HUMAN TECHNOLOGY

An Interdisciplinary Journal on Humans in ICT Environments ISSN: 1795-6889

www.humantechnology.jyu.fi Volume 6 (1), May 2010, 11–37

CRITICAL CONVERSATIONS: FEEDBACK AS A STIMULUS TO CREATIVITY IN SOFTWARE DESIGN

Raymond McCall
Department of Planning and Design
University of Colorado, Denver, USA

Abstract: *Three decades of creating software to support design rationale showed the author how rationale processes can promote generation of novel ideas. Rationale can promote creative design by promoting critical conversations among designers and other project participants. Critical conversations intertwine ideation and evaluation, using feedback about consequences of decisions to challenge designers to devise new ideas. Such conversations take two forms. The first is discussion involving feedback from speculation about consequences of design decisions for implementation and use. The second is discussion involving feedback from actual experiences of implementation and use of the software being designed. The former is purely a process of reflective discourse, the latter a process of situated cognition involving both action and reflective discourse. Thus, the former is pure argumentation, the latter situated argumentation. Exploiting the full potential of critical conversations for creative design requires rethinking rationale methods and integrating them into software supporting implementation and use.*

Keywords: *creativity, software, design, rationale, feedback, situated cognition, action, reflection, planning, reflective practice, design reasoning, argumentative approach, wicked problems.*

INTRODUCTION

This article presents a picture of how feedback-driven rationale processes promote creativity in software design. This picture derives from my three decades of experience in creating software supporting the documentation and use of issue-based rationale for design, that is, the type of rationale pioneered by Horst Rittel (Kunz & Rittel, 1970). This picture is not meant to portray all the ways creativity takes place in design, but it does seek to describe crucial processes that have been largely omitted from other accounts of rationale and creativity, especially the former.

To discuss how rationale promotes creativity in software design, it is useful to define some basic terms. In this paper *software design creativity* refers to the generation of innovative, high-quality ideas for the design of software. The term *ideation* refers to the generation of ideas, especially novel ideas, for artifact design. The term *evaluation* refers to

© 2010 Raymond McCall and the Agora Center, University of Jyväskylä
DOI: http://dx.doi.org/10.17011/ht/urn.20105241905

determination of the value of such ideas. *Feedback* refers to any information about consequences of design decisions that a designer gets from external sources, such as persons or situations. These are narrow definitions, but they serve the purposes of this paper. Note that the definition of software design creativity involves both ideation and the evaluation.

The picture presented here is based on a number of notions that contrast with ideas advocated by others. First of all, it takes a process-oriented view of rationale, while many proposed rationale approaches either eschew process orientation—for example, the question, options, and criteria (QOC) approach (MacLean, Young, Belotti, & Moran, 1996)—or provide only a rationale schema with no indication of processes for eliciting and recording the schematized rationale—such as the decision representation language (DRL; Lee, 1991).

Second, the picture created here is prescriptive in that it not only seeks to record design processes but also to improve them. In particular, it seeks to increase the use of rationale processes that improve design creativity. Not all rationale approaches are prescriptive (Dutoit McCall, Mistrik, & Paech, 2006); some are purely descriptive and seek only to record rather than change what designers think and do, such as QOC (though they might unintentionally improve design).

Third, the picture presented here is based on the view that intertwining ideation and evaluation is a powerful method for promoting creativity. Yet there is much literature both on creativity and on rationale that treats ideation and evaluation as separate phases, that is, not intertwined. Of particular importance here is that Rittel (1966) saw no role for the intertwining of ideation and evaluation in design.

Finally, this paper takes the view that creativity is enhanced if design and its rationale are considered not merely as planning for future action—for example, implementation and use—but also as a type of situated cognition in which design is shaped by feedback resulting from action. Yet, Rittel, who pioneered the field of design rationale, viewed design strictly as planning, in the sense of thinking before acting (Rittel, 1966); he saw rationale as documentation of this preparatory thinking. Most existing approaches to rationale appear to share this view, since they provide no account of rationale being generated in response to actions taken.

The picture presented here of how rationale processes promote creativity in software design can be summarized as follows. Intertwining ideation and evaluation promotes creativity in software design because feedback about consequences of design decisions challenges designers to devise new ideas. This intertwining takes two basic forms. The first involves discussion among designers in which verbal evaluations of proposed ideas prompt them to devise new ideas. The second and more important involves situated cognition in which feedback resulting from actions, especially the actions of implementation and use, prompts designers to devise new ideas.

The commitment to using feedback-driven, critical conversations to promote creativity has crucial implications for rationale methods used in software projects. One implication concerns the type of processes that are modeled. Currently, none of the rationale methods that deal with design decision making explicitly models the ways in which evaluative feedback leads to the generation of new design ideas. When rationale methods cannot model these processes, they not only cannot promote them but may actually discourage them. A second implication concerns the sources of design rationale. Current approaches concentrate almost exclusively on rationale from design discussion (planning). This is sufficient to allow

rationale based on speculative reasoning and the experience of previous projects, but not sufficient to allow rationale based on feedback from actions.

The picture of software creativity as being promoted by feedback-driven critical conversations extends and generalizes Schön's (1983) portrayal of design as a conversation with the situation. It is argued here that Schön's notion of design as both reflection and action provides a better picture of the role of rationale in design than Rittel's. While Rittel saw design as purely argumentation, Schön's theory implies that design is what we might call *situated argumentation*, that is, argumentation informed by feedback from action. Yet Schön's theory by itself covers only a small subset of the situated argumentation that stimulates creativity in software creation. Extending his theory produces a more complete picture of how rationale processes promote creative design. Ironically, extending his theory involves adding ideas of collaborative and participatory design advocated by Rittel.

