Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2008), *22*, 311–324. Printed in the USA.

doi:10.1017/S0890060408000218

Design rationale: Researching under uncertainty

JANET E. BURGE

Computer Science and Systems Analysis Department, Miami University, Oxford, Ohio, USA

(Received May 12, 2007; Accepted May 9, 2008)

Abstract

Rationale research in software development is a challenging area because although there is no shortage of advocates for its value, there is also no shortage of reasons for why rationale is unlikely to be captured in practice. Despite more than 30 years of research there still remains much uncertainty: how useful are the potential benefits and how insurmountable are the barriers? Will the value of the rationale (design and otherwise) justify the cost of collecting it? Although there have been numerous rationale research projects, many, if not most, received little or no empirical evaluation. There also have not been many studies examining what the needs are of the practitioners who would be supported by the rationale. This article discusses the "doom and gloom" predictions of rationale's failure, provides a survey of evaluations of rationale systems, and discusses what we hope is a brighter outlook for rationale research in the future. There are development standards and synergistic research areas that may help with rationale research and its acceptance in the software community with which we should be working. This article also presents the results of a pilot survey of software developers who were asked how they would envision using rationale and what they believe the most important barriers are. Although some results were as expected, there were also some surprises. Research on technology transfer indicates that, among other things, to transition successfully from research into practice we need to understand the need that is being met and demonstrate the value of our approach. Until we have determined how our work is needed by the people we are trying to help we will remain researching under uncertainty.

Keywords: Design Rationale; Empirical Studies; Software Engineering; Traceability

1. INTRODUCTION

Rationale[1] research in software development is a challenging area because although there is no shortage of advocates for its value, there is also no shortage of reasons for why rationale (for design or other phases of software development) is unlikely to be captured in practice. There are numerous proposed uses for rationale: providing additional documentation, assisting new personnel in learning about the design, supporting software maintenance, and many more. There are also many barriers to its capture and use: the effort involved in capturing it, potential liability issues if decisions can be tracked, the potential for disrupting design, and so forth. Despite more than 30 years of research there still remains much uncertainty: how useful are the potential benefits and how insurmountable are the barriers?

Reprint requests to: Janet E. Burge, Computer Science and Systems Analysis Department, Miami University, 230 H Kreger Hall, Oxford, Ohio 45056, USA. Email: burgeje@muohio.edu

[1] Note that in most cases in this article, except when describing prior work specifically on DR, the more general term "rationale" is used rather than the more traditional "DR." This is because in software development, decisions are made in all phases of development and rationale is not restricted to the phase known as "design."

1.1. Rationale capture and project failure

Grudin (1996) describes the danger of adding additional costs to earlier ("upstream") phases of software development and, as part of a discussion of the benefits of upstream investment when many projects are never completed, offers this rather off-putting statement "And any project, by diverting resources to capture design rationale, may reduce its likelihood of surviving or succeeding." This dramatic assertion provides an especially gloomy picture of the future of rationale research.

Software project failure rates are indeed high. A frequently cited source for software failure information is the CHAOS Report, generated every 2 years by the Standish Group. The 1994 report (Standish Group, 1994) states that 31.1% of projects are cancelled prior to completion and 16.2% are completed on time, within budget, and with all features implemented. The remaining projects ("challenged" projects) are completed, but with time and cost overruns and often with not all of the originally planned features implemented. The situation in 2006 is somewhat improved, with 18% having failed and 29% having succeeded (Hartmann, 2006; Standish Group, 2006). Still, it is interesting to note that while only

29% can be viewed as having succeeded, the projects in the challenged category (53% in 2006) are still delivered to the customer (although with possibly reduced functionality). If the usefulness of rationale is dependent on the system being delivered, it has the *potential* to become useful on 82% of software projects.

1.2. Barriers to rationale

Horner and Attwood (2006) investigated barriers to design rationale (DR) use and divided them into four types of limitations: cognitive, capture, retrieval, and usage. The cognitive limitations arise because of human information processing limitations that make it impossible to exhaustively capture rationale. The capture limitations involve the need to collect rationale along with its context, the difficulty of eliciting tacit knowledge, lack of incentives for capturing rationale, the cost of capture versus predicted benefits, and if the collected rationale may pose a risk to the individual (exposing a bad decision) or the corporation (possible liability issues). Retrieval limitations concerned the need to determine what the content of the rationale being retrieved should be and the method of retrieval. When rationale is captured for future use, how do you predict what information will be useful? The final limitation, usage limitations, was concerned with if the applicability of rationale reuse might be constrained by the uniqueness of design problems and that there is a deficiency of methods for assessing how effective rationale is.

Exhaustive rationale capture is certainly unlikely, and is probably also undesirable. It would make far more sense to optimize resource allocation by having some rules or heuristics for which information is most likely to be useable in the future. Another option would be to follow an "incremental formalization" approach (Shipman & McCall, 1997) where information is captured in an unstructured format and formalized as needed. Capture remains a key difficulty to rationale adoption. There is little data available to indicate if the costs are, indeed, greater than its benefits. This is an area where data collection would be critical. Retrieval is an area where technology can be of assistance: integration with development tools and methodology will help capture the context along with the rationale although no technology will help us see the future. The final issue, usage, has similar issues to any reuse of information or technology yet these issues have not discouraged reuse research in the software domain. Software reuse remains a critical goal of software development.

1.3. What use at what cost?

In their article "Argumentation-Based Design Rationale: What Use at What Cost" Buckingham Shum and Hammond (1994) examined a variety of argumentation-based DR approaches to determine if they were able to meet two claims: "Argumentation-based DR is useful" and "Argumentation-based DR is usable," that is, the use and cost of DR. The first claim was examined by looking for three types of evidence: evidence that designer reasoning and deliberation were assisted by working with argumentation, evidence that argumentation recorded earlier was useful, and evidence that using the notation impeded reasoning.

For the first claim, usefulness, they were able to find some of each type of evidence, including evidence of impeding reasoning. The two examples of impeding reasoning were cases where capturing the rationale as argumentation caused designers to get sidetracked onto issues that were not important or not controversial. The examination of the second claim, usefulness, provided some evidence that the semiformal argumentation was not always easy for the designers but the overall result was that more information was needed about how designers might use this kind of an approach. The need for more empirical evidence was highlighted by this article, and the authors expressed concern that technology was being built on "a mutually constructed and self-perpetuating folklore."

1.4. So why bother?

Why should we press on despite these obstacles? Some of the doom and gloom predictions appear somewhat speculative, such as Grudin's assumption that the effort spent in capturing rationale could be the factor delaying development just enough to cause cancellation (Grudin, 1996). There are reports of unsuccessful attempts at rationale use but those results have not been published so there is no way for those who were not involved to understand what happened.

Software projects vary widely in size, scope, and process followed. Developers in companies operating at the higher levels of the capability maturity model (SEI, 1997) are used to spending time on documentation, metrics, and other types of information gathering that go beyond simply writing and debugging code: is rationale collection more arduous than those tasks? What do we need to know to move from guessing about rationale's benefits and barriers toward determining what direction is best for our research?

The remainder of this article looks at what we actually know and do not know in rationale research. Section 2 describes what techniques have been used in the past to evaluate rationale and rationale tools. Section 3 discusses why this is the right time for a renewed interest in the field. Section 4 describes the challenges we face in evaluating our research. Section 5 discusses the importance of obtaining the perspective of practitioners in the field and presents the results of a pilot survey of software practitioners to get their opinions on how, when, and by whom rationale could be used and what they perceive to be the most significant barriers. Section 6 discusses conclusions and future research.

2. EMPIRICAL EVIDENCE

What do we really know about design or other forms of rationale in use? There have been many different approaches to

https://doi.org/10.1017/S0890060408000218 Published online by Cambridge University Press

rationale capture and use proposed but most of them did not receive formal evaluation. Table 1 lists 56 approaches and how they were evaluated. For nearly half, no evaluation was described, and of the remaining approaches, the most common evaluation method was the case study, most of which were informal.

