Consideration of Human Factors for Prioritizing Test Cases for the Software System Test

Christoph Malz[1], Kerstin Sommer[2], Peter Göhner[1], and Birgit Vogel-Heuser[2]

[1] Institute of Industrial Automation and Software Engineering, Universität Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Germany
[2] Department of Information Technology in Mechanical Engineering, TU München, Boltzmannstr. 15, 85748 Garching, Germany
{Christoph.Malz,Peter.Göhner}@ias.uni-stuttgart.de,
{sommer,vogel-heuser}@itm.tum.de

Abstract. A big challenge of software test managers is the limited test time. Especially the system test, where the whole integrated software system is tested shortly before delivery to the customer, is affected by this limitation. During the system test usually several test cycles are needed. However, a test manager cannot execute all available test cases in each test cycle due to the limited test time. He/she has to decide which test cases have to be executed in each test cycle in order to find new possible faults of the software. In this paper the Adaptive Test Management System (ATMS) based on software agents is presented which relieves the test manager from this complex manual work by using software agents for prioritizing test cases based on current information about the software system, the test cases and the human factors of the developers. The goal of the ATMS is to maximize the number of found faults in the available test time with the determined prioritization order.

Keywords: Test case prioritization, human factors, software agents.

1 Introduction

In the scope of industrial automation systems, a relatively high quality of the software system test has to be achieved by the test management in order to guarantee the required reliability of the system. One of the biggest challenges of the test manager is the limited time which he/she has for the software system test. Additionally, a software system usually has to be tested several times, due to possible software changes after each test cycle. Because of limited test time, the test manager cannot repeat all test cases in every test cycle. Somehow he/she has to prioritize the test cases in order to execute the most important ones, i.e. he/she has to find a prioritization order of the test cases that increases the probability to find more faults in the available test time.

Currently, available test management tools [1] offer support for prioritizing test cases by administrating different useful data. However, the big amount of data has to be evaluated by the test manager himself in order to find a prioritization order of the test cases. Automated prioritization techniques, as described in [2], prioritize test cases based on exact information about source code changes. However, they are not

D. Harris (Ed.): Engin. Psychol. and Cog. Ergonomics, HCII 2011, LNAI 6781, pp. 303–312, 2011.

appropriate for complex software systems. Another massive drawback of the existing approaches is the fact that they neglect human factors for prioritizing the test cases. However, since the goal of the prioritization is to increase the probability to find more faults in the available test time and since the faults are made by the developers of the software system, human factors of the developers play an important role in finding the right prioritization order.

A review of the relevant research literature shows that psychological studies as to human factors influencing the performance of programmers have been accomplished since the 1950s. But from the 1990s the focus of interest shifted towards graphical user interfaces (GUIs) resulting in notedly less publications about programming abilities.

Unfortunately, the correlations between the developed programming tests and the job performance of programmers were often quite low. Curtis [3] ascribes that to researchers' deficient understanding of the cognitive requirements of programming, which often resulted in mixing different levels of analysis (e.g. using a test of general programming ability to predict debugging skills). Another reason may be the "well-known failure of a manager's performance ratings to accurately measure individual performance" [3]. Nevertheless, there are undeniably huge differences in performance among programmers when confronted with the same task and there must be something to account for this phenomenon.

According to [3], [4] and [5] the following factors should be considered in order to predict individual programming performance:

- Individual knowledge about the task, the system architecture and the system interface
- Practical experience
- Intellectual aptitudes
- Mental abilities
- Cognitive style (see also [6])
- Motivational structure (especially intrinsic motivation) (see also [7])
- Personality characteristics (e.g. self-confidence) (see also [8])
- Behavioral characteristics

However, only few authors make concrete statements about the degree to which one of these factors affect the performance of the programmer and how they are interrelated. This may be due to the usually very restricted set of affecting variables studied in the same set of data.

Considering previous results [5] as well as ease of measurement, it seems that past practical programming experience is the most promising of the assumed influencing factors.

This and other human factors are considered in the Adaptive Test Management System (ATMS) [9] which is presented in this paper. It automatically prioritizes available test cases for each test cycle with the goal to get a prioritization order of the test cases which increases the probability to find more faults in the available test time. In our approach the ATMS is realized using autonomous software agents [10]. Thereby, each test case and each software module of the software system is represented by a software agent. The software agents prioritize the test cases from their local perspective and in cooperation with the other agents, which reduces the complexity of the prioritization problem.

