2 Three Studies of Design Rationale as Explanation

S.R. Haynes

Abstract: Prior research has pointed out the potential for design rationale to act as a base of explanatory knowledge about an evolving or completed design. One of the benefits of design rationale and its associated techniques and tools is that they help to answer questions about *why* a particular design possesses the structure and behaviors that it does. Answers to these *why* questions are explanations. To date, little empirical work has investigated the challenges and opportunities that emerge when attempting to realize the utility of design rationale as explanations. The three short case studies reported here describe examples of research that explores the use of design rationale as a means to enhance communication and comprehension among the stakeholders in complex systems projects. Lessons learned from the three studies are provided and some areas for future research are identified.

Keywords: design rationale, explanation, case studies, usability

2.1 Introduction

This chapter examines the relationship between the rationale that emerges in the systems design and development context and the explanations constructed in the context of system development, evolution, and use. Motivating this work is the proposition that as systems become more pervasive, complex, and intelligent, better means of explaining their structure and behavior will be required to ensure adoption and effective use. The use of design rationale (DR) as the basis for system explanations is an important part of the DR value equation. Justifying the cost and effort of DR capture involves developing ways to use the products of these efforts more effectively. Access to DR may be particularly important in more complex systems, intelligent, distributed systems, for example, because of the degree of understanding and trust required between these systems and their users as they work together in a problem domain.

DR captures the intentions underlying creation of a system artifact, and the issues, questions, argumentation, and decisions made in the process of navigating a given design space. The knowledge base represented by DR provides the raw material for active construction of system explanations in these contexts. As Dutoit et al. ([12], Chap. 1 in this book) point out there is a range of different uses for DR including, importantly, knowledge

transfer among different stakeholders in a systems development project. DR is about helping to make explicit much of the assumed, tacit knowledge that underlies shared understanding between stakeholders in the same and in different roles. The studies reported here describe three example "use cases" of DR as explanation.

DR as explanation is both a descriptive and a prescriptive thesis. DR is about capturing and communicating the "why" underlying the structure and behavior of a system. DR as explanation is therefore a descriptive idea; it is the essence of the rationale-centric approach to the thinking about the design knowledge. To really leverage the potential power of these explanations, however, requires acknowledging the communicatory power and value of rationales. Explanations are pervasive in the use of DR. Whether among designers or between designers and other project stakeholders, explanations based on the underlying rationale of particular design are the means by which systems are comprehended, adopted, used effectively, enhanced, reused in new settings, and so on. Software engineering is a team-oriented and knowledge-intensive enterprise; DR is the currency that facilitates exchange of knowledge among the project team members.

The chapter first explores complex system explanations and frames these relative to the capabilities provided by access to DR. Prior research in knowledge-based systems and in software engineering has suggested a role for DR as the basis for system explanations, and this work is reviewed in support of the chapter's main arguments. Section 2.2 describes three specific cases of DR as explanation. In the first of these, the *Questions–Options–Criteria* (QOC, [17]) approach was employed to support development of the system help content. In the second, scenarios and claims analysis [7,8,9] are being used to construct a technology transfer package designed to assist potential technology adopters in comprehending the technology and how it might fit with their own organizational objectives and priorities. In the third, scenarios and claims were again employed, this time as a means for evaluating a collaborative system in the field. This last case shows how evaluation results can be transformed into retrospective DR, and how these can be used to develop new design meta-criteria for future system developers.

2.2 Explanations of Complex Systems

Mirel [19] describes complex systems as those used to help structure and solve ill-structured problems. She defines the domains and tasks that give rise to complex systems development as characterized by some core attributes including:

- Indeterminacy of both task goals and criteria for task completion
- Requiring higher order cognitive skill and integrating knowledge from different areas
- Requiring advanced learning and instruction for effective performance

Design for systems created to meet these challenges is correspondingly complex. The indeterminacy of task goals requires the design of flexible systems that rely on abstract software components. The higher the level of abstraction from a particular behavioral or structural requirement, the more difficult it becomes for developers and end users to reconstruct this process of abstraction later, and thereby relate the abstraction to the design deliberations, or rationale, that gave rise to it [3]. The integration of different knowledge in design and use, and the assumption of variable cognitive skills on the part of end users introduce additional complexity to the task of understanding how a particular system form emerges from a particular set of requirements. That these systems require advanced learning on the part of their users raises questions about where they obtain the information needed to facilitate this learning.

