Knowledge Systems Laboratory
Technical Report KSL 92-59

December 1991
Updated February 1993

Generative Design Rationale: Beyond the Record and Replay Paradigm

by

Thomas R. Gruber
Daniel M. Russell

To appear in a forthcoming collection on design rationale edited by Thomas Moran and John Carroll, to be published by Lawrence Erlbaum Associates.

KNOWLEDGE SYSTEMS LABORATORY
Computer Science Department
Stanford University
Stanford, California 94305

Generative Design Rationale: Beyond the Record and Replay Paradigm

Thomas R. Gruber
Knowledge Systems Laboratory
Stanford University
701 Welch Road, Building C
Palo Alto, CA 94304
gruber@sumex-aim.stanford.edu

Daniel M. Russell
Systems Sciences Laboratory
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
dan_russell@parc.xerox.com

Updated February 1993

Abstract. Research in design rationale support must confront the fundamental questions of what kinds of design rationale information should be captured, and how rationales can be used to support engineering practice. This paper examines the kinds of information used in design rationale explanations, relating them to the kinds of computational services that can be provided. Implications for the design of software tools for design rationale support are given. The analysis predicts that the "record and replay" paradigm of structured note-taking tools (electronic notebooks, deliberation notes, decision histories) may be inadequate to the task. Instead, we argue for a generative approach in which design rationale explanations are constructed, in response to information requests, from background knowledge and information captured during design. Support services based on the generative paradigm, such as design dependency management and rationale by demonstration, will require more formal integration between the rationale knowledge capture tools and existing engineering software.

1. Introduction

When a product is designed, the primary output of the design process is a specification of the artifact, such as annotated CAD drawing or a circuit schematic. Additional information, about why the artifact is designed the way it is, is not captured in the artifact specification. Research in design rationale support is concerned with how this additional information might be effectively captured and used.

A design rationale is an explanation of why an artifact, or some part of it, is designed as it is. Such an explanation can include several kinds of information. A rationale may explicate tacit assumptions, such as the expected operating conditions for a device and its intended behavior under those conditions. It may clarify dependencies and constraints among design parameters, such as the set of modules that might need to be changed if a given module is replaced. A design rationale may also justify or validate a design, explaining why particular structures are chosen over alternatives, or how required functionality is achieved by device behavior.

Knowledge of the rationale for a design is needed for engineering tasks throughout the product lifecycle, including the refinement of early conceptual designs, the realization of design specifications through manufacturing, and re-engineering under changing requirements. Design rationale information can help engineers reuse, modify and maintain existing designs. Unfortunately, the design rationale for complex artifacts mainly resides in the heads of the original designers, who are often not available or cannot remember the rationale. Existing methods of documentation do not adequately capture the information needed.

A technology for software support of design rationale is just beginning to be developed. A range of approaches has been proposed, mostly at the laboratory stage.[1] To evaluate the applicability and utility of these techniques, and chart the future direction of work in this area, we must address two fundamental questions: what kind of information should be captured, and how can that which is captured be used to support engineering practice. This paper examines these two questions in depth, and offers prescriptive implications for the design of rationale support tools.

2. What kind of design information should be captured to support rationale?

To address the question of what information to capture for design rationale, we start by asking what kind of information is available and used in existing design practice. It would be of little use to build technology to represent and reason with knowledge that cannot be acquired, or is of little benefit to the information needs of engineers. To address this issue, we turn to data on the information that is available and used by designers when explaining the rationale for designs.

In a survey of design protocol studies (Gruber & Russell, 1992), we looked at how designs are explained in documents and live discussions. We examined the data from a set of empirical studies of designers requesting, communicating, and using design information. The data include protocols of individual designers thinking aloud during innovative design, teams of designers discussing a previous design during redesign, designers asking for information during an initial design, and designers negotiating with stakeholders about a design in progress. The studies covered designs in several domains: architecture (Pirolli & Goel, 1989), electromechanical devices (Baudin, Gevins, Baya, & Mabogunje, 1992; Baudin, Gevins, Baya, Mabogunje, & Bonnet, 1992; Baya et al., 1992; Leifer et al., 1992), (Kuffner, 1990; Kuffner & Ullman, 1990), heating and air conditioning systems for buildings (Garcia, 1991; Garcia & Howard, 1992), user-interfaces in software and hardware (Bellotti & MacLean, 1989), and the design of instructional courses (Pirolli & Goel, 1989).

As an example, consider the following protocol fragment from one of the studies surveyed (Baudin et al., 1992). A mechanical engineer has been given the task of redesigning a computer-controlled automobile shock absorber. The engineer doing the redesign (R) is asking one of the designers of the original device (D) about the design of the damper mechanism.

[1]Many of the approaches are represented in this volume and journal special issue (Carroll & Moran, 1991). Gruber, Boose, Baudin, & Weber (1991) offer a preliminary analysis of the design space for support tools

R: The existing equipment ... What is the problem with it? Why can't it go up to 750 pounds [resistive force]?

D: There are related problems ... heat transfer... the solenoid is ... exposed to a lot of heat. And solenoid is a heat sensitive device, and cannot function above a certain temperature. ... From the vendors we got a limit of 120 degrees centigrade for the solenoid... This is a critical point in how high a force you can get. ... If you are dissipating at high force and high stroke lengths and high speed, then a lot of heat is generated.

R: Did you anticipate that problem?

D: Yes ... The ideal thing would be for us to have the solenoid outside and to get heat convected out by air. But once we put it outside ... we were using up more volume than we had, so we chose to put it inside and to insulate the solenoid.

R: Okay ... The heat problem is associated with the solenoid. If you used a different device, then may not ...

D: We looked at various different kinds of actuators. Solenoid is just an actuator which gives you a certain force when passed it a certain current and voltage. ... there's a tradeoff that we have so little power and we have to generate high force. The solenoid was chosen because with less power it can generate high force at various small stroke length.
...

R: If I get a solenoid with very good insulation. That would be too expensive[?]

D: No, the best you can get can go up to 175 degrees centigrade.

The designer was given a changed requirement for the device to produce a higher resistive force. In the previous design, driving the device to produce the higher force would have resulted in extra heat. He is trying to understand why the previous damper cannot be run at a higher temperature, and is told that the solenoid would not tolerate the heat that would be generated. In exploring alternatives, he wonders why the previous design did not use a different kind of actuator or a solenoid that is adequately insulated. He hypothesizes incorrectly that such a solenoid might have been too expensive. In fact, even the best insulated solenoid available could not tolerate the extra heat.

We analyzed the questions, conjectures, and statements of designers in settings like the above, noting the types of rationales requested, generated and used by designers. Appendix 1 summarizes the categories of information requested, in the form of generic questions asked about a design. Each generic question is abstracted from one or more explicit information requests or think-aloud conjectures made by designers as recorded in the protocols. The data indicate the range of questions that are asked about why a design is as it is.

The survey resulted in a set of general observations about the information underlying design rationale. Each observation motivates a conclusion about the information that should be captured to create design rationales like those we see designers using.

Observation 1. Rationales are based on many kinds of underlying information.

The questions designers asked about the rationales for existing designs included a broad range of sources and subjects, as listed in Appendix 1. The information sources included people, documents, databases, CAD tools, spreadsheets, and notes. The subjects included requirements, constraints, decisions, artifact structure, expected behavior, and intended function.

We found that the scope of design rationale information is very broad—broader than the current literature in design rationale might suggest.