In Section 2 the idea of the ATMS is described in more detail. Section 3 discusses the accounting for human factors for test case prioritization. In Section 4 we present our agent-based prioritization concept. Section 5 concludes this paper.

2 Adaptive Test Management System

The task of the Adaptive Test Management System (ATMS) is the prioritization of test cases for the system test. The goal is to get a prioritization order that allows finding more faults in the available test time.

The ATMS prioritizes the available test cases for each test cycle considering changing information for each test cycle. This is why the system is called "adaptive". Due to the prioritization values the test manager decides which test cases he/she executes in the current test cycle. The scheme of the ATMS is depicted in Fig. 1.

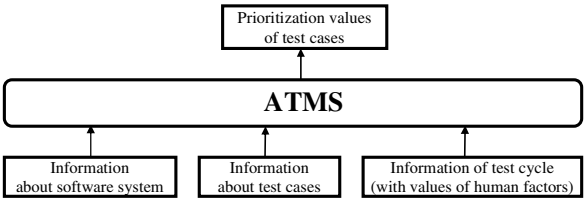

Fig. 1. Scheme of ATMS

The ATMS has three kinds of input. First, it gets information about the software system. The software system consists of several software modules, as shown for a simple example in Fig. 2.

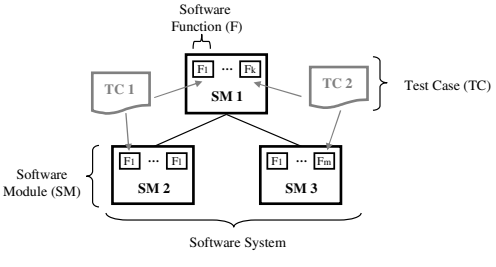

Fig. 2. Software modules, software functions and test cases

The input is a formalized architecture model out of which the ATMS derives information about the existing software modules and their functions, about the calling relations between the software modules and about other properties of the software modules.

Second, the ATMS gets information about the available test cases. This means, the test cases are already specified and can also be even implemented as automated test

scripts. In detail, the ATMS gets the information about which test cases exist, information about different properties of the test cases and about the functions of a software module which a test case executes while running. As depicted in Fig. 2, a test case can execute different functions of different modules during a run. All the information about a test case is retrieved out of a test management tool and its database.

The third and last kind of information is the changing information for each test cycle. The considered information is:

- Information from the test, e.g. information about faults that have been found and their mapping to the software modules.
- Information from the development, e.g. information about software changes, i.e. which function in which software module was changed.
- Information about the values of human factors.

All this information for each test cycle is retrieved out of the databases of the test, fault and change management tools or provided by the test manager.

The outputs of the ATMS are prioritization values for all available test cases. For each test case a value between 0 and 10 is determined as prioritization value. A value of 10 implies very high priority whereas a value of 0 implies very low priority.

After describing the scheme of the ATMS with its goal, task, inputs and outputs, we discuss the account for human factors for test case prioritization in the next section.

3 Accounting for Human Factors for Test Case Prioritization

One of the biggest challenges for the prioritization of test cases using human factors – besides the selection of appropriate influencing factors – is the operationalization of these variables. It is essential that the classification of each programmer is as quick and easy as possible. Ideally, it can be integrated into the everyday work and does not require permanent or recurrent psychological testing.

Bearing this in mind, the most promising of the factors mentioned in the introduction is past practical programming experience. It has shown to have significant effects on programming performance and it can be easily quantified. At the moment it is provided to the ATMS by the test manager judging the experience of the developer with regard to the implementation work he/she has to do. According to Curtis [3], that seems to be rather unproductive. In adoption of the approach described in [5], we plan to prove an experience rating according to the qualification background of the developers (master, bachelor, technician etc.). Moreover, the years of relevant professional experience of the developers will be included. We hypothesize, that the more practical experience (gained during an apprenticeship and/or on the job) a programmer has, the fewer are the faults in the developed software system.

