Is a Design Rationale Vital when Predicting Change Impact? – A Controlled Experiment on Software Architecture Evolution

Lars Bratthall[1], Enrico Johansson[2] and Björn Regnell[1]
[1]Dept. of Communication Systems, Lund University, Sweden
[1]Ericsson Mobile Communications, Lund, Sweden
{lars.bratthall, bjorn.regnell}@telecom.lth.se, enrico.johansson@ecs.ericsson.se

Abstract. Software process improvement efforts often seek to shorten development lead-time. A potential means is to facilitate architectural changes by providing a design rationale, i.e. a documentation of why the architecture is built as it is. The hypothesis is that changes will be faster and more correct if such information is available during change impact analysis. This paper presents a controlled experiment where the value of having access to a retrospective design rationale is evaluated both quantitatively and qualitatively. Realistic change tasks are applied by 17 subjects from both industry and academia on two complex systems from the domain of embedded real-time systems. The results from the quantitative analysis show that, for one of the systems, there is a significant improvement in correctness and speed when subjects have access to a design rationale document. In the qualitative analysis, design rationale was considered helpful for speeding up changes and improving correctness. For the other system the results were inconclusive, and further studies are recommended in order to increase the understanding of the role of a design rationale in architectural evolution of software systems.

1 Introduction

Improvement of the software design process may include the introduction of activities related to better and more elaborated documentation of a system's architecture. Typically, architecture documentation describes *what* is present in the architecture in terms of its constituents. It may also be beneficial to spend time on the creation of Design Rationale (DR) documentation, i.e. descriptions of *why* a software architecture is built as it is. A major argument for introducing DR, is its potential benefit as a tool when making changes to the system in the context of architectural evolution. A change task includes the activity of *change impact analysis*, where changed and added architectural components are identified [3], and it can be argued that the rationale for the architecture is vital information when analysing how new or changing requirements impact the design. However, all expansions of the software development process may affect lead-time negatively, as there of course is a cost associated with documenting a system. This cost must obviously be balanced to the potential gain.

This paper presents an empirical study on the effectiveness of DR in software evolution. The main research question is: How effective is DR as a support for change

impact analysis? The main motivation for the presented research relates to the general need for a better understanding of how process changes relate to product and process quality. In the presented research, the product quality in focus is *maintainability* and the process quality in focus is *development lead-time*.

The importance of having a short product development lead-time has been argued by many, e.g. [11, 26, 35]. A short product development lead-time increases access to the market introduction window. This in its turn increases the chance of market dominance [34] and decreases the risk of market lockout [28]. Being early on the market also enhances the market's education on a particular product, and increases the likelihood that market survey information still is valid at actual market introduction [31]. Not being late to market is also an issue for shareholders; in a study reported in [16], the average stock value drop was more than five percent when an announced product was later than expected. In [4], a company witnessed that being only one week too late to market resulted in the total annihilation of the potential market, and several million dollars of software investments were wasted. These are examples of an important quality of service that software companies are faced with today: Extremely short time to market or time to customer. Improvements to the software development processes is one way of increasing the development speed.

Short lead-time is, however, not only a requirement for the first member of a product family. For example, in the mobile phone industry, new versions built on existing products have to be release regularly to keep the consumer interest high, and new services that work together with existing systems must be regularly introduced at a high pace.

Maintainability and controlled evolution of a system is dependant on the understanding of what is currently present, as changes in design are affected by the prior design [12]. This understanding is perhaps best acquired through experience in working with the system, or at least through communication access to its original developers. Unfortunately, it is not always possible to communicate with the designers of the original system architecture, or they may have problems recalling the structure of the system. Examples include systems developed with a high staff turn-over and systems made by consultants that leave the contractor after the first version of a system has been accepted. For these systems, it is desirable to establish communication which is available over time, and a documentation of DR is, in this situation, assumed to be important for the understanding of the present system architecture.

The problems that occur when changes are being made to poorly understood problems have been highlighted by e.g. [5, 7]. An important problem is architectural erosion [25]: A system that is being changed when the architecture has not been understood erodes into an entity where new change requests are becoming harder and harder to fulfil, and eventually a change request is either impossible to accommodate, or it results in more new errors than those potentially being fixed. This is a serious problem in domains where lead-time is an important issue, since lead-time accelerating activities such as product-family or product-line reuse requires a strong grip of the architecture. In the light of these arguments, we have designed a controlled experiment to investigate the hypothesis of DR documentation being an effective means for supporting change impact analysis.

