Supporting the Regression Test of Multi-Variant Systems in Distributed Production Scenarios

Sebastian Abele and Michael Weyrich
Institute of Industrial Automation and Software Engineering
University of Stuttgart, Stuttgart, Germany
{Abele, Weyrich}@ias.uni-stuttgart.de

***Abstract*—Modern manufacturing systems based on cyber-physical systems with a growing amount of software allow for frequent updates and reconfigurations to adapt the systems to their usage in the production. A diverse system environment arises even for similar or equal subsystems based on the same platform used at different locations. A major challenge for such systems is the regression test after changes or updates. The resources for the regression test, in a dedicated test environment or deployed to the assembly lines, are limited. To plan the test in the best possible way, a lot of dependencies, relationships and experiences from former tests and tests from other locations have to be considered. This paper describes the research on an assistance system which supports the planning of the regression test in distributed manufacturing scenarios by combining manual modeling with automated data processing to improve the planning of the regression test. Therefore the system calculates a cross-location test progress and suggests a prioritized test case sequence.**

I. Introduction

High volume or localized products, e.g. cars, are often produced at different locations. Such locations might be different assembly lines in one or in different factories. The different assembly lines are at least partly equipped with the same production and quality assurance (QA) systems. Often the same system platform is used for all locations. Product features and environmental conditions differ in different locations, thus each system is adapted and configured to run in a specialized use case. The increasing spread of software and cyber-physical systems facilitates frequent updates and reconfigurations. Fig. 1 depicts such a production scenario with the example of a QA-system. QA-systems are exemplary for software-intensive systems based on a platform.

Updates and changes in the particular systems need to be tested before they can be rolled out. Module tests can be executed isolated but the system test depends on the actual production use case and the environment at the assembly lines. Nevertheless, the time and resources for the system test in the use case is limited since the production must not be interrupted for a long time.

Usually, the updates are tested by the supplier and by the manufacturer in dedicated test scenarios with regression tests. Then they are handed over to the maintenance engineers at the different locations. The engineers perform further tests based on their knowledge and experience without or with only few cross-location coordination. To improve the test effectiveness of that regression test, we propose a test management support system which allows for cross-location test progress monitoring and test case prioritization that incorporates test runs from other locations.

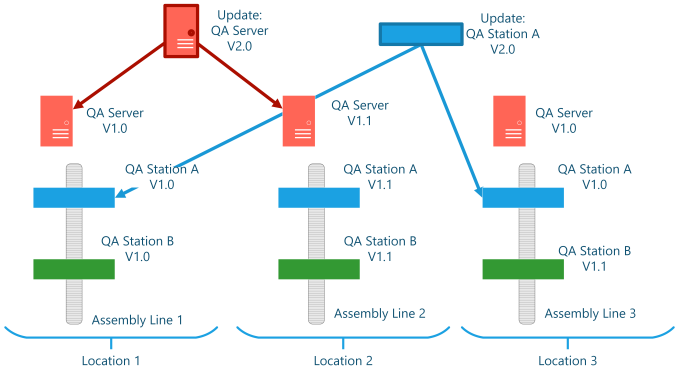

Fig. 1. Exemplary scenrio of a multi-variant quality assurance system

II. Test Case Selection and Prioritization

Test case selection and prioritization methods are used to find the most important test cases for the available execution time. Test case selection methods 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 and prioritization methods can be found in [1] and [2]. Prioritization methods sort the test cases by their importance to increase the test progress as fast as possible. A test case sequence based on prioritization can be interrupted at any time with the maximal benefit possible to the time of interruption.

Every test case prioritization method can also be seen as a selection method when the test cases with the highest priority are chosen. Test case prioritization methods are described in e.g. [3], [4], [5].

The selection and prioritization methods can be categorized by the data they evaluate. Methods which analyze the source code directly to optimize the coverage of changes, e.g. [6], are called white-box methods. Direct access to the source code and its different versions is necessary for these methods. The necessity of source code access limits those methods to software components which are developed by the manufacturer himself. Hardware components and components delivered by suppliers require the usage of other methods.

Black-Box methods have been proposed that use data from the test itself like execution- and fault-histories of test cases to prioritize them, e.g. [7], [8] and [9]. Some other methods are especially concentrating on information about requirements, identifying them as the most important entities for the test case prioritization, e.g. [10] and [11]. Black-Box methods are applicable for both, software and hardware modules and therefore used for the here presented support system.

The benefit of a test case is usually seen as its ability to find as many faults as possible. In order to achieve the best results, other aspects like monetary costs must be considered too [12], [13]. A common approach is to incorporate the fault-proneness or fault probabilities of modules or functionalities into the test case prioritization, e.g. [9], [14] and [15]. Based on the coverage of fault-prone modules, the test cases are selected and prioritized accordingly.

