Test Prioritization Tools

This is a booklet in a series from the EUREKA ITEA Testomat Project - The Next Level of Test Automation

Please follow us on:
Web: www.TestomatProject.eu
Twitter: @TestomatProject
www.youtube.com/TestomatProject

Content

1. Introduction

Software testing plays an important role in software development and increases in value when the software system becomes complicated and large-scale. In order to improve the cost-effectiveness at different stages of the development cycle and in different test activities, two major

approaches are used: The obvious approach is to aim for minimization through test suite reduction, by removing old or redundant test cases or test steps, and secondly is to find an order or a selection through test case prioritization. In test prioritization, the goal is to order test cases to locate faults early in the test execution cycle. Test prioritization is important because a large number of test cases can accumulate over several life cycles of software systems. Running all the test cases is expensive, and thus, prioritization methods are proposed that the test cases based on certain criteria with the goal of detecting faults quickly.

2. Test prioritization techniques

Test prioritization is important to give fast feedback to changes in the software, and put emphasis on what is important. In test prioritization of existing test cases, the goal is to order test cases from most important to least important. A most important test case could differ over time, and phase in testing - and also differ among stakeholders. When large test suites are created, running all the test cases is expensive, and thus, prioritization methods are proposed which order the test cases based on certain criteria with the goal of detecting faults

quickly. There are several benefits to prioritize your tests: i) it provides a way to find more bugs under resource constraint condition and thus improves the reliability of the system quickly; ii) faults are revealed earlier, engineers have more time to fix these bugs and adjust the project schedule.

The prioritization of test cases depends upon various factors. The various factors that classify the prioritization techniques in the different classes are explained below:[1]

Customer requirements: In the customer requirements based prioritization techniques test cases are prioritized on the factors decided on the requirements of customers documented during the phase of requirements gathering.

Coverage based: Based on coverage the prioritization of the test cases are on the quantity of the source code of a program that has been exercised during testing. In this approach the test cases having the capability of testing a larger part of the code are prioritized.

Cost effective: The test cases are prioritized on the basis of the cost factor in this approach. The cost can be the cost of requirement gathering, cost of regression testing, cost of execution and validating test cases, the cost of analyses to select and support a test case, cost of prioritization of

[1] Khandelwal and Bhadauria 2013

test cases or any other implicit cost, e.g. test environment (hardware), competence or other cost pending factors in the development or production cycle.

History based: The test cases are prioritized based upon the history of the test case itself which means priority of test case depends upon its previous execution time, rate of of finding failures and other performance metrics.[2]

Churn: Testing can also be prioritized based on churn,[3] e.g. changes. Meaning that you prioritize the test cases affected by the latest code change. Depending on your architecture, programming language choice and many other development factors e.g. how you associate and connect your tests with the code.

Fault-based: By constantly collecting statistics on every execution of the software, information from e.g. customers, changes, and pass-fail history of the test case, prioritization can be based on the fault history, including severity, occurrence.

2.1 Requirement Prioritization Impacting Test Case Prioritization

In this technique, the requirement is tested first and assigned the higher weight and the test cases covering that requirement are

[2] Khalilian et al., 2012

[3] Knauss et al, 2015

given higher priority of execution. There are four factors to decide the weight of the requirement.

Business value measure: This is the factor in which the requirements are assigned rank according to their importance.

Project Change Volatility: This factor depends on how many times customer modify the project requirements during the software development cycle.

Development Complexity: This factor depends on development efforts, technology used for development, environmental constraints and the time consumed or the complexity of the development phase.

Fault Impact of Requirements : This factor considers requirements which are error prone according to the historical data, failure reported by the customer. Developers can identify requirements that are expected to be error free by using the prior data collected from older versions.

3. Introduction to Tools/Techniques in Testomat Project

3.1 MBTCreator and MBTP (ifak)

The MBTCreator is a tool that combines various functionalities for model-based testing and test prioritization. MBTCreator

offers editors and a graphical user interface for a toolchain that covers all steps from requirements to test case generation and prioritization.

