Reducing Re-Verification Effort by Requirement-based Change Management *

Markus Oertel[1] and Achim Rettberg[2]

[1] OFFIS e.V., Escherweg2, D-26121 Oldenburg, Germany
markus.oertel@offis.de
[2] University of Oldenburg, Ammerländer Heerstraße 114-118, D-26111 Oldenburg, Germany
achim.rettberg@informatik.uni-oldenburg.de

Abstract. Changes in parts of a safety critical system typically require the re-verification of the whole system design. In this paper we present a change management approach that contains the effects of a change within a region of the system. The approach guarantees to maintain the integrity of the system while performing changes. Our approach directly integrates verification and validation activities in the process. Furthermore, the propagation of changes is not based on the interfaces of the components and their interconnections, but exploits the knowledge of the behavior described by the requirements. This approach creates a much more precise set of affected system artifacts. In addition, we propose techniques to analyze the propagation of changes automatically based on formalized requirements and guide the selection of suitable compensation candidates.

Keywords: change management, system consistency, verification and validation, formal methods, safety critical embedded systems, model-based design

1 Introduction

Today's safety critical embedded systems are subject to a strict quality assurance process. Domain specific standards like the ISO 26262 [2] (automotive)or ARP 4761 [1] (aerospace) aim to reduce the risk of harming people by a systematic hazard analysis and structured breakdown of safety requirements and system design. Once having reached a consistent system state in which all verification and validation (V&V) activities have been performed, changes can become extremely expensive since typically the whole system design needs to be verified again.

Current change management tools used in a systems engineering context like Reqtify [11], IBM Change [15] or Atego Workbench [4] focus on the traceability

* The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement n°269335 and the German Federal Ministry of Education and Research (BMBF).

[18][13] between requirements and/or system artifacts. If changes occur, these tools highlight the system artifacts directly connected by trace-links to a changed element. This approach has a couple of limitations: Changing a single component often requires changes in other components as well [12] to re-establish the consistency of the system. Based on the information available by the traceability links and the interconnections of components, it is not possible to contain the propagation of a change. This results again in very broad re-validation and re-certification activities. Furthermore, it is impossible to narrow down the set of system artifacts that are directly affected by the change. All linked elements are potential candidates and need to be checked for unwanted side-effects manually, although only a part of them are relevant to the change. Also the integration of configuration management features into the change management process needs to be improved: In contrast to current development guidelines like CMMI-DEV [8] baselines are set manually, not bound to consistency criteria, and the dependencies between updated system artifacts and verification results are not considered.

In literature change management and change impact analysis is basically approached from two different directions. Estimate the costs of a change before implementing it and identifying affected elements during the change process.

Most of the techniques to quantify the costs of a change do not consider a detailed analysis of the particular change but use knowledge about similar systems and engineering experience. Clarkson et al. [7] use a Design Structure Matrix in combination with a probabilistic approach to determine the likelihood and impact of changes on the different system components. Furthermore scenario based approaches[5] exist to determine quality metrics concerning modifyability for a given architecture. Change scenarios will be defined and for each of them the possible impact will be estimated. The average effort will be calculated based on all considered change-scenarios. This approach is suited to compare two architectures or determine the risk of expensive changes. It is not used in the development process itself to handle introduced changes. Verification and validation activities are not considered and difficult to integrate into the approach since only one unit of measure can be used for the calculations (Lines of code are used in the example).

Approaches for change management during the actual change process are typically based on graph structures [6] and are mostly related to software change management. Extensions exist to object oriented software [19] but the techniques are not applicable for system engineering since the implementation is too late available in the development process (which also might be physical and no code), therefore the requirements of the components are better suited to propagate changes.

In this paper we provide an approach for containing change effects and maintaining the consistency during the change process. Our technique ensures to identify all verification and validation activities that are affected by the change. The approach is based on the system requirements rather than the interconnections of components. Therefore we can limit the change propagation semantically

without explicitly modeling it. Also the set of elements that are directly affected by a change is much smaller compared to other change management systems. I.e., from all connected elements only a subset needs to be checked for malfunctions through the change.

We currently cover the functional and timing aspects of the system within one design-perspective [10]. Perspectives represent the system at different structural stages, like a logical, technical or geometrical perspective. Aspects cover the properties within one perspective that are related to a particular topic like safety, timing or weight. Our approach allows the containment of changes within the mentioned aspects only. Nevertheless adding more aspects and perspectives is systematically possible. It is expected that for a given use case only a small subset of them needs to be added like weight, electromagnetic compatibility (EMC) or heat.