In addition to their practical experience, the workload of the developers implementing a software module of the software system will be taken into account. Currently, it is derived from information about software changes made by a developer, i.e. from the amount of changes, the effort for a change and the time used for implementing the changes. Thereby, the time, when the changes were made, also plays a role, e.g. were the changes made during the day, in the evening or even at the weekend. A major disadvantage of this approach is the fact that it is more a measure of task

difficulty than a measure of workload. That is, it does not account for individually varying coping of the developers when confronted with the same task. In human factors research three valid approaches for measuring workload exist: Performance on a secondary task, physiological reactions (e.g. heart rate variability) and self-reported measures. Because of the mentioned importance of application ease and integration into everyday work (i.e. low task intrusion) a self-assessment of the developers on a one-dimensional rating-scale as to their felt workload seems to be the most appropriate approach. Moreover, compared to the alternatives this method does not require particular equipment, is easily accepted by the developers and produces very low costs. We assume that higher self-reported workload is associated with poorer performance of the programmer. However, one should bear in mind that workload is no static measure but is in a constant state of flux. Therefore, it has to be re-entered by the developers subsequent to the completion of every single programming task and before a new test cycle.

Measuring the other human factors that have been mentioned in the literature – i.e. intellectual aptitudes, cognitive style and motivational structure– is much more complicated. First, most of these factors identify merely the general category without any specification. However, the possible measurement depends greatly on what exactly is meant by the term. Second, some factors require an elaborate testing carried out by psychologists. For that reasons, the mentioned factors will only be included in the final testing procedure after their practical impact on prioritizing test cases for the software system test has been experimentally verified.

After discussing accounting for human factors for test case prioritization, we describe the realization of the ATMS in the next section and explain the functionality of the ATMS.

4 Agent-Based Test Case Prioritization

As described in the first section, the prioritization of the test cases for the system test is executed from the perspective of the test cases and software modules of the software system. Therefore, test cases and software modules are represented by software agents. These software agents realize the ATMS. The software agents together determine the priority of each test case using different information. We call this priority from now on the “global” priority. This means, this is the priority of a test case in comparison to all other test cases. For determining this global priority, the software agents representing the software modules, called the SM-agents, determine a test importance of a software module using information about human factors amongst others. The software agents representing the test cases, called TC-agents, determine local priorities of a test case. Combining test importances of the software modules and local priorities of the test cases, the global priority of a test case is calculated. These steps will be described in more detail in the next sections.

4.1 Perspective of the SM-Agent: Determination of the Test Importance of a Software Module

A SM-agent is generated for each software module specified in the architecture model. The goal of the SM-agent is to increase the number of found faults in the available

test time. Therefore, its responsibility is to determine the test importance of the software module. This test importance indicates the probability with which faults can be found in the system by testing this software module. For determining the test importance the SM-agent retrieves different information from its environment, i.e. from other SM-agents, from the architecture model, from the available databases and from the test manager.

In the current configuration of the ATMS the following information is used for determining the test importance of a software module:

- Complexity of the software module: A more complex software module indicates a high probability for faults in the software module. The value for complexity has to be provided by the developers in the architecture model.
- Number of faults in the software module in the previous test cycle: Faults in a software module indicate a high probability for more faults. The value for the number of faults is retrieved out of the database of the fault management tool.
- Number of changes of the software module: A software module which was changed indicates a higher probability for faults in the software module. The number of changes of the software module is retrieved from the database of the change management tool.
- Number of changes in other modules in which functions are called: Impacts of faults due to changes in other modules can be found in related software modules. Therefore, the test importance of software modules which call changed software modules is increased. The SM-agent is informed by the other agents about changes of other software modules.
- Values of human factors which are currently considered:
  - Workload of the developer of the software module: A developer with a bigger workload is assumed to make more faults. Thus, a higher workload indicates a higher probability for faults in the software module. Currently, the workload is derived from the number of changes, the effort for a change and the time used for implementing the changes. All these values are retrieved from the database of the change management tool.
  - Experience of the developer of the software module: A developer with less experience in a certain task is assumed to make more faults. Thus, a lower experience of the developer indicates a higher probability for faults in the software module. Currently, the experience value is given by the test manager.

The SM-agents determine the test importance of the software module using this information.

Additionally, they need knowledge for evaluating the information. As shown in [11], the knowledge of the test manager is imprecise and can be made available as verbally formulated rules, e.g. “If the workload of the developer of a software module is high, the test importance of the software module is very high”. Furthermore, the determination of the test importance is not a YES/NO decision. It is based on indicators and always afflicted with uncertainty. Therefore, we decided to use fuzzy logic to specify the rules for the determination of the test importance. Fuzzy logic offers a good possibility to reproduce such human evaluation standards, thinking patterns and conclusions, as shown in [12].

