Design Rationale for Software Maintenance
(Doctoral Symposium – Abstract)

Janet E. Burge
Artificial Intelligence in Design Research Group
Department of Computer Science
Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609 USA
508-831-5006
jburge@cs.wpi.edu

1. The Problem

For a number of years, members of the Artificial Intelligence (AI) in Design community have studied *Design Rationale* (DR), the reasons behind decisions made while designing. Standard design documentation consists of a description of the final design itself: effectively a "snapshot" of the final decisions. Design rationale offers more: not only the decisions, but also the reasons behind each decision, including its justification, other alternatives considered, and argumentation leading to the decision [1]. This additional information offers a richer view of both the product and the decision-making process by providing the designer's intent behind the decision. DR is invaluable as an aid for revising, maintaining, documenting, evaluating, and learning the design.

Being able to keep track of what decisions were made, and why, is an important benefit of having rationale and would be especially valuable for software maintenance. One reason for this is that the software lifecycle is a long one. Large projects may take years to complete and spend even more time out in the field being used (and maintained). Maintenance costs can be more than 40 percent of the cost of developing the software in the first place [2]. The panic over the "Y2K bug" highlighted the fact that software systems often live on much longer than the original developers intended. Also, the combination of a long life-cycle and the typically high personnel turnover in the software industry increases the probability that the original designer is unlikely to be available for consultation when problems arise.

If rationale has such potential value, then why is it not in widespread use? One difficulty, despite a good deal of research, is the capture of design rationale. Recording all decisions made, as well as those rejected, can be time consuming and expensive. The more intrusive the capture process, the more designer resistance will be encountered.

Documenting the decisions can impede the design process if decision recording is viewed as a separate process from constructing the artifact [3]. Designers are reluctant to take the time to document the decisions they did not take, or took and then rejected [4]. A real danger is the risk that the overhead of capturing the rationale may impact the project schedule enough to make the difference between a project that meets its deadlines and is completed versus one where the failure to meet deadlines results in cancellation [5].

The key to making the capture worthwhile, as well as providing requirements for DR representation, is the use for, *and usefulness of*, the rationale. There are a number of potential uses for DR. These include:

- *Design verification* – using rationale to verify that the design meets the requirements and the designer's intent.
- *Design evaluation* – using rationale to evaluate (partial) designs and design choices relative to one another to detect inconsistencies.
- *Design maintenance* – using rationale to locate sources of design problems, to indicate where changes need to be made in order to modify the design, and to ensure that rejected options are not inadvertently re-implemented.
- *Design reuse* – using rationale to determine which portions of the design can be reused and, in some cases, suggest where and how it should be modified to meet a new set of requirements.
- *Design teaching* – using rationale to teach new personnel about the design.
- *Design communication* – using rationale to communicate the reasons for decisions to other members of the design team.
- *Design assistance* – using rationale to clarify discussion, check impact of design modifications, perform consistency checking and assist in conflict

mitigation by looking for constraint violations between multiple designers.

- *Design documentation* – using rationale to document the design by offering a picture of the history of the design and reasons for the design choices as well as a view of the final product.

Because *use* is the key behind the value of the rationale, the focus of this work is on how rationale can be *used* to assist in software maintenance.

2. Relevant Research

How the DR can be used depends on its representation format and content [1]. Design Rationale representations vary from informal representations such as audio or video tapes, or transcripts, to formal representations such as rules embedded in an expert system [4]. A compromise is to store information in a semi-formal representation that provides some computation power but is still understandable by the human providing or using the information.

Semi-formal representations are often used to represent argumentation. Argumentation notations provide a structure to indicate what decisions were made (or not made) and the reasons for and against them. Some examples are Questions, Options, and Criteria (QOC) [6], Issue Based Information Systems (IBIS) [4], and DRL (Decision Representation Language) [7].

There are also many different ways to capture DR. One approach is to build the rationale capture into a system used for the design task. One example is RCF (Rationale Construction Framework) [8], which integrates DR capture into an existing design tool.

DR has a variety of uses. Systems such as JANUS [3], critique the design and provide the designers with rationale to support the criticism. Others, such as SYBIL [7], verify the design by checking that the rationale behind the decisions is complete. C-Re-CS [9] performs consistency checking on requirements and recommends a resolution strategy for detected exceptions.

There has also been work on using design rationale in software design. DRIM (Design Recommendation and Intent Model) was used in a system to augment design patterns with design rationale [10]. Co-MoKit [11] uses a software process model to obtain design decisions and causal dependencies between them. WinWin [12] aims at coordinating decision-making activities made by various "stakeholders" in the software development process. Bose [13] defined an ontology for the decision rationale needed to maintain the decision structure. The goal was to model the decision rationale in order to support decision maintenance by allowing the system to determine the impact of a change and propagate modification effects.

Less work has been done to study the usefulness of DR. Field trials were done using itIBIS and gIBIS for software development at NCR [4]. Capturing rationale was found to be useful during both requirements analysis and design. In particular, several errors were found during design that would not have been uncovered until much later when the code was written. IBIS also helped with team communication by making meetings more productive. A study was also performed using DR documents to evaluate a design [14]. In this study, 50% of the designers' questions were about the rationale behind the design and 41% of these questions were answered using the recorded rationale.