The base process is described in section 3. In section 4 the principles behind the containment of the change effects are explained. Section 5 provides an overview over the necessary formalization techniques and activities that can be checked automatically. Based on this formalization we introduce a concept of identifying system elements that can be modified to reach fast a consistent system again. The approach has been implemented in an OSLC based demonstrator. Section 7 introduces this prototype and first evaluation results. A conclusion and future development activities are outlined in the final section 8.

2 System Abstraction and Prerequisites

We analyzed many different meta-models and standards across various domains to identify a basic set of system artifacts and trace-links [17], that cover most of the currently used development artifacts. The change management process is described on these elements to be easy applicable to existing model-based development workflows.

The elements are:

- **Components**: Model-based representation of the interface of a system element. Depending upon the used perspective this can be logical components or concrete hardware or software parts [10].
- **Requirements**: Requirements represent functionality or properties that the component it is attached to shall fulfill.
- **Implementation**: Implementations represent the behavior of a component. This might be software (code or functional model) or a hardware implementation.
- **V&V cases**: Representation of the activity and their results to prove a property of the system.

The trace-links are:

- **Satisfy**: Connects a requirement with a model component. It symbolizes a "shall satisfy" relation.

– **Derive**: A requirement is decomposed into multiple derived requirements.
– **Refine**: A requirement is formalized into another language or representation.
– **Implementation**: Connects an implementation to a component.

These links are verification target, i.e. that it needs to be proved that the claimed property holds. Therefore V&V cases can be connected to each link detailing the activity that needs to be performed to get the necessary evidence.

To establish a change management process that allows to encapsulate the change in a determined area of the development item, it is necessary that a few prerequisites by the development process itself are met. Still, the process shall be as less inversive to the common practice processes as possible.

The most important prerequisite is that the requirements are always linked to a component and formulated in a black box manner, meaning that the requirement specifies the intended behavior on the interface of this component. This is in line with the contract based design paradigms to ensure the composability of the specified elements [9].

There are also some assumptions on the technical setup for the later realisation: As stated in typical recommendations for configuration management [16] we require all system artifacts to be under version control. In addition we require also trace-links to be versioned and pointing to versioned elements. This is necessary since the semantics of a link may not be applied for a changed element, at least not without an analysis. E.g.: A component covers a special requirement, but after changing the requirement is has to be considered again whether the attached component is still the best to implement this requirement. The link is just valid for a special version of a requirement.

3 Change Management Process

During the processing of a change request verification results that are connected to modified elements need to be invalidated. Furthermore all modified elements, also the trace-links and verification results need to be versioned to point to the new targets. The consistency of these elements is necessary to allow a reject of the change request at any point in time. This might be necessary if it is foreseeable that the change is becoming too expensive. Furthermore the correct version of the verification targets is required to fulfill the traceability and documentation requirements from various safety standards.

The process to achieve these goals is depicted in figure 1 and consists of four basic steps that are executed in an iterative manner:

The system $(A, L)_{\mathfrak{v}}$ consists of the set of system artifacts $A = R \cup C \cup V \cup I$ (Requirements, Components, V&V cases and Implementations) and the set of links L. Each state of the system is identified by a version $\mathfrak{v} \in N^+$. In this description of the process a new version of the whole system is created in each step, this is a simplification in the notation, in the implementation, of course, each element is versioned for its own. A baseline $\mathfrak{v} \in B$ is a version in which all verification activities v of the system are executed successfully.

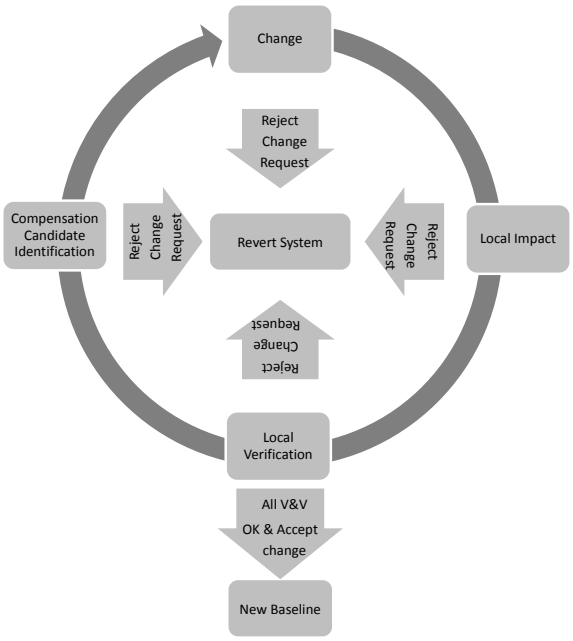


Fig. 1. Basic process for handling Change Requests

$$\mathfrak{v} \in B \rightarrow \forall v \in (A, L)_{\mathfrak{v}} : status(v) == success$$