Today, many routine computing tasks are supported by *high-functionality applications* (HFAs, [29]), which typically include hundreds or even thousands of features and are used to manage large volumes of heterogeneous information. Each feature of such a system may be realized by a number of complexly interacting software components [1]. Intra- and inter-component interactions, as well as the distributed, intersystem interactions that increasingly define the modern computing milieu, make comprehending these systems difficult because of the cognitive load introduced when attempting to comprehend their structure and functionality.

The nature of complex systems suggests that the kinds of explanations required to convey understanding go beyond relatively simple, descriptive knowledge to include *how* and, especially, *why* a design assumes a particular structure and set of behaviors. Explanation requests, compared to, say, requests for instructions, are usually concerned with acquiring some deep knowledge of the event or entity in question. Explanations are provided in response to *why* questions that appeal to the causal chain that resulted in occurrence of the event or existence of the entity.

In the design context, this causal chain can potentially include a large, heterogeneous network of factors and influences that combine to inform the design decisions being made, in other words, the DR. For example, the second of the three development cases reported later in the chapter involved selection and implementation of certain decision-theoretic tools to support antiterrorism planning. The rationale for why a particular set of tools was selected, and how important domain concepts were translated into working software has proven to be of considerable interest to project stakeholders.

Providing even the briefest explanation of a complex system is a knowledge intensive activity. Providing a parsimonious explanation typically requires awareness of substantial implicit and tacit factors including the purpose motivating the explanation request, the current task at hand, what the explanation requestor already knows, and other contextual details that point to the essential information required to fulfill the explanation request [13]. People have adapted to inferring this detail from the environment and do it almost effortlessly. Computer-based information systems, however, are largely unable to ascertain the contextual detail needed to provide focused explanations to even the most well-formed explanation-seeking questions. Section 2.3 discusses related work on identifying appropriate explanation content for complex systems.

2.3 Design Rationale as Explanation Content

Swartout [26, 27] first identified the potential utility of DR as explanation content. His work on the explainable expert system (EES) project was an attempt to address some of the explanation content deficiencies identified in early expert system research. In this earlier work, especially Mycin [4] and derivative projects [10], researchers found significant gaps between the problem-solving strategies systems built to replicate or enhance, and the structural properties of the systems created to realize these strategies. Clancey [10, 11] identified this missing link as the detailed *support knowledge* representing the translation of domain requirements into functional software systems. The nature of this support knowledge presented a knowledge engineering conundrum because it represented a huge and seemingly intractable base of knowledge that was not germane to the application domain per se, but which was required to explain to domain users how system functionality emerged in relation to their domain requirements.

Swartout's attempt to mitigate the effects of this knowledge gap involved construction of an automatic expert system generator that tracked and logged decisions made by the system as it produced rules and control logic based on input in the form of relatively abstract, domain-specific problem-solving goals. The early XPLAIN system and later work on the EES relied on the existence of a domain model and problem-solving principles to translate goals into a system of productions, or rules. The log of automatic design decisions served as justifications for why the resulting system was appropriate given the domain model and principles and the problem-solving goals as expressed by the system developer.

One of the challenges faced by the EES developers was to elucidate the link between domain-independent, strategic concepts and the domain-specific or instance-specific information that is needed to apply a strategic concept in a particular goal scenario. For example, a design principle such as "simplify wherever possible" might be instantiated in a software design as "we can combine these two modules into one with no loss of cohesion." The EES attempts to solve this linkage problem through the concept of *capability descriptions*, which relate system goals to operationalized plans to achieve those goals. Capability descriptions are used to define what the plan does, its competencies. System goals are mapped to plans and associated methods used to achieve those goals through these capability descriptions. The EES was thereby designed to 'understand' the goals that it might be called upon to explain.

Several researchers outside the intelligent and knowledge-based systems community have since highlighted the potential for DR to act as an explanatory knowledge base [14, 15, 18, 20]. These works point to the potential utility of DR as the basis for informed discussion between system designers and system users, and between designers and external stakeholders. Some have claimed that DR's primary benefit is as a facilitator of this cross-party communication, rather than as a cognitive aid to designers or as a form of documentation, as it is often assumed [24].

DR helps to narrow the "gulf of understanding" [21] that exists between users who are domain experts and designers who understand how a particular system was intended to operate within a domain. DR is a critical element in the portfolio of communications tools that are employed in a complex development project. The techniques and tools developed to support DR capture and transfer are communications and organizational memory devices that can help to bridge the knowledge gap between what a given system "knows" about the domain, tasks, and user, and what users know about the complex tools that they use [28].

