Enhancing an Agent-based Test Case Prioritization System by Event Evaluation

Sebastian Abele, Peter Göhner and Michael Weyrich

Institute of Industrial Automation and Software Engineering,
Pfaffenwaldring 47, 70550 Stuttgart, Germany
{sebastian.abele,peter.goehner,michael.weyrich}@ias.uni-stuttgart.de
http://www.ias.uni-stuttgart.de

Abstract. Test case prioritization systems are useful tools to support test managers in planning test runs in regression testing. These systems evaluate information about test cases to find the best order for the test case execution. During a system's life cycle, a lot of test runs are usually performed. To give optimal support and to improve the results from test run to test run, the prioritization systems need to react dynamically to events, which occur during the different phases *development*, *test* and *operation* of the life cycle. They have to consider changes, unexpected faults found during operation, and many other events. This article describes an agent-based approach to improve a test case prioritization system by considering dynamic events for the test plan generation.

Keywords: test case prioritization, software agents, event evaluation

1 Motivation and State of the Art

Developing systems with a high quality is an important factor to succeed in the market. The high competition leads to a decreasing time-to-market and therefore to the need for efficient development and test processes. One of the major parts of the test process is the system test. It is essential to prove the compliance of the system with the quality requirements. The system test planning, specification, execution and evaluation is usually a process, which accompanies the whole development process and cannot be considered isolated. The system test is embedded in the system's life cycle. During its whole lifetime, a system is usually changed and modified to adapt to new requirements and boundary conditions. Modifications of the system are always threaten to introduce faults to the system. Even fault-free modifications can reveal faults that were already inside the system. To minimize this risk of introducing faults, the system needs to be retested after modifications. For this purpose already available test cases can be reused and executed again. The test strategy of repeating already available test cases is called regression test. Over time, the test suites, which contain the test cases for regression testing can grow to very large repositories with thousands of test cases. Executing all the test cases takes a vast amount of time and resources, which are often not available. Especially for minor changes, the execution of the whole test suite is not suitable. Instead a selective testing of the modified system parts and components is needed.

Test planners have to identify suitable test cases to ensure that the quality of the system is not affected by faults, which were introduced with changes. They have to pick test cases out of the available test suite or define new test cases if the already available test cases are not sufficient. In order to address these challenges, test case selection and prioritization techniques are used by support systems to find the most important test cases for the available execution time. Test case selection techniques reduce the test suites by identifying only relevant test cases, for example based on changes of the source code. An overview of different test selection techniques can be found in [3] and [14]. Prioritization techniques go one step further and order the test cases by their expected benefit for the test. Unlike test case selection, a test run that is executed on the basis of prioritized test cases may be interrupted at any time having still the maximal benefit possible by the interruption time. Every test case prioritization technique can also be seen as a selection technique by skipping test cases that have a lower priority than a predefined threshold value. Test case prioritization techniques are described in [2] and [5].

The techniques can be categorized by the data they evaluate to generate the priorities. Many of the techniques proposed in the literature are based on white-box information. They analyze the source code directly to optimize the coverage of changes, e.g. in [13]. Prerequisite for these techniques is the direct access to the source code and its different versions. Due to the distribution of different development tasks to different departments and the incorporation of external modules into the system, this access is not guaranteed in every case. Hence, the black-box methods have been proposed. They use data from the test itself like execution and fault histories of test cases to prioritize them, e.g. [11]. Some techniques especially concentrate on information about requirements, identifying them as the most important entities for the test case prioritization, e.g. [12].

Some approaches have been proposed to adapt the prioritization to the ongoing test process. In [4] a prioritization technique is proposed that adapts immediately to test case results as soon as they are available. In contrast to other prioritization techniques, the test plan can be modified dynamically after every test case execution if the test results necessitate the change of the test plan. In [6] a history-based technique is proposed, which considers differences in the significance of historical data from different test runs. Younger data gets a higher weight when determining the priority. In [10] the incorporation of human factors of the developers is proposed. Their performance is tracked over time and used to find a better test case prioritization.