3. The Approach

There are several different types of changes that may be made during maintenance. These include correcting implementation errors ("bug fixing"), correcting design flaws, and enhancing the system. Design rationale has a number of potential uses for documentation, evaluation, and assistance during all types of software maintenance

Design rationale could be generated at any stage of the design process and describe many different types of decisions:

- *Requirements* – rationale could exist for the existing requirements and for requirements that were considered but then rejected. There will be rationale for the user interface design if the design was performed during the requirements phase.
- *Analysis* – rationale could be associated with use-cases and with the partitioning of the problem into analysis classes and collaboration diagrams.
- *Design* – rationale could be associated with any portion of any design artifact. This could include reasons behind the choice of the design classes, the attributes (including reasons for data types and visibility), the methods, etc.
- *Implementation* – rationale could describe the choice of algorithms, data structures, persistent storage, and more.
- *Maintenance* – rationale could describe both why the modifications were necessary and the reasons behind the design and implementation choices necessary for the modification.

Figure 1 shows the development phases and the rationale that could be generated during each of them. Capturing all this information would present a significant amount of overhead to the software developer. We will initially assume that all the necessary rationale is available.

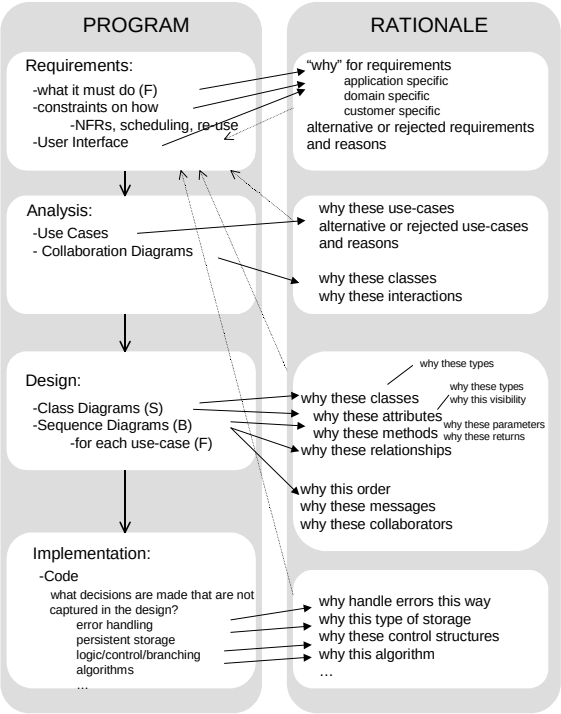

Figure 1: Software Development Phases and Rationale

To drive and evaluate this research, we will develop a system that supports the maintainer. This system will present the relevant DR when required and allow entry of new rationale for the modifications.

The new DR will then be verified against the existing DR to check for inconsistencies. There are several types of checks that should be made: structural checks to ensure that the rationale is complete, evaluation, to ensure that it is based on well-founded arguments, and comparison to rationale collected previously for similar changes to see if the same reasoning was used. In the latter check, the previous rationale could be used as a guide in determining the rationale for the new change. The system will also propagate any necessary changes to the existing DR as well as alerting the maintainer if the code modifications are the same as those made earlier and then rejected.

Our research, and development of this system, will require examining the following questions:

- *How can rationale be used to assist the maintainer?* We hope to use rationale for retrieval, evaluation, traceability to requirements, impact assessment, and as a bridge to other affected portions of the design.
- *How can decisions be represented with enough specificity to be useful yet still general enough to allow for inferencing?* We plan to develop a hierarchy of reasons for modification that can be used at different levels of abstraction to allow comparisons during inferencing.
- *Does rationale differ for different types of software modifications?* We believe that rationale will be used and created at different levels of the design process depending on the type of modification.
- *Does maintenance rationale differ from original rationale?* We expect maintenance rationale to be similar in structure, with reasons for change that are unique to modification.
- *Are there portions of the design or phases of the design process (Figure 1) where rationale capture would be more useful than others?* We expect this to be true. Rationale at early phases is important because the decisions affect more of the resulting implementation. These decisions, however, are less likely to be altered than those later in the process.
- *What is the relationship between rationale collected at different phases? Is it only via the design artifacts?* We have yet to see a direct relationship between rationale at different phases except by using common arguments for and against decisions.

4. Current Status

Inferencing Over Rationale

Some preliminary work has been done in determining ways to inference over the rationale. This resulted in a prototype system, InfoRat [15]. InfoRat supported validation of the rationale and evaluation of the design. Validating the rationale involves verifying that the rationale is structurally complete and that there are no discrepancies, such as decisions made that had no arguments in their favor. Validation is important because explicitly justified decisions may indicate that decisions were well thought out.

InfoRat used rationale to evaluate the design by checking to see if the decisions made were well supported. For example, if a decision has more, or stronger, arguments against it than for it then it may not be the best choice. Also, there may be alternatives that have more support than the ones that were chosen – this may indicate that there is either missing rationale or that the choice made should be reconsidered.

