Software Testing MTAT.03.159
Lab 6 - Project

Dietmar Pfahl
University of Tartu, Institute of Computer Science

April 14, 2013

1 Introduction

This document gives the practical details regarding the project in the course Software Testing. The project in the course is equivalent to 20% of the total course grade. The main objective of the project is to gain a deeper understanding of a specific area within verification and validation (V&V) of software products and also to improve your skills of reading, summarising and presenting scientific literature. A project group consists of 3 persons (+/-1 person, in case the total number of students in a lab group modulo 3 is not equal to 0. The students in a project group perform the project together and will get the same marks. All project members should be involved and the total effort should be evenly distributed among participants.

2 Learning Objectives

The objective of the project is to learn a specific area of software testing and to get a taste of reading scientific literature and scientific writing. Hence, the main learning points of the project are:

- Learn a specific area of software testing
- Collect and summarize research information
- Critical thinking beyond the written information
- Summarise and present information in a structured way

3 Activities

The main activities in the project are:

1. **Decide on a subject.** There are some subjects specified in this document. In each lab group, each topic can only be picked by at most two project groups - first come, first serve. You may also define your own topic (but topic must be confirmed by course instructor).
2. **Find literature.** In each subject area, there is a suggestion of research papers to start from. Additional scientific literature can be found, for example, via ACM Digital Library (http://dl.acm.org/) and IEEE Xplore Digital Library (http://ieeexplore.ieee.org/). If you are logged in with your computer on-campus, you don't need a special account for accessing the digital libraries; you will automatically have access to the PDFs of full papers via the Tartu University Library. You should read and analyse at least as many papers as you are members in your project group (typically: 3). You may pick from the papers listed below, or you can read and analyse additional papers relevant to your specific topic.

3. **Read literature** Read and understand the literature about the chosen topic of software testing; summarise and discuss the essence of each paper.

4. **Outline the report.** Specify section headings and write in short sentences what you plan to include in each section. Follow the structure proposed under Section 6 (Report) below.

5. **Write the report.** The report is to be written in English. The report shall be written in the IEEE template. Follow the structure proposed under Section 6 (Report) below.

6. **Get help from peers and course instructor.** Try to solve any arising difficulties with your task within the group but if you need help during reading and writing, use the message board of the course to get help from peers and the course instrutor.

7. **Submit the report.** Submit the report before Friday, 19:00, of week 36 (i.e., the week before the Lab 6 sessions). Use the submit button on the course wiki. Make sure to have the names of all group members included in your report.

8. **Present the report.** Communicate your topic to your peers. This is done during the presentation session in Lab 6. You have about 8 minutes to describe your work. The presentation must be given in English.

9. **Submit the presentation.** Submit the presentation slides at most 1 hour after the lab session in which you gave your presentation. Use the submit button on the course wiki. Make sure to have the names of all group members included on the title page of your presentation.

4 Schedule

Table 1: Project schedule

Course week	**Activities**	**Hand in to instructor**	**Deadline**
32	Decide on a subject Search literature	Decided subject	Beforet lecture 2 (week 33)
33/34	Search literature Read literature Outline the report		
35/36	Read literature Write report		
37	Finish report Prepare presentation	Completed report	Friday 19:00 (week 37)
38	Presentation session	Presentation slides	1 hour after Lab 6 session

5 Assessment

The work is evaluated according the following characteristics:

- Report/Form (15%) – style, language, references
- Report/Content (60%) – the quality of the conducted work
- Presentation (25%) – the ability to communicate the work (form and content)

6 Report

The report shall be written using the IEEE template[1] and have 4 to 5 pages. The report shall contain:

Abstract: summary of all parts in the report. The purpose of the abstract is to attract people to read your report.

Introduction: introduction into the chosen topic. The purpose is to give an introduction to a reader who is not familiar to the specific area, but knows software engineering and testing generally. This section shall explain the emergence of the topic and motivate why this test topic is of interest/relevance.