There have been a couple of studies to investigate DR use that were not intended to evaluate a specific approach (although they did use specific notations). Karsenty (1996) investigated usefulness of DR by showing DR captured by a scribe observing design meetings that was formalized into the QOC notation (MacLean et al., 1996). All documents were captured on article with one problem per page. Karsenty (1996) conducted an experiment where six designers external to the project were asked to perform a series of tasks given DR documents captured earlier by a scribe who observed design meetings. The data collected from the experiment consisted of the questions asked by the designers (168 questions total) that were grouped into six categories. Of these categories, the majority fit into the category of DR Questions. Of these questions, the DR answered 41%. The unanswered questions were analyzed to determine why the DR was insufficient. In some cases, the DR did not capture all the issues discussed at the design meetings, in some cases the questions were not answered because it involved misconceptions about the designed artifact, whereas in other cases the questions were not answered because the issues asked about were not issues recorded in the rationale because they had not come up at the design meetings.

QOC was also used in the first of three studies performed by Haynes (2006) to investigate the use of DR as explanation. The first study generated DR retrospectively from design meeting transcripts and other design artifacts. They found that it was difficult to capture the chain of reasoning and that the DR did not appear to be complete. Two additional studies were performed using scenarios and claims analysis (Carroll & Rosson, 1992). These studies appeared to indicate that DR in the form of scenarios would be effective in assisting with technology transfer.

Tang et al. (2006) point out that there is a lack of empirical research studying practitioner opinions on DR, how they capture and use DR, and what the barriers are to capturing DR. They conducted a survey that focused on how software architects viewed rationale and received valid responses from 81 architects. The survey investigated the capture and use of nine rationale elements: constraints, assumptions, weaknesses of designs, costs of designs, benefits of designs, certainty of the design, certainty of the implementation, and trade-offs considered. All of these were considered important by their respondents. The ones with the highest support were benefits, certainty, and constraints. These types were also identified as being the most frequently used. The types with the least use were rationales describing weaknesses in the design decisions (design weakness, costs, and complexity). When looking at how frequently they captured the different design elements, the respondents reported that constraints and assumptions are documented frequently. The least documented types were design weaknesses, certainty of design, and certainty of implementation. When investigating barriers, cost and time were the most frequently given reason for not documenting rationale. The overall conclusion of the article was that practitioners *do* believe that DR is important and should be captured.

3. RENEWED INTEREST: HOW AND WHY

Despite the many doom and gloom predictions, there is still a significant amount of interest in this area. There are numerous ongoing research projects, a recent book on rationale management in software engineering (Dutoit et al., 2006), another book, *Rationale-Based Software Engineering* (Burge et al., 2008), and several DR-based workshops. Has the time come for rationale to move beyond being an interesting research problem into something that can become useful in practice?

3.1. Incentives

There are still strong feelings, at least among researchers, that this is an important avenue of research and shows great promise for aiding software development. There are a number of incentives for continuing to pursue this research.

3.1.1. Capability maturity model integration (CMMI)

One incentive for reconsidering rationale use for software engineering can be found in the CMMI (CMMI Product Team, 2006) and its decision analysis and resolution (DAR) process area. The DAR is required for Level 3 of the CMMI and consists of defining a "formal evaluation process" for decision alternative evaluation. The elements of this evaluation include alternative identification, evaluation criteria determination, evaluation method selection and use, and finally, selecting the alternatives based on the identified criteria. The CMMI recommends that part of this evaluation process is defining which decision categories need this formal evaluation and how the evaluation should be performed. The formal evaluation is especially important for decisions that are identified as "high risk."

3.1.2. Knowledge management (KM)

One of the uses of rationale is the ability to capture "corporate knowledge." This is also a goal of KM. KM is critical in software engineering because "intellectual capital" is the key asset for a software organization (Russ & Lindvall, 2002). The challenges faced by the rationale community are similar to those faced by KM: encouraging knowledge sharing, providing incentives, and making knowledge capture part of business practices (Smith & Farquhar, 2000).

The importance of KM has been acknowledged by a number of organizations. Schlumberger (Smith & Farquhar, 2000) uses their intranet and a "knowledge hub" to support "communities of practice," groups of personnel within the organization where information sharing is necessary. This process is supported by a "knowledge champion" who

https://doi.org/10.1017/S0890060408000218 Published online by Cambridge University Press

Table 1. *Design rationale approach evaluations*

Approach	Use	Evaluation
	Field Trials	
Compendium (Buckingham Shum et al., 2006)	Meeting facilitation through dialog mapping	Case studies at 10 organizations (Selvin & Sierhuis, 1999)
Design Rationale editor (DREd; Bracewell et al., 2004)	Goal is to replace the traditional designer notebook	Fifty-four designers at the same aerospace company were trained and encouraged to use DREd. Thirty-two returned a questionnaire and 12 had used DREd (11 of the remaining designers said they did plan to use it).
Logging Rationale for User Interface Designs (LOUIS; Heiliades & Edmonds, 1999)	Capturing information in early software design meetings	User trials with five users
IBIS (Kunz & Rittel, 1970), gIBIS (Conklin & Burgess-Yakemovic, 1996)	Hardware and software development	Field trials at NCR
WinWin, EasyWinWin (Boehm & Kitapci, 2006)	Requires negotiation between stakeholders	Used by students EasyWinWin was used in over 100 "real-world" projects (implemented by students?). There was also a Digital Library Project case study with student developers (In et al., 2001).
Sysiphus (Dutoit et al., 2005)	Sharing system models and rationale between students	Used in software engineering courses; qualitative observations collected
	Laboratory Experiments	
Pattern mining and Architecturally Significant Information extracted from Patterns (ASIP; Barbar et al., 2006)	Software architecture	Empirical study: one observational study and two controlled experiments (mentioned but not described)
Bratthall et al. (2006) textual DR	Software architecture	Laboratory study with 17 participants: 7 industrial senior designers, remaining graduate students and faculty
Software Engineering Using Rationale (SEURAT; Burge & Brown, 2004, 2006)	Software development	Laboratory study with 20 subjects performing three maintenance tasks with or without SEURAT support; subjects were a mix of graduate students and practitioners with average work experience of 6 years.
Decision Goals and Alternatives (DGA; Falessi et al., 2006)	Software architecture	Laboratory study with 50 graduate students
Design Pattern Rationale Graphs (DPRG; Baniassad et al., 2003)	Aid in understanding how a pattern relates to design goals and how those goals are implemented in the code	Two confidence case studies, each with a single developer: one completeness study with two industrial developers and one tool user where the developers identified the design goals, the investigator developed the DPRGs; and the tool user used the DPRGs to detect six out of seven; and one "lightweightness" study with two industrial developers to time DPRG creation
Design process rationale (Brissaud et al., 2003)	Capturing design process rationale to reuse in the current project or other projects	Laboratory experiment where five design actors performed technician roles to design a mechanical system; constructing conjectures and extracting criteria were not difficult.
HERMES (Karacapilidis & Papadias, 2001)	Generic decision support	Evaluated by mechanical engineering graduate students and medical doctors; questionnaires were used to assess the usability of the system and the effectiveness of the environment.
Desperado (Ball et al., 1999)	Provide access to previously considered design options	Fifteen MS students in three sessions (Lambell et al., 2000): one only with paper and pencil, one with an experimental group using Desperado for encoding, and one with an experimental group using Desperado for encoding and retrieval. Results were described as "encouraging" but no statistical analysis was provided. An industry study was planned but results have not been published.

