; Itkonen, 2011; Pfahl et al. 2014). The tester uses the data gathered from the execution of the first set of tests to conduct the next round of tests. ET can find critical defects in a shorter time. Unlike documented test cases, exploratory testing does not follow any testing rules. Testers with strong knowledge of the business and technical domain explore the application. By browsing through and using the

application like a real user, testers are more likely to find issues that customers might face.

An industrial case study to evaluate the impact of education level and experience level on the effectiveness of ET was conducted by (Gebizli and Sözer, 2017). In this study, 19 practitioners, who have different education and expertise levels, were involved in applying ET for testing a Digital TV system. The results show that efficiency regarding the number detected failures per unit of time is significantly affected by both the educational background and experience. Experience also has a significant impact on the number of detected critical failures, whereas education has not. Though the formal education, formal training or test certification is not required and regular end-users regularly find and report bugs in systems even though they are not trained as testing professionals, the testing process encompasses a broad range of skills (planning, design, automation, exploratory testing). A certain level of formal proficiency in these areas is required (Micallef et al., 2016).

Also, a limited related work exists in **combinations of Model-based Testing and Exploratory Testing**. DoNascimento and Machado (2017) proposed using ET approach to acquire knowledge for a model-based testing. The main target area of Exploratory Testing is GUI testing where the tester tries to find defects and break the application by exploring the possible actions in the SUT using his intuition. Testers can start testing new feature immediately since the planning of the testing process is not necessary. In contrary to these advantages, certain disadvantages shall also be mentioned (Shah et al., 2014). It is difficult to assess whether a feature was tested, the process is not monitored and tracked, the quality of testing is unclear and depends on the experience and skills of the tester. The problem is also to re-evaluate the test later.

Kim et al. (Kim and Lee, 2014) focused on generating formal specifications from manually written test cases - a possible synergy between Exploratory Testing and model-based testing. The SUT model is re-engineered to get a better insight into the legacy code.

4. Proposed Solution

The Test Analysis SUT Process Information Reengineering (**Tapir**) Framework is to automate the following activities in the Exploratory Testing process:

1. automated recording of the tests performed by the testers, including used testing data,
2. automated guidance through the system under test (testers are being instructed which pages and functions of the system shall be explored), and,
3. support of teamwork of the exploratory testers, to prevent duplicities in performed tests and test data, for instance.

The framework records the data about all pages visited by the tester and the actions triggered by the tester (button clicked, link clicked, a form submitted) and the data used (applies for forms only). The recorded data is persisted in the database and used later for the tester navigation through the SUT. From the data, the SUT model is being built incrementally.

During the testing process, the **navigational test cases** are created dynamically from the SUT model. In the test cases, the tester is offered the best screen element to test in the next step, the suggested page to go on and the recommended test data to enter. We have developed several **navigational strategies** that select the best candidates for selection of these elements - the criteria being for example the complexity of the link target page (testing the more complex pages first), the type of the element (testing that submitting a form does not result in an error first), the priority of the element (testing the elements more important from the business point of view first), number of previous visits of an element (to achieve higher coverage of original pages and elements of the SUT).

The model of the SUT and the framework directly supports and encourages the **teamwork** in the exploratory testing process. Defined navigational strategies have also their team versions; in this variant, the navigational test cases guide the testers to the parts of SUT which have not been explored by anyone in the team previously. This also applies to the suggested testing data.

As the **SUT model**, we have adopted a model of the web application proposed by Deutsch et al. (2007). The model has been modified and has been extended to meet our requirements. The main definition of the SUT model is the following. The SUT is a tuple (W, w_0, w_e, S, I, A, L, M) where the

- W is a set of all web pages of the SUT containing homepage w_0 and error page w_e; web page contains input, action and link elements (subsets of set defined further in the text), reference to its master page and set of transition rules that indicate which page is loaded when a link or a button (action element) is clicked.
- S is a set of all state values.
- I is a set of input elements in the SUT (input fields on web pages), a range of input values allowed from the business point of view can be defined for input elements.
- A is a set of action elements in the SUT (buttons on web pages).
- L is a set of link elements in the SUT.
- M is a set of hierarchical master pages in the SUT (master page represents part of the HTML pages that is shared across the SUT: page header, application menu or similar fragments).

