Using Requirements Traceability Links At Runtime – A Position Paper

Alexander Delater, Barbara Paech
University of Heidelberg, Institute of Computer Science
Im Neuenheimer Feld 326, 69120 Heidelberg, Germany
{delater, paech}@informatik.uni-heidelberg.de

***Abstract*:** During software development a large amount of varied information is created. It comprises the requirements specification and depending artifacts such as design, code or test cases, as well as supporting information such as traceability links. This information is intended to be used during development time. The research in requirements at runtime has so far focused on using the requirements specification at runtime. This paper explores how to use the existing traceability links between requirements and other artifacts at runtime.

Keywords: *requirements, traceability, development time, runtime*

I. Introduction

Traceability information is created throughout the entire software engineering process using a plethora of different methods [7]. This information is used to support such activities as impact analysis for required changes or to support the developers by providing the related artifacts to a given artifact, e.g. a requirement and its realization in the code. More examples for specific questions that can be answered by traceability links can be found in [8]. However, the traceability information is usually only used during the development time of the system.

In this paper we present our thoughts on how requirements traceability information can also be used during the runtime of a dynamically adaptive system (DAS). In [2] Berry et. al. have defined a DAS as "a computer-based system (CBS) that is capable of recognizing that the environment with which it shares an interface has changed and that is capable of changing its behavior to adapt to the changing conditions".

The paper is structured as follows: in Section II we define the basic terminology and assumptions used throughout the paper. Section III gives an overview of traceability links, their benefits and their use during development time and runtime. This is detailed in Section IV. Section V discusses the results and Section VI provides a conclusion.

II. Basic Terminology and Assumptions

In this section the basic terminology and assumptions used in the paper are defined. First, a short definition of requirements traceability is provided. Second, we define the terms "development time" and "runtime" of a system using the term *release*. Third, we determine how traceability information fits into requirements engineering for DAS.

A. Requirements Traceability

In the field of requirements engineering (RE), traceability is usually defined as the ability to follow the traces to and from the requirements. Gotel & Finkelstein have given a well-accepted definition [5]:

> "Requirements Traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)."

In this paper we assume that the requirements are structured and stored with the help of a requirements management tool such as IBM Rational DOORS. Requirements can either be represented as formalized requirements or natural language requirements. Furthermore, we assume that there exist traceability links between the requirements and further artifacts of the development process, in particular design, code, test cases, rationale description for important development decisions and descriptions of development tasks. Note that we do not consider links between requirements on different levels (e.g. between early and late requirements or between requirements of different granularity). We are not interested in how the artifacts, e.g. requirements, code, test cases etc., as well as the traceability links to depending artifacts were created during the development time. It is assumed that this information is available and ready to use.

B. Development Time and Runtime

Typically, the terms "development time" and "runtime" are defined referencing the complete software system. However, a software system is subject of constant change and is adapted and improved throughout its lifetime. Thus, it is difficult to distinguish development time and runtime clearly. In contrast, a *release* of a software system is a finalized version of the system at a given point of time. Therefore, the runtime of the release is starting when the development time of the release is completed. This leads to the following definitions:

Development Time: During development time, a new release of a system is developed based on a previous release and new requirements. The new requirements might be from the context of use (stakeholders) or induced through new technologies. Furthermore, they might be changes to existing requirements or additions to the existing requirements. A large set of artifacts (requirements, design, code, test cases etc.) is modified and linked to one another. At the end of the development time, a final release of the software is deployed and the set of inter-linked artifacts is completed and refers to the final state of the release.

Runtime: The deployed release of the system is used inside the customer organization. The release supports the users by accomplishing their tasks. The release provides a valuable contribution to the business and is an important asset for the day-to-day business of the customer.

Thus, for a DAS we distinguish

- the development time of the release of the DAS in which the development team creates development artifacts including requirements and traceability links. The code of the release contains adaptation mechanisms and mechanisms to determine when and how to adapt.
- the runtime of the release of the DAS. During the runtime the release can adapt its behavior. That means that due to changes in the environment the code is adapted using the adaptation mechanisms. This creates *a variant* of the release. Clearly, this adaptation also impacts other artifacts and their traceability. Requirements need to be adapted to reflect the current state of the environment and the active variant. Design and test cases need to be adapted to be consistent with the adapted code. Furthermore, test cases need to be performed to make sure that the adapted code satisfies the requirements. Thus, activities performed by the development team at development time are necessary and helpful during runtime. We assume that the development team only becomes active during development time for the next release. So these adaptations can only be performed by the release itself or by the customer or the users.