Description of the chosen topic area: summary description of the chosen topic based on what you read in the selected literature. The purpose is to describe what the chosen topic actually is and how it is useful.

Analysis: analysis and discussion based on what you read (plus your own critical thinking). The aim is to describe the advantages, disadvantages, challanges, possible future research directions, etc.

Conclusion: sumary of the main findings of your literature study and present conclusions based on your analysis and discussion.

7 Presentation

The project will be presented in the project presentation sessionsin Lab 6. Each group will be given 9 minutes to present their chosen topic. The presentation should cover the main points of the report in an engaging and interesting way.

8 Project Topics

Below are some proposed project areas and an overview article related to each. If you would like to choose another area, describe the area to your project supervisor before starting the work. The keywords in Section 9 can be used when you choose a topic search for literature.

8.1 Mutation testing as way to check the effectiveness of test cases

Test cases are developed to check whether a software system is implemented correctly according to the requirements specification. However, if the test cases are badly chosen, they will not trigger failures. There is a method, called Mutation testing, which can be used to check whether the test cases are effective.

Mutation Testing: A. M. R. Vincenzi, J. C. Maldonado, E. F. Barbosa and M. E. Delamaro, Unit and Integration Testing Strategies for C Programs using Mutation, *Software Testing, Verification and Reliability*, 11(3):249-268, 2001.

Improving Mutation Testing: David Schuler, Valentin Dallmeier, and Andreas Zeller. 2009. Efficient mutation testing by checking invariant violations. In Proceedings of the eighteenth international symposium on Software testing and analysis (ISSTA '09). ACM, New York, NY, USA, 69-80. DOI=10.11451572272.1572282 http:doi.acm.org10.11451572272.1572282

Semantic Mutation Testing: John A. Clark, Haitao Dan, Robert M. Hierons, Semantic mutation testing, Science of Computer Programming, Volume 78, Issue 4, 1 April 2013, Pages 345-363, ISSN 0167-6423, 10.1016/j.scico.2011.03.011. (http://www.sciencedirect.com/science/article/pii/S0167642311000992)

Higher Order Mutation Testing: Yue Jia, Mark Harman, Higher Order Mutation Testing, Information and Software Technology, Volume 51, Issue 10, October 2009, Pages 1379-1393, ISSN 0950-5849, 10.1016/j.infsof.2009.04.016. (http://www.sciencedirect.com/science/article/pii/S0950584909000688)

[1] `http://www.ieee.org/conferences_events/conferences/publishing/templates.html`

8.2 Predicting software reliability

Software testing is performed to detect the failures in the software. There are different metrics to use in order to measure whether the software is correct. One such measure is reliability, which is defined as: "The probability for a failure-free operation of a program for a specified time under a specified set of operating conditions." (IEEE 610.12-1990). There are several models and techniques that can be used, for example, reliability growth models, Markov models, statistical usage testing, usage-based testing and operational profile testing.

Reliability growth models: C. Stringfellow, A. A. Andrews, An Empirical Method for Selecting Software Reliability Growth Models, *Empirical Software Engineering*, 7(4): 319- 343, 2002.

Non-parametric reliability growth modeling: Z. Wang, J. Wang and X. Liang, "Non-parametric estimation for NHPP software reliability models," Journal of Applied Statistics, 34 (1), pp. 107-119 (2007).

Reliability-growth modeling with wavelets: X. Xiao and T. Dohi, "Wavelet-based approach for estimating software reliability," Proceedings of 20th International Symposium on Software Reliability Engineering (ISSRE'09), pp. 11-20, IEEE CS Press (2009).

8.3 Verification and validation of requirements and design documents?

Static verification is often used in the beginning of the development of software. There are several different techniques and methods used to check the static representation. The common feature of these is that they have to be manually checked by reviewers. Software inspection, reviews, walkthroughs are common techniques, which are used together with reading techniques.

