An Efficient Specification-Based Regression Test Selection Technique for E/E-Systems

Ralf Nörenberg, Anastasia Cmyrev, Ralf Reißing*, Klaus D. Müller-Glaser**

Daimler AG
G025-BB
71034 Böblingen
[ralf.noerenberg;
anastasia.cmyrev]
@daimler.com

*Hochschule Coburg
96450 Coburg
reissing@hs-coburg.de

**Karlsruher Institut für Technologie
76131 Karlsruhe
klaus.mueller-glaser
@kit.edu

Abstract: Regression testing, a methodology originally developed for software development, is used to revalidate a (software) system in-between release cycles after having implemented changes. In practice there is always limited time to perform a full retest of a system; therefore a random/prioritizing-testing approach is often chosen to perform at least some regression testing. However, the lack of adequate regression testing can lead to exposed errors in untested parts of the system during production or field usage, which may have severe consequences. In order to improve the efficiency of regression testing, so far many approaches were proposed. They intend to select only test cases which cover parts of the system that contain the implemented changes as well as parts that are possibly affected by the change. Unfortunately, most techniques are only available on software level requiring extensive knowledge of the source code, and typically use some additional representation of the software such as a software architecture model. However, in practice, especially within automotive embedded system development, available system models or source code strongly vary in type or design or may even be inaccessible. In order to provide an efficient regression test selection methodology, we propose a novel and light-weight approach primarily based on system requirements and their association with test cases. In addition, substantial similarities between challenges and objectives of regression test selection and product lines testing techniques are identified. Conclusions outline how a potential benefit in reducing overall testing efforts in product lines testing can simply be achieved by applying regression test selection techniques.

1 Introduction

Embedded systems are continually subject to changes during development for a variety of reasons: software bug-fixing, enhanced functionality or added features, changes in system communication or changes in hardware components. After changes are applied, the system needs to be tested to ensure it still behaves correctly in regard to its specification and that the modifications have not had an adverse impact on the quality of the given system. This testing activity is called regression testing.

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html

The most common approach to apply regression testing is to reuse parts of the test suite T_i, used to test a version S_i of the system, for testing the next (modified) version S_{i+1} of the system. Instead of rerunning all available test cases in T_i, so-called **R**egression-**T**est-**S**election techniques (RTS) select a subset $T_i´$ of T_i to test S_{i+1}. In order to improve the efficiency of regression testing, a methodology generally aims at minimizing $T_i´$, so that $|T_i´| <<| T_i|$. In the automotive industry, the general state of practice of regression testing is to rerun all available test cases of the previous system version.

The importance of research on the subject of RTS techniques is given, as they are very effective in reducing the cost of regression testing, which reports estimate consumes already as much as 80% of the overall testing budget and up to 50% of the cost of maintenance at the software level [12]. However, most academic research in the field of regression testing describing the process of a selective revalidation of a modified system so far has been exclusively focused on software-unit and white-box testing [10][3][9], therefore supplying specialized tools for the software level only. Consequently, developed techniques come with the restriction that they strongly rely on complete knowledge of the system structure, or even information about source code as well as test case traces. Furthermore, investigated issues of regression testing are commonly easy to address when handling a small software system, but suddenly become very complicated and costly when applied to complex real-world systems [12]. Thus, there are still many remaining challenges in making the regression test techniques useful in practice and getting the know-how transferred into industrial environments. Also, in industry, integration testing and black-box functional testing have been found to be most effective and useful – a subject which has hardly been investigated in regression testing [7].

This paper gives an overview of the state of research on RTS techniques in order to discuss existing difficulties and challenges of knowledge transfer into industrial environments in to the light of given evaluation criteria. After identifying the differing industrial demands, an adequate, light-weight methodology for regression test selection on system requirements for industrial functional embedded system testing is presented. Finally, an evaluation example and outlook is given.