No single model of the design process (e.g., design as argumentation, design as decision making, design as constraint satisfaction) accounted for a dominant share of the questions. Furthermore, rarely did the language of a rationale explanation in a protocol suggest a unique process model; the protocol reader must reinterpret the natural explanations using the terms provided by the design process model (e.g., "issue", "option", "constraint" etc.). ...

R: What of these pieces already existed? Was it just, what'd I do with them?

D: Right. Yeah. See, these disks are alternate disks. You see, one is friction disk, and the other is a metal disk... and they're made together to get friction. And the concept was used from a motorcycle clutch, although we got these manufactured to our dimensions.

...

This fragment of discussion between designers gives an illustration of a non-issue, non-rationale conversation. D begins the answer by giving some background context before answering. Rationales are more like ongoing conversations: they exist in a problem-solving context that may not be directly modelled by design information alone.

Conclusion: Capture the information that is used to answer designers' questions, not the information that fits a preconceived model of the design process. The observation that designers draw on many kinds of information in explaining designs is not surprising. However, it suggests that design rationale support tools should not limit the captured information to that which fits a particular view of the data. Many of the tools described in the current design rationale literature do just that: they are based on models of the design process that prescribe what information is relevant and provide a representation for capturing it. For example, a tool based on a view that design is constraint satisfaction would only capture formal design constraints among mathematical parameters. However, this tool will not support explanations drawing from natural language documents, CAD drawings, and other relevant sources. Similarly, only capturing information that can be labeled as an argument for or against a design decision may preclude other information that is relevant to the rationale in unforeseen ways, such as descriptions of expected artifact behavior.

Observation 2. Rationales are constructed and inferred.

Rationale explanations are often *constructed* and *inferred* from information that is stored, rather than being stored as complete answers. For example, even though the original damper design was carefully documented using a design notebook, the rationale for rejecting the insulated solenoid alternative was constructed in response to a hypothetical question ("If I get a solenoid...No, the best you can get..."). Often the explanations are constructed in response to *new* questions that arise in the context of redesign. ("[Can I] get a solenoid with very good insulation?") Answering these questions requires inference from stored information, rather than a database lookup for the answer.

Although the original designers in the protocol studies did not often record complete explanations, they did record many of the constraint equations and component specifications that were used to construct explanations. For roughly half of the information requests in the protocols, answers were computed or inferred from information that was stored or could have been accessed; the answers to these rationale questions could not be retrieved directly from a design record.

Conclusion: Acquire data rather than answers. If many rationale explanations are inferred from data, it is more important to capture the data that might be used to infer answers to later questions than to try to anticipate the questions and capture pre-formulated answers. In particular, it is important to identify the parameters of the design that are used in standard inference procedures, such as predicting behaviors from structures, verifying functions, and searching for parts meeting some specifications.

Observation 3. Rationales are not just statements of fact, but explanations about dependencies among facts.

Design rationales are often *explanations*, rather than simple statements of fact. An argument consisting of a set of reasons for and against a position is an explanation, whereas the individual facts offered as evidence are not. In the damper example, the explanation of a choice among alternatives is found in the relationships among several pieces of information: a requirement ("we have to generate high force"), the functional role of a component ("solenoid is just an actuator which gives you a certain force"), and a limiting constraint ("This is a critical point in how high a force you can get. ...If you are dissipating at high force and high stroke lengths and high speed, then a lot of heat is generated."). The rationale explanation is the statement that these particular facts, among all the data available on the design, are combined to explain the decision to choose the solenoid.

Frequently rationale explanations describe *dependencies* among decisions or design parameters. Dependency relations are important for *managing change* in designs. In the protocols surveyed, designers often ask questions or make conjectures about the effects of changes to the design. Here are few examples from several of the protocols covered by the survey.

"Now if we changed GH, would we have to change anything else in the frame?"

"What will I have to do if I try to extend the force range from 500 to 750?"

"What can I do with more current which could not be done with less current?"

"The other thing that would have to change would be the spring, like I said, you adjust the spring constant if you are damping this."

"Supply chilled water temperature should be changed to 45 degrees F. All coils need to be recalculated at 45 degrees F."

Conclusion: Capture dependency relationships. Answering questions such as those above requires knowledge of the dependency relationships among elements of the design (i.e., physical components, model parameters, functional requirements, other requirements, evaluation criteria, and decisions). Thus to support rationale, it would be useful to capture these relationships. It would not be necessary to acquire a formal theory capable of *inferring* the dependencies. Dependency information can be captured as semiformal links (e.g., between the water temperature parameter and the coil parameters). Although these semiformal links represent relationships among design elements that are not formally modeled, the links can be used by tools that perform simple dependency management services. For example, software engineering tools called source management systems are used by program developers and document writers to maintain complex dependencies among modules, yet the source management tool does not know what the modules represent.

Observation 4. Rationale explanations refer to real engineering data and models.

In the protocol review study, we examined the source of each piece of information mentioned in designer questions or used in answers. We found that much of the information used in rationale explanations is either available from information sources already available to the practicing engineer, or could be made available if the documentation were brought on line. For example, information about the availability of solenoids may be found in manufacturer databases, which are accessed routinely by mechanical engineers. The functional description of the solenoid as a force actuator is an example of standard engineering knowledge found in textbooks and handbooks. In addition, many of the more difficult questions asked about designs could be answered by reference to engineering *models*, such as structure models in CAD databases and mathematical behavior models used for simulation and analysis. Such models are used routinely in traditional engineering disciplines. For instance, equations describing relationships among power, stroke length, force, and temperature were maintained on-line in a spreadsheet. The damper designer could then point to those equations when explaining the relationship between a component and design constraints. Although formal models are not available in all domains, where models exist they can play integral roles in constructing rationale explanations.

Conclusion: Capture data and models used in engineering practice. Engineering design is supported by software tools and on-line information sources, and will become increasingly so in the future. Since the data and models manipulated by these tools and databases are constituents of rationale explanations, this information could be captured as a by-product of using these tools. For instance, the annotation facilities in CAD drawing tools could be extended to allow for hypertext-style linking into other design information. This sort of data capture is not possible unless rationale information capture tools are integrated with mainstream engineering tools. This conclusion is elaborated in Section 4.3.

Observation 5: Rationales can be reconstructed from the relevant data.

Although many of the facts that are mentioned in rationale explanations come from formal models and engineering data, justifications for design decisions in the protocols studied were usually informal. Decision rationales were often lists of relevant data of the form, "The following factors were considered: ..." We call such a statement a *weak explanation.* A strong explanation would show *how* the factors led to the decision. In an automated design system that operates on a complete domain model, a strong explanation can be captured for each design step. For example, in explanation-based learning systems, an "explanation" of a design decision is a trace of the formal reasoning done, such as deductive proof of goal satisfaction (Mitchell, Keller, & Kedar-Cabelli, 1986). More generally, one can think of the trace of a design problem solver as a "derivation" that can be "replayed" to justify existing designs and create solutions for similar design problems (Mostow, 1989).

It is not surprising that most justifications were weak explanations, since the designers in the protocols were speaking or writing in natural language to other humans, rather than writing models or proofs for mechanical verification. In the domains we studied, the design process was not automated, nor were the models complete enough to generate strong explanations for most decisions. Fortunately, human engineers can make use of weak explanations, without the ability to simulate the mental processes of the original designer. For example, it is useful to a designer to know that the original designer of the damper "anticipated that [heat transfer] problem," without knowing exactly what the designer thought about heat problem.