C. Requirements Engineering for Dynamically Adaptive Systems

In [2], Berry et. al. have defined four levels of RE for DAS. On the first level the requirements for all variants of the current release are specified. This is the core of requirements engineering activities at development time. So far many approaches have focused on this part by providing new languages to reflect e.g. incomplete information [4].

On the second level the release recognizes that adaptation is necessary and adapts its code accordingly. The release determines (based on the mechanisms implemented during development time) the new functionality to be offered in the active variant. This is part of the runtime of the release. In our context, we assume that this level includes all activities mentioned above to adapt other development artifacts and to perform quality assurance for the adapted code. While these adaptations would typically not considered as RE, Berry et al. argue that this is RE performed by the systems, as it includes understanding the current environments and its requirements.

On the third level RE is concerned with the adaptation mechanisms, that means it identifies suitable mechanisms. Again this is part of the development time of the release.

And finally on level four, RE is concerned with discovering adaptation mechanisms in general. This is independent of the particular release and not considered here.

Traceability links reflecting all variants are part of the first level. In the following we discuss how these links can be useful for RE on the second level. In [12] Welsh and Sawyer also discuss requirements tracing for DAS. However, they concentrate on creating requirements traces for DAS, while we focus on how they can be used at runtime.

III. REQUIREMENTS TRACEABILITY ASPECTS

In Pohl [9], several aspects related to requirements traceability are summarized (see column "Aspect" in TABLE I). These aspects represent different benefits of the availability of traceability links (see also [10] and [11]). We use the aspects presented by Pohl as a basis to identify typical uses of traceability links.

TABLE I. ASPECTS AND LINKS

Link / Aspect	Req. – Req.	Req. – Design	Req. – Code	Req. – Test	Req. – Rat.	Req. – Task
During Development Time & Runtime						
Acceptance			D / R	D / R		
Change Mgmt	D / R	D / R	D / R	D / R	D / R	
Quality Mgmt			D / R	D		
Re-Use	D	D	D / R	D	D	
Allocation			D / R			D
Only During Development Time						
Gold Plating		D	D			
Reengineering		D	D	D	D	
Risk Mgmt	D		D			
Project Progr.		D	D	D		D
Process Mgmt						D

D = usage during development time; R = usage during runtime

As shown in TABLE I, we only consider links from requirements to other artifacts. The examined links are: Requirements (Req.) – Req., Req. – Design, Req. – Code, Req. – Test, Req. – Rationale (Rat.), Req. – Task. A task is an activity performed by a member of the development team. Furthermore, the table summarizes whether these links can be used during development time or during the runtime of the DAS. This judgment is based on our own knowledge of and experience in RE. Details are explained in the next section.

TABLE II summarizes the aspects from TABLE I that benefit from traceability links during runtime and shows how their links are used. It describes which links are traversed in which direction to answer which question. Again the details are explained in the next section.

TABLE II. RUNTIME TRACEABILITY DETAILS

Usage / Aspect	Starting info.	Searched info.	Question/ Decision	Usage at Runtime
Acceptance	Adapted C	C → R R → T	Identify T needed to test adapted C	DAS
Change Mgmt	Adapted C	C → R	Identify side effects of C	DAS
	Adapted C	C → R R → T R → D R → Rat.	Adapt artifacts related to C	DAS
Quality Mgmt	Failed C	C → R	Identify criticality of failure	DAS / user
Re-Use	R	R → C	Identify reusable C	DAS
Allocation	R	R → C	Identify number of executions of C (related to certain R)	DAS

R = req., D = design, C = code, T = test case, Rat = rationale

IV. USAGE OF REQUIREMENTS TRACEABILITY LINKS DURING DEVELOPMENT TIME AND RUNTIME

The purpose of this section is to identify the links that can be used at runtime. We present the aspects from TABLE I and TABLE II in detail. They are all examined in the same structural way: First, a detailed description of the aspect is provided. Second, we explain which links between which artifacts are needed. Third, we look at the usage of these links during the development time. Fourth, we discuss whether these links can be used during the runtime of the release.

A. Acceptance

1) Description

Traceability supports the evidence that a requirement is realized as specified (correct and complete) in the developed release. This evidence increases the acceptance of the release.

2) Links needed

For acceptance the links between requirements and code, as well as between requirements and test cases are most important. The link to the code proves that the requirement has been considered in the implementation, while the link to the test cases proves that the quality of the requirement has been established appropriately.