2 State of Research and Challenges to Industrial Knowledge Transfer

The most prominent class of RTS techniques developed by academic research creates model representations of S_i and S_{i+1} and identifies the set of entities in the models that represent the changes from S_i and S_{i+1} [12].The techniques then use the identified entities to select the subset $T_i´$ of T_i that exercises those entities in S_i to test S_{i+1}. Some techniques use source code representations with entities such as statements [20], branches [18][1], control and data dependencies [11] or program paths [2]. Other, more related techniques to this contribution use activity diagrams [5] or software specifications as representations [6].

Chittimalli [6] uses instrumented code to obtain traceability to specified requirements, so that their automated allocation of test cases can be provided. Both, S_i and S_{i+1}, are then analyzed by an algorithm to identify varying entities within the code and match them to corresponding requirement instrumentations. The previously created traceability of requirements to test cases matrix then provides information on which test cases to select for regression testing. Chen [5] introduces a control-flow similar to "activity diagram" which also contains data-flow information about behavioural dependencies. Using change information on S_i and S_{i+1}, the technique selects test cases required for regression testing by identifying affected nodes and branches. However, this only applies on assumption that information on traceability between requirements and activity diagram and knowledge about test cases traces is manually implemented and therefore comprehensible.

Up to date, all techniques still focus on the software level requiring special white-box representations of the system, e.g. an instrumented source code or formal models [12]. This is one of the main reasons that only few available techniques were evaluated in industrial environments. Also, existing studies are strongly restricted to software unit testing focusing on controlled experiments such as a small set of programs, where as a result, it is not obvious that their findings can be generalized to programs other than the ones considered. Few empirical studies have been performed on real-world, large-scale systems or shown to be useful in practice. Several independent systematic reviews confirming this state of research can be found in [12][8][15].

In order to provide general evaluation criteria to compare and rate various RTS methodologies, a commonly used framework consisting out of the four categories inclusiveness, precision, efficiency and generality was introduced by Rothermel [17].

Inclusiveness is measure to describe to which degree test cases testing modifications in S_{i+1} are selected for regression testing. Objective is the achievement of 100% percent inclusiveness, the RTS then adequately replaces a retest-all methodology by providing a *“safe”* selection. A safe selection thus contains all modification-revealing test cases [17].

Precision is a measure to describe to which degree only the modification-revealing test cases are selected for regression testing. Objective to any RTS methodology is to not select any unnecessary test cases, as their execution requires additional resources. Therefore, a high precision may provide a substantial contribution to the efficiency of the RTS methodology. On the down-side, an upmost precision almost always comes with the drawback of endangering a safe selection of test cases.

The efficiency of RTS techniques is measured in terms of their space and time requirements. Where time is concerned, a RTS technique in general is more economical than the retest-all technique if the cost of selecting $T_i´$ is less than the cost of running the tests in T_i - $T_i´$. Space efficiency primarily depends on test history and analysis information a technique stores. However there’re no fixed criteria on efficiency measures.

The generality of a RTS technique describes its ability to function in a wide and practical range of situations. In an (automotive) embedded system context, generality should be provided to the extent, that the technique can be adapted to most projects, covering the variety of driver assistant systems up to light control systems.

In regard to the evaluation framework, the focus of academic research has so far strongly been put on developing a most precise technique by still remaining *safe* with test case selection. These aims were properly achieved and also good results were published, but certainly these techniques were not rated outstanding in efficiency and generality (see Fig. 1). Especially due to the lack in generality, most techniques are conceivably difficult to adapt and to integrate in given industrial testing structures or tool environments. The major issue of most techniques is to be identified as – due to industrial organizations mostly using system requirements representations – 1) the usage of detailed execution traces of (or the model representation of) S_i with T_i in order to select a subset T_i' of T_i to use for testing S_{i+1} and/or 2) a required fine-grained instrumentation of the source code itself. Both types of approaches are correct and sophisticated solutions, but are not suitable for real-life projects. Subsystems or components are not always developed within one organization and therefore mostly not accessible for the car manufacturer, or if so, at least not necessarily based on the same source code language or model type.