InfoRat demonstrated that intelligent reasoning over DR can provide more beneficial use for the collected DR than just its retrieval and presentation. Such reasoning can provide strategic guidance for the design process as well as a novel way of checking for design quality, as designs with poor rationale are less likely to be of high quality.

Rationale for Software Maintenance

A study was recently conducted to investigate rationale for different phases of software development and how it is used and modified during software maintenance. Three types of modifications were examined: correcting implementation errors (bugs), correcting design flaws, and adding enhancements. The system being modified was designed using a simplified version of the Unified Process [16]. The design artifacts consist of requirements, an initial user interface design, use-cases, collaboration diagrams, class diagrams, event trace diagrams, and source code. Modifications of each type were made and the structure, content, and use of the rationale were studied. This exercise resulted in a better understanding of what DR was for software and provided a research agenda for future work.

5. Expected Results

In the course of this research, and based on the study described earlier, we plan to produce the following:

- A categorization of different ways to use DR during maintenance and what has to be done with the DR to support these uses.
- A method for propagating changes made during maintenance through the rationale to ensure that it is kept current and to capture how the rationale evolves over time.
- A representation for rationale occurring at multiple levels in the development process from requirements through maintenance.
- A design rationale ontology that supports inferencing by indicating the relationships between arguments at different levels of abstraction.
- A way of attaching the rationale to the development artifacts (diagrams and code) so that it can be presented to and modified by the user.
- A prototype system that uses these methods to support the maintainer.

Acknowledgements

I would like to thank my advisor, David C. Brown, for his invaluable help and encouragement.

References

[1] J. Lee, "Design Rationale Systems: Understanding the Issues", *IEEE Expert*, Vol. 12, No. 3, 1997, pp. 78-85.

[2] F.P. Brooks Jr., *The Mythical Man-Month*, Addison Wesley, MA, 1995

[3] G. Fischer, A. Lemke, R. McCall, and A. Morch, "Making Argumentation Serve Design", in *Design Rationale Concepts, Techniques, and Use*, T. Moran and J. Carroll, (eds), Lawrence Erlbaum Associates, NJ, 1995, pp. 267-294.

[4] J. Conklin and K. Burgess-Yakemovic, "A Process-Oriented Approach to Design Rationale", in *Design Rationale Concepts, Techniques, and Use*, T. Moran and J. Carroll, (eds), Lawrence Erlbaum Associates, NJ, 1995, pp. 293-428.

[5] J. Grudin, "Evaluating Opportunities for Design Capture", in *Design Rationale Concepts, Techniques, and Use*, T. Moran and J. Carroll (eds), Lawrence Erlbaum Associates, NJ, 1995, pp. 453-470.

[6] A. MacLean, R.M. Young, V. Bellotti and T.P. Moran, "Questions, Options and Criteria: Elements of Design Space Analysis", in *Design Rationale Concepts, Techniques, and Use*, T. Moran and J. Carroll (eds), Lawrence Erlbaum Associates, NJ, 1995, pp. 201-251.

[7] J. Lee, "SIBYL: A qualitative design management system.", in *Artificial Intelligence at MIT: Expanding Frontiers*, P.H. Winston and S. Shellard (eds), MIT Press, MA, 1990, pp. 104-133.

[8] K. Myers, N. Zumel, and P. Garcia, "Automated Capture of Rationale for the Detailed Design Process", *Proc. of the 11th National Conf. on Innovative Applications of Artificial Intelligence*, AAAI Press, CA, 1999, pp. 876-883.

[9] M. Klein, "An Exception Handling Approach to Enhancing Consistency, Completeness and Correctness in Collaborative Requirements Capture", *Concurrent Engineering Research and Applications*, Technomic Publishing Company, PA, 1997, pp. 73-80.

[10] F. Peña-Mora, and S. Vadhavkar, "Augmenting design patterns with design rationale", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 11, Cambridge University Press, UK, 1996, pp. 93-108.

[11] B. Dellen, K. Kohler, and F. Maurer, "Integrating Software Process Models and Design Rationales", *Proc. of the Conf on Knowledge-based Software Engineering*, IEEE Computer Society Press, 1996, pp. 84-93.

[12] B. Boehm, and P. Bose, "A Collaborative Spiral Software Process Model Based on Theory W", *3rd International Conf. on the Software Process*, IEEE Computer Society Press, CA, 1994, pp. 59-68.

[13] P. Bose, "A Model for Decision Maintenance in the WinWin Collaboration Framework", *Proc. of the Conf. on Knowledge-based Software Engineering*, IEEE Computer Society Press, CA, 1995, pp. 105-113.

[14] L. Karsenty, "An Empirical Evaluation of Design Rationale Documents", *Proc. of the Conf. on Human Factors in Computing Systems*, ACM Press, NY, 1996, pp. 150-156.

[15] J. Burge, and D.C. Brown, "Inferencing Over Design Rationale", *Artificial Intelligence in Design '00*, J. Gero (ed.), Kluwer Academic Publishers, Netherlands, 2000, pp. 611-629.

[16] I. Jacobson, G. Booch, J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, MA, 1999.