3) Activities during development time

The requirements engineer or the project manager uses the links between the requirements and the code to ensure that all requirements were implemented in the source code. In particular, s/he can verify that all non-functional requirements are addressed by one or more parts of the implementation [8].

The customer or the project manager uses the links between the requirements and the test cases to ensure that all requirements are tested.

4) Activities during runtime

As mentioned in Section II, similarly to the verification at development time, it is important to perform verification at runtime. For instance, if a piece of code is adapted at runtime, then the trace to the affected requirements and from them to the corresponding test cases helps to identify the test cases which need to be executed at runtime. Thus, the DAS can identify the test cases and execute them. Of course, this requires also meta-information of test cases as introduced e.g. through built-in-test [3].

B. Change Management

1) Description

If an artifact changes, the traceability information makes it possible to identify linked artifacts that are affected by this change.

2) Links needed

To support the change management, it is necessary to introduce traceability links between the requirements and all depending artifacts that means requirements, design, code, rationale and test cases.

3) Activities during development time

A member of the development team performs an impact analysis that means s/he identifies the artifacts that are affected by the change of the requirement. Furthermore, s/he is able to use this information to predict the cost of the change. Afterwards s/he applies the necessary changes.

4) Activities during runtime

The execution of changes corresponds to the code adaptations at runtime. While the adaptations related directly to a requirements change must be built-in to be performed at runtime (and thus do not need the link from requirements to code), the links help to identify side effects of performed changes. One identifies further requirements (besides the one triggering the change) linked to the changed parts of the code. Thus, e.g. such links trigger further code adaptations. Furthermore, as discussed in Section II, we stipulate that together with the code adaptation also the related artifacts need to be updated. So e.g. the DAS highlights in the artifact repository the artifacts corresponding to the currently active variant.

C. Quality Management

1) Description

The traceability information facilitates the identification of the causes and effects of bugs, the determination of the affected parts of the release and the prognosis of the effort to fix the bug.

2) Links needed

To support quality management, it is necessary to introduce links between the requirements and code as well as between the requirements and test cases.

3) Activities during development time

From a (failed) test case the developer navigates to the related requirement; from this requirement s/he navigates to the affected parts of the code. Thus, the links help to locate the bug. Furthermore, the link to the requirements helps to identify the severity of the failure. If the affected requirements are not critical, fixing the bug can be postponed.

4) Activities during runtime

During runtime built-in test cases are run and may fail. The reaction to the failure must already be built-in to the code, too. Thus, the link from test case to code is not needed at runtime. Similar, the link to the rationale is not required at runtime as well. However, a use of the link from code to requirements seems possible: the DAS notices a failure of the release during runtime, for example a service is not available. Then it knows the affected part of the code. Thus, it can also use the link to the requirements to determine whether these requirements are critical and use this information to decide how to react to the failure. E.g. in case of a critical failure, it can shut down. Similarly, a user could use this link to decide whether a failure is critical.

D. Re-Use

1) Description

Traceability supports the re-use of development artifacts. By performing a comparison between the old and new requirements, artifacts can be identified which can be re-used in the new release.

2) Links needed

To support the re-use of development artifacts, traceability links between all depending artifacts are required, that means between requirements, design, code, test cases and rationale.

3) Activities during development time

A developer has to implement a requirement. S/he searches for a similar requirement already implemented. By using the link between the requirements and design or code, s/he is able to see how the similar requirement has been designed and realized in the code of a previous release. The linked rationale supports the developer during the implementation by providing valuable knowledge and the linked test cases might be adapted for testing the new requirement.

4) Activities during runtime

Identifying an existing piece of code for a given requirement is one of the standard mechanisms used in service-based DAS. Given a specification of a service (e.g. in WSDL[1]) the DAS identifies a corresponding implementation at runtime (e.g. via UDDI[2]). All other links do not seem useful for the re-use during runtime.

E. Allocation

1) Description

Traceability information is useful for mapping the development effort to the individual requirements. By tracing which team member has performed a task related to a requirement and how much time s/he required for the task, it is possible to map the individual development costs to a single requirement. By performing this comparison, the allocation of the resources of the product is improved. It is also interesting to trace the amount of code related to a requirement.

2) Links needed

Traceability links between the requirements and the tasks in the project as well as to the code are required to support the allocation.