2.4 Three Cases of Design Rationale as Explanation

Prior research discussed in Sect. 2.3 highlights the potential for DR to improve the understanding of end users and other stakeholders external to the development team on a complex systems project. Despite this promise, relatively little work has empirically investigated this potential. In the sections that follow, I describe three condensed case studies that explore various aspects of these ideas. The first study was carried out with graduate student participants in a partially controlled environment. The second and third are field studies where DR was captured and is being used, in the first as a vehicle for a technology transfer and in the second as the basis for system evaluation and iterative redesign. The three cases reveal some of the challenges to harnessing the explanatory potential of DR, but also the opportunities for DR to contribute to the comprehensibility of complex systems.

2.4.1 A Transparent User Interface: VentureQuery

The first case is the VentureQuery project, which explored whether concepts and techniques of DR could be leveraged to provide an implementable model of embedded explanations for a software application. As discussed earlier, the theory underlying this research is that DR-based explanations may help to make more transparent the structure of a system by exposing design team deliberations including system requirements, envisioned use scenarios, and the technical, cognitive, organizational, and other constraints applied in the development process.

The VentureQuery project involved analysis, design, and construction of a software system to create and automatically publish electronic questionnaires on the web. A goal of the research design was to provide a project of realistic complexity to act as a source for DR, and to capture and structure the DR in a system capable of providing it back to system users. The team decided that the target application would consist of a web-based question–answer system in the form of a venture capital-seeking "game". The application was intended to help to educate novice e-business entrepreneurs in the venture capital-seeking process.

The project team consisted of 12 graduate students drawn from a Masters of Science course at a UK university. About half of the project team had significant systems development experience and all had a strong interest in the process of system design. Twenty-one meetings of the core design team were recorded on audiotape. An additional three meetings between members of the design team and various project reviewers and

potential users were also recorded in full. Meetings averaged 90 min. Tapes were transcribed to text files resulting in over 400 pages of design meeting dialog. In addition to the design meeting tapes, other project artifacts were analyzed for their contribution to the DR including domain analysis documents, design documents (e.g., flip chart drawings), various Unified Modeling Language (UML) diagrams, meeting agendas and notes, and e-mail between team members.

Design meeting transcripts were analyzed using the Atlas/ti (www.atlasti.de) qualitative analysis software package. Coded transcript fragments were fairly coarse-grained to ensure that the context of a given design deliberation was not lost in analysis. Based on word counts, approximately 52% of the content of the design meetings related directly to design of the application. Design deliberations were coded, extracted, and then captured as DR in a database developed to act as a knowledge base for the systems explanation facility.

The Questions, Options, Criteria (QOC), DR semi-formalism [17] was used as the representational medium for the study. This selection was made based on QOC's balance of ease of use with representational fidelity. QOC is a relatively simple and sparse method for representing DR. This simplicity was deemed an essential trait in the context of this study, as it was felt to most closely parallel the selection criteria likely to be applied in applied project settings, where practitioners are unlikely to invest time learning a potentially more richly expressive, but necessarily more complex and difficult to use formalism.

In the QOC notation, *Questions* highlight issues that have been identified as relevant to the design, *Options* are the potential solution approaches that have been identified to address a given question, and *Criteria* are the reasons that are considered for or against each of the identified options. Whether a criterion is considered a positive or negative factor in the evaluation of a given option is represented in the links, known as *Assessments*, between Options and Criteria. Assessments in QOC are not assigned weights to represent their relative importance to the argument for an Option. Criteria may be instances of *Metacriteria* (such criteria are called bridging criteria by QOC's designers), though this relationship is not required. Finally, Questions may be derived from Options (the Consequent Questions of QOC) as a particular design issue is discussed. In addition, the framework was elaborated with the code *QOC Outline*, which was used to relate a particular element of the DR to the specific application component (generally, a Java class). The code set used and data counts appear in Table 2.1.