The paper is structured as follows. Section 2 gives a brief description of the selected approach to design rationale documentation. Section 3 explains the variables and hypothesis being studied, as well as the design of the experiment. In Section 4 the data from the experiment is analysed, and in Section 5 the results are interpreted in relation to the hypothesis of the experiment together with conclusions and issues of further research.

2 Object of Study: A Specific Approach to Design Rationale Documentation

DR was early suggested as an important part of software architecture [25], and in [1] the importance of documenting DR is also recognized. According, to Shum [29], DR research originates from argumentation formalisms (e.g. [32, 33]), hypertext representational technology [8, 14, 24], design research [10] and software engineering. This paper focuses on software engineering aspects, as it investigates the potential business value of DR in an actual software design environment.

A DR is a kind of documentation, that not necessarily must be produced before the first release of a system. Instead, it can be written after the development of the first generation of a system as a retrospective DR [29], albeit with the risk that the DR is biased by the person writing the DR [21]. A retrospective DR does not delay the market introduction of the first generation of a system, as does a narrative DR [29], written during the initial development.

There are many approaches to both representing and manipulating a DR [17, 20]. A weakness in some approaches to DR, e.g. QOC [29], rIBIS [27], is that they may need substantial training in order to be used effectively. Another weakness is the weak association between a DR document and the actual code. This is a problem, since designers usually dislike making additional documentation [29]. Also, it has been noted that some kind of style guide is necessary if a good DR should be produced, and a structure of DR has been asked for in [29].

Our approach to DR is very simple, yet it tries to address the previously discussed deficiencies: For each aggregation level [6] in a system, the original designer should write a short note on why it is broken down into the lower aggregation levels, and what the purpose of each component is. This ensures that the DR has a strong association with the code, and that the DR's size is limited which is not the case for e.g. a QOC DR [18]. On a system level, comments should be made on only four standardized headings: Organization of system into files, use of language constructs, main dynamic architectural principles and finally, clues to understanding the system. Together, the DR provides explicit information for Kruchten's [19] development view and implicit information on any of the logical, the synchronization, and the physical views. The value of structuring the DR according to the 4+1 View Model [19] is that its information content is field-proven to be useful in architecture level design. The reason why architectural level design is addressed is that during this level of design, decisions with system-wide implications must be taken, which both constrain a system's natural

change room, as well as facilitates changes within this room. Thus an architectural understanding is crucial for maintaining flexibility in a system during its evolution.

The suggested approach has several strengths. First of all, it requires only little training. The writing of a DR for a component at a particular aggregation level can be prompted by a code entry tool, resulting in higher likelihood of actually producing the documentation. Since the documentation follows the structure of the code, the likelihood of the appropriate DR being found when changes are being performed increases. Since the DR documentation is tightly associated with the code, the likelihood that it will be maintained when code changes increases, as a designer does not have to go through the a tedious of updating multiple documents. Apart from this, the approach scales well also to large systems, unlike e.g. QOC [18], and can easily be used with many existing case tools which is important since those generally lack the explicit ability to capture design rationale at different levels of abstraction [13].

In summary, the chosen DR approach which is the object of this empirical study includes natural language explanations of (1) the rationale for the static structure of the system, (2) the organization of the system into files and (3) rationale for other design decisions considered important, when the original designers subjectively thinks about future likely product changes.

3 Experiment Planning and Operation

The experiment is motivated by the need to reduce the time spent in changing complex systems when it is not possible to communicate with the original developers, or the original developers have forgotten much of the system at hand. The aim of this study is to evaluate the effectiveness of having access to a simple textual design description, where care has been taken to motivate why the system is designed in the way it is, i.e. a DR is available.

The experiment is defined as follows:

- Object of study: A textually represented DR combined with some system static architectural description [6] at various aggregation levels [6].
- Purpose: The effectiveness of a DR is studied. Effectiveness is measured in terms of how well change requests are being performed, as well as how long time these change requests take to fulfil.
- Perspective: Single experienced developers coming to a new system, without previous knowledge of it.
- Context: The effectiveness is measured when attempting changes to two real-time systems: A local telephony switch control software and a car cruise control system. Participants are from two groups: Senior industrial designers and a group of Ph.D. students and faculty members in software engineering.