Conclusion: Capture weak explanations (just the relevant data), when a complete justification for a decision is not available. From the first four observations we conclude that rationale support tools should focus on capturing the variety of data and models that are relevant to designers, from existing sources where possible, and manipulated by existing tools and databases. We found that the dependency relations among these data, models, and other information are especially relevant. The emphasis is on capturing the information, but what about rationale — the explanation? The final observation suggests that it is more important to acquire from the designer the relevant set of facts (to be able to reconstruct a rationale) than it is to assemble them into a coherent argument at the point of capture. For example, instead of asking the designer to document the deliberation about how to produce force without generating too much heat, a capture tool might only require the user to *associate* the requirement (perhaps by selecting a region of text in a document), the alternatives under consideration (solenoids, other force actuators, perhaps by references to a manufacturer's database), and their features that are relevant to the deliberation (force, current, heat). The explanation of how these elements justify a design decision could be left to the reader.

This conclusion is not a call to abandon active support for design rationale. When the information is available, a design rationale tool should be able to help generate the explanation. This idea is explored in Section 4.2.

3. How can design rationale support engineering practice?

The purpose of design rationale support is to improve design quality and the productivity of product development. Schemes for capturing, representing, and operating on design rationale should be evaluated with respect to their impact on engineering practice. In other words, to decide what design rationale information to capture, we should consider how that information might be used. In this section, we examine the tasks that might be supported by design rationale. We start by summarizing how the information was used in the verbal design protocols, and then consider the tasks that might be supported by software tools.

3.1 Uses of design rationale information in the design protocols

In our survey of design protocol studies, we classified the designers' questions and answers to identity a set of general categories of information use. Table 1 summarizes these categories, each representing a way that information about an existing design is used in the design process. Appendix 1 summarizes the kinds of questions asked about designs. The categories in Table 1 describe why such questions are asked and how the answers are used in design.

- *Clarifying requirements or assumptions* about the operating environment. ("Does [the pivot arm] flip all the way out, or [are there] two positions?" —manufacturing tool design)
- *Formulating a decision among alternatives.* ("[What materials could be used for the case?] ABS ... Polycarbonate..." —battery case design)
- *Understanding the artifact specification itself.* ("Is this [points to drawing] steel or aluminum?" —manufacturing tool design)
- *Explaining the effects of changes* to the design or a requirement. ("What will happen if I try to extend the force range from 500 to 700 pounds?" —damper design)
- *Explaining the expected operation of the device, often in hypothetical situations.* ("I don't know if you expect some receipt of something like that for your transaction...for parcels and registered letters." —postal ATM design)
- *Verifying and validating the design.* ("The area over the ... loading dock is covered. How do you plan to exhaust truck fumes from that area?" —building design)

Table 1: Categories of use for information about designs

3.2 Computational services that use design rationale information

While human designers can use rationale information in a variety of ways, the focus of this analysis is on computational uses: how design rationale information can be used by software tools to support engineering tasks. We survey four kinds of such services: information retrieval, decision support, dependency management, and rationale by demonstration.

3.2.1 Documentation and Information Retrieval

The service provided by many proposed design rationale tools is documentation, or more generally, information retrieval. The idea is simple: if we can capture design information on-line, we can retrieve the information when it is needed. The efficacy of the information retrieval service depends on the quality of the indexing. Design notebooks provide traditional indexing based on textual search or user-supplied indices (Hwang & Ullman, 1990; Lakin, Wambaugh, Leifer, Cannon, & Sivard, 1989; Uejio, 1990; Uejio, Carmody, & Ross, 1991). Semiformal tools for argumentation, deliberation, design space, and design process history provide an ontology of concept and relation types, which structure the information for graphical browsing (Brown & Bansal, 1991; Chen, Dietterich, & Ullman, 1991; Conklin & Yakemovic, 1991; Fischer, Lemke, McCall, & Morch, 1991; Lee & Lai, 1991; MacLean, Young, Bellotti, & Moran, 1991; Ramesh & Dhar, 1992).

A major obstacle to the practical use of these tools is the burden of structuring the information to rationalize design choices. For example, a design support tool called IDE (Russell et al., 1990) elicited design rationale at design time together with the specification of the designed artifact (instructional courses). Experience of the second author with IDE users over several years revealed that even highly motivated and disciplined designers suffer from "rationale fatigue." Users who started a design by including rationale notes to justify design choices soon began to bypass the rationale capture interface to focus on their primary design task.

Our examination of the design protocols suggested that while some of the design rationale dialog flowed naturally along the lines of argumentation or deliberation, much of the information available to the original designers and needed by others was not structured in this manner. This is an example of a mismatch between the knowledge as available and the knowledge as intended to be used in a rationale.

However, semiformal design documentation can be used for more than information recording. The effort of categorizing and relating notes on the issues, arguments, etc., can pay dividends if the capture tool can provide project management services, such as checklists. If all design issues are labeled as such, and all alternatives are enumerated and categorized, then the support tool can do a simple kind of completeness analysis, looking for unresolved issues and questions. Indeed, the main purpose of tools such as QOC (MacLean et al., 1991) is to guide the designer through a systematic exploration of the design space, producing a better design document and presumable a better design.

3.2.2 Decision Support

Decisions are so ubiquitous in design activity that design is sometimes *defined* in terms of decision making. We found two basic uses for information explicitly related to decisions. First, decision points serve as loci for considering alternatives and linking dependent elements in the design. For example, a designer will ask what decisions in the previous design were related to a specific parameter. The answer is a set of clusters of information (alternatives, criteria). These clusters identify places in the search space where the new design might differ, and link parameters that should be re-examined if the design took a different path (Lee, 1990; Mark & Schlossberg, 1991). This use of decision representations is similar to semi-structured note-taking, and can provide similar information retrieval services.

A second use of decision information is design evaluation. In this context, decision making means choosing alternatives based on evaluation against criteria. Formal tools can help the designer be consistent with normative ideals (i.e., overcoming human bias in judgment). For example, some tools offer spreadsheet-like services for experimenting with tradeoffs among alternatives and criteria in design (Boose, Bradshaw, Koszarek, & Shema, 1992; Boose, Shema, & Bradshaw, 1991; Shema, Bradshaw, Covington, & Boose, 1990). Tools can also help with evaluation tasks such as calculating resource budgets and checking constraints. The most structured formalism for decision making is based on the mathematics of maximizing expected utility, given a network of conditional probabilities (see Howard, 1968, for an introduction). Such a network can determine *exactly how good* a decision would be given information about the probabilities and utilities of consequences of the decision. Of course, greater precision in the decision model comes at the expense of more work in developing the model; in many cases, approximate representations such as partial preference orderings are adequate for documenting important design decisions.

None of the information questions or answers in the protocols we surveyed would have required or used this degree of precision in decision support. However, even seemingly small design decisions can have amplified monetary consequences in products designed by large teams. In such situations, the costs of representing design decisions with the rigor and completeness required of normative formalisms might be outweighed by the benefits.

3.2.3 Dependency Management

A primary reason why designers are interested in the rationale for an existing design is to change it. They want to reuse the good parts of the existing design and modify the bad parts such that the changes don't undo the careful tradeoffs and choices that make up the design. We have already pointed out the need to capture dependency relationships. The rich webs of interconnected dependencies in a realistic design are a source of complexity. Managing this complexity is a problem for designers and an opportunity for software support.

The ubiquitous question "what is affected by a change in this design element?" can only be answered by a program to the extent that the program can *represent* the dependencies and invoke the *reasoning* services that correspond to dependency paths.

