Guidance for Design Rationale Capture to Support Software Evolution

Mathias Schubanz[1],Andreas Pleuss[2],Howell Jordan[2],Goetz Botterweck[2]

[1]Brandenburg University of Technology, Cottbus - Senftenberg, Germany,
[2]Lero – The Irish Software Engineering Research Centre, Limerick, Ireland,
M.Schubanz@b-tu.de, {Andreas.Pleuss, Howell.Jordan, Goetz.Botterweck}@lero.ie

Abstract Documenting *design rationale* (*DR*) helps to preserve knowledge over long time to diminish software erosion and to ease maintenance and refactoring. However, use of DR in practice is still limited. One reason for this is the lack of concrete guidance for capturing DR. This paper provides a first step towards identifying DR questions that can guide DR capturing and discusses required future research.

Introduction Software continuously evolves. This leads over time to *software erosion* resulting in significant costs when dealing with legacy software.

Documenting *design rationale* (*DR*) can help developers to deal with the complexity of software maintenance and software evolution [4, 6]. DR reflects the reasoning (i.e., the "Why?") underlying a certain design. It requires designers to explicate their tacit knowledge about the given context, their intentions, and the alternatives considered [1]. This helps on the one hand to increase software quality and prevent software erosion based on capabilities to 1) enable communication amongst team members [6], 2) support impact analyses [7], and 3) prevent engineers from repeating errors or entering dead-end paths [1]. On the other hand, DR supports refactoring long-living systems to perform the leap towards new platforms or technologies without introducing errors due to missing knowledge about previous decisions.

In general, once documented, DR can support software development in many ways, including debugging, verification, development automation or software modification [4]. This has been confirmed in industrial practise (e.g., [2, 5]).

Problem Despite its potential benefits, systematic use of DR has not found its way into wider industrial practise. Burge [3] outlines that the lack of industrial application is due to the uncertainty connected to DR usage. There are too many barriers to capture DR accompanied by the uncertainty on its potential payoff, as DR often unfolds its full potential late in the software lifecycle. The problem of DR elicitation has been described many times [1, 4, 6]. For instance, engineers might not collect the right information [6]. This – based on the statement that DR answers questions [4] – could be due to posing the wrong or no questions. General questions in the literature, such as "Why was a decision made?", are rather unspecific and ambiguous. This can easily lead to over- or underspecified DR and compromise a developer's motivation. A first approach to guide DR capture has been proposed by Bass et al. [1]. They provide general guidelines on how to capture DR such as "*Document the decision, the reason or goal behind it, and the context for making the decision*". However, considering those guidelines, general questions (e.g., "Why?") alone are not sufficient to cover all relevant aspects and guide developers.

Our goal is provide better support for software evolution by leveraging the benefits from DR management. Hence, we aim to *integrate* guidance for DR elicitation into software design and implementation. For this, we aim to identify concrete, specific DR questions that guide engineers in capturing DR and can be used as a basis for building relevant tool support. To the best of our knowledge, concrete DR questions to ask developers have not been investigated in a systematic way yet. Until now, there is just exemplary usage of DR questions in the literature.

We aim to provide a first step in this paper by analysing DR questions that can be found in the literature up to now. For this we perform the following steps: **(1)** We perform a literature analysis and systematically collect DR questions. **(2)** We normalize the collected questions by rephrasing them. **(3)** We structure them in accordance to common decision making principles. As a result, we suggest a first set of DR questions as a basis towards guiding engineers in capturing DR.

In the remainder of this paper we describe this analysis and the resulting set of DR questions as a first basis towards guiding engineers in capturing DR. Subsequently, the paper discusses the required future work.

Question Elicitation To derive a set of specific DR questions to support software evolution we reviewed existing knowledge in DR related literature in a systematic way. Therefore, we collected all questions that we found in the literature, generalized and structured them, and eliminated duplicates.

Based on an extensive literature review, we found concrete questions for DR capturing in 19 literature sources, for instance "What does the hardware need to do?", "What other alternatives were considered?", or "How did other people deal with this problem?". This resulted in 150 questions that we collected in a spreadsheet.

In the next step, we normalised the questions: Sorting the questions reveals different interrogatives used. Most questions are "how?" (24), "what?" (73) and "why?" (24) questions. The 29 other questions could