In a first step, the tool features methods for formalization of requirements using a notation language, the IRDL (Ifak Requirement Description Language). A state machine can then be generated from formalized requirements, which models all behavior of the SUT as described in the requirements. Test cases are then generated for the state machine using one of several coverage based test goals. Within TESTOMAT, an extension of the toolchain in the form of a separate tool called MBTP (Model Based Test Prioritization) was developed, with which the generated test cases are prioritized via a combination of model-based cluster analysis and a requirements-based evaluation procedure (see figure 3).

The first step is a cluster analysis (of generated test cases) using coverage metrics in regard to the specification model. Here, all test cases are evaluated regarding the selected coverage metric (e.g. state-, transition-, path-coverage etc.). In a second step, the test cases in each cluster are prioritized according to one or several requirement-based criteria (e.g. complexity, volatility, risk, customer priority). The

prioritization goal is to improve test coverage and fault detection rate during early testing.

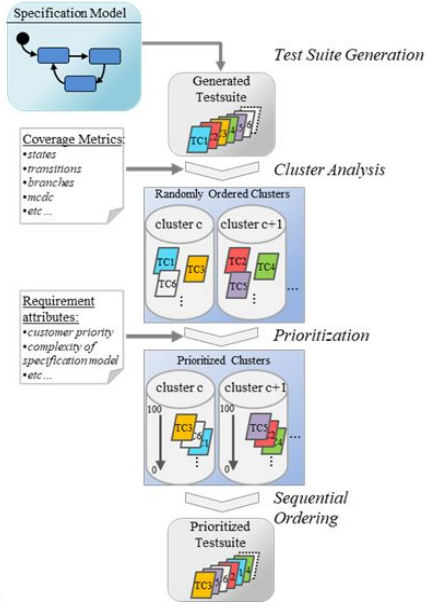

Figure 1. Basic functionality of mbtp

3.2 The SBTTool (Alerion/MGEP)

The SBTTool is a tool including test suite generator and test case prioritizer, and it facilitates simulation-based testing, by automating the generation and prioritization of test cases. The SBTTool can be integrated with a Continuous Integration (CI) tool such as Jenkins, that can be in charge of orchestrating the simulation-based testing components. We use it in Alerion in our simulation environment consisting of ROS [1] and Gazebo[2]. The SBTTool also has a Test Oracle that determines the outcome of the test (see Figure 2).

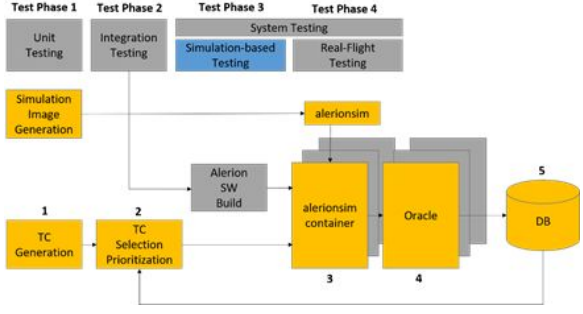

Figure 2. Alerion Simulation based testing process

3.2.1 Test Suite generation in SBTTool

The test case generator is based on a model, a so called Scenario Specification Model (SSM) that defines the relevant parameters of the simulation scenario to be tested, such as the inspection of petroleum pipelines, hydroelectric dams, bridges or wind turbines among other infrastructures.

The SSM, as can be seen in Figure 3, is created by a Test Scenario that comprises a set of initial parameters (Initial Param Set) and execution parameters (Execution Param Set). The Initial Parameters Set is composed of those parameters that must be set before running the simulation. A few examples of these kinds of parameters are, the height of a bridge abutment, the angle of the blade in a windmill, the width of a dam floodgate, etc.