Some dependencies are operational, that is, formulated in a representation in which they can be automatically calculated, propagated, or checked. In the simplest case, semiformal links can be followed when the contents of nodes change (Ramesh & Dhar, 1992). More information can be represented by dependencies that are expressed as *constraints*. For instance, in the damper design, the force produced by the solenoid can be computed by an operational constraint — a closed-form function of current and stroke length. Another kind of constraint is a limit on possible behavior, such as the maximum stroke length produced by the solenoid. Given constraints in this form, programs can help verify the intended function and behavior of components by following the paths of constraints from components to requirement specifications (Baudin, Sivard, & Zweben, 1990; Bowen & Bahler, 1993). Another class of dependencies is called *commitments*. Commitments are constraints that must be satisfied for a particular software module to be incorporated into a larger design (Mark, Tyler, McGuire, & Schlossberg, 1992). Capturing commitments in operational form enables tools that help developers determine what needs to change in a design when a requirement or assumption changes. Another kind of dependency relation is the functional role of a component in a larger system. Explicit descriptions of functional roles can support the retrieval and modification of cases from a design case library (Goel & Chandrasekaran, 1989) and the description of design intent (Chandrasekaran, Goel, & Iwasaki, 1993).

When dependencies are not operational, tools can help elicit them. The deliberation and decision support tools are designed to help the user enumerate and record the dependencies between alternatives and justifying information. The ontologies of argumentation (Kunz & Rittel, 1970; McCall, 1986; Newman & Marshall, 1990) design space possibilities (MacLean et al., 1991), and decision making (Lee & Lai, 1991) are explicit representations of dependencies that are often implicit. When a designer wants to reconsider a decision or design option, he or she can browse the links in the dependency network. An important future direction for such tools is to augment them with more operational dependency management techniques, such as rule-based constraint checkers (Fischer, Lemke, Mastaglio, & Morch, 1990; Fischer et al., 1991), truth maintenance (Lubars, 1991; Petrie, 1990), and mathematical modeling tools (Wolfram, 1991).

3.2.4 Rationale by Demonstration

One of the primary uses of design rationale information is to communicate the intended purpose or expected behavior of the design. Although it is valuable to capture information about expected behavior and contexts of use, it is difficult to specify the *dynamic* and *situational* aspects of a design in static artifact specifications. CAD drawings, program source code, and text-graphic documents are not well suited to the task of communicating time-varying (dynamic) behaviors. Furthermore, specifying the situation in which the artifact interacts with the environment (including the human context of use), requires envisioning in a hypothetical world with detail that is difficult to anticipate. A passive medium offers no assistance with this cognitive task.

We are exploring the idea of using interactive simulation as a communication medium for conveying the intended function of an artifact in an expected operating environment (Gruber, 1990; Gruber, 1991). The technique is called *rationale by demonstration* because one demonstrates the phenomenon of interest, using packaged simulation scenarios. An engineer sets up a simulation *scenario* by specifying structures and behaviors of interest, and the expected operating environment in terms of initial conditions and exogenous variables. The system performs a simulation and generates an explanation of the scenario. The process is illustrated in Figure 1. An intelligent model formulation system actively elicits the engineer's specification, checking that parameters are not under or over constrained and ensuring that component connection topologies meet structural constraints. The model formulation system then generates an operational simulation model, drawing from a library of standard model fragments. The simulation model is given to a simulation system that predicts behavior and generates explanations for human consumption (tracing causal pathways, filtering extraneous detail, etc.). The explanations, possibly including animations of the artifact, serve as demonstrations of the designer's intent. The models and initial conditions can also be saved, so that the demonstration can be regenerated later with slightly different parameter settings to explore alternative scenarios.

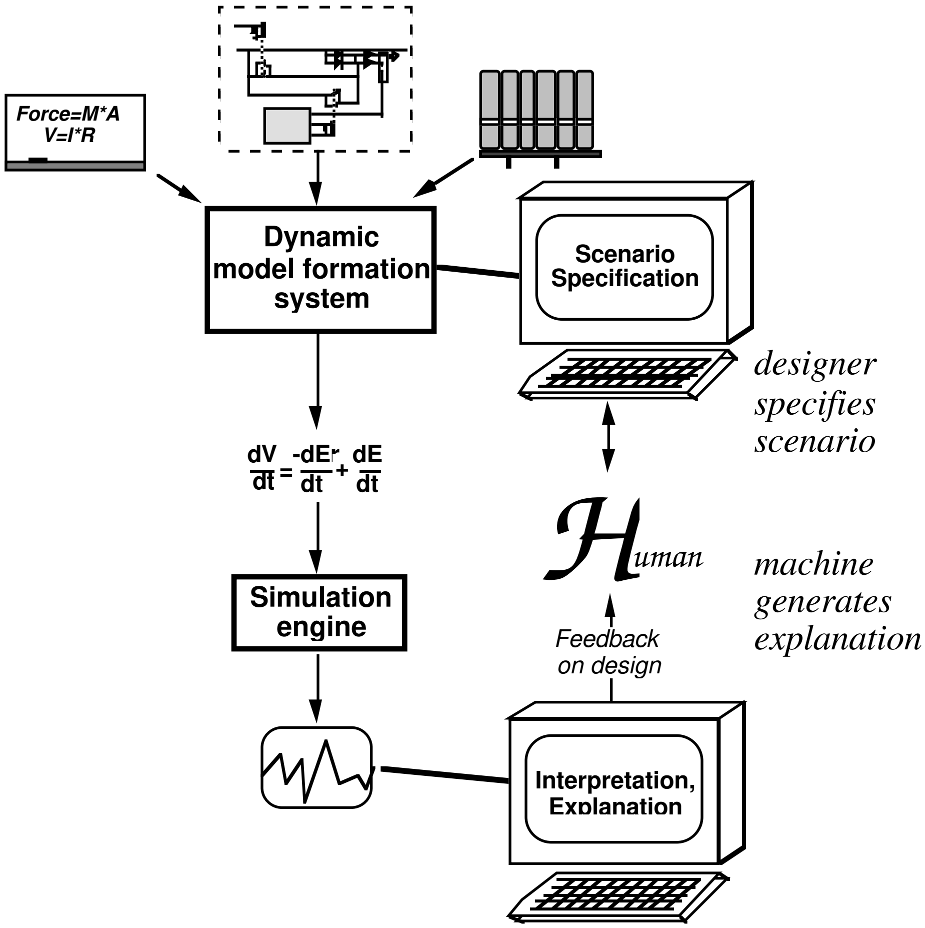

Figure 1: Capturing demonstrations of design intent. The human designer specifies a simulation scenario, which is used to formulate simulation model, which is used to predict behavior. The output of the simulation is presented in text and graphic explanations for human consumption. The designer can use the entire mechanism as a medium for capturing demonstrations of expected behavior and assumed operating conditions. The architecture shown is implemented in the DME system (Iwasaki & Low, 1991).

The basic technology required for rationale by explanation is model formulation assistance (Falkenhainer & Forbus, 1991; Iwasaki & Low, 1991; Nayak, 1992; Palmer & Cramer, 1991) and explainable simulation (Falkenhainer & Forbus, 1992; Gruber & Gautier, 1992). It only works in domains where formal simulation models exist, and is only practical if the model formulation process can be made efficient. (Research on both fronts is progressing rapidly.)