In the $\mathfrak{i}$'s execution of the *change* phase a set of elements of the system $(A, L)_{\mathfrak{i}-1}$ is modified, denoted as $E_{mod_{\mathfrak{i}}} \subseteq (A, L)_{\mathfrak{i}}$.

This modification alone results in an inconsistent system state, since the V&V cases connected to the system elements e that have not been changed are not valid anymore and need to be re-validated. In the *local impact* phase the connected V&V cases are reset:

$$\forall v \in (A, L)_{\mathfrak{i}} | e \in target(v) \wedge e \in E_{mod_{\mathfrak{i}}} : status(v) \rightarrow suspect$$

The set of elements evaluated by a V&V case v is expressed by $target(v)$

The suspect V&V cases $V_{suspect}$ need to be re-validated in the *local verification* phase:

$$v \in V_{suspect} \rightarrow \{failed, success\}$$

For the failed V&V cases it is necessary to adapt the system. The possible compensation candidates E_{comp} are the elements the V&V case is targeting.

$$E_{comp_{\mathfrak{i}}} = \bigcup_{v \in V_{suspect}} target(v)$$

The engineer needs to select one or more elements of this set and modify them so that the failed V&V activities are successful again. These modifications start the cycle again:

$$E_{mod_{i+1}} \subseteq E_{comp_{i+1}}$$

A new baseline can be created if all V&V activities are again successfully executed.

4 Consistency of the System

The described process uses the V&V activities to propagate changes across the system design. We claim to identify the parts of the system that are not affected by a change. In this paper we consider the functional aspect of the system within one perspective only. This means in particular, that "side-effects" over other aspects or perspectives are not considered. A typical example for these side-effects is change propagation through heat distribution. Due to a change on one part of the system other parts get that warm that they need additional cooling. The approach is designed in a way that these additional considerations can be added as needed by the system under development. The set of necessary perspectives can typically be limited in advance since the type of system often constitutes if e.g. EMC, weight or heat aspects have to be considered.

To be able to contain the change effects a defined set of V&V activities needs to be carried out. We use four different types of V&V cases that are necessary to comply with modern safety standards anyway (e.g. ISO 26262 [2] which basically evaluate each trace-link:

- The most important V&V activity is the verification of the correct derivation of requirements, i.e. that a requirement is correctly split up into sub-requirements. If changes in requirements cause this derive-relation between requirements to fail, other requirements need to be adapted as well. Using requirements for change propagation instead of connectors between components has the benefit that a criterion for stopping the change propagation is given. If the split-up is still (or again, after further modifications) correct, there is no further propagation in this part of the system. Therefore requirements build the backbone for change propagation. To prove the correct derivation of requirements a consistency check of these requirements is technically mandatory, since the split-up of inconsistent requirements is correct if the top-level and the sub-requirements result in a system that cannot be build.
- Also modifications in implementations or their components can result in change effect propagation. Therefore the implementation link between a component and an implementation needs to be verified. This check is limited to the interface, since the component itself does not contain any more information. This check is easily automatable.

– Similar to the implementation link, also the satisfy link (between requirement and component) needs to be checked. This analysis is also based in the interface.
– Due to the fact, that the structural system description in form of components is separated from the behavioral description in form of the implementations and requirements an additional V&V activity is needed that checks the relation of the requirement to the implementation. This is typically performed by testing.

The functional relations between the different components are extracted from the requirements. Whether one requirement change propagates towards other components is determined by the result of the test checking the correct derivation of the connected requirements. This step can be automated, see section 5. Therefore no explicit modeling of change propagation is necessary for the functional aspect. To be able to use the requirements for containing change effects they need to be stated in a blackbox manner, i.e. that they are just formulated on the interface of the attached component. This kind of requirements allows virtual integration [9] as needed by our approach. A formal prove that these prerequisites will contain any functional change effects will be published soon.