Table 1 (*cont.*)

Approach	Use	Evaluation
Mission Oriented Architectural Legacy Evolution (MORALE), Software Architecture Analysis Method (SAAM; Richter et al., 1998)	Software architecture	SAAM summary document and SAAMPlayer used to answer questions; experiment looked at how they used the document and video; four subjects participated (Richter & Abowd, 1999).
Design Space Analysis (DSA; MacLean et al., 1996)	Capturing DR to aid in artifact understanding	Pairs of designers worked on a design problem and were recorded; the translation was converted into the QOC notation to see how well the design discussion corresponded to DSA concepts.
Representation and Maintenance of Process knowledge (REMAP; Rhamesh & Dhar, 1994)	System design and maintenance (Ramesh & Dhar, 1994), requirements engineering (Ramesh & Dhar, 1992)	Conceptual model was derived from an empirical study with 20 experienced analysts. Evaluation of REMAP was not described.
Active Design Documents (ADD; Garcia et. al., 1993) and Multi-Agent ADD (MultiADD; Garcia & Vivacqua, 1997)	Supports parametric design and cooperative design	ADD: two tests creating a design proposal (experienced designers), two tests with design documentation users (Garcia et al., 1994); MultiADD example was described.
	Case Studies	
Architecture Rationale and Elements Linkage (AREL; Tang et al., 2007)	Software architecture traceability	Case study: the authors of the paper retrospectively captured DR for an Electronic Funds Transfer system.
Reusable Rationale Blocks (RRB; Hordijk & Wieringa, 2006)	Software design	Three case studies where RRBs were used by the authors
Design Rationale Environment for Argumentation and Modeling (DREAM; Lacaze et al., 2006)	Safety critical systems	Case study where the authors used their tool and notation to model information from meeting minutes (10 meetings discussing prototype systems)
Requirements Engineering Process Improvements (REPI; Framework (Palyagar & Richards, 2006)	Capturing rationale behind RE process improvements	Case study working with a "quality certified organization," unidentified for confidentiality reasons where the authors studied their RE process
Task-Artifact Framework (Carroll & Rosson, 1992, 2003)	Human–Computer Interaction (HCI)	Case study on the MOOsburg community network system
CodeLink (Zaychick & Regli, 2003)	Software design	Informal user case study with four two-person teams of undergraduate and graduate students who were surveyed after 3 weeks of tool use.
Reasoning Loop Model (Louridas & Loucopoulos, 2000)	Reflective design and collaborative design	Examples/case studies: one regarding a major European Electricity Supply Company, capturing rationale for several goals, the other on the evolution of the C++ language (retrospective DR capture)
Integrated Design Information System (IDIS; Chung & Goodwin, 1998)	Rationale capture and use to support design of chemical plants	Five case studies testing the tool with engineers and students
Collaborative Design Support System (C-DeSS; Klein, 1997*a*)	Design geometry	Informal: used to capture design and rationale for a fluid-field measurement device developed at a university research lab; several designers used the tool for several weeks and provided feedback.
JANUS (Fischer et al., 1996)	Kitchen design	JANUS was evaluated informally with designers and computer users at different levels performing learning and design tasks.
Reconstructive Derivational Analogy (RDA; Britt & Glagowski, 1996)	Design process is replayed using new requirements to create a new design	CPU and actual time were compared for a representative set of simple problems. It was not stated if this was tested using more than one designer.
The Inquiry Cycle (Potts et al., 1994)	Requirements analysis	Case study: the four researchers used the process for about 1 month (Potts et al., 1995).
Scenario Claims Analysis (SCA; Carroll & Rosson, 1992; Carroll, 2000)	HCI	Case studies by a number of researchers including digital library usability (Keith et al., 2002) and Interactive Development Environment evaluation (Haynes, 2006).

https://doi.org/10.1017/S0890060408000218 Published online by Cambridge University Press

Table 1 (*cont.*)

Approach	Use	Evaluation
	No Evaluation Described	
Architecture rationale using causal and structural graphs (Bass et al., 2006)	Capturing architectural rationale to see how requirements are satisfied and to determine implications of design modifications	Not described
Software Concordance Design Rationale (SCDR; Gill & Munson, 2006)	Software development	Not described
RE patterns (Hagge et al., 2006)	Sharing requirements of engineering process rationales	Unclear, examples of uses are described
FOCUS (Schneider, 2006)	Capturing knowledge about prototypes	Not described
Risk Analysis Tool (Schneider, 2006)	Risk analysis	Not described
Archium (van der Ven et al., 2006)	Software architecture	Not described
Kuaba Ontology (de Medeiros et al., 2005)	Software design	Not described
Reference Architecture Representation Environment (RARE; Barber & Graser, 2000)	Architecture derivation	Not described
Design Rationale Management (DRAMA; Brice & Johns, 1999)	Engineering design: ranks alternatives, consistency checking	Commercial product: no evaluation data provided
Rationale Construction Framework (RCF; Myers et al., 1999)	Computer-aided design; example applied it to designing a robotic arm	No formal evaluation
Design History System (Shah et al., 1999)	Engineering design	Not described
Engineering History Base (Taura & Kubota, 1999)	Engineering design (Mechanical)	Not described
Design Rationale for the Information Phase of Value Engineering (DRIVE; de la Garza & Alcantara, 1997)	Building design	Not described
Collaborative Requirements Capture System (C-ReCS; Klein, 1997*b*)	Requirements engineering	Not described
Hyper-Object Substrate (HOS; Shipman & McCall, 1997)	Network design	Not described
PHIDIAS (Shipman & McCall, 1997)	2-D, 3-D graphical design: kitchen design example is described	Not described
M-LAP (Brandish et al., 1996)	Uses actions from previous design tasks to group user interface actions into activities to either automate activities during design or predict consequences	Not described
CoMo-Kit (Dellen et al., 1996)	Software process traceability	No evaluation provided, just an example illustrating use
SHARED-Decision Recommendation and Intent Management System (SHARED-DRIMS; Pena-Mora et al., 1995)	Conflict mitigation when developing large-scale engineering systems; DRIM also used to use rationale to select software design patterns (Pena-Mora & Vadhavkar, 1997)	Not described
Documentation System (Lougher & Rodden, 1993)	Sharing software maintenance information	Not described
Galileo2 (Bahler & Bowen, 1992)	Concurrent engineering	Not described
Design Rationale Capture System (DRCS; Klein, 1992)	Airplane design example described	Not described
Process Technology Transfer Tool (PTTT; Brown & Bansal, 1991)	Used to transfer decision-making results from one group to another; major goal is information retrieval	Not described
SYBIL (Lee, 1991)	Qualitative decision management	Used on some small projects but no evaluation described
Device Modeling Environment (DME; Gruber 1990)	Electromechanical design	Not described

encourages participation. This recalls the success of IBIS, where the presence of a champion-supporting rationale was essential (Conklin & Burgess-Yakemovic, 1996). NASA is also promoting KM initiatives to capture case studies, share "stories," describe "lessons learned" and to provide an "expertise locator" (Liebowitz, 2002). DR research and KM research share many of the same benefits and barriers and should study the approaches used in KM research.

https://doi.org/10.1017/S0890060408000218 Published online by Cambridge University Press

3.1.3. Value-based software engineering (VBSE)

VBSE is another synergistic field. The idea behind VBSE is to move away from "value-neutral" approaches to software decision making (Boehm, 2006). Not all software requirements are of equal value to the stakeholder so it makes sense to take value into account when making decisions. Boehm's win–win Theory W (Boehm & Ross, 1989), a methodology supported by capturing rationale, lives at the center of his 4 + 1 theory of VBSE (Boehm & Jain, 2006), which also includes utility theory, dependency theory, decision theory, and control theory. Using value as a key driver in SE decision making requires methods for capturing the criteria that indicate what "value" is as well as methods for evaluating the value of each decision alternative. These are not trivial issues. There may be uncertainty involved in accessing value: costs may be unknown, stakeholders may have conflicting beliefs, intangible benefits are difficult to quantify (Erdogmus et al., 2006). The potentially numerous criteria affecting decisions may interact requiring tradeoff analyses (Vetschera, 2006). Much of the information needed to assess value could be provided by the rationale.

3.2. Technology and tool advances

The previous flurry of rationale research activity occurred when hypertext systems became more popular. Hypertext was a key factor in many of the systems described in Moran and Carroll's (1996) human–computer interaction-focused book. Now we have new tools promising more integrated development. One example is the Eclipse Framework (www.eclipse.org). The ability to create "plugins" that add on to an existing development environment was used in the SEURAT system (Burge & Brown, 2004) so that rationale could be captured and used in the same environment that is being used to develop software. Eclipse is also an open-source project, which means that it is easily accessible to commercial and research organizations. It is no longer necessary to have a large software tool budget to get powerful tools. Eclipse is only one of many potentially useful open-source development environments and development tools that could be extended to support rationale.

3.3. Success stories

Although much of the evidence for rationale has been collected rather informally, there have been some notable successes. The NCR field trials (Conklin & Burgess-Yakemovic, 1996) demonstrated that IBIS and gIBIS could be useful in an industrial setting. The Compendium project (http://www.compendiuminstitute.org/; Buckingham Shum et al., 2006) has been used on a number of different projects and has created a community large enough to sustain an annual workshop. The Compendium Institute e-mail group (http://tech.groups.yahoo.com/group/compendiuminstitute/) has over 1000 members as of December, 2007.