In any case, the potential payoffs to the user are significant. Not only is design rationale information captured as a by-product of normal engineering, but the explanation is guaranteed to reflect the assumptions and input data specified *unambiguously* by the designer. Furthermore, the act of specifying a scenario—particularly in the interactive, qualitative modeling systems—*prompts* the designer to explicate assumptions and enumerate relevant conditions.

Rationale by demonstration shares the intuition of the scenario-driven design, that one can help communicate the intent of a design by describing the content of use and expected behavior (Rosson & Carroll, 1992). However, since rationale by demonstration is based on predictive behavior models and simulation software, it is possible to actively prompt for the information needed to specify a scenario and to verify that the specification is sufficient to model the scenario. The active prompting of input needed for a model coupled with feedback about predicted behavior is similar to what is done by decision support tools (Boose, Shema, & Bradshaw, 1991; Shema et al., 1990), which help the user specify the relevant factors that go into a decision and provide feedback on the consequences of alternative decision outcomes.

4. Implications for the Design of Support Software

Our analysis of the information available for design rationale and the possible computational services that can operate on that information can inform the design of rationale support software. Our major conclusion is that the record and replay paradigm is inadequate. In this final section, we argue for this conclusion, offer an alternative paradigm, and discuss practical implications for the development of tools.

4.1 "Record and Replay" is incomplete.

Many proposals for design rationale support are based on the strategy of recording a complete rationale explanation (argument, design space exploration, decision justification) so that it can be easily replayed (retrieved, presented, browsed) by consumers of the information. We call this the *record and replay* paradigm. The research within this paradigm emphasizes the interesting problem of eliciting knowledge that is often tacit, difficult to formalize, and generally incomplete. However, our analysis suggests that for many types of design rationale information and several potential applications, the capture of complete explanations is not an adequate solution.

A record and replay approach is incomplete with respect to the information needs of designers for several reasons. First, rationale explanations cover a broad range of information requests (see Section 2 and Appendix 1). If design rationales are to be captured and played back, then the design rationale author has to anticipate the space of questions that might be asked by the reader, and formulate possible rationale explanations in advance. A semiformal capture tool can prompt the designer to construct explanations along some dimension, such as deliberation, decision making, or design space exploration. However, the particular nodes and links used in explanations captured at design time will answer only a fraction of the questions asked by designers about existing designs.

Of course, the main objective of semiformal rationale approaches such as QOC (MacLean et al., 1991) is to help the author systematically generate and explore the questions that might of interest to the downstream reader. We are not asserting that the resulting rationales are not useful (see Yakemovic & Conklin, 1990, for evidence on this question). We are questioning whether it is worth asking the designer to construct *complete* explanations at design time, given the cognitive burden of this task.

Second, many uses of rationale information, such as tracking dependencies (Section 3.2.3) and demonstrating design intent (Section 3.2.4), require *inference* from available information to answer questions. In the damper protocol, the original designer explained the reasoning that links the heat transfer problem to the availability of solenoids with desired properties. The original designer explored a space of several competing factors, including functional alternatives to solenoids. The path from overall functionality to solenoid properties is just one of many possible inferences. The space of inferences (and corresponding explanations) that might be made from available design data is much larger than the space of the design data themselves.

Third, the inferences underlying rationale explanations are based on assumptions and design state that can change after a rationale is constructed. This would invalidate "snapshot" explanations. To allow for change, the person recording the rationales would have to anticipate all the assumptions that might change, and explicitly instantiate them for each case (e.g., identify the availability of high temperature solenoids as a supporting argument for some position). If the rationale only included the history of decisions as they actually occurred, the resulting record could end up reflecting an opportunistic hopping among constraints and alternatives (as has been observed in software design (Guindon, 1990)), rather than the structure of dependencies between decisions and the data upon which they are based.

4.2 Generating rationale explanations from what is captured

We conclude that design rationale tools must go beyond the record and replay paradigm if they are to support the kinds of computational services we have described. In particular, we argue for a paradigm in which design rationale explanations are *generated*, in response to information requests, from relevant stored information. The knowledge capture task, then, is to elicit the information that is potentially relevant to explaining a rationale, rather than the explanations themselves. The generation task is to construct an explanation that answers a given query, possibly by way of automated inference or other computation from the captured information and other background knowledge.

4.2.1 A Conceptual Framework for Generative Rationale

The generative view of design rationale can be explained in the terms of a conceptual framework illustrated in Figure 2 (Gruber & Russell, 1990). In this framework, the basic element of information is a design description, which we assume can be captured and represented. Common classes of design descriptions are *structure* (the physical and/or logical composition of an artifact, typically in terms of the composition of parts, and connection topologies), *behavior* (something an artifact might *do*, in terms of observable states or changes), *function* (effect or goal to achieve by artifact behavior), *requirements* (prescriptions concerning the structure, behavior, and/or function that the designed artifact must satisfy, typically specified as constraints), and *objectives* (specifications of desired properties of the artifact other than pure functions, such as cost and reliability, often given as decision criteria). Some of the information used by the designers in the protocol survey are simple facts of one of these types. Answers to many rationale questions involved explanations (see Section 2). Explanations follow paths of inference among design descriptions, shown as directed arcs in the figure. An explanation of why a component was chosen, for instance, might show how the component structure (*S)* implements a behavior (*B*), achieving a function (*F*), which satisfies a requirement (*R*). The generative view of rationale says that such explanations are generated in response to questions about the relationships among design descriptions. The inferences may be performed by engineering tools and/or human engineers.

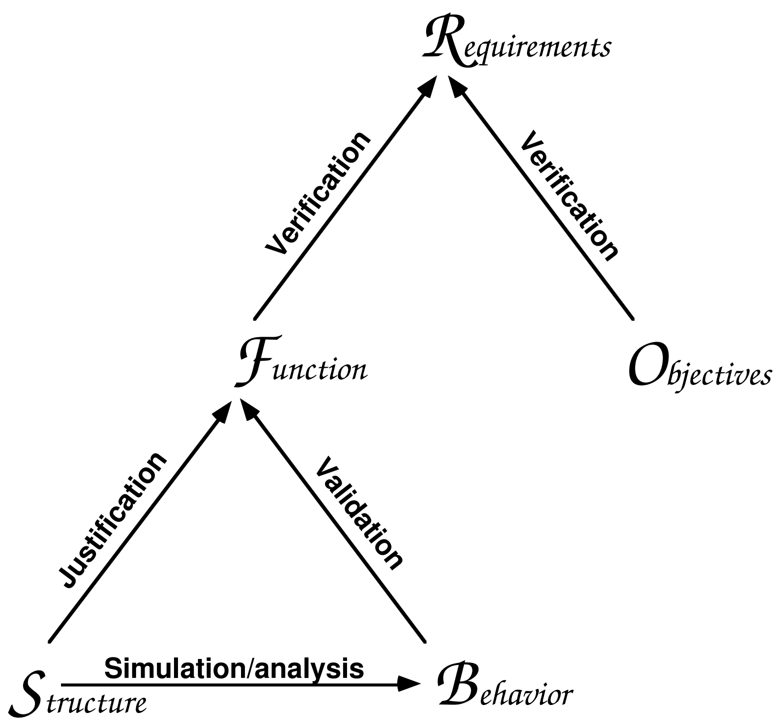

Figure 2: Relationships among design descriptions. Nodes labeled *S, B, F, R,* and *O* are classes of design descriptions. Arcs correspond to *inferences* between descriptions. For example, $S \rightarrow B$ inferences are performed by simulation and analysis tools. *Explanations* follow paths of inference. An explanation of why a component was chosen, for instance, might show how the component structure (*S*) implements a behavior (*B*), achieving a function (*F*), which satisfies a requirement (*R*).