3) Activities during development time

By using the link between the requirements and the tasks in the project, the project manager or customer calculates the exact amount of required time and resources for the realization of each requirement. For example, the customer or project manager identifies which requirement was the most expensive one to implement based on the tasks effort. Moreover, by identifying a very expensive, yet unfinished requirement, the customer could stop the implementation in order to save resources. Similarly, the customer or project manager can identify how much code is related to a requirement.

4) Activities during runtime

Understanding development cost is not necessary during runtime. However, a related question interesting for the customer is the following: how much code related to a requirement is exercised how often at runtime. Thus, the DAS identifies for a given requirement the related code parts and monitors their execution.

The following aspects F-J describe benefits from traceability links that are only useful during development time.

F. Gold Plating

1) Description

Traceability supports the identification of code that was not required in the requirements specification. Therefore, it may have no justification to be part of the release. The development of such code is called “gold plating”.

2) Links needed

To ensure that only required features were implemented in the release, it is necessary to link requirements with the design and with the code.

[1] http://www.w3.org/TR/wsdl
[2] http://www.uddi.org/pubs/uddi_v3.htm

3) *Activities during development time*

All members of the development team as well as the customer use the links from requirements to design or from requirements to the code to check whether all parts of the implementation are justified by a requirement.

4) *Activities during runtime*

As adaptations at runtime are only triggered by environmental changes, a check at runtime, whether these adaptations are justified by the requirements, is not necessary.

G. *Re-engineering*

1) *Description*

Traceability information supports the re-engineering of legacy systems. It is possible to understand which code of the legacy system realizes which requirements.

2) *Links needed*

To support the re-engineering, it is necessary to introduce links between the requirements and the design, code and test cases. Furthermore, to better understand decisions it is necessary to link requirements with rationale.

3) *Activities during development time*

A new team member, who was not participating in the development of a previous release of the software system, uses the linked rationale to better understand the major design decisions of the previous release. For example, a rationale documents a decision made for the usage of a specific technology. Thus, the team member better understands how the requirement has been realized in the design and code of the release. Furthermore, a team member uses the link between the requirements and code or test cases to understand the context of certain code fragments or test cases.

4) *Activities during runtime*

Understanding the code is not necessary at runtime for the DAS itself or the customer or the users.

H. *Risk Management*

1) *Description*

Traceability between requirements and other artifacts (e.g. code) supports the risk management. Artifacts potentially affected by a risk can be identified more rapidly and reliably by the available traceability information.

2) *Links needed*

In order to support the risk management, it is necessary to introduce links between the requirements themselves as well as between the requirements and the code.

3) *Activities during development time*

This aspect is a special form of change management where the impact analysis is used to identify effects of a risk. The requirements engineer or project manager identifies requirements that are maybe affected by a risk early in the development time. If a risk appears, s/he is able to estimate the effort to change the requirement as well as the linked artifacts in the design and code.

4) *Activities during runtime*

For risk management the changes are only analyzed, but not performed. Thus, risk management is not useful at runtime. Clearly, the adaptation after the risk has appeared is necessary at runtime. This is captured under change management.

I. *Project Progress*

1) *Description*

Traceability information supports the tracking of the project progress and the current state of the project. By performing an analysis of the links between the artifacts, one is able to identify which and how many requirements were already included in the code or covered by test cases.

2) *Links needed*

To support the project progress measurement, it is necessary to introduce links between the requirements and design, code and test cases. By introducing links between the project model and system model, a development task can be traced to the artifacts it is related to [6].

3) *Activities during development time*

This aspect is related to acceptance. A project manager uses the links between the requirements and design as well as the links between the requirements and code to identify how much requirements were already realized and implemented. Using the links between requirements and test cases s/he can also track which requirements are already tested. By using the links between requirements and tasks, s/he can track the importance of ongoing tasks.

4) *Activities during runtime*

Understanding project progress or status is not needed at runtime.

J. *Process Management*

1) *Description*

Traceability information supports process management, because it helps to identify problems in the development process and their reasons. The planning and establishment of measurements of improvement can be targeted directly at the causes of the problems.

2) *Links needed*

Traceability links between the requirements and the tasks are necessary to support process management.

3) *Activities during development time*

If a developer has problems while implementing a certain requirement, this affects the overall process to implement the release. The project manager is able to identify the tasks that were not completed in time and find out the reasons for the problems.

4) *Activities during runtime*

Process management is not useful during runtime.

V. Discussion

In this section we summarize the insights from the tables above about the usage of traceability links at runtime.