To guarantee that there is no influence by the change outside of the identified boundary it is assumed that all verification activities are accurate. Many of the mentioned verification activities can be automated using formal methods (see section 5) and therefore reach the desired accuracy. This especially applies for the analysis verifying the correct derivation of requirements which is responsible for the propagation of the changes using the requirements. To be able to get qualified or certified even the manually performed verification activities or testcases need to have a reasonable level of confidence. The same level of confidence also applies for the propagation of the change.

5 Automating Change Impact Analysis

In the previous section we discussed how changes can propagate along the requirements through the system. Therefore, a reliable method of proving the requirements derivation is desired. This method can be realized using the formal requirements specification language (RSL) [10] and the entailment analysis [9].

The RSL is a formal but still human readable language to express requirements. It is based upon predefined patterns with attributes that can be filled by the requirements engineer and was designed to cover a whole range of different types of requirements.

A typical example of a pattern:

`Whenever <EVENT> occurs <CONDITION> holds during [INTERVAL]`

The formalization the the requirement “The emergency light shall provide illumination for at least 10 minutes after the emergency landing has been initiated” looks like:

`Whenever EmergencyLandingInitiated occurs emergencyLight==ON holds during [0s,600s]`

The semantics of these patterns are described using timed automata.

The entailment analysis is a contract based virtual integration technique. Input are the specifications of the top-level component and a set of specifications of the direct sub-level components.

$$ent : r_{top} \times R_{sub} \rightarrow 0, 1 | R_{sub} \subseteq succ(r_{top})$$

The entailment relation is defined on the accepted traces T_a of the specifications [14]. Entailment is given, if the set of traces which are accepted by all sub-requirements is a subset of the set of accepted traces by the top-level requirement. In contrast, if there exists a trace that is accepted by all the sub-requirements but not from the top-level requirement the sub-requirements are not strict enough. These additional traces describe behavior which is forbidden by the specification of the top-level requirement and therefore the requirements breakdown is not correct.

$$ent(r, R) = \begin{cases} 1 & \text{iff } \bigcap_{r_s \in R} T_a(r_s) \subseteq T_a(r) \\ 0 & \text{iff } \bigcap_{r_s \in R} T_a(r_s) \nsubseteq T_a(r) \end{cases}$$

The entailment analysis fits the needs for the verification of derive links. It can be automatically proved that the behavior described by the derived requirements is in line with the top-level requirement. Consequently, if requirements are changed, but the derive-relation towards the top-level and towards the more refined requirements are still correct, the set of accepted traces is unchanged and a propagation of changes outside this boundary can be obviated.

6 Guided Compensation candidate selection by shifting

Using the formalization and entailment analysis the change management process can be further automated. In some cases it is possible to identify a concrete element out of the set of compensation candidates which is a good choice to adapt to a change, because no implementation needs to be adapted but only requirements reformulated.

This guidance mechanism uses the fact that requirements are typically stated with some margins since the actual implementation might not be known at early phases of the development process. This typically includes timing or memory constraints as well as behavioral requirements. Furthermore, the usage of COTS (components-of-the-shelf) enforces this effect since externally purchased components commonly not match the requirements to 100% but are e.g. a little faster or provide additional features not needed at the time of component selection.

In terms of the entailment relation this means that the set of accepted traces by the top-level requirements which are not in the set of the sub-level requirements might be of significant size.

$$\Delta T_a(r_{top}, R) = T_a(r_{top}) - \bigcap_{r_s \in R} T_a(r_s) \neq \emptyset$$

If a requirement gets changed in the system, this buffer can be used to compensate the change not only locally but at a different location of the system. This can be achieved since ΔT_a can be shifted towards the upper level requirements and "collect more" traces that can be used at a chunk. This is realized by replacing the top-level requirement with the parallel composition of the sub-level requirements.