For multi-variant systems, there is another dimension to consider: Test results from other variants or locations give hints about already tested components and interfaces. The test for a particular system can be optimized by covering unique components and interfaces in that system.

III. Concept of the Test Management Support System

As a basis, the support systems needs information about test cases and about the system-under-test. This information is currently modeled manually by the maintenance engineers especially for the support system (see sections A, B and C). Later it is planned to import the information from dedicated design and test tools. With the data available, the support systems calculates the test progress and a test case prioritization (see sections D and E).

A. Test Step and Test Case Management

Often, test cases on the system test level describe complex processes with many different steps to be performed. In various test cases, the same test steps have to be executed. Test steps are single activities that can be executed separately. A test step is for example the stimulation of a sensor or the establishment of a network connection to a server. By using the same test steps, test cases are partly redundant.

The data model of the support systems differentiates in those two levels: Test cases and test steps. A test step can be modeled as part of several test cases (see Fig. 2). This allows the support

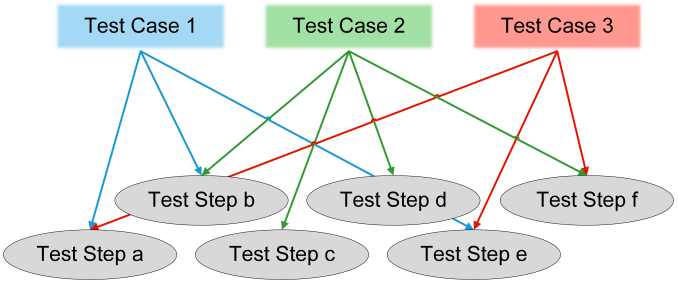

Fig. 2. Test Cases and Test Steps

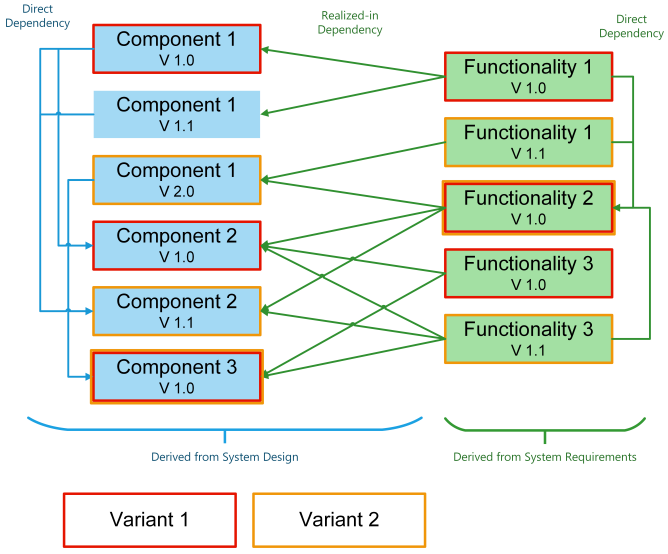

Fig. 3. System Model used by the Support System

system to compare two test cases in terms of test step coverage. If all test steps of a test case were already executed as part of other test cases it might not be necessary or is only necessary with a low priority to execute this test case. Nevertheless it is assumed that a test case covers more than the sum of the test steps contained in the test case. A test case execution of that test case could be beneficial. However, the support system is selecting and prioritizing test cases with the goal to maximize test step coverage.

B. System Representation to Determine Test Coverage

A main challenge for test case prioritization for regression testing is the identification of fault-prone and important to test system parts. Those system parts are usually determined by evaluating the changes is the system since the last test run. The basis for our support system is a simplified system model of the system-under-test which consists of a architectural system representation in system components and a functional representation in functionalitites. Each component and each functionality is versioned. Components and functionalities have dependencies which are also part of the system model. The actual system variants which are used in the production are modeled by selecting the version of each component and functionality which is deployed in that variant. The system model is depicted in Fig. 3. For each component, functionality or test case, additional parameters are stored which help to determine the test case priorities. The parameters are shown in Fig. 4. Dependencies and parameters can change from one version to another. Even with only few components and functionalities a complex dependency network arises which needs to be considered for test case selection. It is one of the main advantages of the support system to be able to consider complex dependencies.