Another success story has been the DREd system (Bracewell et al., 2004). Not only did the designers at Rolls-Royce feel that using DREd helped their design process, it was also given the Rolls-Royce Research and Technology Director's Creativity Award for 2004 (http://www.eng.cam.ac.uk/news/stories/2005/rollsroyce_award/). Using DREd has been made mandatory for design scheme reviews on at least one Rolls-Royce project. The acceptance of DREd by the designers suggests that resistance to the capture of DR may not be as significant as predicted.

4. CHALLENGES IN RESEARCH AND TECHNOLOGY TRANSFER

How do we gather evidence for rationale's success or failure? Controlled experiments collecting and using rationale are challenging. The SEURAT evaluation ran into four types of problems (Burge, 2006): *task problems*, where tasks had to be limited to those that could realistically compare using rationale to not using rationale (eliminating tasks that would require rationale to be successful); *subject availability problems*, where large numbers of subjects could not be recruited for experiments that needed to be performed after working hours and that could take up to 4 h; *time problems*, where tasks had to be simplified to be possible for all levels of user (making them so simple that advanced users would not need assistance provided by rationale); and *noise problems*, where the short duration of the experiment meant that the learning curve introduced by using the rationale tool could eclipse the time saved. Task, time, and noise problems could be mitigated by using the system on more complicated tasks over a longer duration, but the subject availability problems remain. In addition, longer term tasks cannot be performed in a controlled environment that introduces the possibility of more noise.

Ideally, we would like to gather additional data on how rationale could be used in industry. The challenge is to convince practitioners that using rationale will be beneficial. Although it would be nice if developers would participate in evaluations "in the interest of science," the reality is that if they suspect that the evaluation will add to their work burden they are likely to decline participation.

Of course, that does not mean we should give up. Technology transfer is a difficult process for all new innovations in software, not just rationale. Redwine and Riddle (1985) studied technology in the mid-1980s and discovered that it took 15 to 20 years for a technology to become mature. Redwine and Riddle also identified a set of critical factors influencing technology adoption: it must be well developed, it must fill a well-defined and recognized need, it needs to be adaptable to fit user practice, there need to be reports on "prior positive experience," management must be committed, and training must be provided. Of these, two stand out: user needs and positive experience. Does rationale meet a well-defined need or is it a solution in search of a problem? Many uses have been proposed but which ones are really needed?

How can we provide "prior positive experience?" gIBIS was shown to help developers avoid mistakes and provided considerable cost savings but the researchers were not able to find other "disciples" to promote it (Curtis, 2000).

5. SURVEY OF SOFTWARE DEVELOPMENT PRACTITIONERS

Numerous articles on technology transfer point out the criticality of knowing what the technology adopter needs (Redwine & Riddle, 1985; Larson et al., 2006; McGill et al., 2006), yet, as Tang et al. (2006) point out, there has been little research that studies *practitioner views* of rationale. DR has been an active area of research for many years, yet it still has not been adopted in practice except for a few instances. As described earlier, there are many theories why this has been the case but much of this is speculation.

As Redwine and Riddle (1985) pointed out, if a technology is going to transition into practice we need to know what the users need. To get an initial assessment of how software development practitioners viewed rationale, we decided to conduct a pilot survey of software developers to gather opinions.

5.1. Survey design

We had two primary goals for the survey: to determine how software practitioners felt rationale could be the most useful, and to determine what they thought the barriers to rationale adoption were. For the first goal, we looked at how potential uses were valued, when rationale would be used, when rationale would be captured, who would be most likely to provide it and who would be most likely to use it. For the second goal, we asked questions about which barriers were most significant, and what would make rationale more appealing (and therefore motivate its capture). We also asked some questions about rationale presentation and tool integration.

In addition to the questions on rationale, we also collected demographic data about the company/industry of each respondent, their job role, and their level of experience.

5.2. Survey administration

We planned on two methods for survey administration: hardcopy surveys administered and collected at the annual Miami University Computer Science and Systems Analysis (CSA) Alumni Conference and identical Web-based surveys. Participants at the Alumni Conference were invited to attend a talk describing the research and were then invited to fill out the survey. Attendees were also given a hard copy of the talk so that those who were not able to attend the talk could still participate in the survey. This approach proved to be less than successful with only one survey filled out at the conference.

The Web-based survey invitations were sent by e-mail to 168 Miami CSA alumni, 20 professionals who had either participated in SEURAT evaluations or attended demonstrations, and were also sent out with an electronic newsletter to alumni from the author's undergraduate school, Michigan Technological University. Survey invitees were also encouraged to forward the survey to any coworkers whom they felt would be interested in participating. The online survey approach proved to be more successful than the paper one and resulted in 35 responses. The Web-based survey included a text description of what rationale was and how it could be used and a link to the presentation created for the Alumni Conference. Respondents were instructed to view the presentation prior to taking the survey. The average time taken to complete the online survey (not counting viewing the presentation or reading other information provided about rationale) was 14 min.

5.3. Demographics

The end of the survey asked a series of questions to provide demographic information about the respondents companies and level of experience. This resulted in the following information about the respondent's employers:

1. Nearly 45% of the respondents reported the average lifetime for a product developed by their company as 5–10 years. Only 14% reported longer than 10-year lifetimes.
2. Fifty percent of respondents reported the average time from conception to product release as 1–2 years. Less than 17% reported lifecycles longer than 2 years.
3. New releases of a product typically happened at least once per year (41.67% 6 months to 1 year, 38.89% less than 6 month intervals between releases).
4. Seventy-five percent of the organizations maintained the software they developed.
5. Only 25% of the respondents gave the biggest risk of software failure as mission failure (16.67%) or lives lost (8.33%). The largest number of responses was 36% where business process disruption was the greatest risk.
6. Team sizes varied with the largest number of respondents having teams of 6 to 10 people (30.56%), the next 3 to 5 people (27.78%), 2-person teams were at 16.67% and 1-person teams at 8.33%. Less than 17% had teams of greater than 10 people.

Many respondents (41.67%) did not feel their organization fit into any of the listed types (defense at 22.22%, aerospace at 2.78%, financial at 8.33%, educational at 2.78%, computer technology at 22.22%, and entertainment at 0%).

We also asked the respondents about what type of development they did: developing systems sold by their organization (50%), systems used by their organization (19.44%), and providing consulting services to other organizations (30.56%). Their current job roles were primarily technical (55.56%), followed by management (33.33%), and with 11.11% giving their role as "other" (two project managers, one VP of government

https://doi.org/10.1017/S0890060408000218 Published online by Cambridge University Press

sales, and a fourth in Web development and systems administration). Our respondents had an average of 16 years of experience, with the lowest being 1 and the highest being 30.

The survey also asked if the respondents had any prior experience using rationale tools. Most had not (86.49%). The three respondents who used rationale tools indicated SEURAT, self-made decision-tree documentation programs, and a system the respondent had built for him or herself.

5.4. Survey results

We designed the survey to receive input on two categories of questions: uses of rationale to investigate if practitioner needs matched our research goals and the perceived barriers of use to compare practitioner barriers with those described in the literature.

5.4.1. Uses of rationale

Our first set of questions looked at how and when software development practitioners thought rationale would be useful and used. The first question on the study asked the respondents to rate the importance of several uses of rationale. Table 2 gives the results shown in the same order presented to the respondents. The last column of the table shows the combined percentages of the top two levels.

No rationale uses were ranked as being not useful by more than one person. Many were ranked in the upper two levels of usefulness. For the top levels combined, 7 out of 17 uses were given a combined score of greater than 50%. Uses that appear to be the most compelling to the respondents were capturing assumptions, providing traceability to functional requirements, and assisting in impact assessment when goals change. The item of least interest was post mortem reporting.

The next five questions asked for additional information about how rationale could be captured and used. For these questions, we had the respondents rank the different options so we could determine relative importance or value with a value of one being the most likely. We give the percentages for each ranking in Tables 3–6.