Every SM-agent determines the test importance for the software module it represents autonomously by an internal fuzzy-unit, as depicted in Fig. 3.

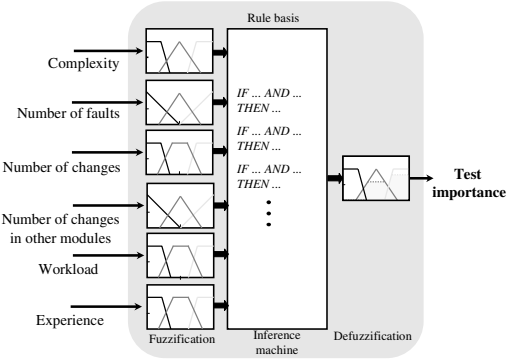

Fig. 3. Fuzzy-unit for determination of the test importance

First, the input data is fuzzified, this means crisp values are transformed into grades of membership for linguistic terms (e.g. low, middle, high) of fuzzy sets. A membership function is used to associate a grade to each linguistic term.

The fuzzified input data is then combined together by the fuzzy inference machine with rules of the type "if … and … then…" For example, rules for the workload are specified like this:

- If (workload is high) then (test importance is very high)
- If (workload is middle) then (test importance is high)
- If (workload is low) then (test importance is low)

Similarly, the rules for all input data are described. Since a crisp value of input data can have several grades of membership for different linguistic terms, no unwarranted precision is introduced in the prioritization.

Finally, the test importance is determined through defuzzification. The resulting test importance value has a range from 1 (very low) to 10 (very high).

After considering the perspective of the SM-agents in this section, we want to consider the perspective of the TC- agent in the next section.

4.2 Perspective of the TC-Agent: Determination of the Local Priorities of a Test Case

For each test case specified in the database of the test management tool a TC-agent is generated. The goal of the TC-agents is, as it is for the SM-agents, to increase the number of found faults in the available test time. Therefore, its first capability is to determine the local priorities of a test case with respect to each software module, which the test case executes when it runs. Thus, the local priority (LP_{xi}) is the priority of a test case "x" with respect to a software module "i" (see Fig. 4).

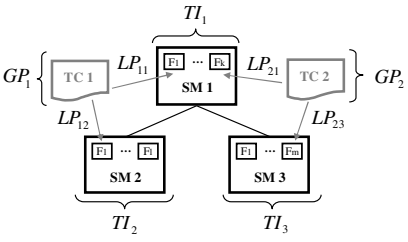

Fig. 4. Test importance, local priority and global priority

The local priority indicates the probability with which the test case can find faults in the related software module. A test case can have different local priorities with respect to different software modules. For determining the local priorities the TC-agent retrieves different information from its environment, i.e. from the available databases and from the SM-Agents.

In the standard configuration of the ATMS the following information is used for determining the local priority of a test case with respect to a software module:

- Number of found faults in the software module by the test case in average: A higher number indicates a high probability with which the test case can find faults in the related software module. The value for the number of found faults is retrieved from the database of the fault management tool.
- Number of changed functions in the software module which are executed by the test case: When a test case executes changed functions in a software module, this indicates a higher probability with which the test case can find faults in the related software module. In order to derive this information the TC-agents ask the related SM-agents if any software modules have been changed. In case of changes, the TC-agent gets information about the changed functions from the SM-Agents.

With this information each TC-agent determines the local priorities of its test case autonomously for each software module which its test case executes. As the rules for determining the local priorities have the same characteristics as for determining the test importance, we also decided to use fuzzy logic. Thus, as for the test importance, an internal fuzzy-unit is used for the determination of the local priorities in each TC-agent.

The resulting values for the local priorities have a range from 1 (very low) to 10 (very high) as for the test importance.

At this time, the software agents have determined the test importances (TI_i) of the software modules "i" and the local priorities (LP_{xi}) of the test cases "x" in parallel. The next step is to determine the global priority (GP_x) of each test case "x", as depicted in Fig.4. This is described in the next section.

4.3 Perspective of the TC-Agent: Determination of the Global Priority of a Test Case

As mentioned above, the global priority is the priority of the test case in comparison to all other test cases. It is also determined out of the perspective the TC-agent. The

TC-agent determines the global priority based on information about the local priorities and the test importance. The TC-agent has the information about the local priorities. The information about the test importance of the software modules, which are executed by its test case, is requested by the TC-agent from the corresponding SM-agents.