Table 2.1. QOC code set and data counts

QOC element	# captured
QOC outline elements	25
total questions	151
total options	339
total criteria	122
meta-criteria	21
bridging criteria	87
consequent questions new questions that arose as a result of a selected option.	32
assessed option–criterion Pairs option–criterion pairs for which an explicit assessment, + or −, was derivable directly from the meeting transcripts and other materials.	114
un-assessed option–criterion pairs option–criterion pairs for which no explicit assessment, + or −, was derivable directly from the meeting transcripts and other materials.	322

After being coded as QOC, elements of the DR were cross-coded to identify the explanatory information represented by each QOC component. The code set used was based on a taxonomy of explanation types derived from prior research on the philosophy of explanation see [15]. The purpose of this cross-coding was to identify the types of explanations provided by DR and to serve as a schema for explanation delivery at application runtime. Explanation types are divided into two types: *operational* explanations that provide basic information about the design, and *why* explanations conforming more closely to conceptions of explanation content as appealing to deeper knowledge about the application domain. Both the operational and *why* explanation code sets and data counts appear in Tables 2.2. and 2.3.

Though the operational explanation types are straightforward, the *why* explanation types require further definition. Deductive-nomological (law-based) explanations are those based on the constraints (laws) imposed by the underlying technical aspects of the system and from the need to conform to standards and legislative statutes. Functional explanations are those that relate directly to the purpose or requirement of a system or component, for example, use scenarios and desired outcomes from use.

Table 2.2. Operational explanation code set and data counts

operational explanations	count (%)	examples
What is it?	56/37	what is user answer?
		what are the attributes of user answer?
how do I use it?	35/23	what user answer input formats are supported?
		can question wording be varied?
how does it work?	54/36	how are user answers validated?
		how are question dependencies managed?
other	6/4	who will system test the application?
		who will own the rights to the application?

Table 2.3. *Why* explanation code set and data counts

why explanations	count (%)	Examples
D–N (law-based) explanation	14/11	*we are constrained by http* *EU privacy laws prevent us from storing that*
functional explanation	108/89	*what is the purpose of user answer?* *we need user square path to tailor questions based on prior responses*

Discussion and Lessons Learned

One of the most important results from the VentureQuery case study was that much of the design deliberation, including crucial assessments of criteria against design options, as well as the actual process of deciding on elements of a final design, were not made explicit in the design process, as shown in Table 2.1. Though this retrospective approach to rationale capture did help to work around some of the design process disruption associated with integrating QOC into a project "ecology" [5], in the context of explanation content capture the costs of not following the approach appear to be too great. The decision not to follow an explicit DR process meant that deliberations on a particular design issue did not always result in a complete QOC structure, with, for example, multiple Options generated for each Question, and each Criterion explicitly applied to the evaluation

of each Option. This finding lends weight to Schön's argument [23] that deliberate techniques must be applied in technological design in order to promote explicit consideration and reflection during the design problem solving process.

Though it has been claimed that the most significant issues in any software project are discussed in design meetings, rather than informal discussions or not at all [24], it is also possible that certain implementation decisions are made in isolation by individual members of the project team, and therefore, never deliberated and never recorded as part of the rationale. Such "rationale ambiguity" may be an unavoidable, even essential characteristic of technological design and construction [2]. If large, complex design and development projects are to be completed within their inherent resource constraints, not every decision and relevant factor can be deliberated, and the challenge becomes one of defining an acceptable level of ambiguity rather than eliminating it altogether. That said, this ambiguity poses a significant challenge to providing comprehensive explanations.

Another problem that emerged was that of explanatory completeness with respect to design *questions* and how they appear to be answered in practice. Analysis of full-text meeting transcripts suggests that design options sometimes emerge almost mystically from design discussions. It was sometimes difficult to see the chain of reasoning that led to a particular design option being proposed and then being either accepted or rejected. This problem was especially acute in situations where a design option took the form of a high-level design object, for example, a class, and then candidate object components were enumerated in rapid succession. We might expect such cases to generate a rich set of rationales, but the conversation moved so quickly between foci that much of the information required to populate the QOC was found to be missing. This again highlights the potential role of a reflective design process in helping to make these assumptions more explicit.

There was an apparent asymmetry with respect to the amount of discussion allocated to certain features over others. This asymmetry was especially acute with respect to *what* questions vs. *how* questions. Relatively little discussion was evoked by the identification of a new candidate entity for the system, while discussions of new processes more often resulted in long discussions. This seemed to lead in many cases to the inclusion in the design of system entities that were poorly defined and poorly understood outside the context of the processes in which they played a role. This is problematic in the context of object-oriented design, where the generation of a complete justification and description for a given entity can assist with the creation of more modular system objects with more well-defined semantics and behaviors.