The formal definition of the SUT and Exploratory Testing problem contains more concepts which are presented in the Dissertation Thesis. The Tapir framework allows the team lead (or a user with appropriate access level) to apply meta-data to desired elements. One example of the meta-data was already mentioned - the allowed data range. Another type of the meta-data is the business type of the values that can be filled into an input field (telephone number, email address, …); or the priority of the page or of an element. The priority is reflected in the automated navigation of the tester in the SUT, which is provided by the framework.

The defined **SUT model is incrementally built** during the testing process. The system starts with an empty model of the SUT and then, during the initial stage, the model is dynamically created and authored by the test lead. The Tapir browser extension analyses every page of the SUT the user visits and retrieves information about the page and every input, action and link element on that page. Extracted data is sent to the server, analysed and used to create (or update) the persisted model of the SUT. The test lead can manually modify the model by applying meta-data to selected elements.

The Tapir Framework composes of three main modules, which cooperate closely (the overall architecture is depicted in Figure 1).

Tapir Browser Extension records activities of the exploratory testers. Individual captured events are sent to the TapirHQ server. In the browser, the extension also highlights the elements of the SUT user interface. Elements suggested to explore are highlighted to the tester; elements, which have been analysed before and are part of the SUT model can be highlighted. Currently, The extension is available for the Chrome browser. In the project roadmap, development of a Firefox browser extension is planned for 2018. **TapirHQ** module gets the captured events from the Browser Extension and constructs the SUT model based on this information. From the model, this component prepares the navigational test cases and presents them to the testers. TapirHQ interacts with the testers via special test management application (TapirHQ Front-end). In this application, navigational test cases and information about explored SUT parts are displayed, test lead or administrator can edit the SUT model or priorities can be set to individual model parts. **Tapir Analytics** module visualises the recorded SUT model. Using this visualisation, the testing team gets an overview, which parts of the SUT have been explored so far during the exploratory testing process.

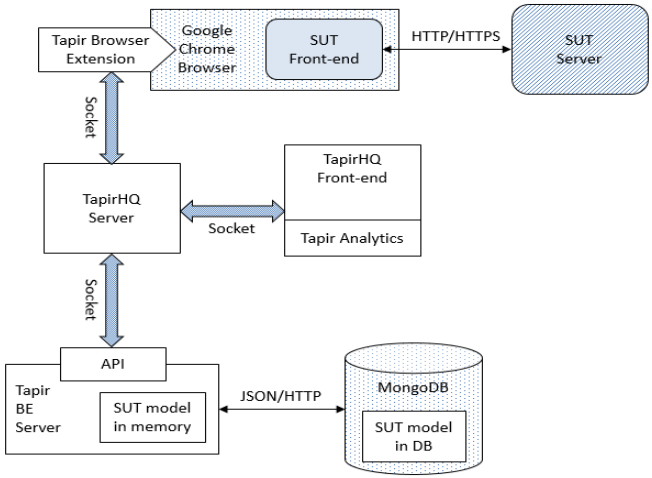

Figure 1: The high-level architecture of the Tapir framework

5. Experiments and Summary of the Results

To analyse the practical applicability of the proposed Tapir Framework and its efficiency when used to support Exploratory Testing process, we conducted four case studies, which we describe in the Thesis.

The aim of Case Studies 1 and 2 was to compare the efficiency of the Tapir Framework with Exploratory Testing process performed manually. In these case studies, we split the testers to two principal groups. The first group was using the support of the Tapir Framework. The second group was only monitored using the Browser Tracking extension. During this process, the SUT model was constructed, but no navigational support was provided to the testers of this group. In both case studies, we compared data collected from constructed models. In the Case Study 1, we also collected testers reports on time spent on individual tasks in the testing process. In the Case Study 2, we relied more on time measurement connected to model construction. The Case Study 2 was performed using a more recent prototype of the Tapir Framework.