The global priority is calculated by the TC-agent as the weighted average value of the local priorities:

$$GP_x = \frac{\sum_{i=1}^{n} LP_{xi} TI_i}{\sum_{i=1}^{n} TI_i} \qquad \begin{array}{l} \text{Weighting factor} \\ TI_i \geq 0 \\ \text{n = number of software modules} \end{array} \tag{1}$$

This means the local priorities are weighted by the test importances and then added together. They are divided by the sum of the test importances for scaling. In the example of Fig. 4 the global prioritiy GP_1 is calculated as follows:

$$GP_1 = \frac{(LP_{11} * TI_1) + (LP_{12} * TI_2)}{TI_1 + TI_2 + TI_3} \tag{2}$$

Considering the global priorities the test cases can be compared and ordered. The test manager can decide up to which priority value test cases should be executed in a test cycle.

5 Conclusion

This paper presented an approach for prioritizing test cases by software agents for software system test considering human factors. Software modules of the software system and test cases are represented by software agents. Using rules based on fuzzy logic the agents prioritize the test cases for each test cycle. The reached prioritization order allows finding more faults in the available test time.

A prototype of the ATMS was implemented using the agent development framework JADE and the fuzzy logic package jFuzzy Logic. The implemented prototype is connected to the databases of the test management tool Testopia and the fault and change management tool Bugzilla. The architecture model is currently provided to the ATMS in XML format. The experience of a developer is provided to the ATMS by the test manager using the GUI. Currently, we are evaluating the prototype using data from a finished software project of a company from the automotive industry in order to compare which test cases were executed without ATMS and which would be executed with ATMS. The first evaluation results are very promising. After software changes the ATMS prioritizes the test cases in a way that many test cases with very low priority can be excluded. Without ATMS almost all of such test cases were repeated after the software changes but did not find any faults. Currently, we are continuing the evaluation with more data from a bigger software project.

Furthermore, we will extend the consideration of human factors for the prioritization of test cases by selecting further human factors due to experiments and by improving the operationalization of the human factors.

References

1. Illes, T., Pohlmann, H., Roßner, T., Schlatter, A., Winter, M.: Software –Testmanagement. Heise Verlag, Hannover (2006)
2. Rothermel, G., Untch, R.H., Harrold, M.J.: Prioritizing Test Cases For Regression Testing. IEEE Transactions on Software Engineering 27(10) (2001)
3. Curtis, B.: Five Paradigms in the Psychology of Programming. In: Helander, M. (ed.) Handbook of Human Computer Interaction, Elsevier Science B.V., North-Holland (1988)
4. Carroll, J.: Mental Models in Human-Computer Interaction. In: Helander, M. (ed.) Handbook of Human Computer Interaction, Elsevier Science B.V., North-Holland (1988)
5. Friedrich, D., Vogel-Heuser, B.: Evaluating the Benefit of Modelling Notations for PLC-Programming Quality. In: Human Computer Interaction (HCI), Las Vegas(2005)
6. Bishop-Clark, C.: Cognitive Style, Personality, and Computer Programming. Computers in Human Behavior 11(2), 241–260 (1995)
7. Bergin, S., Reilly, R.: The influence of motivation and comfort-level on learning to program. In: Romero, P., Good, J., Acosta Chaparro, E., Bryant, S. (eds.) 17th Workshop of the Psychology of Programming Interest Group, Sussex (2005)
8. Bergin, S., Reilly, R.: Programming: Factors that Influence Success. In: Proceedings of the thrity-fifth SIGCSE technical symposium on Computer Science Education, St. Louis (2005)
9. Malz, C., Jazdi, N.: Agent-based test management for software system test. In: Malz, C., Jazdi, N. (eds.) IEEE International Conference on Automation, Quality, Testing, Robotics (2010)
10. Wooldridge, M., Jennings, N.R.: Intelligent Agents: Theory and Practice. Knowledge Engineering Review, vol 10(2), 115–152 (1995)
11. Xu, Z., Gao, K., Khoshgoftaar, T.M.: Application of Fuzzy Expert System In Test Case Selection For System Regression Test. In: Xu, Z., Gao, K., Khoshgoftaar, T.M. (eds.) IEEE International Conference on Information Reuse and Integration, pp. 120–125 (2005)
12. Avineri, E., Köppen, M., Dahal, K., Sunitiyoso, Y., Roy, R.: Applications of Soft Computing - Updating the State of the Art. Springer, Heidelberg (2009)