This question, on use time, yielded some of the most surprising results of the survey. Many researchers, including the author of this article, predicted maintenance as the aspect of software development where rationale would be the most valuable. Instead, nearly half the practitioners surveyed ranked it as the least. It is possible that if the respondents did not have a preference between the last three elements they simply ranked them in order of appearance. It is interesting, however, that less than 20% of respondents ranked maintenance as first or second. During design appears to be the most likely with during requirements development coming in second.

These results were not surprising; most respondents felt that design would be the phase when rationale was most likely to be captured.

These results are consistent with the earlier question, and suggest that the designer is most likely to provide rationale.

The consensus appears to be that rationale is most useful during development by developers wanting to understand why someone else made a decision or in looking back on their own decisions. They felt that rationale would be the least useful for managers to review system progress.

The survey also investigated when rationale should be presented to the user. Some rationale systems, such as JANUS (Fischer et al., 1996) act as critics and present rationale when there appears to be a problem with the decisions being made; other systems, such as SEURAT (Burge & Brown,

Table 2. *Potential rationale uses*

	Not Useful	Low				High	Top Levels Combined
Capturing relationships between requirements	0.00%	5.41%	16.22%	21.62%	**37.84%**	18.92%	56.76%
Providing traceability from functional requirements to code	0.00%	0.00%	8.11%	29.73%	24.32%	**37.84%**	66.16%
Providing traceability from nonfunctional requirements to code	0.00%	8.11%	10.81%	**35.14%**	32.43%	13.51%	51.35%
Connecting the rationale to the code	0.00%	5.41%	21.62%	21.62%	**32.43%**	18.92%	51.35%
Connecting the rationale to the design documents (UML, etc.)	2.70%	5.41%	18.92%	**24.32%**	24.32%	24.32%	48.64%
Capturing assumptions made during development	2.70%	0.00%	2.70%	18.92%	**51.35%**	24.32%	**75.67%**
Performing impact assessment when system goals change	0.00%	2.70%	10.81%	21.62%	**35.14%**	29.73%	64.874%
Providing additional documentation describing the system	0.00%	2.70%	16.22%	**37.84%**	29.73%	13.51%	43.24%
Supporting project postmortem reporting	0.00%	18.92%	27.03%	**29.73%**	18.92%	5.41%	24.33%
Evaluating support for alternatives	0.00%	8.11%	16.22%	**35.14%**	29.73%	10.81%	40.54%
Supporting collaborative development by capturing deliberation from multiple developers	2.70%	5.41%	8.11%	**37.84%**	29.73%	16.22%	45.95%
Capturing deliberation during design reviews	2.70%	8.11%	16.22%	29.73%	**32.43%**	10.81%	43.24%
Capturing deliberation during code inspections	0.00%	10.81%	24.32%	27.03%	**35.14%**	2.70%	37.84%
Pointing out inconsistencies in reasoning that may indicate inconsistencies in the requirements, design, or implementation	0.00%	2.70%	8.11%	**37.84%**	27.03%	24.32%	51.35%
Teaching new personnel about the system	2.70%	10.81%	13.51%	**35.14%**	27.03%	10.81%	37.84%
Assisting with reuse by showing which portions of the system would require change to support new requirements	0.00%	5.41%	8.11%	**37.84%**	35.14%	13.51%	48.65%
Ensuring that previously rejected alternatives were not selected	0.00%	5.41%	16.22%	29.73%	**32.43%**	16.22%	48.65%

https://doi.org/10.1017/S0890060408000218 Published online by Cambridge University Press

Table 3. *Most likely use time: When do you think rationale is most likely to be used?*

	First	Second	Third	Fourth	Fifth
During requirements analysis	27.03%	24.32%	13.51%	18.92%	16.22%
During design	**43.24%**	**37.84%**	5.41%	13.51%	0.00%
During implementation	10.81%	18.92%	27.03%	21.62%	21.62%
During integration and test	8.11%	10.81%	**35.14%**	**27.03%**	18.92%
During maintenance	10.81%	8.11%	18.92%	18.92%	**43.24%**

Table 4. *Most likely capture time: When do you think rationale is most likely to be captured?*

	First	Second	Third	Fourth	Fifth
During requirements analysis	36.11%	27.78%	19.44%	13.89%	2.78%
During design	**47.22%**	**41.67%**	5.56%	2.78%	2.78%
During implementation	13.89%	11.11%	**44.44%**	19.44%	11.11%
During integration and test	2.78%	13.89%	19.44%	**50.00%**	13.89%
During maintenance	0.00%	5.56%	11.11%	13.89%	**69.44%**

Table 5. *Rationale provider: Who is most likely to provide rationale?*

	First	Second	Third	Fourth	Fifth
Requirement analyst	33.33%	30.56%	19.44%	11.11%	5.56%
Designer or architect	**55.56%**	**27.78%**	8.33%	8.33%	0.00%
Coder	11.11%	22.22%	**41.67%**	11.11%	13.89%
Tester	0.00%	13.89%	13.89%	**41.67%**	30.56%
Maintainer	0.00%	5.56%	16.67%	27.78%	**50.00%**

2006), display, or indicate, rationale when the associated artifact is being viewed or modified. Other systems store rationale but only present it to the user upon request. Nearly half (45.95%) preferred to see the rationale when they were editing or viewing an associated artifact with the remainder of the vote split between when they take an action that signifies a potential problem (24.32%) or upon request (29.73%).

There are also different mechanisms available for visualization. Some of the most successful approaches, including Compendium (Buckingham Shum et al., 2006) and DREd (Bracewell et al., 2004), use a graphical format. Others, such as SEURAT (Burge & Brown, 2006), use a tree format. Of the respondents of this survey, 67.57% preferred the tree structure and 27.03% preferred the graph. Two respondents chose "other" for their selection. One was not sure what representation would be preferable, the other said that a tree would be insufficient and that they would want to see indented free text with hyperlinks. It is possible that because the materials provided to them describing rationale used SEURAT as an example they may have been biased toward the tree structure. Alternatively, they could find a tree representation more natural because software developers are used to working in development environments that use a tree structure to represent code directories.

5.4.2. Barriers and assistance

When designing the survey, we were especially interested in determining if the barriers to rationale were as described in the literature. Table 7 ranks several factors that have been proposed as potential barriers to the use of rationale.

The first two items, cost and tediousness of rationale capture, were ranked as first or second by the majority of respondents. A quarter of those responding put reluctance to record decisions as being the greatest barrier. The concern that documenting reasons behind decisions hampers the process was ranked as last by over a third of those surveyed.

We were very curious as to practitioners' opinions on what could be done to make rationale capture and use appealing. The results of that survey question are given in Table 8.

Augmenting code comments with rationale scored quite high. This is not an approach that has been taken by many researchers, with one exception being the Archium project (van der Ven et al., 2006). Traceability matrixes were also of interest to the practitioners. Using rationale to produce post mortem reports was ranked low which is consistent with results reported in Table 2.

There has been some discussion among researchers about how integration with development tools could assist in rationale capture and use. Table 9 ranks several types of

Table 6. *Rationale user: Who will find rationale the most useful?*

	First	Second	Third	Fourth	Fifth	Sixth
Managers reviewing system progress	8.33%	5.56%	5.56%	13.89%	13.89%	**52.78%**
New people joining the team who need to understand the system	5.56%	16.67%	**27.78%**	**25.00%**	19.44%	5.56%
Developers wanting reminders of why they made earlier decisions	13.89%	**44.44%**	13.89%	11.11%	11.11%	5.56%
Developers wanting to understand why someone else made a decision	**50.00%**	22.22%	13.89%	8.33%	0.00%	5.56%
Maintainers trying to understand the system	19.44%	5.56%	**27.78%**	22.22%	16.67%	8.33%
Quality assurance checking if the software meets its requirements	2.78%	5.56%	11.11%	19.44%	**38.89%**	22.22%

https://doi.org/10.1017/S0890060408000218 Published online by Cambridge University Press