A record and replay approach would capture completed explanation graphs (perhaps structured as decisions among alternatives). In contrast, the generative approach captures the nodes involved in a decision (the "relevant facts") and the inference processes that link them (if they are available on-line). The explanations are generated when a consumer of the rationale information makes a query, such as "what functions depend on this structure?" or "how is this requirement met?"

4.2.2 Applications of Generative Design Rationale

The generative approach can be applied to several of the computational uses of design rationale described in Section 3. The cases of dependency management and rationale by demonstration illustrate the idea.

For dependency management, in the record and replay paradigm one would explicitly capture the dependencies between each pair of related design elements. Perhaps the dependency would be couched in the terminology of evidential support for a position. In the generative approach, one would capture only the local dependencies. At explanation time a process would traverse the dependency network, uncovering unanticipated relationships among design elements. If the value of one of the nodes in a dependency network changes, a new explanation can be generated.

The technique of rationale by demonstration is an inherently generative approach. The analog in the record and replay paradigm is the hand-crafted mock-up or story-board animation. This is a known technique for flushing out assumptions and documenting intent, especially in domains such as user interface design. However, the medium is essentially static and passive. In the generative approach, one captures an operational specification of a scenario, which can be run. The specification is given as the inputs to the model formulation system: the basic device model, input conditions, behavior of interest, and operating conditions. The explanation of how the device operates in the specified scenario is *generated* by the simulation program. This can be used to produce an active, interactive document. Textual and graphical explanations are generated in response to user queries. Users can ask questions such as “what caused this behavior to occur?” and “what else might have happened at this point?”

When the knowledge consumer views the demonstration, he or she has access to all the detail specified by the original designer. Note that the “author” had to specify an *operational* behavior model and an *unambiguous* set of initial conditions to run the simulation. This information is more useful than a static, textual summary of expected behavior, for two reasons. First, since the explanation is generated from the specification, it is guaranteed to be consistent with the engineering models. Second, when the models or assumptions change, the explanations are automatically kept up to date and accurate. Of course, the cost to the user of specifying operational models is higher than jotting down notes. However, if model formulation can be made easier and useful in earlier stages of the design process, the capture of operational demonstrations of design intent can be a by product of the normal use of simulation and analysis tools.

Another example of generative rationale support is Garcia’s ADD system (Garcia, 1992; Garcia & Howard, 1992). ADD supports the documentation of parametric designs. During design, designers make choices for parameters in light of requirements and constraints. ADD monitors the decision making process, as an apprentice. The program has a strong a priori model of the domain (design parameters for heating and air-conditioning systems), design process (the ordering dependencies among design decisions), and decision making (how to evaluate alternatives considering multiple criteria and constraints). The designer enters requirements, constraints, and decision criteria for a particular case. Then, as the designer makes design decisions, the system uses the domain, process, and decision models to predict expected choices for parameters. If the designer’s choices do not match the system’s expectations, ADD asks for more information from the user to help justify the choices. If they do match, ADD can justify the choices. As a result, all decisions in a design can be explained by ADD.

ADD is an excellent example of the generative approach. It acquires rationale information from the designer that is used to generate explanations to the user of the document. It produces focused explanations in response to user queries, rather than asking the designer to anticipate the reader's needs. The explanations are constructed from models and acquired information (parameters, requirements, constraints, criteria). As with simulation-based explanations, the ADD’s explanations are guaranteed to be consistent and up to date with the design choices.

In sum, the generative approach enables more powerful computational services yet requires the capture of more operational information. This is practical if the capture of dependency relations and scenario specifications results from the routine use of engineering support tools. This brings us to a final issue: the impact of this perspective on the design of software.

4.3 Architectural Implication: Integration with Existing Engineering Tools

We believe that much of the information to be captured for design rationale is either available now from existing engineering tools and information sources, or will be. Knowledge-based CAD, advanced programming environments, on-line engineering handbooks, simulation and modeling programs, design checking tools, and electronic mail are all to be found in the designer's toolbox. If more and more of design will be done in the context of such tools, then there will be information freely available in machine-readable form. However, "freely available" does not imply "ready to use."

To bring these tools into the service of design rationale support requires *integration*. Consider again the damper example. The information on properties of available solenoids might come from a database. The equations relating voltage, current, and force might be in a model library. The decision to use a short stroke length might be found in a design history, and hyper-linked to a diagram in the requirements document. To construct complete explanations in response to user questions, a design rationale tool needs to reason about the *dependencies* among these heterogeneous information sources. This requires a common conceptualization of the shared data and a mechanism for representing relations that cross tool boundaries. In addition, since some of the relationships among design elements are *computed* by engineering tools, reconstructing the explanation when assumptions change requires re-invoking the tools in a coordinated fashion.

Fischer and his colleagues have shown the value of integrating critics (design checker tools) with artifact specification (e.g., domain-specific CAD drawing) and hypertext documentation (Fischer et al., 1991). This work showed that the explanations given by critics could serve the same kinds of information roles as the hypertext documentation (i.e., giving reasons for rejecting alternatives). Furthermore, the explanations seem better grounded and more informative when presented in the same context as the textual annotations and CAD drawings.

Critics that generate their explanations from engineering models, such as the industrial design checkers used to validate the manufacturability and testability of electronic circuits, could be incorporated into the documentation of designs in a similar manner. One can view the output of the generative tools as part of the documentation. Since the tools can be rerun with different input conditions (alternative designs), the documentation can answer hypothetical questions not covered by the original document writer and survive change in the life of the design.

However, achieving this vision of the generative documentation of rationale requires greater integration of tools than exists today. The tightest integration in existing engineering tools occurs in *CAD frameworks* — families of programs designed together to share data in standard formats and common domain models. However, we believe that large scale rationale capture and generation will require the integration of design rationale capture tools with sets of *independently* developed, *heterogeneous* engineering tools of the designer's choosing (across CAD frameworks if desired). One should be able to specify dependencies between a section of text in a requirements document, a drawing in a CAD database, textual annotations to CAD objects, models of the components represented by those objects, constraint equations and simulation models specifying the physical behavior of those objects, and plans generated by a process planner for manufacturing the components. An architecture called SHADE is under development that will provide this capability (Gruber, Tenenbaum, & Weber, 1992).

Acknowledgments

We would like to thank the participants in the Stanford design rationale seminar in the Fall of 1991 and the members of the Palo Alto Collaboration Testbed consortium (Cutkosky et al., 1993) for many discussions that contributed to this work. We are grateful to the researchers who generously allowed us to analyze their hard-earned protocol data, much of which was previously unreleased. They include Catherine Baudin, Vinod Baya, Jean-Charles Bonnet, Jody Gevins, Larry Leifer, and Ade Mabogunje at NASA Ames and the Stanford Center for Design Research; Ana Cristina Bicharra Garcia from the Stanford Civil Engineering Department, Thomas Kuffner and David Ullman at Oregon State University (NSF grant DMC.87.12091); Pete Pirolli and Vinod Goel at the U. C. Berkeley School of Education; and Victoria Bellotti and Alan MacLean at EuroPARC. Conversations with Tom Dietterich, Bill Mark, Richard Pelavin, Mark Stefik, Marty Tenenbaum, and Jay Weber were particularly enlightening. Reviews by Tom Moran and Jack Carroll were extremely helpful. Work by the first author is supported by NASA Grant NCC2-537, NASA Grant NAG 2-581 (under ARPA Order 6822), and DARPA prime contract DAAA15-91-C-0104 through Lockheed subcontract SQ70A3030R, monitored by the U. S. Army BRL.