3.1 Variables and Hypotheses

The independent variables are variables that we can control and manipulate [37]. There are two independent variables: The provision of a DR - or not, and the system where change requests are applied on. Our main hypothesis is that people design dif-

ferently when having access to a DR. Three more formal hypotheses are shown in table 1. All hypotheses are tested at $p \leq 0.10$ on a Windows PC running SPSS version 9.0. The level of significance is chosen beforehand to avoid fishing for particular results.

Table 1. Hypotheses. $Sys \in \{\text{System A, System B}\}$

Hypothesis
$H_{t,\,Sys,\,1}: \overline{t_{Change,\,Sys,\,WithDR}} \neq \overline{t_{Change,\,Sys,\,NoDR}}$ There is a difference in how long time it takes to complete the change tasks depending on whether a DR is available or not. Null hypothesis: There is no difference in this aspect.
$H_{PercOK,\,Sys,\,1}: \overline{PercOK_{Sys,\,WithDR}} \neq \overline{PercOK_{Sys,\,NoDR}}$ There is a difference in how large percentage of the required changes are correctly suggested when a DR is available and when it is not. Null hypothesis: There is no difference in this aspect.
$H_{NoExtra,\,Sys,\,1}: \overline{NoExtra_{Sys,\,WithDR}} \neq \overline{NoExtra_{Sys,\,NoDR}}$ There is a difference in how many superfluous, incorrect or unsuitable changes are suggested when a DR is available and when it is not. Null hypothesis: There is no difference in this aspect.

3.2 Design

There are 17 participants in the experiment of which 7 are industrial senior designers active in a designated software architecture group. The rest are faculty members or Ph.D. students with varying industrial experience. All participants have received training in the formality of the source models used, the form of the change requests, and how to indicate where they believe that changes are requested in order to fulfil each change request. The participants have been exposed to two or three systems, and a number of change requests for each system. One group of participants has had access to a DR, and the other group has not.

Before the experiment, all participants filled in a form describing their knowledge and experience. In total, 17 aspects of their experience and knowledge have been asked for. The experience from real-time systems and the modelling language have been used to randomize participants regarding their access or no access to a DR. The reason that these two aspects have been used to randomize participants is that these aspects have been seen in a prior experiment to have a relatively large correlation with the ability to quickly identify errors in distributed real-time systems [2] of similar complexity as those studied in this experiment.

In the beginning of the experiment, an introduction and an overview of the model language used to describe code, SDL [30], were given. SDL describes the code graphically using extended finite state machines, which can be hierarchically grouped. The introduction was guided by slides. These slides were available to all participants as handouts. These handouts were used as a reference when the code was studied, in case something in the modelling language should be unclear. Through the introduction and the model language overview, it was made sure that all participants had a reasonable understanding of the code modelling language and initial learning effects would not

affect the experiment. It was also made certain that everyone knew what to do and how to fill in the forms.

All participants were first assigned four change requests in random order, that required changes to a local telephony system – system A (described in table 2). An example change task is shown in Figure 1. The change requests are realistic requests for new services. The author writing the DRs has had no knowledge of the change tasks to come, thus reflecting reality. This seem to differentiate this study from e.g. [15] and several studies in [29].

Purpose:
The system must be updated in order to maintain speed and safe distance from other vehicles. A highly sensitive radar will be used to precisely locate the position of the cars ahead and behind. The cruise control should be able to take advantage of this new functionality. All the described functionality must be implemented.

Description:
Detailed description (e.g. it must be possible to set the distance using particular switches)