2.4.2 Design Rationale for Technology Transfer: ATFP

The second case is an investigation into the use of DR as a facilitator of technology transfer. Since the Spring of 2002, we have been working with the United States Marine Corps on a decision model and cognitive support system to aid effective allocation of antiterrorism and force protection (ATFP) resources at Marine Corps installations. A central concern for the ATFP work is the migration or transfer of the technology across institutional boundaries and its adoption into local practices. In this particular case, technology developed in an academic partnership with a unit within the Marine Corps is to be transferred to other units both within the Marine Corps and to other services and government organizations. The work is ongoing and the report here is only a preliminary treatment.

The decision model and system developed for Marine Corps antiterrorism officers, facilities planners, public works officers, and military police provides support for asset prioritization, calculation of antiterrorism mitigation project utilities, resource allocation, and acts as a repository for organizational learning in the ATFP domain. Requirements have been gathered and refined through a series of briefings, informal and formal design reviews of the evolving prototype, and cognitive walkthroughs [22] with prospective users at Marine Corps installations. Over 100 Marine Corps officers and civilian personnel have reviewed the project, and over 30 have participated in focused cognitive walkthroughs.

In addition to the core decision model and cognitive support system that implements it, the project involved development of a range of knowledge resources to aid users working in the domain including a training module and explanation facility. The project's Website includes a scenario editor that captures details of real and envisioned interactions with the system in response to a range of decision making and planning problems collected in the field.

Discussion and Lessons Learned

We have found that a range of factors impact opportunities for successful adoption of the ATFP system at different installations. These include 'microscopic' issues such as domain terminology, which is unfamiliar to many planners facing ATFP problems for the first time, and 'macroscopic' issues such as whether the Department of Defense or Headquarters Marine Corps would mandate the use of a particular ATFP planning approach and supporting tools.

An initial finding from this work is that DR, in the form of scenarios of use and associated claims analyses [8] can act as boundary objects [25] to facilitate knowledge sharing, adoption decision-making, and technology

evaluation across organizations. Boundary objects are "those objects that are plastic enough to be adaptable across multiple viewpoints, yet maintain continuity of identity" [25]. Central to the utilization of boundary objects as a theoretical orientation for technology transfer is Star's claim for the efficacy of boundary objects as touchstones for understanding among members of distributed and culturally diverse communities. According to Star, a 'good' or effective boundary object has many identities, definitions, and interpretations. One important insight is that the most effective boundary objects are those that are able to evoke and make explicit the largest quantity of tacit knowledge in a particular problem context [6]. In this way, the use of DR as boundary objects serves as the basis for the *active construction* of explanations for end users and other project stakeholders. In reviews and walkthroughs of the ATFP system, we witnessed cases where prospective users developed their understanding of the application by reflecting on scenarios supplied by previous users and how they were employed in development of a decision model, as exemplified in the walkthrough quote:

...if we had more of these [scenarios], *more well fleshed out and then these linked to these models, and someone could browse the scenarios and say, okay, that's sort of like my scenario, kind of like what I'm going to do here.*

We are actively exploring ways in which scenarios and other elements of the ATFP system DR can be most effectively captured and then packaged to facilitate explaining complex systems, and thereby fostering their adoption, and use. For example, we are exploring the use of a claims taxonomy tuned to the requirements of technology transfer as suggested in prior research. One such taxonomy focuses on relating each scenario to aspects of system *comprehensibility*, development organization *credibility*, and system adoption *cost*. The ultimate objective of this work is to provide prospective system adopters with means to understand how a particular system design relates to specific scenarios of use, and what this mapping entails for the technology adopting organization.

2.4.3 Design Rationale from Evaluations: PMLAV

The third and final case involves another system project from the Marine Corps domain. This project does not involve design per se, but rather an evaluation project in which DR concepts and representational tools have been used as focusing principles to guide the work and organize its results. As in the second case, techniques from scenario-based design and claims analysis were used, although adapted here to the task of evaluation of a

complex, computer-supported cooperative work (CSCW) system that supports product lifecycle management (PLM).

Once again, the setting for the study is a unit of the United States Marine Corps, the office of the program manager, light armored vehicles (PM LAV). The PM LAV has implemented an integrated digital environment (IDE) to support the cooperative and collaborative work of both civilians and Marines in their use, maintenance and evolution of the LAV. The IDE is used by 70 PM LAV personnel as well as at Marine maintenance depots in the United States and in the field.