The parallel composition [20] of a set of requirements is also defined by the accepted traces. The composition includes only the traces that are compatible to all requirements.

$$T_a(r_1||r_2) = T_a(r_1) \cap T_a(r_2)$$

If entailment is given ($ent(r_{top}, R) = 1$) it is obvious that the top-level requirement r_{top} can be replaced by the parallel composition $||_{r \in R}$ without altering the result since entailment is always given this way:

$$\begin{aligned} & ent(||_{r_s \in R}, R) = 1 \\ \Rightarrow \quad & \bigcap_{r_s \in R} T_a(r_s) \subseteq T_a(||_{r_s \in R}) \\ \Rightarrow & \bigcap_{r_s \in R} T_a(r_s) \subseteq \bigcap_{r_s \in R} T_a(r_s) \end{aligned}$$

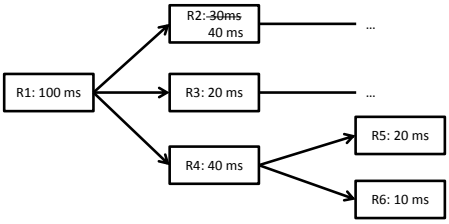
While reducing the set of accepted traces for the top-level requirement by $\Delta T_a(r_{top}, R)$ a failed entailment relation using r_{top} as a sub-requirement might be successful again.

An example with timing requirements is depicted in figure 2. The requirements $R1$ to $R6$ represent time budgets, the arrows represent derive links. If requirement $R2$ is changed to $50ms$ because it was not possible to find an implementation that could fulfill this requirement the entailment relation $ent(R1, \{R2, R3, R4\})$ would fail (see figure 2(a)), causing additional changes in the system. In this case requirement $R4$ can be replaced by the parallel composition of $R5$ and $R6$, namely $30ms$, and the entailment relation $ent(R1, \{R2, R3, \{R5||R6\}\})$ is still valid (figure 2(b)).

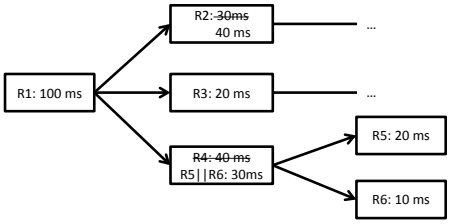
Applying this technique over multiple levels and starting from more than one sub-requirement also non-trivial cases of shifting can be identified resulting only in requirement changes and no implementation changes. Furthermore, it is not necessary to re-validate the changed requirements.

7 Prototype

The tool landscape used for model-based development of safety-critical embedded systems is characterized by a high degree of distribution and heterogeneity [17]. Different development teams are working with different tools (for requirements management, verification, test, modeling, simulation, etc...) and different repositories on the final product. Therefore an integrated change management solution needs to overcome tool, supply-chain and data-format boundaries. Therefore, our prototype implementation "ReMain" uses an OSLC [3] based communication to access the different system artifacts across different repository



(a) Entailment fails because of changes in R2



(b) Corrected entailment by shifting

Fig. 2. Simplified example using timing requirements

locations. The tool consists of a server component handling the change requests and initiating the versioning of artifacts and a client that displays the affected system parts to the user and handles his input.

In the current demonstrator setup requirements can be stored in IBM DOORS or MS Excel, components are represented as EAST-ADL or AUTOSAR models and Implementations are read from Simulink models. The V&V cases are managed by the ReMain tool itself. Missing V&V cases which were necessary (see section 4) for our approach can be identified and generated. A dedicated V&V repository is planned for future versions.

The typical workflow should not be touched, so developers can automatically trigger the change management process by changing elements with their known modeling tool (e.g. Simulink or DOORS). The OSLC adapters of these tools will detect the change and send a modification event to the server part of the ReMain tool.

After receiving the modification event the ReMain server will perform the "local impact" activity. In particular, this includes the versioning of the connected links and V&V cases and setting their status to "suspect". The client will display only the part of the system that is currently affected by the change.

The status of the different V&V cases is represented by colored bubbles after the name of the element. The engineer gets a direct overview which activities still have to be performed and how much the change already propagated through the system. If formalized requirements are used the entailment analyzes can be executed automatically directly from the user interface of the ReMain client. If a verification activity fails, compensation candidates are highlighted.

If all verification activities have been executed successfully a new baseline can be created containing the consistent system elements. The baseline is currently stored externally but shall be directly integrated in existing configuration management tools in the future.

First evaluation results have shown that it is much easier to track changes in distributed development environments. Compared to existing approaches the decision how to react on changes can be made much faster. We use an ABS braking system as a test platform in which a saving in verification activities of more than 80% could be reached depending upon the artifacts that have been changed. For conclusive numbers more systems need to be analyzed in a systematic way.

8 Conclusion and Outlook

We presented an approach to identify the affected system parts during changes of system components for the functional aspect of the design. Using proper requirement formulation, traceability between system artifacts and a defined set of verification activities it is possible to reduce the re-certification effort of products since changes outside of the identified boundary have no effect on the system behavior.