Software inspection and reading techniques: A. Aybuke, H. Petersson, C. Wohlin, State-of-the-art: Software Inspections after 25 Years, *Software Testing, Verification and Reliability*,12(3):133-154, 2002.

Effectiveness and efficiency of inspections: Kai Petersen, Kari Rönkkö, and Claes Wohlin. 2008. The impact of time controlled reading on software inspection effectiveness and efficiency: a controlled experiment. In Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM '08). ACM, New York, NY, USA, 139-148. DOI=10.1145/1414004.1414029 http://doi.acm.org/10.1145/1414004.1414029

8.4 Methods that help decide when to stop testing

Stopping criteria are used in the testing phase to determine when a certain quality level has been achieved. The quality can for example be defined as the reliability or just the number of faults left in the system. There are several techniques to use as a stopping criterion. One could, for example, estimate the number of faults left after an inspection or to estimate the number failures left in testing.

Fault content estimation: L. C. Briand, K. E. Emam, B. G. Freimut, O. Laitenberger, A Comprehensive Evaluation of Capture-recapture Models for Estimating Software Defect Content, *IEEE Transactions on Software Engineering*, 26(6):518-540, 2000.

Stopping criterion in testing: J. D. Musa and A. F. Ackerman, Quantifying Software Validation: When to Stop Testing?, *IEEE Software*, 6(3):19-27, 1989.

8.5 Techniques for deriving and generating test cases

Software test cases are often derived from documents produced during the software development. There are different techniques and languages that can be used. Use cases can be used to derived test cases and two methods that use this approach are usage-based testing and operation profile testing. Furthermore, there are techniques that are automated and use UML diagrams.

Usage-based testing: B. Regnell, P. Runeson and C. Wohlin, Towards Integration of Use Case Modelling and Usage-Based Testing,*Journal of Systems and Software*, 50(2):117-130, 2000.

Operational profile testing: J. Musa, Operational Profiles in Software-Reliability Engineering, *IEEE Software*, 10:(2):14-32, 1993.

Specification-based testing: J. Offutt, S. Liu, A. Abdurazik and P. Ammann, Generating Test Data from State-based Specifications, *Software Testing, Verification and Reliability*, 13(1):25?53, 2003.

8.6 Evaluation of testing tools

Tools are important in order to implement software testing effectively in an software organization. However, although tools are needed, they do not solve the problems in the testing phase. Several tools exist and they have to be evaluated before a software organization purchases one.

Evaluation of testing tools: Poston, R. M. and Sexton, M. P., Evaluating and Selecting Testing Tools, *IEEE Software*, pp. 33-42, 1992.

8.7 Techniques that help evaluate different testing techniques (methods)

One research methodology is to use empirical methods in order to evaluate which testing methods are best to use. The methodologies are often divided into experiments, surveys and cases studies. All these methods are important in order to help software organizations to choose the right testing technique for their purpose.

Experiment: Sun Sup So, Sung Deok Cha, Timothy J. Shimeall and Yong Rae Kwon, An Empirical Evaluation of Six Methods to Detect Faults in Software, *Software Testing, Verification and Reliability*, 12(3):155?171, 2002.

Survey: P. Runeson, C. Andersson and M. Höst, Test Processes in Software Product Evolution – A Qualitative Survey on the State of Practice, *Journal of Software Maintenance and Evolution*, 15(1):41-59, 2003.

Case study: T. Berling and T. Thelin, An Industrial Case Study of the Verification and Validation Activities, *International Software Metrics Symposium*, pp. 226-238, 2003.

8.8 Testing in agile projects

Test-first is a principle that is often used in Agile processes. The principle is based on that test cases are first derived and then these are used to specify how the system should work as well as used as test cases when the system is developed. There are some questions, which are of interest within this area, for example, what is the relation between requirements and test cases, what is the final quality of such a system and how easy is it to maintain such a system? Similar questions can be asked, e.g., for exploratory testing, acceptance testing, etc.