Fig. 1. Example of change task

Table 2. Description of systems

System A is a real-time system that controls the operation of a local PBX. The software is based on asynchronous processes, and it is modelled in SDL [30], which is a high-level well-defined language. The requirements specification is rather complete and well-written. The system has previously been described in [36]. The system is described at 3 aggregation levels containing seven different static software processes. The maximum number of concurrently executing software processes is 28.
System B is a real-time system written in SDL that consists of two distinct parts: i) A car cruise control and ii) a test-driver stub for i). The test-driver allows the simulation of a road and driver actions while allowing monitoring of speed and dash-board indicators. The system is in industrial use. The requirements are described on less than a half page of text. This specification is incorrect as well as incomplete - a too common situation in an industrial context. The system is described at 5 aggregation levels containing 18 different static software processes. The maximum number of concurrently executing software processes is 18.
System C is a building access control real-time system written in SDL that consists of two distinct parts: i) A part that is a central database for key-cards and their authorization and ii) A part that is the real-time handler for each door. The requirements are described on four pages, and the requirements are considered as being correct and complete. Their is also a short definition of the vocabulary used. The system is described at 3 aggregation levels containing 2 different static software processes. The maximum number of concurrently executing software processes is potentially unlimited, depending on the number of doors connected to the system. In the program provided to the participants, there are 2 concurrently executing software processes.

For practical reasons, the maximum time allowed for each change request was nine minutes, not including maximally 5 minutes for studying the extent of documentation and reading each change task. The suggested change impact is recorded in a form where it is possible to indicate components requiring internal change, as well as addition of new components at various aggregation levels. Parts of such a form is illustrated in Figure 2.

Indicate where you believe you will have to make changes for the current change task here.

What (Components in software)	An X indicates that you believe you would like to make a *change*	An X indicates that you would like to *add* a SDL-process or an SDL-block in this block
Cruise_Requirements Analysis		
Cruise_Requirements Analysis_Domain		

Fig. 2. Parts of form for recording where changes are likely, i.e. “change points”

After the change tasks related to system A, the participants have been exposed to either of two experimental designs. The reason for this is that it was seen that the first pilot experimental design did not work well - in short, there was so little time available for the change requests that the participants did not succeed at all in delivering answers to the change requests.

Design I (pilot): After the telephony switch control system, both a car cruise control system and a building access control system was provided in random order, together with four change requests for each system, with the change requests for each system in random order. The time limit for each change request was maximized to nine minutes for practical reasons. In practise, this limit proved to be far to low for the cruise control system (system B, described in table 2), and the building access control (system C) proved to be so simple that almost all participants gave equal answers. Therefore, only data from the telephony switch control systems are retained and analysed in this paper.

Design II (main run). After system A had been addressed, all participants were faced with the cruise control system. Only two change requests were provided in random order. These change tasks were sampled from a commercial patents database. The time-limit for these two change requests were maximized to thirty minutes each. In practise, this design worked well, since there was ample time to comprehend the cruise control system.

Finally, after completing all change tasks, the participants were interviewed to get some subjective data. The interviewer used a predefined order of asking defined questions, i.e. schedule-structured interviews have been performed. This kind of interview ensures that variations in answers to as large extent as possible are attributable to the respondents and not from variations in the interviews [23].

Each change task is compared to a system expert written solution, containing required change-points (both additions and changes in Figure 2). Indicated changes that are not part of the expert solution are called superfluous change points.

3.3 Threats to Validity

The validity of the findings is highly dependent on how well threats have been handled. Four types of validity threats [9] are analysed: Threats to conclusion validity, construct validity, internal validity, and external validity. The relationships between these are illustrated in Figure 2.

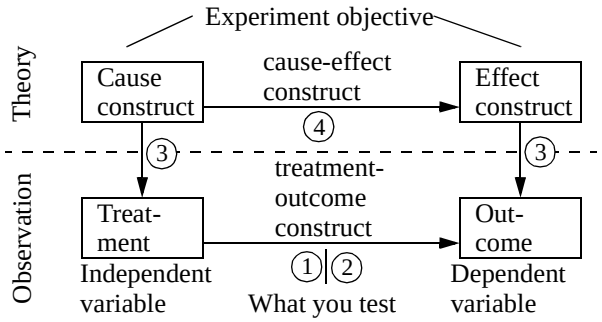

Fig. 2. Experiment principles as described in [37]

Conclusion validity. Conclusion validity (marked 1 in Figure 2) concerns the relationship between the treatment and the outcome. Care has been taken not to violate any assumptions made by the statistical tests. All time measurements have minutes as their unit, and the participants have been instructed to use the same clock during all measurements. Combined with the fact that there is no gain for a participant in adjusting their measurements, the reliability of measures should be good. There is a risk that participants have used the supplied DR in different ways, but this threat has been addressed by explicitly allowing time for studying the available documentation for each system, before the change tasks were delivered to the participants. There are no random irrelevancies (such as mobile phone calls during the experiment) that we are aware of. The threat of a large random heterogeneity of subjects have been balanced by proper randomization of participants and treatments. Each participant has been assigned individually to a group firstly depending on their self-assessed knowledge of real-time systems, secondly depending on their knowledge of system A (some participants have seen this system before). To the best of our beliefs, the individual performance has been cancelled out, as has the effect of learning the systems during the experiment.