Technique:	**Evaluation Criteria**			
	Inclusiveness	**Precision**	**Efficiency**	**Generality**
Chittimalli	very high	very high	very low	low
Chen	very high	very high	very low	average
Industrial Requirements	high	average	very high	very high

Figure 1: Evaluation of RTS techniques and objective for the regression test technique under investigation.

This is the main reason why most organizations commonly use the more abstract approach of selecting test cases by their traceability to requirements and only few are reported to perform automated RTS by applying described techniques [12]. Thus, changes mapped to requirements can be properly traced to a selection of adequate test cases. A major issue of this approach is that test cases not being associated with changed or new requirements but being affected by the change may be missed, which is not *safe*. This problem was addressed by making additional information available for impact analysis (information about changes in interfaces or data and historical test case effectiveness), yet having to deal with the tradeoff of achieving less generality. On that account, many organizations still rerun T_i on S_{i+1} completely.

The basic challenge of developing a RTS technique for industry is to find a good balance between all evaluation criteria, thus having no greater drawbacks on inclusiveness and precision. The technique shall be general enough to be applied to any kind of embedded system available and be primarily based on system requirements and test cases. Despite generality, the main goal is to achieve an increase of efficiency in comparison to a full retest.

Consequently industrial interests lie especially within techniques that are easily applied to various projects, i.e. own a high generality, and next to that, of course, efficiently select necessary test cases by achieving a total saving in resources. In respect to given tradeoffs, a required more light-weighted analysis of system dependencies will therefore most likely result in a lower inclusiveness and precision of test case selection. This may lead to the technique not being *safe* as well. However, regression testing in an embedded system environment is only to be used for increasing testing efficiency within inner release cycles. For full release versions, it is still required to rerun a complete retest (see ISO 26262 [13] standard for automotive verification processes). The whole process of testing therefore is *safe*. Thus, achieving a *safe* selection of test cases is certainly still desirable, but should not lead to major drawbacks within efficiency and generality.

3 Towards an Efficient Specification-based RTS Technique for E/E-Systems

Embedded systems (E/E-systems) typically consist of a number of functions representing the entire functionality of a system. This system functionality is mainly provided by software which is implemented into a main component. Several cross-linked components contributing to the realization of the system functionality are regarded as an E/E-system. E/E-systems are designed and tested according to the V-Model [4], thus the verification process includes various test levels, namely software-, component- and system-in-the-loop test platforms.

The state of practice representation of an E/E-system within the automotive industry is given by a system specification and several component specifications. Both typically contain requirements written in natural language. The system specification describes the system functionality and therefore is to be considered as its central representation. Component specifications define functionality provided by each component in order to realize the system functionality (please note that hardware aspects are not in focus of this discussion). System requirements to be verified via testing are linked to at least one test case being an element of a separate documentation named system test specification. A system test specification typically contains all test cases derived for the system, thus include test cases covering multiple if not all test levels (software is commonly provided and (unit-) tested by third parties only and therefore may often not be available to the car manufacturer.

In order to develop a methodology for black-box functional RTS, the system specification and the system test specification have to be considered as fundamental, available basis of information. The concept for regression test selection on system requirements contains three aspects:

1) The basis for a successful application of the regression test strategy is enabled a standardized test strategy. Next to the required ISO 26262 conformance of the test specification, the strategy assures the integrity of the test specification as well as provides information about which key point each test case verifies within the test object. Knowledge about the latter will be mandatory to obtain a comprehensible methodology about which test cases may possibly be selected or omitted for regression testing.

2) A key requirement to enabling regression testing is the development of a structured specification framework (system representation). In order to enable ripple effect analysis in between system functions and contributing components, the system representation needs to contain supplemented dependency information which provides a networked grey-box view of the system.

3) The ripple effect analysis and the subsequent regression test selection are part of the third step, the RTS methodology itself. A new aspect provided by the presented technique is the identification and selection of test cases covering requirements on system level that have not been changed but possibly have been affected by the modifications.