As one can see from TABLE I traceability links between requirements and code are particularly useful at runtime. But all the other links between requirements and system artifacts (in contrast to the tasks which are project artifacts) are also useful, especially during change management. After an adaptation the DAS can use the links to highlight development artifacts relevant to the current variant of the release. Thus, the DAS monitors the knowledge important for the current variant. Similarly, (as can be seen in TABLE II) for the allocation usage at runtime, the DAS uses the links for monitoring. The other runtime uses are more directly related to the execution of the DAS: the re-use usage is a key mechanism needed in service-based systems. During the usage for acceptance, change management (side effects) and quality management, the link from code to requirements is used to provide more knowledge to the DAS which supports the execution of the DAS: the test cases for runtime verification and the side effects and criticality information to support the adaptation.

In our view, implementing the monitoring and the execution support is straightforward. However, one could also think of more elaborate change management usages, where the artifacts are not only highlighted, but also adapt themselves. Furthermore, we have so far only explored whether development time usages could be transferred to runtime. It might be that totally new usages of traceability links at runtime can be identified, when we have better understood the implications and implementations of DAS.

We have argued above that only part of the traceability information is helpful during runtime. However, we think that this information is particularly useful, as it does not require any further preparation of the requirements. While other approaches for requirements at runtime [1] require an extended or formalized representation of the requirements, the usage described above only uses the link as the formal element. This information is available and ready to use and does not require a specific requirements representation.

VI. Conclusion

In this paper we have discussed how specific requirements traceability information can be used during runtime. Based on traceability aspects representing benefits at development time we have identified possible uses for some links at runtime. These uses do not require a particular requirements representation (formalized or natural language) besides the traceability links. In our view this broadens the understanding of requirements at runtime as well as our understanding of traceability. As already argued in [3] for the case of built-in test, the support of runtime activities supports also the performance of these activities at development time. In that case, integration test was supported through insights from built-in test. In the case of traceability links, we believe that an approach for using the links at runtime will also support the automated performance of these activities at development time.

References

[1] Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A., Letier, E.: Requirements Reflection: Requirements as Runtime Entities. In ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, New York, NY, USA, pp. 199–202 (2010)

[2] Berry, D. M., Cheng, B. H. C., Zhang, J.: The Four Levels of Requirements Engineering for and in Dynamic Adaptive Systems. In: REFSQ '05: Proceedings of the 11th International Workshop on Requirements Engineering Foundation for Software Quality (2005)

[3] Brenner, D., Atkinson, C., Malaka, R., Merdes, M., Paech, B., and Suliman, D.: Reducing verification effort in component-based software engineering through built-in testing. Information Systems Frontiers, Vol. 9, Issue 2-3, pp. 151–162 (2007)

[4] Cheng, B. H. C., Giese, H., Inverardi, P., Magee, J., de Lemos, R.: Software Engineering for Self-Adaptive Systems: A Research Road Map. In Dagstuhl Seminar Proceedings (Schloss Dagstuhl, Germany - Leibniz - Zentrum fuer Informatik, Germany) (2008) http://www.dagstuhl.de/08031/

[5] Gotel, O., Finkelstein, A.: An Analysis of the Requirements Traceability Problem. In: Proceedings of the International Conference on Requirements Engineering, Colorado Springs, CO, USA, pp. 94–101 (1994)

[6] Helming, J., Koegel, M., Naughton, H.: Towards Traceability from Project Management to System Models. In TEFSE '09: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering, Washington, DC, USA, pp. 11–15 (2009)

[7] von Knethen, A., Paech, B.: A Survey on Tracing Approaches in Practice and Research. IESE (2002)

[8] Maletic, J. I., Collard, M. L.: TQL: A Query Language to Support Traceability. In TEFSE '09: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering, Washington, DC, USA, pp. 16–20 (2009)

[9] Pohl, K.: Requirements Engineering, dpunkt.verlag (2008)

[10] Ramesh, B.: Factors Influencing Requirements Rraceability Practice. Communications of the ACM, Vol. 41, No. 12, pp. 37–44 (1998)

[11] Spanoudakis, G., Zisman, A.: Software Traceability: A Roadmap. In: Handbook of Software Engineering and Knowledge Engineering, World Scientific Publishing, pp. 395–428 (2004)

[12] Welsh, K., Sawyer, P.: Requirements Tracing to Support Change in Dynamically Adaptive Systems. In REFSQ '09: Proceedings of the 15th International Working Conference on Requirements Engineering: Foundation for Software Quality, pp. 59–73, Springer, Heidelberg (2009)