Table 7. *Potential barriers to rationale use*

Potential Barriers	First	Second	Third	Fourth	Fifth	Sixth	Seventh	Eighth
Cost of capturing the rationale	25.00%	**27.78%**	8.33%	17.14%	8.57%	5.71%	2.86%	2.86%
Rationale capture is likely to be tedious	**36.11%**	25.00%	**22.22%**	8.57%	8.57%	0.00%	0.00%	0.00%
Developer reluctance to record reasons for their decisions	25.00%	11.11%	16.67%	17.14%	14.29%	14.29%	2.86%	0.00%
Developer reluctance to document their "mistakes" (alternatives tried and then rejected)	0.00%	16.67%	**22.22%**	**20.00%**	14.29%	14.29%	2.86%	8.57%
Liability issues if software failures can be traced to faulty decisions	2.78%	2.78%	8.33%	11.43%	**22.86%**	17.14%	11.43%	22.86%
Lack of tool integration	8.33%	11.11%	11.11%	14.29%	14.29%	**25.71%**	5.71%	11.43%
No need to go back and look at reasons behind decisions after initial development is complete	2.78%	0.00%	2.78%	8.57%	14.29%	2.86%	**51.43%**	17.14%
Documenting reasons behind decisions hampers the decision-making process	0.00%	5.56%	8.33%	2.86%	2.86%	20.00%	22.86%	**37.14%**

Table 8. *Possible appeal: What would make capture and use more appealing?*

	First	Second	Third	Fourth	Fifth
Producing a record of meeting discussions	11.11%	13.89%	**33.33%**	16.67%	25.00%
Producing requirements for traceability matrixes	25.00%	**27.78%**	16.67%	22.22%	8.33%
Producing postmortem reports	2.78%	13.89%	11.11%	**33.33%**	**38.89%**
Augmenting code comments with rationale	**55.56%**	**27.78%**	13.89%	2.78%	0.00%
Producing status reports on development progress	5.56%	16.67%	25.00%	25.00%	27.78%

Table 9. *Integration priorities: Importance of integrating rationale with different tools*

	First	Second	Third	Fourth	Fifth	Sixth	Seventh
Interactive development environment (code compilation and editing)	**38.89%**	13.89%	13.89%	13.89%	13.89%	2.78%	2.78%
Design tools	25.00%	**36.11%**	16.67%	8.33%	8.33%	2.78%	2.78%
Requirement management tools	36.11%	19.44%	**22.22%**	8.33%	8.33%	2.78%	2.78%
Problem reporting tools	0.00%	8.33%	16.67%	**22.22%**	19.44%	22.22%	11.11%
Configuration management tools	0.00%	2.78%	8.33%	11.11%	16.67%	**30.56%**	**30.56%**
Workflow management tools	0.00%	5.56%	16.67%	16.67%	**19.44%**	16.67%	25.00%
Project management tools	0.00%	13.89%	5.56%	19.44%	13.89%	22.22%	25.00%

tools by how important it would be to integrate them with rationale.

The first three types of tools on the list, interactive development environments (IDEs), design tools, and requirements management tools were ranked as the most important. There was little interest in integrating rationale with the other types of tools.

5.5. Experiment summary

Some survey results were what one might predict. One example is the barriers to rationale: the greatest barriers involved the tediousness, the cost, and designers reluctant to record their reasons. The biggest surprise was *when* developers thought rationale would be the most useful. Grudin (1996) cautioned that rationale capture required resources "upstream" while providing savings "downstream" during maintenance that would weaken incentives for capture. The results of the survey described here indicated that our respondents felt the rationale would be most likely to be used during requirements development and design and would be least likely to be used during maintenance.

The 36 developers who responded to the survey are not a representative sample of the industry as a whole. In particular, it would be interesting to receive responses from more developers working on safety-critical and mission-critical systems. One question that, in retrospect, should have been on the survey was if the developer's organization followed the CMM or CMMI. The CMMI process area on decision analysis and resolution requires formal evaluation of alternatives, yet less than half the respondents rated evaluating support for alternatives in the top two value categories.

Another interesting observation from the experiment is that more than half the respondents listed managers reviewing

https://doi.org/10.1017/S0890060408000218 Published online by Cambridge University Press

system progress as being those who had the least use for rationale. Informal discussions with developers had suggested that providing that kind of insight to managers might be helpful in promoting rationale acceptance yet these results suggest otherwise.

6. SUMMARY AND CONCLUSIONS

Rationale, design and otherwise, has been an active research area for many years yet we still have had only limited success in putting rationale into practice. Although technology transfer rates in software engineering are typically slow, we should be concerned that rationale has not received wider adoption. On a more positive note, advances in software tool technology are making it possible to provide support for integrating rationale into the development process in ways that were not possible before. There is also an increasing acknowledgement of the importance of knowledge as a "key asset" to corporations (Liebowitz, 2002). Rationale has the potential to make a key contribution toward that knowledge. There remain two major obstacles toward rationale's acceptance that need to receive more attention by the research community. One is that we need to understand the needs and problems of the practitioners we are trying to support. The other is that we need to provide more concrete evidence of the value of our solutions through formal empirical evaluations of both existing and new approaches. Until then, we will remain researching under uncertainty.

REFERENCES

Bahler, D., & Bowen, J. (1992). Design rationale management in concurrent engineering, *Proc. Workshop on Design Rationale Capture and Use, 10th National Conf. Artificial Intelligence*.

Ball, L., Lambell, N., Ormerod, T.C., Slavin, S., & Mariani, J. (1999). Representing design rationale to support innovative design reuse: a minimalist approach. *Proc. 4th Design Thinking Research Symp*., pp. I.75–I.87.

Baniassad, E.L., Murphy, G.C., & Schwanninger, C. (2003). Design pattern rationale graphs: linking design to source. *Proc. 25th ICSE*, pp. 352–362.

Barbar, M., Gorton, I., & Kitchenham, B. (2006). A framework for supporting architecture knowledge and rationale management. In *Rationale Management in Software Engineering* (Dutoit, A., McCall, R., Mistrík, I., & Paech, B., Eds.), pp. 237–254. Heidelberg: Springer–Verlag.

Barber, K.S., & Graser, T. (2000). *Reference Architecture Representation Environment (RARE), A Tool to Support Object-Oriented Software Architecture Derivation and Evaluation*, Technical Report TR00-UT-LIPS-SEPA-04. Austin, TX: University of Texas at Austin, Laboratory for Intelligent Processes and Systems.

Bass, L., Clements, P., Nord, R.L., & Stafford, J. (2006). Capturing and using rationale for a software architecture. In *Rationale Management in Software Engineering* (Dutoit, A., McCall, R., Mistrík, I., & and Paech, B., Eds.), pp. 255–272. Heidelberg: Springer–Verlag.

Boehm, B. (2006). Value-based software engineering: overview and agenda. In *Value-Based Software Engineering* (Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., & Grünbacher, P., Eds.), pp. 3–14. Heidelberg: Springer–Verlag.

Boehm, B., & Jain, A. (2006). An initial theory of value-based software engineering. In *Value-Based Software Engineering* (Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., & Grünbacher, P., Eds.), pp. 15–37. Heidelberg: Springer–Verlag.

Boehm, B., & Kitapci, H. (2006). The WinWin approach: using a requirements negotiation tool for rationale capture and use. In *Rationale Management in Software Engineering* (Dutoit, A., McCall, R., Mistrík, I., & Paech, B., Eds.), pp. 173–190. Heidelberg: Springer–Verlag.

Boehm, B., & Ross, R. (1989). Theory-W software project management: principles and examples. *IEEE Transactions on Software Engineering 18(7)*, 902–916.

Bracewell, R., Ahmed, S., & Wallace, K. (2004). DREd and Design Folders, a way of capturing, storing, and passing on knowledge generated during design projects. *Proc. ASME 2004 Design Automation Conf*.

Brandish, M.J., Hague, M.J., & Taleb-Bendiab, A. (1996). M-LAP: a machine learning apprentice agent for computer supported design. *AID'96 Machine Learning in Design Workshop*.

Bratthall, L., Johansson, E., & Regnel, B. (2000). Is a design rationale vital when predicting change impact?—a controlled experiment on software architecture evolution. *Proc. Int. Conf. Product Focused Software Process Improvement*, pp. 126–139.

Brice, A., & Johns, B. (1999). *Improving Design by Improving the Design Process*, Technical Report QSL-9002A-WP-001.

Brissaud, D., Garro, O., & Poveda, O. (2003). Design process rationale capture and support by abstraction of criteria. *Research in Engineering Design 14*, 162–172.

Britt, B., & Glagowski, T. (1996). Reconstructive derivational analogy: a machine learning approach to automating redesign. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 10*, 115–126.