The PM LAV IDE includes communications (e-mail, videoconferencing), workflow, document management, project management, collaborative engineering, and performance reporting functionality. The PM LAV monitors a fleet of about 800 light armored vehicles deployed worldwide in a variety of configurations and tasked with a range of missions. The program manager is responsible for monitoring LAV health and field performance, developing enhancements to the vehicle, and directing vehicle maintenance. The IDE is designed to support this work with an integrated environment for communications and information management.

We interviewed PM LAV personnel across the organization from staff assistants to division chiefs with a range of responsibilities including engineering, logistics, and business operations. We used a semi-structured interview guide. The guide was designed to elicit scenarios and claims (DR tools) as the basic unit of analysis for the evaluation. An abbreviated version of the interview guide appears in Fig. 2.1.

1. **General questions: elicit roles, task goals, & setting**
   - Position, role, key tasks and priorities, collaborators, etc
2. **Questions related to current system use: scenarios & claims**
   - Describe scenarios of use with the IDE
   - How do these scenarios contribute to the PM LAV mission? (evaluative claims)
   - (follow-on questions, probes)
3. **Questions related to prospective system use**

Fig. 2.1. PM LAV IDE interview guide

The interviews were recorded, transcribed, and then coded. We were particularly interested in obtaining participant descriptions of their use scenarios and how they felt that IDE support for these scenarios contributed to their work and to the mission of the PM LAV.

Analysis of the interview transcripts yielded 43 unique scenarios. Twenty seven of these are scenarios describing actual, current use of the system, and 16 were scenarios envisioned by study participants. We identified 464 total claims in the transcripts, where claims were propositions

made about the system's support for scenarios. Following the claims analysis technique, these propositions were assessed as either positive or negative. Individual scenarios were related to between 0 and 22 claims each, with a mean of 5.6. In addition, the method identified 223 claims that were disconnected from any one scenario but in general represented statements about the IDE system as a whole. After the scenarios and claims had been gathered, we went back to the PM LAV to validate our scenario-claim sets in focus groups. Results from this part of the study suggested that our scenarios and claims were relatively complete and fairly representative of PM LAV perceptions towards the system.

Discussion and Lessons Learned

The scenarios and claims we extracted from the study may be seen as a reverse engineering of the original DR. They represent in many ways the consequences of the original DR, and describe how the system is perceived by its users in use. They help explain how the system is performing and how it is perceived by users. At the same time, they serve as a blueprint for redesign of the system and, further, account for the use context that is impossible to predict at design time.

Among the interesting aggregate findings from the study was the extent to which the scenario-based technique was able to ground evaluation in situations where the system is heavily used and in situations where use is especially consequential in the daily work life of the study participants. Of the 74 discrete features of the IDE (identified by us at a relatively arbitrary level of granularity), scenario coverage identified only 19 of these as being all important or impactful to the organization's mission and priorities. This finding has important implications for requirements engineering, at least in this case, and suggests that the use of scenario-based design in the early stages of the project may have helped focus development time and money on the most important aspects of the system.

The technique was able to elicit evaluative claims spanning a range of topics including how the organization's technical infrastructure, the design of the IDE itself, psychological and social–psychological, and organizational factors were all implicated in either or both the success or failings of the system. In terms of the IDE design, for example, we identified a number of critical areas for redesign including areas where the system possessed insufficient functionality, problems with usability and ease of use, inflexible task support, performance and reliability issues, and problems with security and accessibility of the system. Because claims in each of these areas are linked to specific scenarios of use, they implicate

particular system features and functionality and suggest how they can be improved through future development efforts.

One area where the evaluation technique failed was in helping to identify the contributions gained from implementation of the IDE within the organization. Costs of systems such as this are relatively easy to measure; just sum the invoices from the development contractors and system integrators (though this of course does not account for the true lifecycle costs of these systems). Effective techniques to measure the benefits of distributed, collaborative systems, however, remain elusive. One of the chief aims of the method as described here was to link perceptions of system benefits, in the form of claims, to the specific scenarios supported by the IDE. Of the 212 claims we identified as relating to the system's contributions, only 6% were truly measurable benefits, with 26% being what we classified as tangible but immeasurable benefits, and 68% intangible.