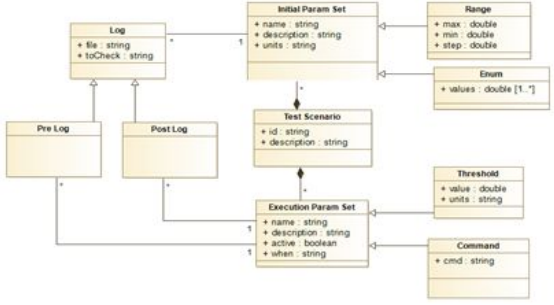

Figure 3. Scenario Specification Model

The Execution Param Set type parameters are used to define those events that can occur during the simulation. Such as problem in a hardware/software module, communications problems or atmospheric phenomena, like rain, that can occur with a predefined intensity during the simulation.

The test case generator uses an adaptive random algorithm to create test cases. This algorithm was selected to cover the widest spectrum with the fewest test cases, which will allow more faults to be found. To create a test suite, the test case generator needs four inputs: (i) the Scenario Specification Model, (ii) the number of test cases to generate, (iii) the number of test case candidates and (iv) the algorithm to measure the distance between the test cases.

3.2.2 Test case prioritization

The prioritization will be performed in two phases. In the first phase, the primary goal is to analyze and create (gather) historical data about the simulations, failures found, execution times, coverage achieved, and so on. In the second phase, historical data generated in the first phase will be then considered for prioritization.

In the first phase, during analysis, the prioritizer receives as input the test suite, the number of test cases to execute and the algorithm to calculate the distances between test cases. The system first selects a test case randomly from the test suite and inserts it in the selected test cases. Then it searches for the most different test cases in the test suite, considering all test cases selected until that moment. To calculate the differences between test cases, the prioritization parameters values are considered and the Euclidean distance is calculated to select the test case with the highest value. This process is repeated until the desired number of test cases is obtained.

The aim of the second phase is to carry out prioritisation based on the historical data obtained from the first phase. The prioritizer will receive a test suite and the desired number of test cases. Unlike in the first phase, the test suite will also include historical data about previous test case executions which will be used to carry out prioritization.

4. TESTONA & MODICA (Expleo)

With TESTONA and MODICA, Expleo offers two major tools in the area of test

prioritization. Both these tools can be used independently of a specific test execution platforms such as JUnit, EXAM and Canoe.

TESTONA is a test design, test generation and test prioritization tool based on the classification tree method that classifies tests (and other artifacts) into a tree of different classes.

The classes can represent variations in configuration data, tested steps or different Systems under Test. The tree can also be used to model a test oracle using dependency rules that formalize allowed combinations of classes. Especially when

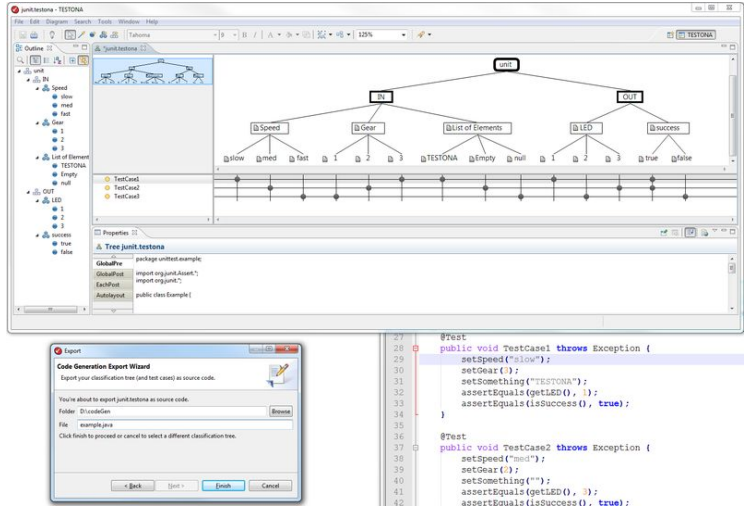

Figure 4. TESTONA overview

the number of classes is big or their dependencies are complex, a main purpose

of TESTONA is not simply organizing tests but generating them in the first place. This generation can be restricted and parameterized in many ways to prioritize certain kinds of test goals.