Brown, D.C., & Bansal, R. (1991). Using design history systems for technology transfer. In *Computer Aided Cooperative Product Development, Lecture Notes in Computer Science* (Sriram, D., Logcher, R., & Fukuda, S., Eds.), Vol. 492, pp. 544–559. New York: Springer–Verlag.

Buckingham Shum, S., & Hammond, N. (1994). Argumentation-based design rationale: what use at what cost? *International Journal of Human–Computer Studies 40(4)*, 603–652.

Buckingham Shum, S., Selvin, A., Sierhuis, M., Conklin, J., Haley, C., & Nuseibeh, B. (2006). Hypermedia support for argumentation-based rationale: 15 years on from gIBIS and QOC. In *Rationale Management in Software Engineering* (Dutoit, A., McCall, R., Mistrík, I., & Paech, B., Eds.), pp. 111–132. Heidelberg: Springer–Verlag.

Burge, J. (2006). Anatomy of an experiment: difficulties in evaluating rationale-based systems. *Workshop on Design Rationale: Problems and Progress, DCC'06, Conf. Design Computing and Cognition,* Eindhoven.

Burge, J., & Brown, D.C. (2004). An integrated approach for software design checking using rationale. In *Proc. Design Computing and Cognition '04*, pp. 557–576. Dordrecht: Kluwer Academic.

Burge, J., & Brown, D.C. (2006). Rationale-based support for software maintenance. In *Rationale Management in Software Engineering* (Dutoit, A., McCall, R., Mistrík, I., & Paech, B., Eds.), pp. 273–296. Heidelberg: Springer–Verlag.

Burge, J.E., Carroll, J.M., McCall, R., & Mistrík, I. (2008). *Rationale-Based Software Engineering*. Heidelberg: Springer–Verlag.

Carroll, J., & Rosson, M. (1992). Getting around the task-artifact cycle: how to make claims and design by scenario. *ACM Transactions on Information Systems 10(2)*, 181–212.

Carroll, J., & Rosson, M. (2003). Design rationale as theory. In *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science* (Carroll, J.M., Ed.), pp. 431–461. San Francisco, CA: Morgan–Kaufmann.

Carroll, J.M. (2000). *Making Use: Scenario-Based Design of Human–Computer Interactions*. Cambridge, MA: MIT Press.

Chung, P.W.H., & Goodwin, R. (1998). An integrated approach to representing and accessing design rationale. *Engineering Applications of Artificial Intelligence 11(1)*, 149–159.

CMMI Product Team. (2006). *CMMI for Development*. Version 1.2, CMU/SEI-2006-TR-008.

Conklin, E.J., & Burgess-Yakemovic, K.C. (1996). A process-oriented approach to design rationale. In *Design Rationale Concepts, Techniques, and Use* (Moran, T., & Carroll, J., Eds.), pp. 393–427. Hillsdale, NJ: Erlbaum.

Curtis, B. (2000). From MCC and CMM: technology transfers bright and dim. *Proc. 22nd ICSE*, pp. 521–530.

de la Garza, J.M., & Alcantara, P.T. (1997). Using parameter dependency network to represent design rationale. *Journal of Computing in Civil Engineering 11(2)*, 102–112.

Dellen, B., Kohler, K., & Maurer, F. (1996). Integrating software process models and design rationales. *Proc. Knowledge-Based Software Engineering*, pp. 84–93.

De Medeiros, A.P., Schwabe, D., & Feijo, B. (2005). Kuaba ontology: design rationale representation and re-use in model-based designs.

Conceptual Modeling—ER 2005, 24th Int. Conf. Conceptual Modeling, pp. 241–255.

Dutoit, A., McCall, R., Mistrík, I., & Paech, B., Eds. (2006). *Rationale Management in Software Engineering*. Heidelberg: Springer–Verlag.

Dutoit, A.H., Wolf, T., Paech, B., Borner, L., & Ruckert, J. (2005). Using rationale for software engineering education. *Proc. 18th Conf. Software Engineering Education and Training*, pp. 129–136.

Erdogmus, H., Favaro, J., & Halling, M. (2006). Valuation of software initiatives under uncertainty: concepts, issues, and techniques. In *Value-Based Software Engineering* (Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., & Grünbacher, P., Eds.), pp. 39–99. Heidelberg: Springer–Verlag.

Falessi, D., Cantone, G., & Becker, M. (2006). Documenting design decision rationale to improve individual and team design decision making: an experimental evaluation. *Proc. ISESE*, pp. 134–143.

Fischer, G., Lemke, A., McCall, R., & Morch, A. (1996). Making argumentation serve design. In *Design Rationale Concepts, Techniques, and Use* (Moran, T., & Carroll, J., Eds.), pp. 267–293. Hillsdale, NJ: Erlbaum.

Garcia, A., Howard, H., & Stefik, M. (1993). *Active Design Documents: A New Approach for Supporting Documentation in Preliminary Routine Design*, Technical Report 82. Stanford, CA: Stanford University.

Garcia, A., Howard, H., & Stefik, M. (1994). *Design Rationale for Collaboration: The Active Document Approach*, Technical Report MCC08/94. Rio de Janeiro: Universidade Católica do Rio de Janeiro, Departmento de Informática Pontifícia.

Garcia, A.C.B., & Vivacqua, A.S. (1997). *MultiADD: Multiagent Active Design Documents*. Accessed at http://citeseer.ist.psu.edu/242337.html on April 14, 2007.

Gill, S., & Munson, E. (2006). A version-aware tool for design rationale. *Proc. WebMedia'06*, pp. 20–26.

Gruber, T. (1990). Model-based explanation of design rationale. *Proc. AAAI-90 Explanation Workshop*.

Grudin, J. (1996). Evaluating opportunities for design capture. In *Design Rationale Concepts, Techniques, and Use* (Moran, T., & Carroll, J., Eds.), pp. 454–470. Hillsdale, NJ: Erlbaum.

Hagge, L., Houdek, F., Lappe, K., & Paech, B. (2006). Using patterns for sharing requirements engineering process rationales. In *Rationale Management in Software Engineering* (Dutoit, A., McCall, R., Mistrík, I., & Paech, B., Eds.), pp. 409–427. Heidelberg: Springer–Verlag.

Hartmann, D. (2006). Interview: Jim Johnson of the Standish Group, InfoQ. Accessed at http://www.infoq.com/articles/Interview-Johnson-Standish-CHAOS on April 30, 2007.

Haynes, S.R. (2006). Three studies of design rationale as explanation. In *Rationale Management in Software Engineering* (Dutoit, A., McCall, R., Mistrík, I., & Paech, B., Eds.), pp. 53–71. Heidelberg: Springer–Verlag.

Heliades, G., & Edmonds, E. (1999). On facilitating knowledge transfer in software design. *Knowledge-Based Systems 12*, 391–395.

Hordijk, W., & Wieringa, R. (2006). Reusable rationale blocks: improving quality and efficiency of design choices. In *Rationale Management in Software Engineering* (Dutoit, A., McCall, R., Mistrík, I., & Paech, B., Eds.), pp. 353–371. Heidelberg: Springer–Verlag.

Horner, J., & Attwood, M.E. (2006). Effective design rationale: understanding the barriers. In *Rationale Management in Software Engineering* (Dutoit, A., McCall, R., Mistrík, I., & Paech, B., Eds.), pp. 73–90. Heidelberg: Springer–Verlag.

In, H., Boehm, B., Rodgers, T., & Deutsch, M. (2001). Applying WinWin to quality requirements: a case study. *Proc. 23rd Int. Conf. Software Engineering*, pp. 555–564.

Karacapilidis, N., & Papadias, D. (2001). Computer supported argumentation and collaborative decision making: the HERMES system. *Information Systems 26(4)*, 259–277.

Karsenty, L. (1996). An empirical evaluation of design rationale documents. *Proc. SIGCHI Conf. Human Factors in Computing Systems: Common Ground*, pp. 150–156.

Keith, S., Blandford, A., Fields, R. & Theng, Y.L. (2002). An investigation into the application of Claims Analysis to evaluate usability of a digital library interface. *Proc. JCDL Workshop on Usability.*

Klein, M. (1992). DRCS: an integrated system for capture of designs and their rationale. *Proc. Artificial Intelligence in Design '92* (Gero, J., Ed.), pp. 393–412. Dordrect: Kluwer Academic.