Test-first principle: M. M. Muller, and O. Hagner, Experiment about Test-first Programming, *IEE Proceedings Software*, 149(5):131-136, 2002.

Effectiveness of exploratory testing: Juha Itkonen, Mika V. Mantyla, and Casper Lassenius. 2007. Defect Detection Efficiency: Test Case Based vs. Exploratory Testing. In Proceedings of the First International Symposium on Empirical Software Engineering and Measurement (ESEM '07). IEEE Computer Society, Washington, DC, USA, 61-70. DOI=10.1109/ESEM.2007.38 http://dx.doi.org/10.1109/ESEM.2007.38

Rule-based exploratory testing: Theodore D. Hellmann and Frank Maurer. 2011. Rule-Based Exploratory Testing of Graphical User Interfaces. In Proceedings of the 2011 Agile Conference (AGILE '11). IEEE Computer Society, Washington, DC, USA, 107-116. DOI=10.1109/AGILE.2011.23 http://dx.doi.org/10.1109/AGILE.2

8.9 Techniques for testing object-oriented software systems

When implementing a system in an object-oriented fashion, i.e. not only using an object-oriented language, special conditions are present for e.g. integration.

Object-oriented testing: Y. Labiche, P. Thvenod-Fosse, H. Waeselynck, M.-H. Durand, Testing Levels for Object-Oriented Software, *Proceedings of the 22nd International Conference on Software Engineering*, pp. 136-145, 2000.

Object-oriented testing (short paper): Mauro Pezz and Michal Young. 2004. Testing Object Oriented Software. In Proceedings of the 26th International Conference on Software Engineering (ICSE '04). IEEE Computer Society, Washington, DC, USA, 739-740.

Object-oriented testing without oracles: T. H. Tse, Francis C. M. Lau, W. K. Chan, Peter C. K. Liu, and Colin K. F. Luk. 2007. Testing object-oriented industrial software without precise oracles or results. Commun. ACM 50, 8 (August 2007), 78-85. DOI=10.1145/1278201.1278210 http://doi.acm.org/10.1145/1278201.1278210

8.10 Techniques for testing distributed software systems

Many software systems are distributed over different computers. For example, internet applications most often are distributed over alt least two: the server and the client.

Testing of distributed systems: S. Goeschl and H. M. Sneed, Case study of testing a distributed internet-system, *Software Testing, Verification and Reliability*, 12(2):77 - 92, 2002.

Stress-testing of distributed systems: Vahid Garousi. 2011. Fault-driven stress testing of distributed real-time software based on UML models. Softw. Test. Verif. Reliab. 21, 2 (June 2011), 101-124. DOI=10.1002/stvr.418 http://dx.doi.org/10.1002/stvr.418

8.11 Techniques for web-application testing

The quality of web applications mostly relies on the skill of the individual developer. In order to meet the quality decided by end-users, the processes of developing and testing web applications need to be formalized.

Web application testing: Di Lucca G. and Fasolino, A., Testing Web-based Applications: The State of the Art and Future Trends, *Information and Software Technology*, 48(12):1172- 1186, 2006.

Statistical testing of web applications: Jianhua Hao and Emilia Mendes. 2006. Usage-based statistical testing of web applications. In Proceedings of the 6th international conference on Web engineering (ICWE '06). ACM, New York, NY, USA, 17-24. DOI=10.1145/1145581.1145585 http://doi.acm.org/10.1145/1145581.1145585

Data-flow testing for web applications: Yu Qi, David Kung, and Eric Wong. 2006. An agent-based data-flow testing approach for Web applications. Inf. Softw. Technol. 48, 12 (December 2006), 1159-1171. DOI=10.1016/j.infsof.2006.06.005 http://dx.doi.org/10.1016/j.infsof.2006.06.005

8.12 Tools and techniques for static code analysis

Static code analysis tries to help developers localise areas in their code that is probably incorrect. The challenge is to avoid 'false positives', i.e., trying to avoid alerting the developer of potentially incorrect code sequences which actually are correct.