Those techniques are well-suited to optimize the usage of the knowledge, which is contained within the historical database. Based on that data, the proposed techniques are able to generate a very good test plan. However, the techniques are limited to the evaluation of the data that is stored during testing. To generate a optimal test run, other factors need to be considered, too. Dynamic events like the unexpected occurrence of faults at the customer's site or spontaneous changes of requirements, lead to the necessity to adapt the test plan accordingly. Currently, these adaptations are done by test managers based on their experience. In the underlying PhD work, I propose an agent-based test management support system, which is able to react to such dynamic events. By drawing conclusions from events, test plan adaptations are performed to relieve the test manager. Therefore the system will be able to provide better support than state of the art systems.

2 Basis: Agent-based Test Case Prioritization

In a preliminary work done by Malz et al. an agent-based test case prioritization system has been developed, which generates a test plan considering test case priorities and test resource utilization [7], [8], [9]. The prioritization is performed in four steps:

Malz et al. recognized the problem of test case prioritization as a distributed problem: Parts of the system, e.g. components and modules need to be tested. They compete for as much test effort as possible for them. On the other hand, the test cases, which provide those tests, compete for test resources, which are necessary to execute them. They decided to model their test case prioritization problem as a multi-agent system and established agents representing software components and agents representing test cases. Additionally the system uses interface agents to gather needed data. The following steps are performed by the agents:

Step 1: Gather Data – Each interface agent is responsible for one data source. Data sources can be test support tools and databases that store relevant test data. The agents use interfaces, which are provided by the linked tools or use a direct database connection. Therefore they need knowledge about the stored data and about the data structures.

Step 2: Approximate components' fault pronenesses – Each system component is represented by an agent called *Component Agent*. This agent holds the information about its component, which have been delivered by the interface agents in step 1. With this information and a special knowledge base, it determines a value for the expected fault proneness of the component. The rule-base is described in [9].

Step 3: Approximate test cases' fault finding probabilities – After the fault pronenesses of the component has been determined, test cases are searched that are suited to test especially the fault-prone components. Therefore each test case is represented by a dedicated agent called *test case agent*. The test case agent calculates the fault finding probability of its test case for each component. For the estimation of the fault finding probabilities, tester-given rules are used as well.

Step 4: Calculate test case priorities – In the final step, the previously calculated values for the fault proneness of components and the fault finding probabilities of test cases are combined to a priority value for the test cases. The more components with a high fault-proneness are well tested by a test case, the higher raises its priority. The priority is calculated as the mean value of a test case's fault finding probability weighted with the corresponding fault-proneness values of the covered components.

3 New Approach: Adding Response to Dynamic Events

The approach presented here has the goal to extend the agent-based test management system with the capability to react to dynamic events, which occur during the development of a system. The system has three possibilities to react to such events:

Modifying or amending data – If the system recognizes that the data it is evaluating is wrong, or if an event brings new findings, which are not contained in the data yet, the system can modify the stored data directly. Especially user-given values like complexity and criticality approximations can be verified in later development stages when more empiric data is available.

Adapting or reparameterizing the prioritization algorithm – Long term changes or slight deviations in the system require a long-term adaptation of the prioritization. Therefore the prioritization algorithm is adapted itself. Parameters used for calculations or rules used for direct inferring can be changed if necessary.

Modifying the resulting test plan directly – Serious events may require an immediate change of the test plan as an adequate reaction. This immediate change cannot be achieved by modifying the data or by adapting the prioritization algorithm in a long-term evolution step. Rather, a direct intervention in the test plan itself is needed. S

3.1 Dynamic Events During a System's Life Cycle

This section describes the events, the agent system has to react to. Events are classified into three categories: Events from the Development, events from the test and events from the system in use.

Events from development Events from development are triggered during the development process - either while adding new features to the system or while correcting faults that have been found in a preceding test run.

– Source code change to fix a fault

- Source code modification to fix a recurred fault
- Source code modification to implement a new requirement
- New test case available

Events from test Events form test are triggered during the test phase. Usually a test phase starts after a development phase to get sure that the changes from the development phase didn't introduced faults to the system. Events from test can refer to those from the development. So, an event history with an appropriate assessment is needed.

- Test case executed
- Test run completed

Events from operation Events from system in operation are usually triggered by a customer's input. Since those events occur at the customer's site, they are the most critical ones. They usually need a fast reaction.