Klein, M. (1997*a*). An exception handling approach to enhancing consistency, completeness and correctness in collaborative requirements capture. *Concurrent Engineering Research and Applications*, pp. 37–46.

Klein, M. (1997*b*). Capturing geometry rationale for collaborative design. *Proc. 6th Int. Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*.

Kunz, W., & Rittel, H. (1970). *Issues as Elements of Information Systems*, Working Paper 131. Berkeley, CA: University of California, Berkeley, Center for Urban and Regional Development.

Lacaze, X., Palanque, P., Barboni, E., Bastide, R., & Navarre, D. (2006). From DREAM to reality: specificities of interactive systems development with respect to rationale management. In *Rationale Management in Software Engineering* (Dutoit, A., McCall, R., Mistrík, I., & Paech, B., Eds.), pp. 155–172. Heidelberg: Springer–Verlag.

Lambell, N.J., Ball, L.J., & Ormerod, T.C. (2000). The evaluation of Desperado: a computerised tool to aid design reuse. In *People and Computers XVI: Usability or Else!* (McDonald, S., Wearn, Y., & Cockton, G., Eds.), pp. 437–453. Cambridge: Cambridge University Press.

Larsson, M., Wall, A., Norström, C., & Crnkovic, I. (2006). Technology transfer: why some succeed and some don't. *Proc. Int. Workshop on Software Technology Transfer in Software Engineering*, pp. 23–28.

Lee, J. (1991). SIBYL: a qualitative design management system. In *Artificial Intelligence at MIT: Expanding Frontiers* (Winston, P.H., & Shellard, S., Eds.), pp. 104–133. Cambridge MA: MIT Press.

Liebowitz, J. (2002). A look at NASA Goddard Space Flight Center's knowledge management initiatives. *IEEE Software 19(3)*, 40–42.

Lougher, R., & Rodden, T. (1993). Group support for the recording and sharing of maintenance rationale. *Software Engineering Journal 8(6)*, 295–306.

Louridas, P., & Loucopoulos, P. (2000). A generic model for reflective design. *ACM Transactions on Software Engineering Methodology 9(2)*, 199–237.

MacLean, A., Young, R.M., Bellotti, V., & Moran, T.P. (1996). Questions, options and criteria: elements of design space analysis. In *Design Rationale Concepts, Techniques, and Use* (Moran, T., & Carroll, J., Eds.), pp. 201–251. Hillsdale, NJ: Erlbaum.

McGill, K., Deadrick, W., Hayes, J.H., & Dekhtyar, A. (2006). Houston, we have a success story: technology transfer at the NASA IV&V facility. *Proc. Int. Workshop on Software Technology Transfer in Software Engineering*, pp. 49–54.

Moran, T., & Carroll, J., Eds. (1996). *Design Rationale Concepts, Techniques, and Use*. Hillsdale, NJ: Erlbaum.

Myers, K., Zumel, N., & Garcia, P. (1999). Automated capture of rationale for the detailed design process. *Proc. 11th National Conf. Innovative Applications of Artificial Intelligence*, pp. 876–883.

Palyagar, B., & Richards, D. (2006). Capturing and reusing rationale associated with requirements engineering process improvement: a case study. In *Rationale Management in Software Engineering* (Dutoit, A., McCall, R., Mistrík, I., & Paech, B., Eds.), pp. 391–408. Heidelberg: Springer–Verlag.

Peña-Mora, F., Sriram, D., & Logcher, R. (1995). Design rationale for computer-supported conflict mitigation. *ASCE Journal of Computing in Civil Engineering 9(1)*, 57–72.

Peña-Mora, F., & Vadhavkar, S. (1997). Augmenting design patterns with design rationale. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 11*, 93–108.

Potts, C., Takahashi, K., & Anton, A. (1994). Inquiry-based requirements analysis. *IEEE Software 11(2)*, 21–32.

Potts, C., Takahashi, K., Smith, J., & Ota, K. (1995). An evaluation of inquiry-based requirements analysis for an Internet service. *Proc. 2nd IEEE Int. Symp. Requirements Engineering*, pp. 172–180.

Ramesh, B., & Dhar, V. (1994). Representing and maintaining process knowledge for large-scale systems development. *IEEE Expert: Intelligent Systems and Their Applications 9(2)*, 54–59.

Ramesh, B., & Dhar, V. (1992). Supporting systems development by capturing deliberations during requirements engineering. *IEEE Transactions on Software Engineering 18(6)*, 498–510.

Redwine, S., & Riddle, W. (1985). Software technology maturation. *Proc. 8th ICSE*, pp. 189–2000.

Richter, H., & Abowd, G. (1999). *Automating The Capture of Design Knowledge: A Preliminary Study*, Technical Report GVU-99-45. Atlanta, GA: Georgia Tech.

Richter, H., Schuchhard, P., & Abowd, G. (1998). *Automated Capture and Retrieval of Architectural Rationale*, Technical Report GIT-GVU-98-37. Atlanta, GA: Georgia Tech.

Rus, I., & Lindvall, M. (2002). Guest editors' introduction: knowledge management in software engineering. *IEEE Software 19(3)*, 26–38.

Schneider, K. (2006). Rationale as a by-product, In *Rationale Management in Software Engineering* (Dutoit, A., McCall, R., Mistrík, I., & Paech, B., Eds.), pp. 91–109. Heidelberg: Springer–Verlag.
SEI. (1997). *Integrated Product Development Capability Maturity Model*, Draft Version 0.98. Pittsburgh, PA: Carnegie Mellon University, Enterprise Process Improvement Collaboration and Software Engineering Institute.
Selvin, A., & Sierhuis, M. (1999). Case studies of project Compendium in different organizations. *Workshop on Computer-Supported Collaborative Argumentation Conf. Computer-Supported Collaborative Learning*.
Shah, J., Rangaswamy, S., Qureshi, S., & Urban, S. (1999). Design history system: data models and prototype implementation. In *Knowledge Intensive CAD* (Tomiyama, T., Mäntylä, M., & Finger, S., Eds.), pp. 91–114. Dordrecht: Kluwer Academic.
Shipman, F., & McCall, R. (1997). Integrating different perspectives on design rationale: supporting the emergence of design rationale from design communication. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 11*, 141–154.
Smith, R., & Farquhar, A. (2000). The road ahead for knowledge management: an AI perspective. *AI Magazine 21(4)*, 17–40.
Standish Group. (1994). *CHAOS Report*. Accessed at http://www.standishgroup.com/sample_research/chaos_1994_1.php on April 6, 2007.
Tang, A., Babar, M., Gorton, I., & Han, J. (2006). A survey of architecture design rationale. *Journal of Systems and Software 79*, 1792–1804.
Tang, A., Jin, Y., & Han, J. (2007). A rationale-based architecture model for design traceability and reasoning. *Journal of Systems and Software 80(6)*, 918–934.
Taura, T., & Kubota, A. (1999). A study on engineering history base. *Research in Engineering Design 11(1)*, 45–54.
van der Ven, J.S., Jansen, A.G.J, Nijhuis, J.A., & Bosch, J. (2006). Design decisions: the bridge between rationale and architecture. In *Rationale Management in Software Engineering* (Dutoit, A., McCall, R., Mistrík, I., & Paech, B., Eds.), pp. 329–348. Heidelberg: Springer–Verlag.
Vetschera, R. (2006). Preference-based decision support in software engineering. In *Value-Based Software Engineering* (Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., & Grünbacher, P., Eds.), pp. 67–89. Heidelberg: Springer–Verlag.
Zaychik, V., & Regli, W.C. (2003). Capturing communication and context in the software project lifecycle. *Research in Engineering Design 14(2)*, 75–88.

Janet E. Burge is an Assistant Professor in the Miami University Computer Science and Systems Analysis Department. She has worked in industry as a researcher and software developer for over 20 years. She received her PhD and MS in computer science from Worcester Polytechnic Institute and her BS in computer science from Michigan Technological University. Dr. Burge's major research interests are in software engineering and artificial intelligence. Her primary research area is in DR, with a focus on DR for software maintenance. She is a coauthor of the book *Rationale-Based Software Engineering*.