The first two steps of providing a transparent and systematic documentation of a system are not only crucial for RTS, but for all selective testing strategies which aim for a redundant free reuse of test cases, the SPL testing techniques in particular.

4 Deriving A Standardized Testing Strategy

Up to date, testing strategies and their specific interpretation to derive test cases for verification of E/E-systems in development strongly depend on expertise of the testing personnel being in charge. Therefore test strategies do strongly vary within one organization, its various departments and even its numerous projects. This approach consequently leads to varying quality and integrity of obtained test specifications and its level of contained, hence available information. This state of practice is insufficient in order to develop a regression test selection methodology which naturally relies strongly on structured information frameworks.

In order to obtain an equal level of available information within the test specification a standardized methodology to derive project specific test strategies needs to be developed. The approach taken at Daimler is published in [16] and [14]. The resulting test strategy, achieved by a harmonized and universal verification plan, primarily consists of 8 standardized test goals, 7 test coverage criteria to evaluate the achievement of given test goals and 12 pre-set test methods, and provides two essential foundations for a development of a regression test selection methodology:

I. The integrity and comprehensibility of the derived test specification. Logically, selecting test cases out of a fragmentary test specification is not adequate. The test strategy therefore represents the approach to ensure that all aspects of testing were considered for test case derivation for a specific object under test.

II. Information about the type of given test cases. Every test case is annotated with the test goals it contributes to (e.g. proof of correctness of: functionality, safety mechanisms, diagnosis, interfaces, …). This, in most cases, allows a regression test selection based on the type of change. For example, a change in parts of the system describing diagnostic elements does not require to rerun a full test on all interfaces.

Next to a sufficient test specification, the RTS requires a standardized system representation to allow a ripple effect analysis within the system boundaries. This second step of the concept is described in the following chapter.

5 Requirements on the System Specification Framework

Analysis of numerous system specifications reveals, similar to the discussed situation of derived test specifications, that many deviations from the provided template and structures are spread within the organisation. Also, in order to apply a potential (automated) RTS methodology, the analysis shows the necessity to improve existing framework conditions towards being more stringent.

For the purpose to enable a grey-box view of the system, its functions and contributing components it is advisable to provide a traceable affiliation between defined test objects and associated test cases. As a result of the previously discussed testing strategy and the state of practice, for automotive systems where the “component-view” is more and more superseded by the “system-view”, the documentation of requirements based on the test object “function”, describing end-to-end user-experienced functionality (which is named function orientation), has proven to be best.

Three hierarchy levels are introduced: 1) The first level separates all functions f_n of a system (e.g. function "brake light" of system "outside light control"). 2) To obtain an even more detailed view, each function f_n is subdivided into its execution sequence aspects f_{nm}. This second hierarchy level contains the categories precondition, trigger, function execution and end condition (f_{nm}, $1 \leq m \leq 4$). 3) The third hierarchy level is the requirement level, therefore containing the requirements. With assorting requirements into this hierarchy, an association with the test object "function" and its corresponding test cases is obtained.

In order to provide a grey box view of the system being adequate for an RTS technique, it is mandatory to not only take the system functionality but also networking components, their contributing functions and the communication into consideration. For example, a targeted goal of the RTS technique on system level is to reveal potential impacts of changes in contributing components (which not necessarily have to be considered to be object under test) on given system functions. In order to provide an efficient and general system representation, which includes elements of component specifications, a sophisticated documentation framework is required. To address this problem, we propose the following documentation framework to associate requirements with:

System functions (f_{nm})

A system function specifies a function f_{nm} of the system functionality realized by the software implemented on the main hardware component C^0. Subordinated requirements specify the function, whereas n represents the identification of the function (e.g. brake lights, turning lights) and m the element of its execution sequence (see above). All functions f_{nm} are test objects and documented in the system specification (supreme level: "System Functions").