Static code analysis: P. Louridas, "Static Code Analysis," IEEE Software, vol. 23, no. 3, July 2006, pp. 5861.

Static code analysis for embedded software: Ben Chelf and Christof Ebert. 2009. Ensuring the Integrity of Embedded Software with Static Code Analysis. IEEE Softw. 26, 3 (May 2009), 96-99. DOI=10.1109/MS.2009.65 http://dx.doi.org/10.1109/MS.2009.65

Defect density prediction based on static code analysis: Artem Marchenko and Pekka Abrahamsson. 2007. Predicting software defect density: a case study on automated static code analysis. In Proceedings of the 8th international conference on Agile processes in software engineering and extreme programming (XP'07), Giulio Concas, Ernesto Damiani, Marco Scotto, and Giancarlo Succi (Eds.). Springer-Verlag, Berlin, Heidelberg, 137-140.

8.13 Techniques for effective and efficient regression testing

Regression testing is the process of validating modified software to detect whether new errors have been introduced into previously tested code. Because of time and resource constraints for testing, regression test selection techniques have been proposed, to reduce the expenses.

Regression test selection: E. Engström, P. Runeson and M. Skoglund, A Systematic Review on Regression Test Selection Techniques, *International Software Technology*, 52(1):14-30, 2010.

Regression testing practice: E. Engström and P. Runeson, A Qualitative Survey of Regression Testing Practices, Proceedings 11th International Conference on Product-Focused Software Process Improvement, (PROFES), pp. 3-16, 2010.

8.14 Techniques for automatically generating test cases from the code

A class of testing methods, called search-based testing, generate test criteria from the code. The approach is promising, but there are of course limitations to it.

Search-based testing: M. Harman and P. McMinn. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. *IEEE Transactions on Software Engineering*, 36(2):226-247, 2010.

Static code analysis: Kostyantyn Vorobyov and Padmanabhan Krishnan. 2012. Combining Static Analysis and Constraint Solving for Automatic Test Case Generation. In Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST '12). IEEE Computer Society, Washington, DC, USA, 915-920. DOI=10.1109/ICST.2012.196 http://dx.doi.org/10.1109/ICST.2012.196

8.15 Techniques for model-based testing

Model-based testing is a popular term for generating test cases from various models, for example UML specifications. The advantage is that new test cases can be generated at low costs, once the model is defined. Maintenance of the test model may also be lower than maintaining individual test cases for evolutionary development.

A. Pretschner, W. Prenninger, S. Wagner, C. Kühnel, M. Baumhartner, B. Sostawa, R. Zölch, T. Stauner, One Evaluation of Model-Based Testing and its Automation, *Proceedings. 27th International Conference on Software Engineering (ICSE)* pp. 392-401, 2005.

Q.Malik, A.Jaaskelainen, H.Virtanen, M.Katara, F.Abbors, D.Truscan, and J.Lilius. Model-based testing using system vs. test models - what is the difference? *17th IEEE International Conference and Workshops on Engineering of Computer Based Systems*, pp. 291-299, 2010.

8.16 Techniques for combinatorial testing

The combinatorial explosion is enormous in software systems. Not only input combinations, but different hardware variants and software configurations, create combinations which may have to be tested. There are some theoretical models, and other heuristic models which guide the selection of combinations.

Survey of combinatorial testing: Changhai Nie and Hareton Leung. 2011. A survey of combinatorial testing. ACM Comput. Surv. 43, 2, Article 11 (February 2011), 29 pages. DOI=10.1145/1883612.1883618 http://doi.acm.org/10.1145/1883612.1883618

Automated combinatorial testing: R. Kuhn, R. Kacker and Y. Lei, Automated Combinatorial Test Methods – Beyond Pairwise Testing, *Crosstalk, Journal of Defense Software Engineering*, vol. 21, no. 6, June 2008.