Internal validity (2). This validity concerns matters that may affect an independent variable's causality, without the knowledge of the researcher. Maturation effects have been countered by making sure that the experiment does not take more than two and a half hour to conduct. The random order of change requests should cancel any other maturation and learning effects. There may be selection effects present, since the participants have been unusually well educated or well-experienced designers. However, we do not think that this affects the results other than that the difference between the group that had access to DR and the other group, without DR, may be smaller than in a

less experienced group of participants. There has been no mortality during the experiment. Since the experiment was given with and without DR at the same time, the control group (without DR) cannot have learned about the treatment (access to DR) in advance, and thus cannot have imitated the solution strategies used by the DR-equipped group.

Construct validity (3). Construct validity concerns whether we measure what we believe we measure. There are two main threats to construct validity: Design threats and social threats.

As there are several change tasks and two different systems, the effect of peculiarities of a single system/change task should be low. We know of no confounding level of constructs, and the participants have been randomized as to cancel any effect of having different knowledge and experience a priori the experiment. Given the careful presentation of the experiment and the full anonymity provided, there should be no social threats.

External validity (4). This last validity concerns generalization of the findings to other contexts and environments than the one studied. There may be some problem with generalizing the results to less experienced groups, such as students going directly from university to industry, since all participants are either very well-educated or have a senior designer experience level. However, we believe that the difference between the group receiving DR and the control group, without DR, should be bigger in a group with less experienced designers. The change tasks are considered realistic for system A and very realistic for system B since the change tasks are sampled from a patents database. Thus they should be representative in an industrial environment. System B as well as its accompanying documentation are industrially operational system, while system A is an educational system, but of such size and complexity that we believe it to be comparable to industrial systems.

In summary, great care has been taken in the design of the experiment, so the threats to validity should be under control.

4 Data Analysis

The purpose of this section is to present results and statistical analysis of data collected. The experiment provides quantified data. The hypotheses have been tested using the non-parametric Mann-Whitney test, at $p \leq 0.10$. The results are presented in sections 4.1 and 4.2. The subjective data from interviews are investigated in section 4.3.

4.1 Analysis of Experiment Data, System A

For the change-tasks for system A, there is a statistically significant difference in both the time spent on the change tasks as well as the quality of the predicted changes,

based on data from 57 completed change tasks. All data are illustrated in Figures 3-5, and the results of the statistic analyses are summarized in Table 3. No participants are treated as outliers in the analysis. The boxplots all show an improvement (Figures 3 and 4) or at least no clear difference (Figure 5) between the group having access to a DR, and the group that does not. The statistic tests reflect this, by rejecting the null hypotheses $H_{t,\,SysA,\,0}$ and $H_{PercOK,\,SysA,\,0}$.

The null hypothesis $H_{NoExtra,\,SysA,\,0}$ is not rejected. This is interpreted as that there is no clear difference between the two groups in this aspect. This interpretation is strengthened from figure 5.

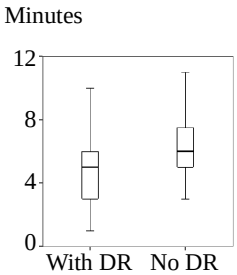

Fig. 3. System A, time used for change tasks

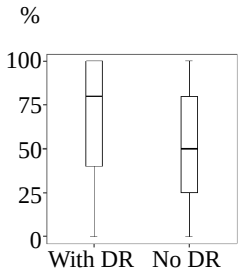

Fig. 4. System A, percentage of required change points

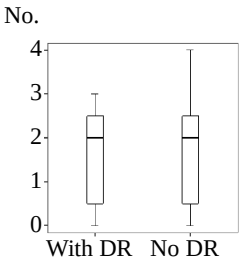

Fig. 5. System A, number of superfluous change points

Table 3. Summary of statistical analysis at significance level $p = 0.10$

	Sys = System A		*Sys* = System B	
	Mann-Whitney	Illustration	Mann-Whitney	Illustration
$H_{t,\,Sys,\,0}$	Reject	Figure 3	No reject	Figure 6
$H_{PercOK,\,Sys,\,0}$	Reject	Figure 4	No reject	Figure 7
$H_{NoExtra,\,Sys,\,0}$	No reject	Figure 5	No reject	Figure 8

