The Knowledge Engineering Review, Vol. 7:4, 1992, 363–366

Design Rationale Management Research

JINTAE LEE
Department of Information and Computer Sciences, University of Hawaii, Honolulu, HI 96822, USA

1 Introduction

Many benefits potentially stem from a structured representation and use of "design rationales", i.e. the deliberations underlying a software design process. Explicitly-represented rationales help us understand the design better so that we can produce a better design, maintain the resulting artifact better, and exploit the cumulated knowledge when we need to redesign it. Explicit representation of the rationales also provide a basis for reviewing or justifying the decisions that have been made, for communicating with other members of the design team more easily, and for defining computer services that support various design activities, such as keeping track of dependencies or managing multiple viewpoints. Technologies that have recently become available, such as multi-media and distributed databases, provide the necessary ingredients for pursuing these potential benefits seriously. As a result, in the past few year we have seen growing interest in design rationale management.

At AAAI'92 (San Jose, USA, in July), I chaired a one-day workshop on design rationale capture and use, in which over 40 people from academia as well as from industry participated. The foremost goal of the workshop was to bring together the people working on the topic in various disciplines and promote interactions among them. The fields represented included artificial intelligence, software engineering, mechanical engineering, civil engineering, computer-supported work, and human computer interaction. Our hope was that, through the workshop, people would become aware of the issues to consider in design rationale management capture, learn what other people have done to resolve them, and transfer techniques across disciplines (e.g. between software and VLSI designs). This article reports on what I view as the important issues raised and discussed at the workshop (though it represents only the chairman's view). The workshop concluded with a wrap up session, where we discussed possible future collaborations.

The workshop began with an introduction (J. Lee) and overview of the general issues (T. Gruber, J. Jagannathan). The rest of the workshop was divided into six sessions, each of which is shown below together with its moderator and the presenters for the session.

- Rationale Management in Hardware Design
  (W. Mark: D. Bahler, B. Chandrasekaran, A. Garcia, W. Mark, A. Wong)
- Rationale Management in Software Design
  (G. Arango: I. Baxter, B. Durney, I. Hulthage, E. Kant, W. Swartout)
- Constructing Rationales
  (T. Gruber: G. Arango, B. Britt, R. Cohen, T. Gruber, G. Kim)
- Rationales in Supporting Coordination / Integration
  (J. Lee: J. Bradshaw, J. Glicksman, P. Johnson, M. Klein, C. Petrie)
- Semi-Formal Approaches to Rationale Management
  (T. Moran: K. Singley, J. Conklin, F. Lakin, B. Ramesh & V. Dhar, B. Reeves & F. Shipman)
- Linking Semi-Formal Approaches to Formal Approaches
  (G. Fischer: D. Barman, M. Eisenberg, J. Lee, K. Nakakoji, J. Ostwald, L. Ruecker)

Although some of the discussions in these sessions were specific to the chosen topics, many issues raised were general and relevant to most of the sessions. Hence, this report takes the form, not of a chronological account, but of issues that have been raised and answered during the workshop. These issues are grouped into the following categories: the nature of design rationale, services, representation, production of rationales, evaluations, and future collaborations.

1 Nature of Design Rationale

What is design rationale? A proposed answer (Gruber) was "an explanation that answers a question about why an artifact is designed as it is." This definition is appealing because it can be used to generate further questions such as Who is the audience? When and who generates this explanation? What kind of questions does it answer? Each of these questions is discussed later.

Is the nature of rationale different in different design domains? This question was raised explicitly (Moran), but also was implicit in the juxtaposition of the two sessions, software design rationale and hardware design rationale. Our goal was to see whether the research in one is transferable to the other and how much of the research was domain-specific. It turned out that most of the issues raised and the discussions held in these two sessions were relevant to both hardware and software designs. For example, it was pointed out (Arango) that the transformational approaches (e.g. Baxter), most visible in software design, would be equally suitable for other domains, where you can formally specify the requirements, their decomposition, and a set of legal transformations (e.g. VLSI design). Therefore, we might say that the form of the rationales can be different in different domains if we define a domain in terms of the nature of reasoning involved, but not if defined in terms of the objects involved.