The aim of Case Study 3 was to assess proposed navigational strategies and suggest the best option to be evolved further on. We compared team and individual navigational strategies proposed in the Dissertation Thesis, as well as alternative ranking functions, which are used by these navigational strategies.

The aim of Case Study 4 was to assess the applicability of the Tapir Framework to different types of SUTs and to assess its applicability when used on an SUT with dynamically generated front-end HTML pages, including the structure of the pages and URL format.

The first two Case Studies were performed to compare the efficiency of exploratory testing supported by the proposed framework with exploratory testing performed manually without such support. The results of these Case Studies are promising: In the Case Study 1, significant effort was saved for the part of subtasks of the ET process when performed with the aid of an automated support. The savings were achieved mostly in the subtasks related to the documentation of the test case (steps which have to be taken) and overall documentation of the explored parts of the SUT. Overall time savings in the case study were 23.5% when the proposed solution was used. In the subtasks

which were not directly supported by the used version of Tapir Framework (e.g. specification of the test expected result, or defect report), no significant improvement was achieved. Further on, there was a slight increase in an average number of detected defects for the aided exploratory testing in comparison to the manually performed ET process by 6%.

In the Case Study 2, the results also show that the exploratory testers supported by the Tapir framework have explored the larger original extent of the SUT pages and functions than the testers performing the exploratory testing process without such support. For instance, an average number of SUT pages explored per time unit was better by 7%, when the testers were guided by the Tapir Framework. When we consider unique explored pages only, this improvement was even more significant, 39%. With the support of the Tapir framework, the testers could activate more artificial defects than in the Case Study 1. The improvement was 31% in this case.

The goal of the Case Study 3 was to select the best navigational strategy and related ranking function to construct the navigational test cases. Several conclusions have been made from these experiments. Guidance of the exploratory testers based on the analysis of the complexity of the target pages provided the best results from explored possible strategies. Also, the navigational strategy based on the teamwork principle was more efficient regarding the time spent on the testing activities. This strategy was also performing better when we analysed the extent of original SUT pages and elements, which have been explored during the testing.

Finally, the Case Study 4 explored applicability of the Tapir Framework to different styles of SUT front-end coding. In the case of OFBiz and Moodle systems, customization of Browser Extension was entirely feasible, and the Tapir Framework was working without technical limitations. Our concern was an application with dynamically generated front-end pages, where only a little stable elements and element IDs are present. As an example of this application, we selected JTrac. As the customisation of browser tracking extension has shown, dynamically generated URL of the front-end pages represented a problem to reconstruct the SUT model; framework was running partially, but the model reconstruction was inaccurate. This represents an application limit of the proposed framework. It is worth to mention that for an automated testing based on the front-end such dynamic generation of HTML content also represents the significant obstacle.

6. Conclusion

The main application area of the Exploratory Testing technique is the software development projects, in which the design documentation (which serves as a basis to create the test cases) is missing, obsolete or inconsistent. Several issues can influence the efficiency of this technique. When testers' actions in the SUT are not systematically documented during the testing process, it usually leads to repetitive test cases, including repetitive data combinations.

In such situations, it is difficult to assess, whether particular SUT feature has been tested already. This makes decisions what to test in the next steps difficult, especially, for more junior members of the testing team. Also, this situation usually requires a strong managerial presence of a Test Lead, who must organize the testing efficiently. Furthermore, problems when reporting defects can be experienced, as testers often do not remember the explored path exactly. This impacts the quality of defect reports, which adds additional overhead to the development and testing part of the project (more information is required by the development team to reproduce these defects). Besides the extra costs, this effect also prolongs the defect fixing by the developers. Consequently, the quality of testing strongly depends on the experience and skills of the testers.

With the aim to minimise these problems, we designed and experimentally implemented a framework for automated support of the Exploratory Testing process. This framework is designed for the systems under tests to web-based applications and information systems, providing an HTML-based user interface. Browser tracking extension records actions in the SUT front-end performed by the testers and based on this information, the SUT model is created and dynamically updated during testers' exploration of the SUT. During this exploration, navigational test cases are automatically created and presented to the testers by guideline application running side-by-side with the SUT.