In many contexts, classes or their combinations can be linked to metrics such as usage frequency, risks, code complexity, etc. Armed with this data, TESTONA can both prioritize existing test suites and generate test suites that are optimized accordingly. Therefore the user can select different algorithms and metrics for prioritizing the test suite. Examples would be to generate a test suite containing pairwise combinations of all classes that have minimal execution time or has the maximal probability of containing faults. Furthermore, the generated tests can easily be annotated with traceability data that can be used when prioritization is required in a later process stage (e.g. when the prioritization also depends on variables not available during test design time).

MODICA is a tool for model based testing (MBT). It allows to graphically model systems and their environment using hierarchical UML-state machines.

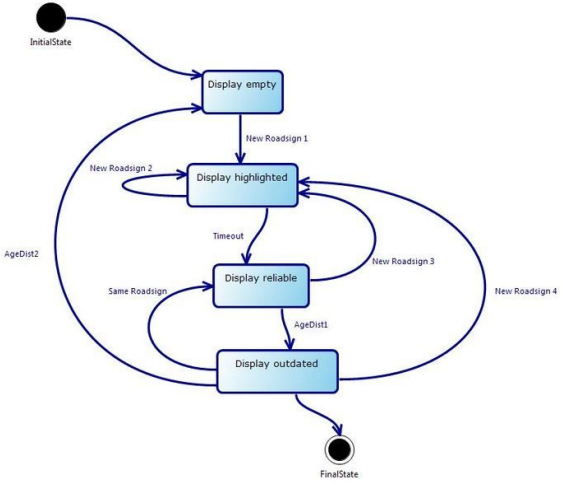

Figure 5. A simple state machine in MODICA

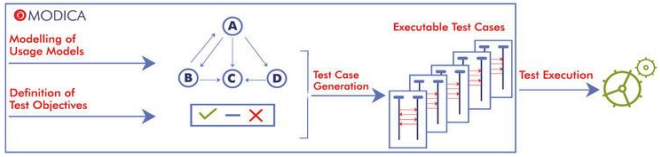

Figure 6. Workflow in MODICA

By default, the test generation maximizes the coverage of all structural statechart elements (states, transitions, …) and all of the generation aspects can be customized. There are different presets that the user can

select and more advanced prioritization schemes that match the importance of these elements individually or their combinations can be defined. Multiple sets of these settings can be defined for a single model for example when test suites for different test stages are automatically generated with different prioritizations. Similarly to TESTONA, MODICA also allows to automatically calculate metrics relevant for prioritization based on the path taken through the model. These metrics include requirement coverage, amount of conditional branchings and coverage of states or transitions. These metrics can also be combined. MODICA will then present a test suite sorted by the used metrics to the user. This data can then be used when prioritization should be done in later process stages.

5. Codescene by EMPEAR

CodeScene applies machine learning algorithms to identify and prioritize technical debt in both application code and test code. The tool goes beyond code as CodeScene considers the organization and people side of the system, as recorded in version-control history. That way, the tool can predict off-boarding risks and detect inter-team coordination needs as well.

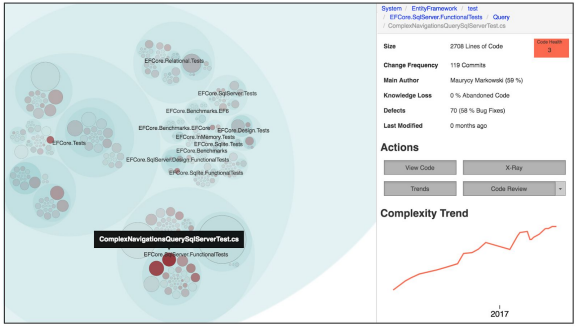

Figure 7. CodeScene prioritizes technical debt in automated tests

CodeScene's virtual code reviewer includes automated detection of test code smells. Based on these findings, an organization can refactor their automated test code based on where code improvements have the most likely return on investment.

In addition, CodeScene's predictive analytics detect delivery risks, which lets an organization focus additional testing and verification activities to where they are likely to be needed the most.