2 Services: What Good are Design Rationales?

What are the services that we can provide with design rationale? Some of the suggested categories were: documentation and retrieval support (Lakin), maintenance support (Baxter, Durney), dependency management (Mark), generation of explanations (Gruber, Kant, Swartout), organizational design (Hulthage & Gasser) , simulation and diagnostics (Chandrasekaran), requirement engineering (Lee, Ramesh, Dahr), design methodology support (Bahler, Singley, Wong), and collaboration support (e.g. project management, communication among different members, training of new members).

How can we use rationales to support coordination/integration? There were several papers which pointed out the use of rationales for coordination among designers or to integrate different aspects of design. For example, explicitly represented rationales can serve as a basis for communication among designers, for project management such as keeping track of unresolved issues, and their dependencies, or in order to update new comers. Rationales can also be used for producing consensus: e.g. on type definitions (Johnson) or on ontology (Bradshaw). A model of rationales was also proposed that supports distributed subgoaling as a result of decision making and propagation of the effects of decision revision through a distributed dependency network (C. Petrie). Other issues addressed in this context include how to develop a shared vocabulary for representing rationales for coordination (Bradshaw), how to use multi-media email to support rationale-sharing effectively (Glicksman), and how the decision making and the artifact aspects of the design process can be integrated (Klein).

Who is the audience, or who can be supported through design rationales? Thinking about who we want to support with design rationales makes clear what kind of services we should aim at. For example, the captured rationales may be used to support designers at design time, designers at redesign time, maintainers, trouble-shooters, or new trainees. For example, if we want to support designers at design time, the ability to manage dependencies becomes quite important. If it is the designers at redesign time that we want to support, it becomes important to define a similarity metric (e.g. requirements overlap) that will help retrieve and reuse relevant parts of the design rationales. Of course, these different requirements create different cost economics, e.g. how much and how formally information needs to be captured.

3 Representation: What information is worth capturing and reusing?

The representations proposed at the workshop range from unstructured (e.g. electronic notebooks that record rationales in natural language), to semi-structured (e.g. templates of different types such as requirements, specifications, goals, alternatives, arguments), to completely formal (e.g. states/causal connections/annotations). The differences among these representations reflect the different services that they are designed to support. For example, if the primary goal of managing rationales is to help people archive, retrieve, and examine the reasons for their decisions, for example, the representation can be semi-structured, i.e. only parts of the representations need to be understood by the computer and the rest

can be only interpretable by humans. (cf. Semi-Formal Approaches). On the other hand, if the goal is to manage dependencies and suggest new alternatives, then the representation has to capture the domain knowledge more formally.

Is there a generic structure of design rationales? An interesting observation is that despite this diversity, there was some convergence, suggesting that there might be some generic structure of design rationales. For example, in many representations (Bradshaw, Lee, Petrie), decision constructs like alternatives, criteria, and evaluations play important roles. They are often complemented by constructs for representing requirements (e.g. requirements, goals, specifications, excludes[requirement, requirement]), representing arguments for evaluations (e.g. claim, supports[claim, claim], denies[claim, claim], qualifies[claim,claim]) and/or representing different parts of what is being designed (e.g. artifact, attribute, module, interface, has-attribute).

4 Production of Rationales

What are alternative ways of producing rationales? There are at least three major ways that have been suggested for producing rationales (Gruber): record and replay, post-hoc reconstruction, generation from domain knowledge. In record and replay, the rationales are captured as they unfold, i.e. as the designers deliberate over possible alternatives, criteria, answer questions, and so on. For example, people can use a shared database to raise issues, propose new alternatives and criteria, enter evaluations, and so on. The rationales can be captured in unstructured (e.g. electronic notebook) form, in structured representations, or in the form of annotations (Reeves & Shipman). And they can be captured synchronously (e.g. in a capture room, where people use a shared public screen to respond to each other) or asynchronously (e.g. electronic mail). This capture methodology usually implies that the representation cannot be too rich or too formal because such a representation would create excessive overhead and disrupt the flow for the designers.