4.2 Analysis of Experiment Data, System B

For the change-tasks for system B, there is a no statistically significant difference in neither the time, nor any of the quality measurements taken for the change tasks. This is based on data from 20 completed change tasks. All data are illustrated in Figures 6-8, and the results of the statistic analyses are summarized in Table 3. No participants are treated as outliers in the analysis. Judging from the medians in the figures, there is a trend showing that it is beneficial to have access to a DR. For example, the median time for accomplishing each change task decreases from 20 minutes to less than 15 minutes, while the median of correctness in answers increases. These results are similar to those from system A, which strengthens the position that a DR can be beneficial. However, the Mann-Whitney test does not detect any significant difference in any case tested.

It is possible that the statistical tests cannot detect a significant difference between the two groups, given the lower number of data-points (20), and rather small difference between the groups. Therefore, the results are inconclusive for this system.

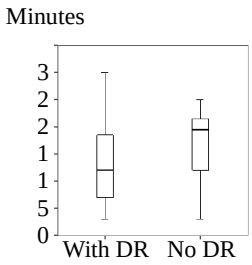

Fig. 6. System B, time used for change tasks

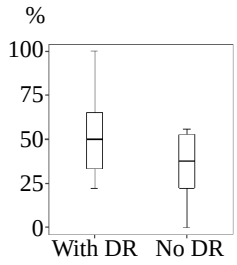

Fig. 7. System B, percentage of required change points

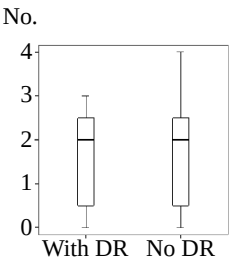

Fig. 8. System B, number of superfluous change points

4.3 Analysis of interview data

This section presents some subjective data elicited during the interviews. One of the questions asked was “How much faster can you solve the change task (comfortably well) with access to a DR?”. The results are presented in figures 9 and 10. The participants believe that there is some improvement in development lead-time with access to a DR for the less complex system A, and a high degree of improvement for the more complex system B.

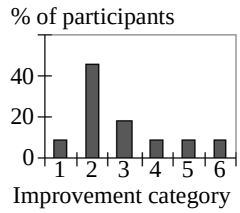

Fig. 9. System A, change in lead-time with access to DR

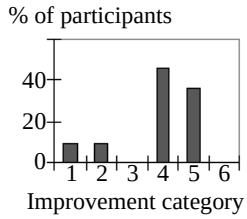

Fig. 10. System B, change in lead-time with access to DR

	Improvement categories
1	No opinion
2	0-19% faster with DR
3	20-39% faster with DR
4	40-59% faster with DR
5	60-79% faster with DR
6	80+ % faster with DR

Another question was “To what degree do you think that a DR increases your correctness in change predictions?” with results presented in figures 11 and 12. The participants indicate no or a little improvement for the less complex system A, and a much higher degree of improvement for the more complex system B.

It should be noted that the participants appreciated the DR more for the complex system B than for the less complex system A. No participants claimed that having access to a DR was harmful. Several participants witnessed that they believe that the effectiveness of the DR decreases as the system at hand gets better known.

5 Summary and Conclusions

This section gives an interpretation of the results presented.

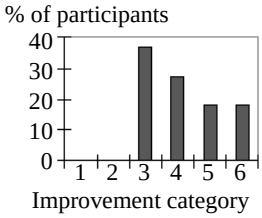

Fig. 11. System A, change in impact prediction correctness with access to DR

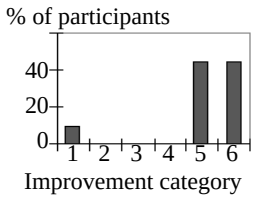

Fig. 12. System B, change in impact prediction correctness with access to DR

Improvement categories	
1	No opinion
2	It makes predictions worse
3	It does not affect correctness at all
4	I become marginally more correct
5	I become more correct
6	I become a lot more correct

Regarding system A, it did take significantly shorter time for the participants to accomplish the change-tasks when having access to a DR, and the quality of the results were significantly better or possibly equal than for the group that did not have access to a DR. These objective results are further reinforced by the subjective interview data.