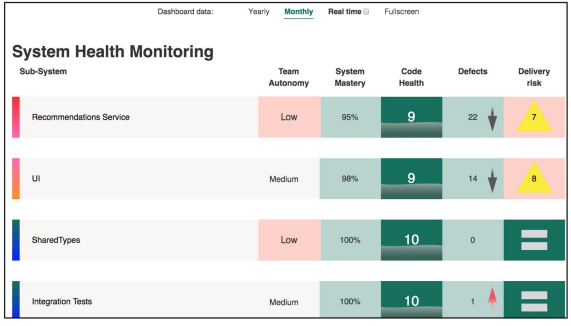

Figure 8. CodeScene's real-time detection of delivery risks on system and component levels

CodeScene supports a goal-oriented workflow to make its technical debt priorities actionable and visible to the whole software organization. For this purpose, CodeScene lets stakeholders plan goals and record decisions directly in the tool. Those goals are then automatically supervised, and feedback and progress is delivered via the following channels:

- Quality gates in a build pipeline.
- Automated code review comments on pull requests.
- Auto-generated PDF reports targeting architects/managers/testers.
- Warnings for violated goals on the analysis dashboards.

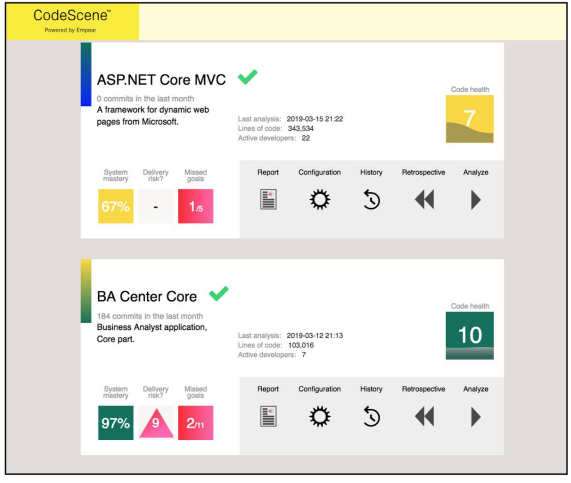

Figure 9. A high-level dashboard presents the technical debt and goal fulfillment for all products in the organisation

6. sepp.med MBTsuite

Together with sepp.med AKKA extended the MBTsuite for better prioritization by introducing the dynamic tag filter. MBTsuite is a specialized test case generator based on the model-based-testing method. The MBTsuite automatically generates executable test cases and test data from graphical (UML) test design models. All generated test cases are platform-independent and may be exported into various formats or test management tool to use for manual testing or any test automation tool.

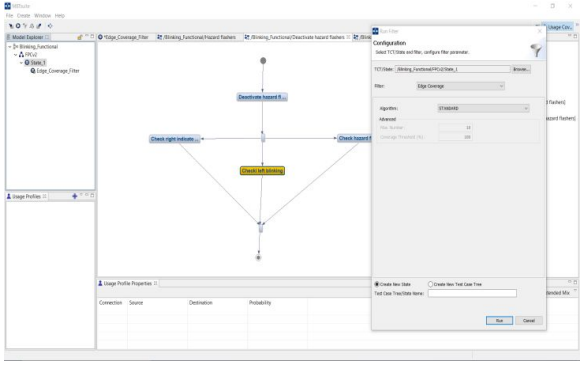

Figure 10. MBTsuite screenshot

The user benefits from the models as a basis for discussion and decision-making. With the MBTsuite you have the chance to synchronize the test process with the development process, as you constantly actualize the test model parallel to the development progress. MBTsuite supports test management by defining meaningful test coverage and traces the requirements to test cases down to the test steps.

The Dynamic Tag Filter allows the users to filter test cases very individually. Logical expressions can be used to filter exactly for those test cases which meet certain criteria, based on attributes the user defines.

In a customer project AKKA used this dynamic tag filter to prioritize test cases based on risk analysis, user experience analysis, and FMEAs. In this way test sets were based on more important test cases in

respect of security, but also allowed a general coverage of the testing space.