Currently, the model is managed manually by the test engineer. The support system provides functions like the automatic

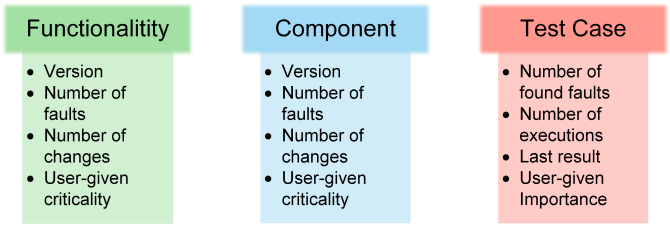

Fig. 4. Parameters for functionalitites, components and test cases in the data model

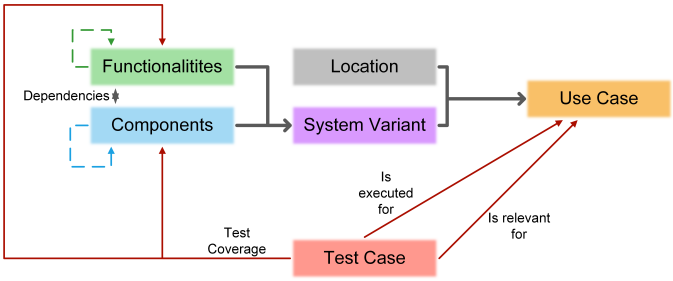

Fig. 5. Data Model

generation of a new version of a component or function with the same dependencies like the older component based on release notes which are provided with a new version. The engineer only has to maintain the dependency changes of new versions.

C. Connecting Test Cases and the System Model

To get a representation of the actual production use cases, the data model is complemented with the actually used systems at the production locations. Therefore the data entity *Use Case* is introduced and added to the system model. A use case represents an actual system at a special location. Test cases are executed for one of these use cases.

The test cases themselves have three different connections to the system model. First, they have a test coverage. The test coverage determines which components and which functions in which versions are tested by the test case. This information is independent from actual variants and use cases. To consider the real use cases, it is stored for which use case a test case is relevant. With the relevance different configurations or different usages of the same system can be considered. Only relevant test cases are used to determine the test coverage and to test a system used in a use case. The third connection for test cases is the execution of a test case. Test cases are executed for particular use cases. With this information, the support system is capable to analyze if a test case already has been executed for the same system in another use case and thus lowering the priority to execute this test case again.

D. Test Progress Analysis

The test progress analysis determines for each component and each functionality how good they are currently tested incorporating test results from the last test runs and information about changes since those test runs. The determination allows the test engineer in the assessment where to spend the most test effort and when to stop the test and bring the system into operation. Additionally, it supports the engineer in finding coverage gaps.

The procedure of determining the test progress for components and functions is identical. In the following the process is only described for components. Functionalities are treated the same way with the same parameters but with different dependencies which allows another view to the system.

The following parameters are used for each component to calculate the test progress:

- Untested changes
- Untested changes in dependent components
- Ratio of executed test cases to available test cases for the current use case
- Ratio of executed test cases to available test cases for other use cases

The parameters related to use cases are use case specific because each use case might have different relevant test cases. Thus, the test progress value for each component is use case specific as well.

In the current approach of the support system, the parameters are just summed up to calculate the test progress value. The actual calculation method is still part of the ongoing research. It has to be figured out if the parameters have to be weighted and in which circumstances the transferred results of other use cases are trustful.

E. Test Case Prioritization

Based on the current test progress, the next test cycle has to be planned. This task is supported by the test management system with an automatic prioritization of already available test cases.

To calculate the test case priorities, first the test importance of the components has to be determined. This is done by extending the test progress with the following parameters which are indicators how important it is to test a component:

- Number of already found faults in the component
- User-given criticality estimation
- User-given complexity estimation

As a second step, the test cases are evaluated to find the most appropriate to test the most important components. Therefore the following parameters are summed up for each test case:

- Number of already found faults by the test case
- Not yet covered test steps
- User-given test importance estimation

Like with test progress calculation, the influence of the parameters has to be investigated in further research.

IV. Prototypical Implementation

The test management support system has been realized as a web-based prototype. The usage of web technology allows a location-independent access to the support system and also access from mobile devices. The system has been divided into four components that are depicted in Fig. 6. The front-end has

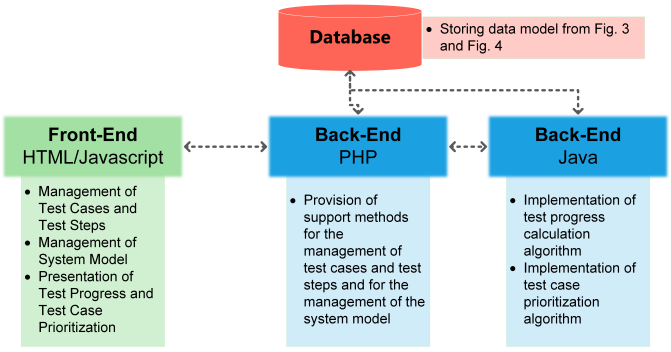

Fig. 6. Architecture of the prototypical implementation of the support system

	Use Case 1 Location 1	Use Case 2 Location 2	Use Case 3 Location 3	...
QA Server	V 2.0	V 2.0	V 1.0	
QA Station A	V 2.0	V 1.1	V 2.0	
QA Station B	V 1.0	V 1.1	V 1.1	
...				