Function contributions k_l

A function contribution k_l specifies the interface between functions f_{nm} and one component function from the viewpoint of the system under test. Subordinated requirements specify the contribution k_l demanded by one or multiple system functions f_{nm}. Function contributions k_l are named identical to the corresponding component function c^j_{nm} and are documented in the system specification (supreme level: "Networking Component Functions"). Function contributions are subject to the system test of connected f_{nm}.

Component functions c^j_{nm}

A component function c^j_{nm} specifies the functionality provided by one of the participating electronic control units (ECU). Subordinated requirements specify the function, whereas j represents the identification of the ECU, n the function (e.g. realize lights) and m the element of its execution sequence (see above). All functions c^j_{nm} are documented in component specifications (supreme level: "Component Functions") and are generally not subject to verification of the system.

The separation of requirements in functions f_{nm} and c^j_{nm} create – mapped into a two dimensional graph – a matrix, with the y-axis representing a detailed view of the system function structure, and with the x-axis representing the component functions c^j_{nm}, an architectural layout of the system integrating all components c^j (Fig. 2). The function contributions are to be seen as the interfaces of a crossing point between f_{nm} and c^j_{nm}.

Although this framework provides hierarchy elements to assort requirements specifying different types of functions, it does not yet outline any communications dependencies between functions and the networking devices of a system. This, however, is an essential requirement to run a ripple effect analysis. We propose three kinds of dependencies to connect functions f_{nm}, k_l and c^j_{nm}: linkage, signals and parameters.

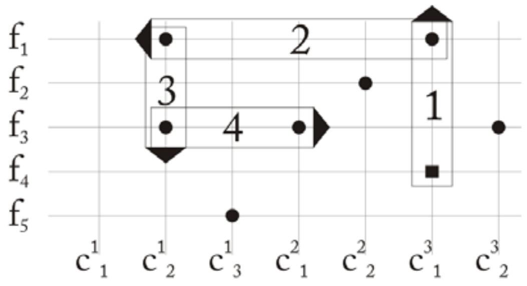

Figure 2: Schematic plot of a 2-dimensional matrix mapping the dependencies between system functions f_n and component functions c^j_n, with the dots representing the function contributions k_l. The arrows represent a path of the ripple effect analysis identifying possible impacts of a change (square) throughout a system.

Linkage

The implementation of links (a functionality provided by most requirements management tools) is typically used to create directed dependencies between two entities of any specification. Within this concept, this type of manual links (next to providing traceability between requirements and test cases) is used to illustrate dependency information in between system functions f_{nm}. These dependencies are to be set manually as they are typically not accessible on system level because they are integrated in the software based on its defined architecture. However, as requirements are grouped in an end-to-end user's manner, knowledge about the architecture and the execution sequence aspects of the functions easily allow creating dependencies between functions, their execution sequence elements (e.g. function execution of function f_1 is linked to trigger of function f_3) and even their requirements (although this is not recommended). Also, functions f_{nm} are linked to function contributions k_n which they require at given points of the execution sequence.

Signals

Signals are a entity of communication between components. They are defined as a signal name with a corresponding value. Signals are documented within the function contribution k_l and are attributed either as a "send"-signal or as a "receive"-signal (from the viewpoint of the system). Signals are derived from the car communication matrix and therefore represent fixed dependencies.

Parameters

Parameters are a set of values to control system function behavior. They are defined as a parameter name with a corresponding value and store data within the main component on which the system functionality is implemented. Parameters are documented within f_n and provide information about the function "reading" or "writing" its value during execution (in an automotive embedded system context these parameters usually remain static once set).

Implementing these types of communication information into the specifications offers a comprehensive representation of the systems functionality and its networking dependencies. Certainly, the implementation has to be performed precisely (not subject of discussion), which keeps documentation efforts low and allows the quite considerable amount of information to be managed. However, implementing this properly, the combination and merge of the previously discussed information data allows the construction of a sufficient system representation model to be used for ripple effect analysis, thus RTS.