Regarding system B, the picture is not as clear. The median time used for the change tasks is shorter for the group having access to DR than the non-DR group. The median percentage of correctly indicated change points is also better for the DR-group. However, this result is not statistically significant. There may be many reasons for this, such as unrealistic experimental procedures, too few data points (there are much fewer data points for system B, since we discarded all data related to this system from the pilot run of this experiment), or that qualities in the system itself affect the effectiveness of having access to a DR. Regardless, it calls for further analysis.

Information from the interviews suggests that the participants liked having access to a DR. They believed that could work both faster and better, and a Mann-Whitney test shows that the group that had access to a DR believed they would need significantly ($p \leq 0.10$) shorter time to solve the change-tasks related to system B than the group without a DR. All participants believe that they work faster and better with access to a DR, than when no DR is available, when no other documentation is available than the source code and the requirements specification.

In short, we conclude the following and suggest some future lines of work:

It is likely that having access to a DR expressed in the suggested way have a positive impact on both lead-time as well as quality when experienced designers are faced with the task of predicting where changes must be performed on an unknown real-time system. However, it is possible that there are better ways of achieving the same results using other models or ways of transferring the sought for knowledge to maintainers. For example, participants frequently indicated that they needed a static architecture overview and sequence diagrams/MSCs. Further experimentation is needed to find out what is “the best” model for various purposes.

In projects where lead-time is important it is possible that a sharp schedule prohibits the creation of models during design. In that case, writing a DR in the suggested manner after initial system release may be a cheap, yet effective way to facilitate future system evolution, without prolonging the time to initial system release. This result can easily be incorporated in standard development processes.

Acknowledgements

This work was partly funded by The Swedish National Board for Industrial and Technical Development (NUTEK), grant 1K1P-97-09690. We thank the designers at Ericsson Mobile Communication AB, Lund, Sweden for interest and participation in this study. Prof. Claes Wohlin at the Dept. of Communication Systems, Software Engineering Research Group, Lund University has given insightful comments on this paper, as well as Dr. Magne Jørgensen at the Dept. of Informatics, Industrial Systems Development Group, Oslo University. Lars Bratthall is currently at Oslo University.

References

1. Bass, L., Clements, P., Kazman, R. *Software Architecture in Practise.* Addison Wesley. 1998
2. Bauer, N., Olsson, T., Runeson, P., Bratthall, L. “Lead-time Impact of Distributed Testing on Distributed Real-time Systems”. Under submission. Lund University, Sweden. Dec. 1999
3. Bohner, S., Arnold, R. (Eds). *Software Change Impact Analysis*. IEEE Computer Society Press. 1996
4. Bratthall, L., Adelswärd, K., Eriksson, W., Runeson, P. “A Survey of Lead-Time Challenges in the Development and Evolution of Distributed Real-Time Systems”. Submitted Oct. 1999
5. Bratthall, L., Runeson, P. “Architecture Design Recovery of a Family of Embedded Software Systems - An Experience Report”. In Proc. First IFIP Working Conf. on Software Architecture. San Antonio, Texas. Feb. 1999
6. Bratthall, L., Runeson, P. “A Taxonomy of Orthogonal Properties of Software Architectures”. Proc. 2nd Nordic Software Architecture Workshop. Ronneby, Aug. 1999
7. Brooks, F. *The Mythical Man-Month: Essays on Software Engineering*. Ingram Int’l., USA. 1995
8. Conklin, J. “Hypertext: An Introduction and Survey”. IEEE Computer, Vol. 20, No. 9, pp 17-41. 1987
9. Cook, T.D., Campbell, D.T. *Quasi-Experimentation - Design and Analysis Issues for Field Settings*. Houghton Mifflin Company. 1979
10. Cross, N. “The Nature and Nurture of Design Ability”. Design Studies, Vol. 11, No. 3, pp 127-140. 1990
11. Datar, S., Jordan, C., Kekre, S., Rajiv, S., Srinivasan, K. “New Product Development Structures and Time To Market”. Management Science. Vol. 43 No. 4. Apr. 1997
12. Fisher, G., Lemke, A.C., McCall, R., Morch, A.I. “Making Argumentation Serve Design”. Human-Computer Interaction 6(3&4). 1991
13. Grundy, J. “Software Architecture Modelling, Analysis and Implementation with SoftArch”. Submitted. Jan 2000
14. Halasz, F.G. “Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems”. Comm. of the ACM, 31, pp 836-852. 1988
15. Hamilton, F., Johnson, H. “An Empricial Study in Using Linked Documentation to Assist Software Maintenance”. In Human-Computer Interaction (Interact ‘95), pp 219-24. Chapman & Hall, London. 1995