7. Atlassian Jira Xray for Test Management

Jira provides as plugin an overarching tool for Test Management called Xray. It can be utilized during the entire testing life cycle and provides an efficient solution for each phase of the life cycle: test planning, test design, test execution or test reporting. In the test design phase and once the test cases are identified, a wide range of options are provided to define pre-conditions, label the testing level, priority of the test case and the testing methodology to be used for execution of test cases etc. The tester doing the test design based on his understanding of the requirement has to define the priority of each test case. Thus, this predefined prioritization of test cases makes it easier to include the test case in the final test execution plan.

Xray provides a comprehensive report of the entire test execution activity. Once the test execution tasks have been completed, a report can be generated automatically and the report contains a graphical matrix about the total number of successfully executed test cases, failed test cases and the number of bug tickets created. Moreover, the test execution plan also provides the execution

history of each test case i.e. how many times a test cases passed and failed.
Therefore, based on matrix in the Xray report, it becomes comparatively easier for the testers to identify test cases for future test execution. Additionally, it also helps in determining the quality of the product based on the number of times a test case failed and the number of bug tickets being created.

8. Intel: Coverage based prioritization: tselect (part of commercial C++15 compiler)

The test prioritization tool, also known as the tselect tool, enables the profile-guided optimizations on all supported Intel®[4] architecture, on Linux, Windows, and OS X operating systems, to select and prioritize tests for an application based on prior execution profiles.The tool offers a potential of significant time savings in testing and developing large-scale applications where testing is the major bottleneck. The test

[4] User and Reference Guide for the Intel® C++ Compiler 15.0

prioritization tool lets software developers select and prioritize application tests as application profiles change.

The test prioritization tool provides an effective testing hierarchy based on the code coverage for an application. The following list summarizes the advantages of using the tool:

- Minimizing the number of tests that are required to achieve a given overall coverage for any subset of the application: the tool defines the smallest subset of the application tests that achieve exactly the same code coverage as the entire set of tests.
- Reducing the turn-around time of testing: instead of spending a long time on finding a possibly large number of failures, the tool enables the users to quickly find a small number of tests that expose the defects associated with the regressions caused by a change set.
- Selecting and prioritizing the tests to achieve a certain level of code coverage in a minimal time based on the data of the tests' execution time.

Usage Model: The following figure illustrates a typical test prioritization tool usage model.

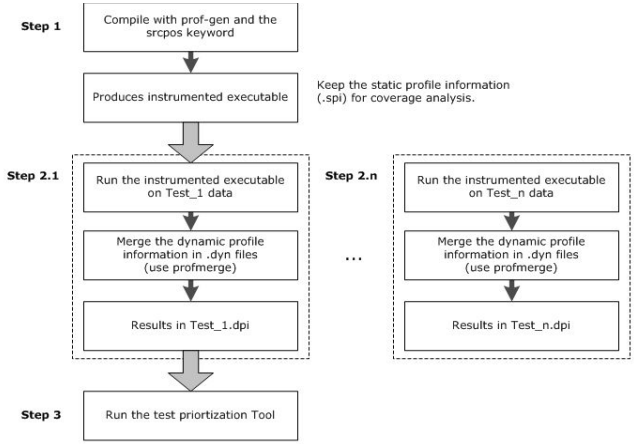

Figure 11. Usage Model

9. Coverage based prioritization: (Java): Jnan Test Prioritization Tool

An Automated coverage collection tool that can capture the statement coverage for the program under test and then use the information to prioritize the test classes in the test suite. The tool uses ASM bytecode manipulation framework to manipulate the bytecode. Bytecode manipulation is performed on the fly by a Java Agent which makes use of the Instrumentation API. A JUnit listener is used to capture the start and end events for each JUnit test method. The agent jar file and the JUnit listener class can be integrated with any maven

project to perform code coverage. This is done by updating the pom.xml file present in the project root directory.[5]

10. SmarTest

SmarTest is a testing module for accelerating the detection of faults in Drupal.[6] Also, SmarTest allows the testers to prioritize the executions of the test modules in order to detect faults as fast as possible.