6 The RTS Methodology

The existing system representation framework provides various entry points to map modifications to the specification: system function f_{nm}, function contributions k_l, component functions c^j_{nm}, each function execution sequence elements, single requirements, signals, parameters or even whole components. In order to be able to match bug-fixing changes to the software changes to the system specification, the following approach is chosen: Each change is to be mapped to at least one software module represented as a documented entity within a software specification (which is linked to the system specification) or documented element within a function f_n. However, some software modules might be subordinated to more than one system function; certain general software modules may even be associated to all of them. Once a change is located, automated ripple effect analysis can be applied using system dependencies in the system specification.

For ripple effect analysis within the defined documentation framework, we propose a four-step approach, which in regard to the discussed matrix, describes a "full circle" investigation on possible impacts of a change. The analysis thus consists of two vertical and two horizontal paths through Figure 2, each containing the three sub-steps impact identification, function analysis and impact analysis on further entities. The four steps of analysis (Fig. 2) will now be explained on basis of the entry point of the validation example, where a change to one of the components c^j is implemented and is traced throughout the system (f_{nm}) and back to further components c^j.

Step 1: Analysis of c^j_{nm} (vertical)

In this step 1) changes implemented to a component function c^j_{nm} are located, 2) further impacts on neighbor functions c^j_{xm} are analyzed and 3) possible impacts on function contributions k_l of the system in focus determined. Obviously, information data type "signal" has to be "sent" by c^j and "received" by k_l (thus f_a).

Step 2: Analysis of f_{nm} (horizontal)

In this step 1) impacts on system functions f_{am} are determined, 2) impacts on the execution sequence of the functions f_{am} analyzed and, according to the testing strategy, required test cases selected and 3) further possible impacts of the change on component contributions k_t (information data type "signal" equals "send") examined.

Step 3: Analysis f_{nm} and c^j_{nm} (vertical)

In this step 1) further impacts on other system functions f_{bm} as well as component functions c^k_{nm} targeted by previously (2.3) determined function contributions k_t are identified. Then 2) potential impacts on c^k_{nm} and neighbor component functions are analyzed (please remember that c^k_{nm} are not subject to verification and are just used for further ripple effect analysis, thus no test cases are selected) in order to 3) identify rebounding information transfer having an impact on further k_o.

Step 4: Analysis of f_m (horizontal)

In this step 1) any potential impacts on functions f_{bm} and functions f_{cm} targeted by k_o in step 3 will be analyzed and 2) adequate test cases selected. Consequently, the third step would be the connecting step in order to restart a new iteration with step 1. This however, has proven to be unnecessary, as during validation for the systems chosen, ripple effect analysis never exceeded step 4.

7 Validation and Outlook

The presented concept of an RTS technique based on system requirements is currently validated within a scope two separate but related systems, namely Outside Light Control (OLC) and Intelligent Light System (ILS), containing several f_{nm} each. Both systems individually depend on function contributions k_n provided by the same component named Light Control Device (LCD). The OLC contains various basic light functions such as "brake lights", "turning lights" or "high beam", whereas system 2, the Intelligent Light System (ILS), contains more advanced light functions such as "automatic high beam function". The component LCD contributes two basic functions c^j_{nm}, namely "control lights" and "dim lights", to both systems in order to realize their light functions. Within the scope of validation, one system at the time is the object under test. An overview of the extent of the systems and the component is given in Fig. 3.

Project/System	Functions	Total Test Cases
OLC	20	>1000
ILS	10	>600
LCD	2 in focus	N/A

Figure 3: Systems/components used for evaluation of the RTS technique

Each of the following validation examples analyzes the possible impacts of a change within the scope of all 3 systems/components by using the discussed concept described in this contribution. The validation results are presented in Fig. 4. Column "Possibly Affected Functions" gives the number of possibly affected functions identified for each system/component (a partly affected function counts as 1 as well). The column "Selected Test Cases" shows the number of test cases identified to verify a potential impact of the change and therefore are selected for regression testing.