Appendix 1: Categories of Information Requested about Designs

Following are a set of generic questions one might ask about a design, derived from an analysis of designers talking about designs in the protocol studies surveyed (Gruber & Russell, 1992). Each generic question is derived from segments of design protocols in which a subject requested the information directly, or made a conjecture or statement about the answer to the question in a think-aloud setting. Given the nature of the protocol data, the relative frequency of each question type is not reported (see Kuffner & Ullman, 1990, for such data). Instead, the generic questions characterize the broad range of knowledge used to create a design rationale statement.

1 Requirements

What are the given requirements?
Is this constraint a requirement?
Give more detail about this parameter of the operating environment.
Can I assume this fact about the operating environment?
What are the requirement constraints on this parameter?
Is this parameter constrained by external requirements?
What is the expected behavior of this artifact in the scenario of use?
Should I assume that this functionality is required?
Can I modify this requirement?

2 Structure/form

What are the components?
What class of device or mechanism is this part?
What is the geometry of this part? (qualitative)
What material is this part made of?
How do these components interface?
What are the locations of parts, connections, etc. (for constraint checking)?
What are the known limitations (strengths) of this part/material class?
What affects the choice of artifact components?

3 Behavior/operation: what does it do

What is the behavior of this parameter in the operating conditions
What is the behavioral interaction between these subsystems?
What is the range of motion of this part?
What is the cause of this behavior?
What are the expected failure modes in the scenario of use?

4 Functions
What is the function of this part in the design?
What is the function of a feature of a part in the design?

5 Hypotheticals
What happens if this parameter changes to this new value?
What is the effect of this hypothetical behavior on this parameter?
Adapt equation to this changed parameter and recompute
What will have to change in the design if this parameter changes to this new value?

6 Dependencies
What are the known dependencies among the parts?
What are the constraints on this parameter?
Is this parameter critical (involved in a dominant constraint)?
How is this subassembly related to this parameter?
What is the source of this constraint?

7 Constraint checking
Is this constraint satisfied?
Does this structure have this behavior that violates this constraint?
What are the known problems with this design?
Would a part with this functionality satisfy this constraint?

8 Decisions
What are the alternative choices for this design parameter?
What decisions were made related to this parameter?
What was an earlier version of the design?
What decisions were made related to satisfying this constraint?
Which parameter, requirement, constraint, or component should be decided first?
What design choices are freed by a change in this input parameter?
What alternative parts that satisfy this constraint could substitute for this part?
Where did the idea for this design choice come from?

9 Justifications and evaluations of alternatives
Why this design parameter value?
Why is design parameter at value V1 instead of normal value V2?
Why was this alternative chosen over that alternative?
What is person P's evaluation of these alternatives?
Why not try this alternative?

10 Justifications and explanations of functions
Why is this function provided?
Why is this function not provided?
Why can't the current design achieve this new value of this functional requirement parameter?

11 Validation explanations
How is this requirement satisfied?
How is this function achieved?
How is this functional requirement achieved?
How will this part be maintained?

12 Computations on existing model
Compute a parameter value given other parameters
What are the trajectories of parameters

13 Definitions
What does a term in the documentation mean?

14 Other design moves
Search for information expected to be in documentation, e.g., equation or diagram.
Change this requirement constraint and update design.
Have all the arguments for/against this alternative been checked?

References

Baudin, C., Gevins, J., Baya, V., & Mabogunje, A. (1992). Dedal: Using Domain Concepts to Index Engineering Design Information. *14th Annual Conference of the Cognitive Science Society*, Cognitive Science Society.

Baudin, C., Gevins, J., Baya, V., Mabogunje, A., & Bonnet, J.-C. (1992). *The variable damper experiment* (Technical Report FIA-TR-92-08). NASA Ames Research Center, Moffett Field, CA.

Baudin, C., Sivard, C., & Zweben, M. (1990). Recovering rationale for design changes: A knowledge-based approach. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics,* Los Angelos, CA, pages 745-749, IEEE.

Baya, V., Gevins, J., Baudin, C., Mabogunje, A., Toye, G., & Leifer, L. (1992). An experimental study of design information reuse. *4th International Conference on Design Theory and Methodology.*

Bellotti, V., & MacLean, A. (1989). *Transcription of: Two designers discussing a proposed design for a Fast Automated Teller Machine* (Internal Technical Report AMODEUS RP6/DOC1). Rank Xerox EuroPARC.

Boose, J. H., Bradshaw, J., Koszarek, J. L., & Shema, D. B. (1992). Better group decisions: Using knowledge acquisition techniques to build richer decision models. *Hawaii International Conference on System Sciences*.

Boose, J. H., Shema, D. B., & Bradshaw, J. M. (1991). Knowledge-based design rationale capture: Automating engineering trade studies. In M. Green (Eds.), *Knowledge Aided Design*. London: Academic Press. in press.

Bowen, J., & Bahler, D. (1993). Constraint-based software for concurrent engineering. *IEEE Computer*, *26*(1), 66-68.

Brown, D. C., & Bansal, R. (1991). Using Design History Systems for Technology Transfer. *Proceedings of Computer-Aided Product Development, MIT-JSME Workshop,* MIT, Cambridge, MA, 1989, Computer Science Department, Worcester Polytechnic Institute.

Carroll, J. M., & Moran, T. P. (1991). Special Issue on Design Rationale. *Human-Computer Interaction*, *6*(3-4), .

Chandrasekaran, B., Goel, A. K., & Iwasaki, Y. (1993). Functional representation as design rationale. *IEEE Computer*, *26*(1), 48-56.

Chen, A., Dietterich, T. G., & Ullman, D. G. (1991). A computer-based design history tool. *NSF Design and Manufacturing Conference,* Austin, Texas, pages 985-994.

Conklin, J., & Yakemovic, K. B. (1991). A Process-oriented Paradigm for Design Rationale. *Human Computer Interaction*, *6* (3/4), 357-392.

Cutkosky, M., Engelmore, R. S., Fikes, R. E., Gruber, T. R., Genesereth, M. R., Mark, W. S., Tenenbaum, J. M., & Weber, J. C. (1993). PACT: An experiment in integrating concurrent engineering systems. *IEEE Computer*, *26*(1), 28-37.

Falkenhainer, B., & Forbus, K. (1992). Self-explanatory simulations : Scaling up to large models. *Proceedings of the Tenth National Conference on Artificial Intelligence,* San Jose, CA, pages 685-690, AAAI press / MIT press.

Falkenhainer, B., & Forbus, K. D. (1991). Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, *51*, 95-143.

Fischer, G., Lemke, A. C., Mastaglio, T., & Morch, A. I. (1990). Using Critics to Empower Users. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'90)*, pages 337-347.

Fischer, G., Lemke, A. C., McCall, R., & Morch, A. I. (1991). Making Argumentation Serve Design. *Human-Computer Interaction*, *6*(3/4), 393-420.

Garcia, A. C. B. (1991). Selections from Request for Rationale materials in HVAC design. Stanford University, Department of Civil Engineering. Personal Communication, November, 1991.

Garcia, A. C. B. (1992). *Active Design Documentation.* Doctoral Dissertation, Stanford University. Available as a technical report from the Department of Civil Engineering.

Garcia, A. C. B., & Howard, H. C. (1992). Acquiring design knowledge through design decision justification. *Artificial Intelligence ior Engineering, Design, and Analysis in Manufacturing*, *6*(1), 59-71.