The SUT model includes pages of the SUT front-end, input, link and action elements of these pages, as well as structures for management of test data used in the exploratory testing process. History of SUT exploration by individual testers is also recorded in the model.

To evaluate the efficiency and applicability of the proposed solution, we conducted four Case Studies, which we presented in this Dissertation Thesis. These Case Studies documented effort savings in part of subtasks of the ET process with the support of the Tapir framework. This automated support also ensured exploration of more considerable extent of new SUT parts during the tests and capacity of the proposed framework to support detection of more defects in the system under test.

The contributions of this Dissertation Thesis can be summarised as the following:

- Design of the framework, which contributes to conduction of Exploratory Testing process in the more efficient way regarding spent resources, the extent of explored SUT and found defects.
- Combination of Reverse-Engineering, Model-Based Testing and Exploratory Testing, which we consider innovative (during extensive literature study on the related topic, we have not identified a research or software industrial project, which addressed the problem in the way similar to our approach).
- A formal model of the underlying system under test, which can be (apart from being used as a basis for the Tapir Framework processes) further used for modelling of Exploratory Testing process or measurement of the efficiency of software testing process in general.
- Design of the initial navigational strategies, ranking functions, and test data strategies, which can be further explored and more efficient variants can be found. This includes the focus on the teamwork and efficient usage of the individual tester's resources.
- The practical applicability of the proposed framework to industrial software development and testing projects.
- Possible alternative usages of the Tapir Framework, as a measurement tool for the testing process efficiency or assessment of the efficiency of a particular set of test cases.

References

Bandyopadhyay, A. and Ghosh, S. Test input generation using UML sequence and state machines models. In Software Testing Verification and Validation, 2009. ICST '09. International Conference on, pages 121-130, April 2009.

Brambilla, M. and Fraternali, P. Large-scale model-driven engineering of web user interaction: The WebML and WebRatio experience. Science of Computer Programming, 89:71-87, 2014.

Brambilla, M. and Fraternali, P., et al. The interaction flow modeling language (IFML). Technical report, version 1.0. Technical report, Object Management Group (OMG), http://www.ifml.org, 2014.

Brambilla, M., Mauri, A. and Umuhoza, E. Extending the interaction flow modeling language (IFML) for model driven development of mobile applications front end. In Mobile Web Information Systems, volume 8640 of Lecture Notes in Computer Science, pages 176-191. Springer International Publishing, 2014.

Bubna, K. and Chakrabarti, S. K. Act (abstract to concrete tests)-a tool for generating concrete test cases from formal specification of web applications. In ModSym+ SAAAS@ ISEC, pages 16-22, 2016.

Deutsch, A., Sui, L. and Vianu, V. Specification and verification of data-driven web applications. Journal of Computer and System Sciences, 73(3):442-474, 2007.

Dias Neto, A.C., Subramanyan, R., Vieira, M., and Travassos, G. H. A survey on model-based testing approaches: a systematic review. In Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007, pages 31-36. ACM, 2007.

do Nascimento, L. H. and Machado, P. D. An experimental evaluation of approaches to feature testing in the mobile phone applications domain. In Workshop on Domain specific approaches to software test automation: in conjunction with the 6th ESEC/FSE joint meeting, pages 27-33. ACM, 2007.

Eldh, S., Hansson, H., Punnekkat, S., Pettersson, A. and Sundmark, D. A framework for comparing efficiency, effectiveness and applicability of software testing techniques. In Testing: Academic and Industrial Conference - Practice and Research Techniques, 2006. TAIC PART 2006. Proceedings, pages 159-170, Aug 2006.

Ferri, F. Visual Languages for Interactive Computing: Definitions and Formalizations. Premier reference source. Information Science Reference, 2008.

Gebizli, C. S. and Sözer, H. Impact of education and experience level on the effectiveness of exploratory testing: An industrial case study. In 2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pages 23-28, March 2017.