Fig. 7. Presentation of the test progress as a test management dashboard

been realized in HTML and Javascript. It shows the graphical user interface. The back-end is split into a PHP and an Java part and a data base. The data base stores the data model from Fig. 4 and Fig. 5. the PHP back-end provides supportive functions to manage test cases and the system model. The Java back-end realizes the calculation algorithms for the test progress calculation and the test case prioritization.

Fig. 7 shows the result of the test progress calculation for the scenario shown in Fig. 1. A table based dashboard presentation of the results has been chosen. The rows represent the system structure represented by the components of the system. The columns contain the production use cases. The individual cells show the version of the component contained in the use case and the current test progress which is encoded in a color ranging from green to red.

To evaluate the prototype, data following the scenario of Fig. 1 has been generated in cooperation with experts for automotive QA-systems. With this data it could be shown that the test progress dashboard shows correct and helpful results as assessed by the experts. The test case prioritization also provides traceable and good results that have been assessed by the experts. Nevertheless a long-term evaluation with multiple update cycles and the usage of the prioritized test case lists still has to be done to evaluate the whole method.

V. Conclusion and Further Work

A test management support system to support the regression test in distributed industrial production scenarios is presented. The system provides support functions to determine the current test progress and to generate a prioritized list of test cases to improve the test progress of a particular system variant used in a special production use case. Therefore the system uses a system model and a test case model. These models are updated with system changes and test case executions. To provide the support function, the system evaluates this model. The main functionality of the test progress determination is the evaluation of dependencies and coverage given in the model. The test case prioritization then generates a list of test cases intended to maximize the test progress with few effort.

Our current research focuses on the application of the concept and the prototype to real production scenarios performing a long-term evaluation. This allows to optimize the parameters and assumptions which were made in the concept and to investigate methods to better incorporate expert knowledge and experience. Furthermore, methods are searched which allow more automation in the provision of the needed data for the support system.

References

[1] E. Engström, P. Runeson, and M. Skoglund, "A systematic review on regression test selection techniques," *Information and Software Technology*, vol. 52, no. 1, pp. 14–30, 2010.

[2] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[3] S. Elbaum, A. Malishevsky, and G. Rothermel, "Test case prioritization: a family of empirical studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182, 2002.

[4] Z. Li, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization," *IEEE Transactions on Software Engineering*, vol. 33, no. 4, pp. 225–237, 2007.

[5] G. Rothermel, R. Untch, Chengyun Chu, and M. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.

[6] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos, "Time aware test suite prioritization," in *Proceedings of the 2006 international symposium on Software testing and analysis*. ACM, 2006, pp. 1–12.

[7] B. Qu, C. Nie, B. Xu, and X. Zhang, "Test case prioritization for black box testing," in *31st Annual International Computer Software and Applications Conference - Vol. 1- (COMPSAC 2007)*. IEEE, 2007, pp. 465–474.

[8] H. Park, H. Ryu, and J. Baik, "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*, 2008, pp. 39–46.

[9] C. Malz, N. Jazdi, and P. Göhner, "Prioritization of test cases using software agents and fuzzy logic," in *IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST)*, 2012, pp. 483–486.

[10] J. Karlsson and K. Ryan, "A cost-value approach for prioritizing requirements," *IEEE Software*, vol. 14, no. 5, pp. 67–74, 1997.

[11] H. Srikanth, L. Williams, and J. Osborne, "System test case prioritization of new and regression test cases," in *International Symposium on Empirical Software Engineering*, 2005, pp. 62–71.

[12] M. Kumar, "Towards multi-faceted test cases optimization," *Journal of Software Engineering and Applications*, vol. 04, no. 09, pp. 550–557, 2011.

[13] A. Malishevsky, G. Rothermel, and S. Elbaum, "Modeling the cost-benefits tradeoffs for regression testing techniques," in *Proceedings of the International Conference on Software Maintenance*, 2002, pp. 204–213.

[14] N. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 675–689, 1999.

[15] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller, "Predicting faults from cached history," in *Proceedings of the 29th International Conference on Software Engineering*. IEEE Computer Society, 2007, pp. 489–498.