The second way of producing rationales is to reconstruct rationales after the fact. The deliberations are captured in the most unobtrusive form, e.g. in video. Then the rationales are recast into the constructs of a particular representation. This methodology has the advantage that reconstruction forces reflections and the representation can be more formal and more systematic. However, reconstruction may be a luxury that many people cannot afford. The third way of producing rationales is to construct rationales from the domain knowledge. Some of the suggested methods for constructing rationales were: simulation (Gruber & Russell), context monitoring (Cohen), reverse engineering (Kim & Bekey), product modeling (Arango), interactive verification (Garcia & Howard), and rule-based construction of composition hierarchy (Britt & Wick), and a domain-independent decision revision theory encapsulating domain-specific informal objects (C. Petrie). This strategy has the high initial cost of compiling the knowledge needed to construct the rationales, but has the appeal of creating rationales at no cost to the user later and of being able to maintain consistent and up-to-date rationales .

The following have been suggested as the appropriate technologies for capturing and accessing rationales (T. Moran): scribbling tools, audio-video, groupware, argument representation, hypertext, expert systems, document management, design tools. The following have been also suggested as possible candidates for constructing rationales (T. Gruber): designers at design time, designers at documentation time, designers' support tools, programs at re-engineering time.

5 Semi-Formal Approaches

The first thing that would probably strike anybody looking at the literature is the division between formal and informal approaches to design rationale. This division cuts across all the aspects that have been discussed so far: service, representation, and production of rationales. As Fischer pointed out, formal representations have the advantage of being interpretable by computers and having well-established inference procedures, but they are hard to create and comprehend. Also, the domain knowledge needed to create formal representations is often missing. On the other hand, informal representations are easy to create and natural, but they are not interpretable by computers and rely on unarticulated background knowledge.

Semi-formal approaches take the middle ground and have some parts of the representation interpretable

by the computer, but others only by human users. For example, task-specific objects (e.g. decision, goal, claim) can be formal objects with their own attributes and formally related, but the system may allow these attribute values to be filled in by designers in the form of informal descriptions or other formal objects that the designers might choose to create. The system then processes the descriptions to the extent that they have been formalized, but leaves others for human processing. The appeal of this approach is that there is relatively less overhead in capture (in fact, semi-formal representation can be easier to deal with than informal representations by suggesting what information is expected and defaults, e.g. forms), yet we can define computational operations exploiting the formalized part of the representation. The degree of formalization can also be controlled by different representations.

How should we integrate semi-formal approaches with formal approaches? The following have been suggested as ways of linking more formal representations with semi-formal representations so that we can define more automated design support as we understand more of the domain: creating rules from the keywords assigned to different parts of the rationales (Nakakoji), incremental formalization through an issue base (Lee), a programmable design environment combining knowledge-rich cooperative problem-solving systems and programmable applications (Eisenberg), a representation combining hypertext and knowledge representation (Barman), an evolving artifact approach (Ostwald), or extending semi-formal representation with domain-specific formal representation (Ruecker). Some of the issues discussed for all of these approaches were (Fischer): what are the roles of human and computer in these systems? How are the two linked? How do they communicate? How does the distribution between human and system change over time? Is there shared knowledge between human and system? If so, what kind? How does it get created? How does it change over time?

6 Future Collaboration

In the wrap up session, we discussed ways in which we can continue to work together on design rationale after the workshop. One idea suggested was the use of a set of canonical examples. Detailed records of a few design processes (in video or in transcripts) would provide the researchers with the data. Also, they would provide common examples and vocabulary that should facilitate comparison and communication among the different models underlying the design rationale research. A few examples, such as Lift, ATM, HVAC, were suggested and we decided to look into them further. Another related idea for facilitating future collaboration was to set up a clearing-house, not only for the canonical examples, but for other kinds of resource as well. For example, we can share through the clearing-house task-specific ontologies, domain-knowledge, or even inferencing modules. We decided that this idea would be pursued in the context of the on-going attempts to share knowledge.

Many other issues were raised and discussed at the workshop. For example, it was pointed out and agreed that the success of a rationale management system critically depends on successful resolution of non-technical issues, such as the need to integrate such a tool with standard operating systems and tying proper incentives and resource allocation to the usage and refinement of parts, e.g. software modules (Dahr).

Limitations on space prevent us from expanding on these discussions here, but the papers presented at the workshop have been compiled in the form of notes, and request for further information, or comments can be addressed to the author.

Acknowledgment

I thank all the people who participated in the workshop, and the program committee members, each of whom have put a lot of effort into reviewing the papers and moderating the sessions, deserve special thanks. I also appreciate the comments on this report from Tom Gruber, Bob Halperin, Charles Petrie, and Tom Moran.