Hendrickson, E. Explore it!: reduce risk and increase confidence with exploratory testing. The Pragmatic Programmers, 2014.

Itkonen, I. and Rautiainen, K. Exploratory testing: a multiple case study. In Empirical Software Engineering, 2005. 2005 International Symposium on, pages 10-pp. IEEE, 2005.

Itkonen, J. et al. Empirical studies on exploratory software testing. 2011.

Jena, A. K., Swain, S. K., and Mohapatra, D. P. A novel approach for test case generation from uml activity diagram. In 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), pages 621-629, Feb 2014.

Jorgensen, P. C. The Craft of Model-Based Testing. CRC Press, 2017.

Kaner, C., Bach, J. and Pettichord, B. Lessons learned in software testing. John Wiley & Sons, 2008.

Kansomkeat, S., Thiket, P., and Offutt, J. Generating test cases from uml activity diagrams using the condition-classification tree method. In Software Technology and Engineering (ICSTE), 2010 2nd International Conference on, volume 1, pages 62-66, Oct 2010.

Kim, D.K., and Lee, L.S. Reverse engineering from exploratory testing to specification-based testing. International Journal of Software Engineering and Its Applications, 8(11):197-208, 2014.

Kumar, B. and Singh, K. Testing UML designs using class, sequence and activity diagrams. International Journal for Innovative Research in Science and Technology, 2(3):71-81, 2015.

Lochau, M., Peldszus, S., Kowal, M., and Schaefer, I. Model-based testing. In Advanced Lectures of the 14th International School on Formal Methods for Executable Software Models - Volume 8483, pages 310-342, New York, NY, USA, 2014. Springer-Verlag New York, Inc. ISBN 978-3-319-07316-3.

Micallef, M., Porter, C. and Borg, A. Do exploratory testers need formal training? An investigation using HCI techniques. In 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pages 305-314, April 2016.

Morgado, I. C., and Paiva, A. C. R. Testing approach for mobile applications through reverse engineering of UI patterns. In 2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW), pages 42-49, Nov 2015.

Pfahl, D., Yin, H., Mäntylä, M. V., and Münch, J. How is exploratory testing used? a state-of-the-practice survey. In Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, page 5. ACM, 2014.

Shah, S. M. A., Gencel, C., Alvi, U. S., and Petersen, K. Towards a hybrid testing process unifying exploratory testing and scripted testing. Journal of Software: Evolution and Process, 26(2):220-250, 2014.

Sharma, H. K., Singh, S. K. and Ahlawat, P. Model-Based Testing: The New Revolution in Software Testing. Database Systems Journal, 5(1):26-31, May 2014.

Utting, M. and Legeard, B. Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007. ISBN 0123725011, 9780080466484.

Yue, T., Ali, S., and Briand, L. Automated transition from use cases to UML state machines to support state-based testing. In Modelling Foundations and Applications, pages 115-131. Springer, 2011.

List of Author's Publications

Articles in impacted journals

Frajták, K., M. Bureš, and I. Jelínek. Exploratory testing supported by automated reengineering of model of the system under test. Cluster Computing, 20(1):855{865, 2017. ISSN 1386-7857. [33%]

Bureš, M., T. Černý, K. Frajták, and B. Ahmed. Testing the consistency of business data objects using extended static testing of crud matrices. Cluster Computing, Aug 2017. ISSN 1573-7543. [25%]

Conference papers indexed in ISI Web of Science

Frajták, K., M. Bureš, and I. Jelínek. Model-based testing and exploratory testing: Is synergy possible? In Proceedings of the 6th International Conference on IT Convergence and Security (ICITCS 2016), pages 329-334, Red Hook, US, 2016. ISBN 978-1-5090-3765-0. [33%]

Frajták, K., M. Bureš, and I. Jelínek. Formal specification to support advanced model based testing. In Federated Conference on Computer Science and Information Systems (FedCSIS 2012), pages 1311-1314, New York, US, 2012. ISBN 978-1-4673-0708-6. [33%]