Example		System / Component	Possibly Affected	Selected System Test Cases
1	Analysis of the impact of a change in "sending brake light status" of the electronic stability program (ESP) on ILS	OLC	3	6%
		ILS	2	18%
		LCD	-	-
2	Analysis the impact of a change in the component function "dim lights" of LCD on ILS and OLC	OLC	3	23%
		ILS	3	22%
		LCD	1	20%
3	Analysis of a parameter change within LCD and its consequences on LCD itself, ILS and OLC	OLC	2	6%
		ILS	6	44%
		LCD	2	100%
4	Analysis of a bug fixing change of light function "corner lights" of ILS and its impact on ILS itself and LCD	OLC	-	-
		ILS	2	<25%
		LCD	1	N/A
5	Analysis of a change in function "tourist mode" of ILS (inverting light functions left/right lane drive) on ILS itself	OLC	-	-
		ILS	5	18%
		LCD	1	N/A

Figure 4: Validation results of the RTS technique applied to a real-life project. Data marked with N/A is either out of focus or not completely available.

As a result, the presented RTS technique shows that regression testing is possible on an abstract system level and also contains great potential to reduce overall testing effort. Since informality of requirements does not affect the approach, the presented RTS technique is applicable to a wide range of projects in an embedded system environment. However, the approach requires domain specific information as it uses ISO 26262 refinements and is therefore specialized for an automotive environment.

Considering the amount of information required and used by this light-weight RTS technique, a very good efficiency is achieved. As increasing system complexity does not directly lead to an increasing system specification complexity and thus an information overflow, no major constraints are expected regarding this matter. Also, validation is performed using two large-scale systems.

Also, another great benefit of this technique is given as it does not only support verification of systems under development, but also applies to product line testing. The technique is capable of managing the exchange of whole components (equal to an impact on all c^j_{nm}) when, for example, integrating the system under test into a new car series while also the system functionality itself may remain unchanged.

Due to overall selection of requirements, first reviews of the inclusiveness of the technique lead to the conclusion that – assuming completeness and correctness of the system and test specification – the technique appears to be *safe* within the four iteration steps. However, due to the abstract level of analysis, no proof can be provided. Therefore the great benefit of the methodology primarily has to be seen reducing inner-release verification efforts it is advisable to run a full retest on at least the final release of the system.

The precision of the presented RTS technique is under current investigation. Current research is focused especially on how the knowledge about the test goals of each test case can further improve and optimize the selection of test cases. At this level a further potential increase in precision, thus efficiency is still to be expected.

In terms of the amount of selected test cases, the results already show a great benefit with respect to a complete retest. In terms of the initial goals of this light-weight RTS technique and the abstraction level of the system and test specification, results are to be rated as excellent. Furthermore, it is if putting too much energy into achieving high precision is worth the effort, as due to the abstraction level certain limits (given by the function hierarchy and test goal attribute for test cases) will ease continuous improvements of this criterion. Additionally, the results are expected to fluctuate more strongly within the projects compared to white-box technique performances. This is also due to the abstraction level.

Similarities to (Software) Product Line Testing

Testing of (software) product lines (PL) is one further approach which aims for a reduction of testing effort. Amongst others, it describes methodologies on how to efficiently verify further developed versions or newly introduced configurations of a product. Incremental testing [19], for example, is a technique which attempts to test a PL by selecting one individual product and testing it thoroughly, then deriving further versions with the required testing effort. Subsequently, it identifies commonalities and differences between the individual product and the PL and selects only those parts for testing which vary from the fully tested product configuration. The objective of RTS and SPL testing techniques thus are quite similar as both aim at minimizing the number of test cases in T_i' for re-verification of the product.