- Fault reported by customer
- Requirements changed
- Requirements' importance changed

3.2 Adding event-evaluation to the agents

To add the capability of reacting to dynamic events, the agents firstly need to recognize the events. After an event is recognized, the agents need to draw the correct conclusions. Some events, like the completion of a test run, can be recognized by observing the data sources of the other test tools. As far as the results are available, the event is triggered. Other events need special interfaces or human interaction because they cannot be recognized automatically.

After the events are recognized and sent to the concerning agents, they have to decide, which of the possibilities described above they use. To recognize faults in the data sources the agents use consistency and plausibility checks and rules describing wrong parameters. The long-term evolution by modifying the prioritization algorithm with a learning algorithm has already been investigated and is described in [1]. A genetic algorithm is used to optimize the rule-weights of the rules used to calculate the fault-pronenesses of components.

The possibility to alter the test plan directly is to be investigated. A rule-based approach with learning capability is imaginable here.

4 Summary

During the life cycle of a system, a system test is performed periodically. To maximize the test efficiency in the available test time, an state of the art agent-based test case prioritization systems is used, which order the test cases due to their performance. Based on the prioritization, a test plan is created. To enhance the test plan generation over the life time of the system, dynamic events need to be recognized and considered by the prioritization system. In the approach presented here, those events are recognized and evaluated by the agents in order to improve the test case prioritization over time.

References

1. Abele, S., Göhner, P.: Improving proceeding test case prioritization with learning software agents. In: International Conference on Agents and Artificial Intelligence. pp. 293–298 (2014)
2. Elbaum, S., Malishevsky, A., Rothermel, G.: Test case prioritization: a family of empirical studies. IEEE Transactions on Software Engineering 28(2), 159–182 (2002)
3. Engström, E., Runeson, P., Skoglund, M.: A systematic review on regression test selection techniques. Information and Software Technology 52(1), 14–30 (2010)
4. Hao, D., Zhao, X., Zhang, L.: Adaptive test-case prioritization guided by output inspection. In: 2013 IEEE 37th Annual Computer Software and Applications Conference (COMPSAC). pp. 169–179 (2013)
5. Li, Z., Harman, M., Hierons, R.M.: Search algorithms for regression test case prioritization. IEEE Transactions on Software Engineering 33(4), 225–237 (2007)
6. Lin, C.T., Chen, C.D., Tsai, C.S., Kapfhammer, G.M.: History-based test case prioritization with software version awareness. In: 2013 18th International Conference on Engineering of Complex Computer Systems (ICECCS). pp. 171–172 (2013)
7. Malz, C., Jazdi, N.: Agent-based test management for software system test. In: IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR). pp. 1–6 (2010)
8. Malz, C., Göhner, P.: Agent-based test case prioritization. In: IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). pp. 149–152 (2011)
9. Malz, C., Jazdi, N., Göhner, P.: Prioritization of test cases using software agents and fuzzy logic. In: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST). pp. 483–486 (2012)
10. Malz, C., Sommer, K., Göhner, P., Vogel-Heuser, B.: Consideration of human factors for prioritizing test cases for the software system test. In: Engineering Psychology and Cognitive Ergonomics, Lecture Notes in Computer Science, vol. 6781, pp. 303–312. Springer Berlin Heidelberg, Berlin and Heidelberg (2011)
11. Park, H., Ryu, H., Baik, J.: Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing. In: Second International Conference on Secure System Integration and Reliability Improvement. pp. 39–46 (2008)
12. Srikanth, H., Williams, L., Osborne, J.: System test case prioritization of new and regression test cases. In: 2005 International Symposium on Empirical Software Engineering. pp. 62–71 (2005)
13. Walcott, K.R., Soffa, M.L., Kapfhammer, G.M., Roos, R.S.: Time aware test suite prioritization. In: Proceedings of the 2006 international symposium on Software testing and analysis. pp. 1–12. ACM, New York and NY (2006)
14. Yoo, S., Harman, M.: Regression testing minimization, selection and prioritization: a survey. Software Testing, Verification and Reliability 22(2), 67–120 (2012)