A Test prioritization tool for drupal, includes support for cyclomatic complexity, relevant commits for module, code covered by tests, module size, test fault history. SmarTest enables a dashboard with statistics about the Drupal system in real time. This information allows to guide the testing the system through faults propensity data in different parts of the code.

11. Dextool Mutate (Saab Aeronautics)

One of the plugins for Dextool, called Mutate, is a Mutation Testing tool for C/C++. This tool contains several different reporting options for mutation testing that allows the

[5] Paul and Balasubramanian, 2017

[6] Sánchez, 2015

user to prioritize test based on different criteria. In general, developers use a combination of the options in order to better prioritize their tests.

11.1 Minimal set

With time the number of test cases increases and the runtime with it. Sooner or later the test suite is too slow to run continuously for a development team, but the team does not want to leave the testing to the end. They want a collection of test cases that have a high probability of catching the common mistakes. One of the report options can be used to generate such a smoke screen test suite automatically. The report collect a set of test cases that achieve the current mutation score. This has been shown by experience to always be less than the total number of test cases (30%-80%).

11.2 Test Case Uniqueness

In order to establish, or get an indication, that test cases have become redundant and can be removed from the test suite, an option for reporting the test case uniqueness is provided in Dextool Mutate. This options reports what test cases that kill mutants "uniquely" and test cases that have no unique killed mutants. Using this feature can help prioritize which tests that needs to

be reworked, can be removed completely, or just subject to further investigation.

11.3 Test Case Similarity

A team that are working with a test uniqueness report to understand how to reduce the current test cases may want help to compare the available test cases with each other. The test case similarity report calculate how test cases intersect with each other and present these intersections and differences to the developer. This has successfully been used to merge test cases that have a high similarity, or to remove parts of test cases that are "overtested".

12. Axini Modeling Suite

The Axini modeling suite (AMS) consists of the components: modeling, testing and application generation. AMS is used in the financial, high-tech and rail industries. Examples are pension administration systems, medical devices and 24/7 rail control systems.

In the modeling environment modelers create, visualize, debug and document their models. A strong characteristic of the modeling language is that it supports actions, data and time. The testing environment automates the entire test-process: test-case generation,

execution of test-cases and evaluation of the test-result. For models with enough detail it is possible to generate applications and/or smart-stubs.

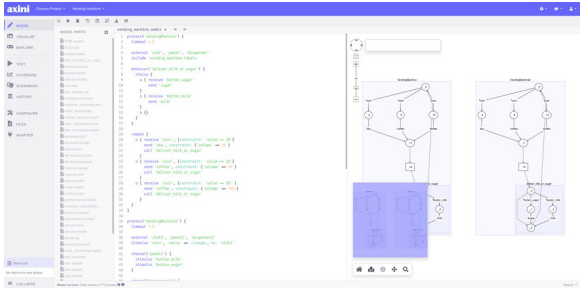

Figure 12. AMS editor and visualization

For test-case prioritization there are several test-strategies. There are generic strategies that focus on coverage of model-properties like state-coverage, transition-coverage, data-coverage; it is also possible to focus on specific states, transitions or actions. Another generic strategy handles the type of test-cases: good-weather, bad-weather, ugly weather. The specific strategies focus on specific areas of the system or use-case that are important for the client. Examples are the coverage of specific requirements, use-cases and error-conditions.

13. Conclusion

In order to improve the cost-effectiveness of test activities, test case prioritization have been proposed. The main purpose of test

case prioritization is to rank test cases execution order to deliver fast feedback to developers and testers, and detect fault as early as possible which means secure what is most important at the moment.

In this booklet, we have presented a small selection of test prioritization tools used, within the TESTOMAT Project, where some of our tools are also available to procure. Our goal is to make our research accessible and useful to both the academic and the industrial community. We are aware that there is an abundance of other prioritization tools on the market, and also locally produced, which we have not addressed, as this booklet is not to be seen as complete tools overview on the subject.