16. Hendricks, K.B., Singhal, V.R. “Delays in New Product Introduction and the Market Value of the Firm: The Consequences of Being Late to the Market”. Management Science. Vol. 43 No.4. Apr. 1997
17. Jarczyk, A.P.J., Løffler, P., Shipman III, F.M. “Design Rationale for Software Engineering: A Survey”. Proc. 25th Annual Hawaii Int’l Conf. on System Sciences. pp 577-86. 1992
18. Jørgensen, A.H., Aboulafia, A. “Perceptions of Design Rationale”. In Human-Computer Interaction (Interact ‘95), pp61-6. Chapman & Hall, London. 1995
19. Kruchten, P. “The 4+1 View Model”. IEEE Software 12(6), 1995
20. Lee, J., Lai, K.”What’s in Design Rationale?”. Human-Computer Interaction. Vol. 6 No.s 3,4. 1991
21. Lee, J. “Design Rationale Systems: Understanding the Issues”. IEEE Expert, pp 78-85. May/June 1997
22. Message Sequence Charts (MSC), ITU-T Standard Z.120. International Telecommunication Union, 1996
23. Frankfort-Nachmias, C., Nachmias, D. *Research Methods in the Social Sciences, Fourth Edition.* St. Martin’s Press, United Kingdom. 1992
24. Nelson, T.H. “A File Structure for the Complex, the Changing, and the Indeterminate”. Proc. ACM National Conference. pp 84-100. 1965
25. Perry, D.E., Wolf, A.L. “Foundations for the Study of Software Architecture”. Software Engineering Notes. Vol. 17 No. 4, pp 40-52. Oct. 1992
26. Porter, M.E. *Competitive Strategy - Techniques for Analyzing Industries and Competitors.* The Free Press, New York, USA. 1980
27. Rein, G.L., Ellis, C.A. “rIBIS: A Real-time Group Hypertext System”. Int’l. Journal of Man-Machine Studies. 24, pp 349-367. 1991
28. Shilling, M.A. “Technological Lockout: An Integrative Model of the Economic and Strategic Factors Driving Technology Success and Failure”. Academy of Management Review. Vol. 23 No. 2. 1998
29. Shum, S.J. *A Cognitive Analysis of Design Rationale Representation.* Ph.D. Thesis, York University, Great Brittain. Dec., 1991
30. Specification and Description Language (SDL), ITU-T Standard Z.100. International Telecommunication Union, 1992
31. Stalk, G. “Time - the Next Source of Competitive Advatage”. Harward Business Review. Vol. 66 No. 4. 1998
32. Stefik, M., Foster, G., Bobrow, D.G., Kahn, K., Lanning, S., Suchman, L. “Beyond the chalkboard: Computer Support for collaboration and problem solving in meetings”. Comm. of the ACM, Vol. 30, No. 1, pp 32-47. 1987
33. Toulmin, S. *The Uses of Argument*. Cambridge University Press. Cambridge, Great Brittain. 1958
34. Urban, G.L., Carter, T., Gaskin, S., Mucha, S. “Market Share Rewards to Pioneering Brands: An Empirical Analysis and Strategic Implications”. Management Science. Vol. 32 No. 6. June 1986
35. Wheelwrigt, S.C., Clark, K.B. *Leading Product Development - The Senior Manager’s Guide to Creating and Shaping the Enterprise*. The Free Press, New York, USA. 1995
36. Wohlin, C. “The Challenge of Large Scale Software Development in an Educational Environment”. Proc. Conf. on Software Engineering Education & Training, pp 40-52. Virginia Beach, Virginia, USA. 1997
37. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Boston, MA, USA. 1999