Conference papers indexed in Elsevier Scopus

Frajták, K., M. Bureš, and I. Jelínek. Manual testing of web software systems supported by direct guidance of the tester based on design model. World Academy of Science, engineering and Technology, 80(0):243-246, August 2011. ISSN 2010-376X. [33%]

Frajták, K., M. Bureš, and I. Jelínek. Pex extension for generating user input validation code for web applications. In Proceedings of the 9th International

Conference on Software Engineering and Applications, pages 315-320, Setúbal, PT, 2014. ISBN 978-989-758-036-9. [33%]

Frajták, K., M. Bureš, and I. Jelínek. Reducing user input validation code in web applications using Pex extension. In ACM International Conference Proceeding Series, Volume 883, ACM International Conference Proceeding Series, pages 302-308, Rousse, BG, 2014. ISBN 978-1-4503-2753-4. [33%]

Frajták, K., M. Bureš, and I. Jelínek. Using the interaction flow modelling language for generation of automated frontend tests. In Position Papers of the 2015 Federated Conference on Computer Science and Information Systems, Annals of Computer Science and Information Systems, pages 117-122, Warsaw, PL, 2015. ISBN 978-83-60810-77-4. [33%]

Frajták, K., M. Bureš and I. Jelínek. Transformation of IFML schemas to automated tests. In Proceeding of the 2015 Research in Adaptive and Convergent Systems (RACS 2015), pages 509{511, New York, US, 2015. ISBN 978-1-4503-3738-0. [33%]

Other publications

Frajták, K., M. Bureš, and I. Jelínek. Web software systems testing supported by model-based direct guidance of the tester. Proceedings of International Conference on Information Technologies, 2012(26):45-52, 2012. ISSN 1314-1023. [33%]

Anotace

Metoda průzkumného testovaní (angl. Exploratory Testing) je metoda testování softwarových projektů vhodná v situacích, ve kterých není k dispozici žádná návrhová dokumentace nebo je neúplná či nekonzistentní a není možné ji využít pro tvorbu testovacích scénářů. Principem je souběžné prozkoumávání testovaného systému, tvorba testovacích scénářů a testování tohoto systému. Pro zajištění efektivity této techniky je klíčová dokumentace prozkoumaných částí systému i vykonaných testovacích scénářů.

Rekonstrukci modelu testovaného systému, společně se sledováním aktivit testera v testovaném systému a připojenými metadaty, lze využít pro automatickou navigaci testera. Tím se dají snížit náklady nutné na pořizování dokumentace v průběhu testování, čímž se zvyšuje transparentnost a efektivita průzkumného testování. V této práci představujeme model a framework, který tuto podporu poskytuje.

Rozšiřující modul webového prohlížeče automaticky nahrává akce prováděné testerem v systému. Tato data slouží k průběžnému vytváření modelu systému a k jeho aktualizaci. Framework dynamicky vytváří navigační testovací scénáře na základě dostupného modelu a aktuální pozice testera v systému a tím zajišťuje vyšší efektivitu průzkumného testování.

V průběhu dynamického generování navigačních testovacích scénářů může být použito několik navigačních strategií, které jsou popsány v textu práce. Tyto strategie využívají řadu vstupů, jako je informace o předchozích návštěvách dané stránky konkrétním testerem nebo jeho kolegy z daného týmu, priorita stránek nebo komplexnost stránek z hlediska počtu a druhu sledovaných prvků.

Provedené experimenty ověřily, že metoda průzkumného testovaní prováděného s automatickou podporou navrženého frameworku, ve srovnání s průzkumným testováním prováděným pouze manuálně, zvyšuje efektivitu této techniky v několika oblastech. Navržená automatická podpora snižuje čas potřebný na jednotlivé úkoly a vede testery k otestování větší části systému. Pokusy se zanesenými umělými defekty v testovaném systému ukázaly, že navržený framework pomohl testerům odhalit větší množství těchto defektů oproti skupině testerů pracujících bez jeho podpory.