References:

Elbaum, S., Rothermel, G., & Penix, J. (2014). “Techniques for improving regression testing in continuous integration development environments”. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 235-245.

Khalilian, A., Azgomi, M. A. and Fazlalizadeh, Y. (2012). “An improved method for test case prioritization by incorporating historical test case data”. Science of Computer Programming Vol. 78, pp. 93–116.

Khandelwal, E. and Bhadauria, M. (2013). “Various techniques used for prioritization of test cases”.

International Journal of Scientific and Research Publications, Volume 3, Issue 6.

Khatibsyarbini, M., Isa, M. A., Jawawi, D. N., & Tumeng, R. (2018). “Test case prioritization approaches in regression testing: A systematic literature review”. Information and Software Technology, pp. 74-93.

Knauss, E., Staron, M., Meding, W., Söder, O., Nilsson, A., & Castell, M. (2015). “Supporting continuous integration by code-churn based test selection”. In Proceedings of the Second International Workshop on Rapid Continuous Software Engineering, IEEE Press, pp. 19-25.

Ma, Z. and Zhao, J. “Test case prioritization based on analysis of program structure”. Department of Computer Science Shanghai Jiao Tong University, China.

Paul, J. and Balasubramanian, N. (2017). “Jnan Test Prioritization Tool”, University of Texas at Dallas (UTD) https://github.com/Nandita-93/Test-Prioritization-Tool

Sampath, S., Bryce R.C, Jain S. and Manchester, S. (2011). “A Tool for combinatorial-based prioritization and reduction of user-session based test suites,” 27th IEEE International Conference on Software Maintenance (ICSM).

Sánchez, A.B., Segura, S. and Ruiz-Cortés, A. (2013). “A comparison of test case prioritization criteria for software product lines”. Applied Software Engineering Research Group University of Seville, Spain.

Sánchez, A.B. (2015). “SmarTest”. Departamento de Lenguajes y Sistemas Informáticos Escuela Técnica

Superior de Ingeniería Informática Universidad de Sevilla Spain http://www.isa.us.es/smartest/index.html

User and Reference Guide for the Intel® C++ Compiler 15.0
https://software.intel.com/en-us/node/522744

References for Tools/Contact Person

MBTCreator and MBTP (ifak): It is not open source yet, but in the long term they will be open source

Contact person: martin.reider@ifak.eu

The SBTTool (Alerion/MGEP):
The type of license for the tool is still to be defined. In case you are interested in using it, please get in touch with one of the contact persons, for more details.
Contact persons: Oier Peñagaricano: oier@aleriontec.com and Leire Etxeberria letxeberria@mondragon.edu .

TESTONA (Expleo): TESTONA and all information is available at http://www.testona.net/

MODICA (Expleo): MODICA and all information is available at https://www.expleo-germany.com/en/products/modica/

Codescene by EMPEAR: CodeScene is available as a SaaS at https://codescene.io/ and in an on-prem version via https://empear.com/

Contact person:
adam.tornhill@empear.com

Dextool Mutate (Saab Aeronautics):
https://github.com/joakim-brannstrom/dextool/tree/master/plugin/mutate

Contact person:
joakim.k.brannstrom@saabgroup.com, niklas.pettersson2@saabgroup.com

MBTsuite (sepp.med):
https://www.seppmed.de/de/portfolio/mbtsuite/

Contact person:
martin.beisser@seppmed.de

Axini Modeling Suite:
https://www.axini.com/

Contact person:
Dr. Ir. Machiel van der Bijl
vdbijl@axini.com

Acknowledgements:

This booklet is produced by
EUREKA ITEA3 TESTOMAT PROJECT
The Next Level of Test Automation
Find out about us on the web:
https://www.testomatproject.eu/

Follow us on Twitter
@Testomatproject

This booklet was produced by a research collaboration between the following partners:

The Testomat Project is sponsored by:

We'd love to hear feedback from you! Contact us via Twitter or this feedback form: https://goo.gl/J5wnjm