Model Element	#	Question	Response Type
Decision	#1	What is the purpose of the decision?	Text
	#2	What triggered the decision to be taken?	Text
	#3	When will the decision be realized?	Text
	#4	What are the options?	Option[]
Option	#5	What are the actions to be done?	Action[]
	#6	What judgements have been made on this option?	Judgement[]
	#7	What are the anticipated consequences of this option?	Consequence[]
	#8	Who is responsible?	Text
Selected Option	#9	Why was this alternative selected?	Text
Rejected Option	#10	Why was this alternative not selected?	Text
Action	#11	What artefacts will be added/changed?	Text/Link
	#12	What other artefacts are related to this addition/change?	Text/Link
Add/Change Artefact	#13	What is the status before the action?	Text/Link
	#14	Why is the new/changed artefact specified in this way?	Text
	#15	Who are the intended users of the new/changed artefact?	Text
	#16	How should the new/changed artefact be used?	Text
Judgement	#17	What are the criteria according to which this judgement is made?	Criterion
	#18	Who provided the judgement?	
Consequence	#19	What are the anticipated scenarios in which this consequence may occur?	Scenario[]
	#20	What are open issues associated with this consequence?	Open Issue[]
	#21	What are risks and conflicts associated with this consequence?	Text
Open Issue	#22	What needs to be done?	Text
	#23	Who will be responsible?	Text
	#24	When will it need to be addressed?	Text
Decision Context	#25	What are the current criteria for success?	Criterion[]
	#26	What are the intended future scenarios?	Scenario[]
Criterion	#27	Which stakeholders does this criterion represent?	Text
Scenario	#28	What events could trigger this scenario?	Text

Table 1: Refined set of questions found in the literature.

be rephrased to start with an interrogative. Based on that it seems useful to consider also the other main interrogatives ("who?", "when?" and "where?") and we added them as generic questions to the overall set. In several iterations we then rephrased each question in a more generic way using one of the interrogatives and removed redundancies which resulted in 47 questions. We then further selected, summarized, and rephrased the questions according the guidelines from [1], resulting in a set of 28 questions, as shown in Table 1.

As DR is closely related to decision making concepts, the resulting questions can be grouped according to them. For instance, some questions refer to available options and others to the consequences of a decision. We structure them in a data model with, e.g., *Option* and *Consequence* as entities, and links between them.

Table 1 shows the resulting questions (middle column), structured by the identified entities (left column). The right column indicates the response type, which is either *text*, a *link* to a development artefact (e.g., a design model), or a reference to another entity. The resulting data model could be implemented as a tool-supported data model or metamodel.

Research Agenda Besides a few successful cases (e.g., [2, 5]) the use of DR in industrial practise is still an exception. One reason is the lack of information about the practitioners concrete needs. First work has been conducted by Burge [3] and Tang et al. [8] each performing a survey on the expectations and needs in relation to DR usage. They found that practitioners consider DR as important, but also that there is a lack of methodology and tool support. They also stress the need for more empirical work to close this gap.

We think that there is no *one-fits-all approach*. Therefore, Table 1 is just a first step to overcome the uncertainty connected to DR usage. As we intend to provide concrete guidance to designers for capturing DR, concrete 1) application domains, 2) team structures, and 3) the employed development processes need to be considered. Thus, to successfully guide designers when capturing DR, further work to elicit questions to be answered need to be developed under the consideration of these three dimensions. If this is not done carefully, DR questions will remain on an abstract level. Hence, they would merely serve as a guideline for DR capture (similar to [1]) instead of concrete guidance to DR capture.

Within future work we intend to get more insights into the process of DR documentation by taking the three dimensions from above into account. Software engineering in regulated domains, including certification-oriented processes seems to be a promising candidate as the need for careful documentation is already well established and first successful industry cases exist (e.g. [2]). Hence, we aim to focus on the automotive domain as a first starting point and intend to create questionnaires and perform interviews with practitioners of corresponding industry partners.

Acknowledgments

This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero – the Irish Software Engineering Research Centre http://www.lero.ie/.

References

[1] L. Bass, P. Clements, R. L. Nord, and J. A. Stafford. Capturing and using rationale for a software architecture. In *Rationale Management in Software Engineering*, pages 255–272. Springer, 2006.

[2] R. Bracewell, K. Wallace, M. Moss, and D. Knott. Capturing design rationale. *Computer-Aided Design*, 41(3):173 – 186, 2009.

[3] J. E. Burge. Design rationale: Researching under uncertainty. *AI EDAM (Artificial Intelligence for Engineering Design, Analysis and Manufacturing)*, 22(4):311, 2008.

[4] J. E. Burge, J. M. Carroll, R. McCall, and I. Mistrk. *Rationale-Based Software Engineering*. Springer, 2008.

[5] E. J. Conklin and K. C. B. Yakemovic. A process-oriented approach to design rationale. *Human–Computer Interaction*, 6:357–391, 1991.

[6] A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech. Rationale Management in Software Engineering: Concepts and Techniques. In *Rationale Management in Software Engineering*, pages 1–48. Springer, 2006.

[7] J. Liu, X. Hu, and H. Jiang. Modeling the evolving design rationale to achieve a shared understanding. In *CSCWD*, pages 308–314, 2012.

[8] A. Tang, M. A. Babar, I. Gorton, and J. Han. A survey of architecture design rationale. *Journal of Systems and Software*, 79(12):1792–1804, 2006.