8.17 Defect handling in large-scale software engineering

Defect reports are important means for communication within an organization, for example between between testers and developers. However, in large-scale software engineering, the number of defect reports quickly grows large, and hence become costly. One problem is duplicate reporting of known problems. Another is the quality of teh defect reports as such.

Defect reporting quality: T.Zimmermann, R.Premraj, N.Bettenburg, S.Just, A.Schroter, and C.Weiss. What makes a good bug report? IEEE Transactions on Software Engineering, 36(5):618–643, 2010.

Duplicate detection: P. Runeson, M. Alexandersson and O. Nyholm, Detection of Duplicate Defect Reports Using Natural Language Processing, *Proceedings of 29th Int. Conference on Software Engineering*, pp. 499-508, Minneapolis, MN, USA, 2007.

9 Keywords in Software Testing

The keywords in Table 2 can be used if you want to decide on a subject not specified above. Another source that is appropriate to use is the Software Engineering Book of Knowledge. The latest version can be downloaded on http://www.swebok.org.

10 Journals and Conferences

Below in Table 3, there are some journals and conferences listed which publish results of software testing research.

Table 2: Keywords in software testing

Black-box testing	**Gray-box testing**	**White-box testing**
Equivalence partitioning		Reference models from code-based testing (flow-graph, call-graph)
Boundary value analysis		Control flow-based criteria
Decision table		Data-flow based criteria
Finite-state machine-based		Mutation testing
Testing from formal specification		Coverage measures
Error guessing		
Random testing		
Operational profile		
Application testing	**Testing in the software process**	**Test methods**
Object-oriented testing	Connection between requirements and testing	Statistical usage testing (usage based)
Component-based	Architecture Testability	Specification-based software testing
Web-based		Code-based
GUI testing		Fault-based
Testing of concurrent programs		Testing from formal specification
Protocol conformance testing		Random testing
Testing of distributed systems		
Testing of real-time systems		
Testing of scientific software		
Objectives of testing	**Evaluation of the testing**	**Testing metrics and measurements**
Acceptance testing	Coverage measures	Test process
Installation	Fault seeding	Documentation and test-ware
Alpha, beta	Mutation score	Test organization vs. company
Conformance, functional, correctness	Comparison and relative effectiveness of different techniques	Size/Cost/Effort estimation and other metrics
Regression	Stopping criteria	Termination and stop criterion
Performance, stress		Test reuse and test patterns
Recovery		
Configuration		
Usability		
Estimations in software testing	**Empirical methods in software testing**	**Quality models and software testing**
Reliability	Compare testing techniques	CMM
Reliability growth models	Compare inspection and testing	SPICE
Types and classification of faults		Testing with SQA
Remaining faults and faults density		Testing and certification
Inspections	**Test management**	**Testing targets**
Process		Unit
Reading techniques		Integration
		System
		Alpha, Beta
Automated software testing (tools)	**Simulation of the test process**	**Testing connected to development process**
Tools for different techniques	Discrete event models	XP
	System dynamics models	Cleanroom
		Risk-based

Table 3: Journals and conferences

Journals	**Conferences**
ACM SIGSOFT Software Engineering Notes	International Conference on Software Testing, Verification and validation (ICST)
ACM Transactions on Software Engineering and Methodology	International Conference on Software Engineering (ICSE)
Empirical Software Engineering	International Conference & Workshop on the Engineering of Computer-Based Systems (ECBS)
IEEE Software	International Conference on Empirical Software Engineering and Measurement (ESEM)
Information and Software Technology	International Conference on Automated Software Engineering (ASE)
Journal of Systems and Software	International Symposium on Software Reliability Engineering (ISSRE)
Software Engineering Journal	International Conference on Software Testing and Analysis (ISSTA)
IEEE Transactions on Software Engineering	
IEE Proceedings Software Engineering	
Software Quality Journal	
Software Testing, Verification and Reliability	