Goel, A., & Chandrasekaran, B. (1989). Functional representation of designs and redesign problem solving. *IJCAI-89,* Detroit, pages 1388-1394.

Gruber, T. R. (1990). *Model-based Explanation of Design Rationale* (Technical Report KSL-90-33). Knowledge Systems Laboratory, Stanford University. Also appears in *Proceedings of the AAAI-90 Workshop on Explanation*, Boston, MA, July 1990.

Gruber, T. R. (1991). Interactive acquisition of justifications: Learning "why" by being told "what". *IEEE Expert*, 6(4), 65-75, August.

Gruber, T. R., Boose, J., Baudin, C., & Weber, J. (1991). Design rationale capture as knowledge acquisition: Tradeoffs in the design of interactive tools. In L. Birnbaum & G. Collins (Eds.), *Machine Learning: Proceedings of the Eighth International Workshop,* Chicago, pages 3-12, Morgan Kaufmann.

Gruber, T. R., & Gautier, P. O. (1992). *Machine-generated Explanations of Engineering Models: A compositional modeling approach* (Technical Report KSL 92-88). Knowledge Systems Laboratory, Stanford University. Submitted to IJCAI'93.

Gruber, T. R., & Russell, D. M. (1990). *Design Knowledge and Design Rationale: A Framework for Representation, Capture, and Use* (Technical Report KSL 90-45). Knowledge Systems Laboratory, Stanford University.

Gruber, T. R., & Russell, D. M. (1992). *Derivation and Use of Design Rationale Information as Expressed by Designers* (Technical Report KSL-92-64). Knowledge Systems Laboratory, Stanford University.

Gruber, T. R., Tenenbaum, J. M., & Weber, J. C. (1992). Toward a knowledge medium for collaborative product development. In J. S. Gero (Eds.), *Artificial Intelligence in Design '92*. Boston: Kluwer Academic Publishers.

Guindon, R. (1990). Knowledge exploited by experts during software system design. *International Journal of Man-Machine Studies*, *33*, 279-304.

Howard, R., & Matheson, J. E. (Eds.). (1984). *Readings on the Principles and Applications of Decision Analysis*. Menlo Park, CA: Strategic Decisions Group.

Hwang, T. S., & Ullman, D. G. (1990). The design capture system: Capturing back-of-the-envelope sketches. *International Conference on Engineering Design (ICED 90),* Dubrovnik, Yugoslavia.

Iwasaki, Y., & Low, C. M. (1991). *Model Generation and Simulation of Device Behavior with Continuous and Discrete Changes* (Technical Report KSL-91-69). Stanford University, Knowledge Systems Laboratory. To appear in *Intelligent Systems Engineering*, 1993.

Kuffner, T. A. (1990). *Mechanical Design History Content: The Information Requests of Design Engineers.* Master's thesis, Oregon State University.

Kuffner, T. A., & Ullman, D. G. (1990). The information requests of mechanical design engineers. *Design Studies*, *12*(1), 42-50.

Kunz, W., & Rittel, H. (1970). *Issues as Elements of Information Systems* Center for Planning and Development Research, University of California at Berkeley.

Lakin, F., Wambaugh, H., Leifer, L., Cannon, D., & Sivard, C. (1989). The electronic design notebook: Performing medium and processing medium. *Visual Computer: International Journal of Computer Graphics*, *5*(4), 214-226.

Lee, J. (1990). SIBYL: A tool for managing group decision rationale. *Proceedings of the conference on Computer Supported Cooperative Work (CSCW-90),* Los Angelos, pages 79-92.

Lee, J., & Lai, K.-Y. (1991). *A Comparative Analysis of Design Rationale Representations* (Technical Report 121). MIT Center for Coordination Science.

Leifer, L., Baudin, C., Gevins, J., Toye, G., Baya, V., & Mabogunje, A. (1992). A Tool and Protocol for Developing Reusable Design Knowledge. *First AIAA Aerospace Design Conference.*

Lubars, M. D. (1991). Representing design dependencies in an issue-based style. *IEEE Software*, 8(4), July, 81-89.

MacLean, A., Young, R., Bellotti, V., & Moran, T. (1991). Questions, Options, and Criteria: Elements of A Design Rationale for User Interfaces. *Human Computer Interaction*, *6*(3/4), 201-250.

Mark, W., & Schlossberg, J. (1991). Interactive acquisition of design decisions. *Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop,* Banff, Canada.

Mark, W., Tyler, S., McGuire, J., & Schlossberg, J. (1992). Commitment-based software development. *IEEE Transactions on Software Engineering*, , 18(10), 870-885 .

McCall, R. (1986). Issue-serve systems: A descriptive theory for design. *Design Methods and Theories*, *20*(8), 443-458.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, *2*(1), 46-80.

Mostow, J. (1989). Design by derivational analogy: Issues in the automated replay of design plans. *Artificial Intelligence*, *40*, 119-184.

Nayak, P. P. (1992). *Automated modeling of physical systems.* Ph.D. Thesis, Computer Science Department, Stanford University. Technical report STAN-CS-92-1443.

Newman, S. E., & Marshall, C. C. (1990). *Pushing Toulman too far: Learning from an argumentation representation scheme* Technical Report Xerox PARC.

Palmer, R. S., & Cramer, J. F. (1991). *SIMLAB: Automatically creating physical systems simulators* (Technical Report TR 91-1246). Department of Computer Science, Cornell University.

Petrie, C. (1990). *REDUX: An Overview* (Technical Report TR ACT-RA-314-90). Microelectronics and Computer Technology Corporation.

Pirolli, P., & Goel, V. (1989). Three Transcripts of design protocols: (1) Architecture, (2) Mechanical Engineering, (3) Instructional Design. UC Berkeley, School of Education. Personal correspondence, December, 1989.

Ramesh, B., & Dhar, V. (1992). Supporting systems development using knowledge captured during requirements engineering. *IEEE Transactions on Software Engineering*, (June), 18(6), 498-510.

Rosson, M. B., & Carroll, J. M. (1992). *Extending the Task-Artifact Framework: Scenario-based Design of Smalltalk Applications* (Research Report RC 17852 (#78516)). IBM Research Division.

Russell, D. M., Burton, R. R., Jordan, D. S., Jensen, A. M., Rogers, R. A., & Cohen, J. (1990). Creating instruction with IDE: Tools for instructional designers. *Intelligent Tutoring*, *1*(1), 3-16.

Shema, D., Bradshaw, J., Covington, S., & Boose, J. (1990). Design knowledge capture and alternative generation using possibility tables in CANARD. *Knowledge Acquisition*, *2*(4), 345-364.

Uejio, W. H. (1990). Electronic Design Notebook for the DARPA Initiative in Concurrent Engineering. *Proceedings of the Second Annual Concurrent Engineering Conference,* Morgantown, WV, pages 349-362.

Uejio, W. H., Carmody, S., & Ross, B. (1991). An electronic project notebook from the electronic design notebook (EDN). *Proceedings of the Third National Symposium on Concurrent Engineering,* Washington, D.C., pages 527-535.

Wolfram, S. (1991). *Mathematica: A System for doing Mathematics by Computer.* Menlo Park, CA: Addison-Wesley.

Yakemovic, K. B., & Conklin, J. (1990). Observations on a Commercial Use of an Issue-Based Information System. *Proceedings of Computer Supported Cooperative Work,* Los Angeles, CA.