In context of PLs, evolutionary steps within a product line leading to alternative versions and configurations of the product primarily represent nothing but a change (e.g. replacement, enhancement, addition or deletion of system parts) in the architectural design of the system functionality (f_{nm} or k_{nm}) or the system itself (f_{nm} or k_l or c^j_{nm}). Thus, impacts of such steps on unchanged parts of the system or its components may be detected by the RTS technique as well, leading to an optimized verification of the new product configuration.

Next to analogues objectives, also remaining challenges, especially the one of transferring the concept of SPL testing techniques into industrial system levels, are quite similar to the ones discussed. The developed approach on RTS might therefore support obtaining a solution to SPL testing as well. Especially on system level, where the system specification has to be regarded as a substantial basis for analysis, SPL testing can greatly benefit from an efficient and general RTS methodology by using the same evaluation criteria, documentation framework as well as a related approach for selecting required test cases.

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html

References

[1] H. Agrawal, J. R. Horgan, E. Krauser, S. London, Incremental regression testing, In Proc. Conf. on Softw. Maint., 1993

[2] T. Ball, On the limit of control flow analysis for regression test selection, In Proc. Int'l Symp. on Softw. Testing and Analysis, 1998.

[3] S. Biswas, A Model-Based Regression Test Selection Approach for Embedded Applications IIT Kharagpur, India GM India Science Lab, 2009

[4] B. Boehm, Guidelines for Verifying and Validating Software Requirements and Design Specification, EURO IFIP 79, P.A. Samet (eds.) North Holland, 1979

[5] Y. Chen, R. L. Probert, D. P. Sims, Specification-based Regression Test Selection with Risk Analysis, IBM Centre for Advanced Studies Conference, 2002

[6] P. K. Chittimalli, M. J. Harrold, Regression Test Selection on System Requirements, 1st India Software Engineering Conference, 2008

[7] E. Engström, A Systematic review on regression test selection techniques, Department of Computer Science, Lund University, 2010

[8] E. Engström, Exploring Regression Testing and Software Product Line Testing - Research and State of Practice, Department of Computer Science, Licentiate Thesis, Lund University, 2010

[9] K. Gallagher, Reducing Regression Test Size by Exclusion, Durham University, 2007

[10] R.P. Gorthi, A. Pasala, K.K.P. Chanduka, and B. Leong, Specification-Based Approach to Select Regression Test Suite to Validate Changed Software, in Proc. APSEC, 2008

[11] R. Gupta, M. J. Harrold, M. L. Soffa. Program slicing based regression testing techniques. Journal of Softw. Testing, Verif., and Rel., 1996.

[12] M. J. Harrold, A. Orso, Retesting During Development and Maintenance, IEEE Conference on Software Maintenance, 2008

[13] International Organization for Standardization, ISO FDIS 26262 BL19, Road vehicles – Functional safety

[14] R. Nörenberg, A. Cmyrev, R. Reißing, K. D. Müller-Glaser, Efficient verification planning for ISO-conformant functional testing of automotive applications, Technische Akademie Esslingen, 2011

[15] R. Nörenberg, Effizienter Regressionstest von E/E-Systemen, Dissertationsproposal, Karlsruher Institut für Technologie , 2011

[16] R. Nörenberg, R. Reißing, J. Weber, ISO 26262 Conformant Verification Plan, 8th Workshop Automotive Software Engineering, 2010

[17] G. Rothermel, M. J. Harrold, Analyzing Regression Test Selection Techniques IEEE Transactions on Software Engineering, 1996

[18] G. Rothermel, M. J. Harrold, A safe, efficient regression test selection technique. ACM Trans. Softw. Eng. Meth., 1997

[19] S. Oster, A. Wübbeke, G. Engels, A. Schürr, Model-Based Software Product Lines Testing Survey in, J. Zander, I. Schieferdecker, P. Mosterman (eds.): Model-based Testing for Embedded Systems, CRC Press/Taylor & Francis, 2010

[20] F. Vokolos, P. Frankl, Pythia: A regression test selection tool based on text differencing, In Proc. Int'l Conf. on Rel., Qual., and Safety of Softw. Intensive Sys., 1997