2.5 Challenges and Opportunities for Design Rationale as Explanation

A number of challenges and research opportunities emerge from consideration of the findings from these three studies. Results from the VentureQuery case study highlight the difficulties associated with capturing complete DR when design activities occur not only in formal meetings, but also in informal and individual forums. If completeness is a critical attribute of the DR for a system, such as when it is used to provide an explanatory knowledge base, then design knowledge capture is one of the most pressing challenges for research. Design is a ubiquitous activity and can happen as often in the mind of a single individual riding on the train as it does in more formal contexts where it is amenable to capture. The challenge of pervasive design capture raises many questions to occupy researchers.

Our experience on the VentureQuery project also suggests that adoption of a DR process, in addition to notations and supporting tools, may help ensure capture of a more complete design knowledge base. Process prescriptions for experienced designers are, however, notoriously difficult to enforce as they are seen as disempowering these creative individuals. Explorations into better ways of integrating DR techniques into the day-to-day work of designers may help ease this problem, as might better tool support for DR capture.

Our work with the Marine Corps on antiterrorism planning decision models and tools has suggested a role for DR, in the form of scenarios,

claims, and related systems artifacts, as a vehicle to facilitate transfer of technology between organizations. This work is ongoing and our results are tentative but experiences with antiterrorism planners in the field suggests the potential utility of DR, in the form of scenarios and claims analyses, as explanatory transfer packages to help prospective system adopters evaluate new technologies. Questions have already emerged about the form such a package should take, and the kinds of DR that are most useful for technology adopters in different roles and contexts.

Our work with the Marine Corps PM LAV on complex, distributed system evaluation suggests that there may be valuable linkages to be developed between evaluation techniques and tools and those used for DR. Developing techniques to relate DR to evaluation, then forward to redesign and subsequent evaluation in a cycle of learning and artifact improvement may be one way to achieve a truly progressive systems design science. Still under-researched are the downstream consequences of design decisions made when a system is still an abstract model as unrealized in working software. Repositories of DR that provide a longitudinal view of design deliberations and their consequences may help us better understand the effectiveness of the different design methodologies and tools created to support the systems design process.

It is not expected that the cases presented here and the lessons learned from their analysis will apply to all settings in which DR and design capture are attempted. In particular, situations less contrived than the 'zoo' study reported in the first case, and less structured and formal than in the second and third, may exhibit very different characteristics and outcomes. Still, empirical studies of DR are lacking and it is hoped that these cases can contribute to the evolving base of experiences with DR in both controlled and field settings.

2.6 Conclusion

This chapter has described three cases of design rationale in use as explanations. The theory motivating this work is that access to DR by an expanded group of project stakeholders, to include end users, may have the potential to significantly increase the comprehensibility of systems tools. This potential may be greatest for users of sophisticated software applications in complex domains, especially those users who require or desire a deeper understanding of the contextual factors that guide and constrain the design process. DR techniques and tools may have the added benefit of facilitating a kind of *virtual participatory design* in which users are able to provide meaningful input to the evolution of their systems. For design

rationale to make sense may depend on showing how the costs of capture can be recovered through new and innovative uses of these design knowledge stores. Focusing on DR as a means to facilitate explanation and other communication between development project stakeholders may represent one way to expose this value proposition.

Acknowledgments. This work was supported by the United States Marine Corps through the Marine Corps Research University. The chapter was substantially improved by three anonymous reviewers who commented on an earlier draft.

References

[1] Bar-Yam, Y (1997) Dynamics of complex systems, Addison-Wesley, Reading, MA
[2] Bowker G, Leigh-Star S (1994) Knowledge and infrastructure in international information management: Problems of classification and coding. In: Bud-Frierman L (ed) Information Acumen: The Understanding and Use of Knowledge in Modern Business, Routledge, London, pp. 187–216
[3] Brooks, FP (1987) No silver bullet: Essence and accidents of software engineering. IEEE Computer, 20(4): 10–19
[4] Buchanan, BG, Shortliffe, EH (1984) Rule-based expert systems: the MYCIN experiments of the Stanford Heuristic Programming Project. Addison-Wesley, Reading, MA
[5] Buckingham Shum S, MacLean A, Bellotti V, Hammond N (1997) Graphical argumentation and design cognition. Hum.–Comput. Interact., 12(3): 267–300
[6] Carlile PR (2002) A pragmatic view of knowledge and boundaries: Boundary objects in new product development. Org Science, 13(4): 442–455
[7] Carroll, JM (1995) Scenario-based design: Envisioning Work and Technology in System Development, Wiley, New York
[8] Carroll, JM (2000) Making Use: Scenario-Based Design of Human–Computer Interactions, MIT Press, Cambridge, MA
[9] Carroll, JM, Rosson MB (1992) Getting around the task-artifact cycle: How to make claims and design by scenario. ACM Trans. Inform. Sys., 10(2): 181–212
[10] Clancey WJ (1983) The epistemology of a rule-based expert system – A framework for explanation. Artificial Intelligence, 20: 215–251
[11] Clancey WJ (1987) Knowledge-Based Tutoring: The GUIDON Program, MIT Press, Cambridge, MA
[12] Dutoit, AH, McCall R, Mistrik I, Paech B (2006) Rationale Management in Software Engineering. In: Dutoit AH, McCall R, Mistrik I, Paech B (eds.) Rationale Management in Software Engineering, Springer, Berlin Heidelberg New York

