Available online at www.sciencedirect.com
ScienceDirect
The Journal of Systems and Software 81 (2008) 395–413
The Journal of Systems and Software
www.elsevier.com/locate/jss

Software Engineering Using RATionale

Janet E. Burge [a,*], David C. Brown [b]

[a] Department of Computer Science & Systems Analysis, Miami University, Oxford, OH, USA
[b] Computer Science Department, WPI, Worcester, MA, USA

Received 13 October 2006; received in revised form 15 April 2007; accepted 22 May 2007
Available online 25 May 2007

Abstract

Many decisions have to be made when developing a software system and a successful outcome depends on how well thought out these decisions were. One way that the decisions made, and alternatives considered, can be captured is in the rationale for the system. The rationale goes beyond standard documentation by capturing the developers' intent and all alternatives considered rather than only those selected. While the potential usefulness of this information is seldom questioned, it typically is not captured in practice. We feel that the key to motivating capture is to provide compelling uses and tool support integrated with the development environment. Here we describe the Software Engineering Using RATionale system which inferences over the rationale to evaluate decision alternatives and perform impact assessment when requirements, development criteria, and assumptions change.

Keywords: Assumptions; Inference; Rationale; Requirements; Software maintenance

1. Introduction

Any system development effort, software or otherwise, requires making a series of decisions in order to determine what to build and then how to build it. The success of the system is dependent on the results of these decisions. How good were the alternatives chosen? Were alternatives selected for the right reasons? Did the developers investigate more than one alternative? Were there assumptions made that no longer hold true?

The answers to these questions, and more, can be found in the *rationale* for the system. Rationale differs from other types of documentation because it documents more than the results of each decision: it documents what the decisions were, what alternatives were considered and rejected, and what arguments were used in making the alternative selections. This serves as more than just a snapshot of the results of the decisions, it provides the intent of the decision-makers (Sim and Duffy, 1994).

The rationale can serve as a form of corporate knowledge by providing insight into the history and reasoning behind the system. This information is especially valuable during software development. Software often requires change after delivery to either repair problems or add new functionality (Bennett and Rajlich, 2000). Problems with sharing expertise increase as companies outsource their software development projects and rely on external consultants in addition to or instead of in-house developers (Edwards, 2003).

Rationale, often referred to as Design Rationale, has been studied for many years for a variety of applications including Engineering Design (Lee, 1997), Human Computer Interaction (Moran and Carroll, 1995), and Software Development (Dutoit et al., 2006). There seems to be widespread agreement about the importance of rationale but it is still rarely captured in practice (ASME, 2005). One reason for this is a persistent belief that the cost of capturing the rationale is high and may not outweigh the benefit of having the rationale available. To motivate the capture, and to help determine what types of rationale should be captured, this work has focused on developing ways that the rationale can be *used*.

* Corresponding author. Tel.: +1 513 5299760.
E-mail address: burgeje@muohio.edu (J.E. Burge).

0164-1212/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2007.05.004

We have developed the SEURAT (Software Engineering Using RATionale) system (Burge and Brown, 2004, 2006; Burge, 2005) to support capture, display and use of the rationale. SEURAT also inferences over the rationale to look for potential problems in both the rationale itself (such as incompleteness or contradictions) and the software system (such as poor or inconsistent choices).

In this paper, we will describe the design and implementation of the SEURAT system and our initial research results. Section 2 discusses rationale and software development. Section 3 provides our approach to using rationale to support software development. Section 4 describes the design of the SEURAT system, while Section 5 discusses its implementation. Section 6 describes how we evaluated our approach, Section 7 discusses related work, and Section 8 gives a summary and some conclusions.

2. Rationale and software development

Much of the early research focus on rationale has been on *Design Rationale*—the reasons behind decisions made when designing. In software development, while there is a specific phase that is called design, decisions are made throughout the development process starting with requirements and continuing through software maintenance and until the system is eventually retired.

2.1. Uses of rationale in software maintenance

There are many ways that rationale can be used in software maintenance. Rationale can serve as *documentation* by capturing knowledge of the original developers for use by new people joining the team. This is particularly crucial for software maintenance since the maintainers are often not the original developers (Charette et al., 1997) and may not even work for the same company (Levery, 1998). By recording those alternatives considered but not selected, rationale provides two useful services to the developer: it indicates which alternative decisions are not good, and the reasons, and also provides a list of decisions that were not chosen but that may be worth a second look. This is information that would have to be painfully recreated by trial and error if it were not present in the rationale.

Because rationale captures the relationships between decisions, it can also be used to *analyze the impact* of design changes (revisions). The rationale can be used to determine which upstream and downstream decisions would need to be revisited if the proposed design change were made. This impact analysis is very valuable as it provides insight into how difficult the change is likely to be and by ensuring that all the affected portions of the design are known so that they can be changed as needed.

Rationale can also *assist in changes* needed if the technology changes. The reasons given in the rationale can be used to infer where decisions were driven by the technology available at that time. This information can be used to see where the system requires modification to exploit new technology and indicate if decisions rejected previously should now be reconsidered.

2.2. Encouraging rationale use in software development

Rationale is only useful if the developers actually use it. If the rationale support tools are integrated into tools already used by the developers then it might be possible to present the rationale *exactly when it is needed* without extra effort from the developer. For example, if the developer is viewing source code in the editor they should be able to know when there is rationale available and be able to access it without having to bring up an additional tool or search for it in a repository. Integrated tools can also provide more active assistance, using the existing rationale to evaluate current decisions and provide feedback to the designer.

We have addressed these issues in our approach by integrating our rationale tools into a software development environment already used by many developers and maintainers. In this case, the environment chosen was the Eclipse Development Framework (www.eclipse.org), a framework used as the IDE on many Java development projects.

The developer/maintainer can view, modify, and analyze the rationale without leaving the development environment. This supports the need to have the rationale become an integrated part of the process. The rationale is associated directly with the source code that implements the decisions. Its presence is indicated to the developer/maintainer in a non-intrusive manner when the code is viewed/edited.

3. Approach

Crucial decisions are made at many points in the software development process. Documentation of these decisions, and the rationale for them, can be very useful in subsequent development phases and also when developing new systems with similarities to the current one. The primary obstacle to decision documentation is that the rationale is not easy to capture and while there are many potential uses, the tools and methods are not in place to support them.

There are many different approaches that have been proposed to assist in the capture of rationale. These include analyzing the patterns of changes made in a Computer Aided Design (CAD) tool (Ishino and Jin, 2006), using a trained facilitator to structure rationale generated during discussion (Buckingham Shumm et al., 2006), and associating code with e-mail messages that contain rationale (Zaychik and Regli, 2003). While these techniques have promise, we feel that in order for rationale to be useful (and worthy of the effort required to capture it), the key to success is to find ways in which it can be *used*. To this end, while our tools support the capture of rationale, our primary research emphasis has been on how the rationale can be used by the software developer.