In contrast to existing change management approaches the propagation of changes is not based on connected interfaces, but on the stated requirements of the system. This ensures semantically correct change effect propagation without the need of explicitly modeling the propagation. Furthermore, the set of system artifacts that need to be investigated during the process is reduced compared to approaches using the interconnections of components. Using formalized requirements the propagation process can be automated and even suggestions to possible good compensation candidates for failed V&V activities can be given. In the prototype implementation "ReMain" the process has been integrated in a highly distributed development environment consisting of commonly used development tools and formats like DOORS, Excel, Simulink, AUTOSAR and EAST-ADL.

Still, a couple of features are not yet discussed or implemented: Separating a requirement into an assumption and a promise (contract), the propagation of changes can even further reduced, also the compensation candidate selection is likely to benefit from this approach. Furthermore, the process needs to be extended to cover more than one perspective and more than the functional aspect. This includes the integration of allocation decisions. It is the aim to provide a change management solution where additional aspects (like EMC or heat distribution) can be added to cover the "side-effects" that are needed for the system under development. Also the accuracy of the approach needs to be investigated if the confidence in the executed V&V activities is low.

References

1. Arp-4761:aerospace recommended practice: Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment (1996)
2. Iso 26262: Road vehicles - functional safety (November 2011)
3. Oslc: Open services for lifecycle collaboration. http://open-services.net/ (December 2012)
4. atego: Atego workbench. http://www.atego.com/products/atego-workbench/ (November 2012)
5. Bengtsson, P., Bengtsson, P., Bengtsson, P.: Architecture-level modifiability analysis. Journal of Systems and Software 69 (2002)
6. Bohner, S.: Extending software change impact analysis into cots components. In: Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE. pp. 175 – 182 (dec 2002)
7. Clarkson, P., Simons, C., Eckert, C.: Predicting change propagation in complex design. Journal of Mechanical Design(Transactions of the ASME) 126(5), 788–797 (2004)
8. CMMI Product Team: Cmmi for development, version 1.3: Improving processes for developing better products and services. http://www.sei.cmu.edu/reports/10tr033.pdf (November 2010)
9. Damm, W., Hungar, H., Josko, B., Peikenkamp, T., Stierand, I.: Using contract-based component specifications for virtual integration testing and architecture design. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2011. pp. 1 –6 (march 2011)
10. Damm, W., Hungar, H., Henkler, S., Stierand, I., Josko, B., Reinkemeier, P., Baumgart, A., Büker, M., Gezgin, T., Ehmen, G., Weber, R.: SPES2020 Architecture Modeling. Tech. rep., OFFIS e.V. (2011)
11. Dassault Systems: Reqtify. http://www.3ds.com/products/catia/portfolio/geensoft/reqtify/ (Nov 2012)
12. Eckert, C., Clarkson, P., Zanker, W.: Change and customisation in complex engineering domains. Research in Engineering Design 15, 1–21 (2004), http://dx.doi.org/10.1007/s00163-003-0031-7
13. Gotel, O., Finkelstein, C.: An analysis of the requirements traceability problem. In: Requirements Engineering, 1994., Proceedings of the First International Conference on. pp. 94 –101 (apr 1994)
14. Hungar, H.: Compositionality with strong assumptions. In: Nordic Workshop on Programming Theory. pp. 11–13. Mälardalen Real–Time Research Center (11 2011)
15. IBM: Rational change. http://www-01.ibm.com/software/awdtools/change/ (November 2012)
16. International Organization for Standardization: Iso 10007: Quality management systems guidelines for configuration management (June 2003)
17. Rajan, A., Wahl, T. (eds.): CESAR - Cost-efficient Methods and Processes for Safety-relevant Embedded Systems. No. 978-3709113868, Springer (2013)
18. Ramesh, B., Powers, T., Stubbs, C., Edwards, M.: Implementing requirements traceability: a case study. In: Requirements Engineering, 1995., Proceedings of the Second IEEE International Symposium on. pp. 89 – 95 (mar 1995)
19. Ryder, B.G., Tip, F.: Change impact analysis for object-oriented programs. In: Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering. pp. 46–53. PASTE '01, ACM, New York, NY, USA (2001), http://doi.acm.org/10.1145/379605.379661
20. SPEEDS: SPEEDS core meta-model syntax and draft semantics (2007), SPEEDS D.2.1.c