[13] Graesser AC, Person N, Huber J (1992) Mechanisms that Generate Questions. In: Lauer TW, Peacock E, Graesser AC (eds.) Questions and Information Systems, Lawrence Erlbaum, Hillsdale, NJ, pp. 167–187
[14] Gruber T (1991) Learning why by being told what. IEEE Expert, 6(4): 65–74
[15] Gruber TR, Russell, DM (1996) Generative design rationale: Beyond the record and relay paradigm. In: Moran TP, Carroll JM (eds.) Design Rationale: Concepts, Techniques and Use. Lawrence Erlbaum, Mahwah, NJ, pp. 21–51
[16] Haynes SR (2000) Explanation in information systems: Can philosophy help? Paper presented at The Eighth European Conference on Information Systems (ECIS), July 3–5, 2000, Vienna, Austria
[17] MacLean A, Young RM, Bellotti VME, Moran, T (1996) Questions, Options, and Criteria: Elements of Design Space Analysis. In Moran TP, Carroll JM (eds) Design Rationale: Concepts, Techniques and Use, Lawrence Erlbaum, Mahwah, NJ, pp. 21–51
[18] MacLean A, McKerlie, D (1995) Design Space Analysis and Use-Representations. In: Carroll JM (ed.) Scenario-Based Design: Envisioning Work and Technology in System Development, Wiley, New York
[19] Mirel B (1998) Minimalism for complex tasks. In: Carroll JM (ed.) Minimalism beyond the Nurnberg Funnel, MIT Press, Cambridge, MA, pp. 179–218
[20] Moran TP, Carroll JM (1996) Design Rationale: Concepts, Techniques, and Use. Lawrence Erlbaum, Mahwah, NJ
[21] Norman DA (1986) Cognitive Engineering. In: Norman DA, Draper SW (eds) User Centered System Design: New Perspectives on Human–Computer Interaction. Lawrence Erlbaum, Hillsdale, NJ, pp. 31–61
[22] Polson PG, Lewis C, Rieman J, Wharton C (1992) Cognitive walkthroughs: A method for theory-based evaluation of user interfaces. Int. J. Man–Mach. Stud., 36: 741–773
[23] Schön DA (1983) The reflective practitioner: how professionals think in action. Basic Books, New York
[24] Shipman, FM, McCall, RJ (1996) Integrating Different Perspectives on Design Rationale: Supporting the Emergence of Design Rationale from Design Communication (PDF CSDL 96-001). College Station, TX: Center for the Study of Digital Libraries, Texas A&M University
[25] Star SL (1989) The Structure of Ill-Structured Solutions: Heterogeneous Problem-Solving, Boundary Objects and Distributed Artificial Intelligence. In: M. Huhns M, Gasser L (eds.) Distributed Artificial Intelligence, Vol. 2, Morgan Kauffmann, Menlo Park CA, pp. 37–54
[26] Swartout W, Paris C, Moore J (1991) Design for explainable expert systems. IEEE Expert (June): 58–64
[27] Swartout WR (1983) XPLAIN: A system for creating and explaining expert consulting programs. Artif. Intell., 21: 285–325

[28] Winograd T (1995) Forward. In: Newman WM, Lamming MG (eds.) Interactive System Design. Addison-Wesley, Reading, MA
[29] Ye Y, Fischer G (2002) Information Delivery in Support of Learning Reusable Software Components on Demand. Paper presented at the International Conference on Intelligent User Interface (IUI), January 13–16, 2002