The following sections of this paper expand on the above-stated ideas. The next section explains the background and motivation for the ideas presented here. The section following that explains the prescriptive and process-oriented approach used here to analyze rationale and creativity. I then look at the relationship between ideation and evaluation in both rationale processes and creative processes. I also contrast views of design as planning for action versus as situated cognition. After that, I identify implications for rationale processes that support creativity in software design. Finally, I summarize the conclusions of this paper and look at ideas for future work.

HISTORICAL BACKGROUND

Rittel (Kunz & Rittel, 1970) pioneered the field of design rationale with his work on Issue-Based Information Systems (IBIS). As a student of Rittel's, I devised a new approach to IBIS called Procedural Hierarchy of Issues (PHI; McCall, 1979, 1986, 1991) and began a series of software projects aimed at using PHI to improve the quality of designed artifacts. These projects revealed previously unforeseen potentials and limitations of rationale in design. In particular, they showed how the generation of novel ideas for software can be supported by processes in which the consequences of design ideas are identified. This paper describes what these projects revealed about the connections between rationale and creativity.

The PHI-based projects created the following software:

- PROTOCOL (McCall, 1979), a text-only hypertext system that elicited rationale from users in PHI form
- MIKROPLIS (McCall, 1989; McCall; Lutes-Schaab, & Schuler, 1984), text-only hypertext supporting user-controlled authoring and navigation of PHI rationale
- JANUS (Fischer, Lemke, McCall, & Morch, 1996; McCall, Fischer, & Morch, 1990), a system for kitchen design using loosely coupled subsystems for 2D computer-aided design (CAD), knowledge-based critiquing, and hypermedia for delivery of PHI rationale
- PHIDIAS (McCall, Bennett et al., 1990; McCall, Bennett, & Johnson, 1994; McCall, Ostwald, Shipman, & Wallace 1990), a system for building design using a hypermedia system to implement 3D CAD and knowledge-based agents, as well as authoring and delivering PHI rationale with multimedia

- HyperSketch (McCall, Johnson, & Smith, 1997; McCall, Vlahos, & Zabel, 2001), a pen-based system for designing by creating a linked collection of hand-drawn sketches.

The later systems were designed using lessons learned from the earlier systems. These projects are stages in a larger project meant to find out (a) how rationale can help designers create better artifacts, and (b) what software support is needed for such use of rationale.

In addition to documenting rationale for design of physical artifacts, all of the above-listed systems except JANUS were also used to document rationale for their own design. The experiences of this documentation effort revealed that the ways in which new ideas emerged involved processes not described anywhere in the rationale literature. In particular, the creative rationale processes in our projects were not supported either by Rittel's (Kunz & Rittel, 1970) IBIS or my PHI method. Furthermore, our creative processes were incompatible with parts of Rittel's theory about design processes and problems. This article looks at these differences and their implications for rationale approaches and software supporting creative software design.

The above-listed projects changed my understanding of rationale processes and creativity. To understand how, I should begin by describing what that understanding was at the start. Simply put, it was based on Rittel's (1972) ideas about (a) the need for an argumentative approach to design, and (b) how IBIS was to help achieve that goal. Rittel's advocacy of an argumentative approach was based on his theory that design problems are "wicked problems" (Rittel & Webber, 1973). By this he meant that they are ill-defined and ill-behaved in a variety of ways that, for example, go far beyond the difficulties of "ill-structured problems" (Simon, 1973). Wicked problems systematically violate conditions required for use of rigorous scientific method to understand and solve them. Rittel (1972) therefore called for a collaborative and participatory approach that involved stakeholders in defining requirements and evaluating proposed designs. Instead of relying on the unexplained judgments of "experts," however, he called for a process in which the reasoning of designers was open to inspection and criticism by others. This implied the need for an argumentative approach, that is, an approach in which all of design was treated as argumentation about design decisions.

Rittel used the term *argument* with the meaning of explicit reasoning, and not with the colloquial English meaning of heated verbal disagreement, as in, "We had an argument about who was to blame" (Rittel, personal communication, 1977). In other words, he used the word *argument* with the meaning it has in his native German language as well as in philosophical discourse in English. Unfortunately, his intentions were often misunderstood by his American students. In the later years of his life, he told his colleague Jean-Pierre Protzen that, because of this, he wished he had called his approach *deliberative* rather than *argumentative* (Protzen, personal communication, 1992).

Further promoting misunderstanding was the fact that, despite Rittel's insistence that the term *argument* was not a reference to disagreement, he felt that controversy was an intrinsic part of design and that forceful debate was the most valuable type of design discussion. He devised IBIS not as a general means of handing all argumentation in design but rather as a way of handing disagreement through debate. IBIS centered on the discussion of issues, but Rittel (1980) defined IBIS' issues as controversial design questions. All other design questions he labeled "trivial issues," and excluded them from IBIS discourse.

These days, all issue-based approaches to design rationale, as well as similar approaches like QOC and DRL, have abandoned Rittel's exclusive focus on controversy and adversarial argumentation. Rittel's focus on controversy, however, is more than an interesting historical

footnote, because it apparently led him and others to neglect the collaborative, constructive argumentation described here as a driving force of design creativity.

To clarify discussion, it is useful to briefly describe IBIS and to explain how PHI differs from it. IBIS was intended both as a method for discussing issues and as a means for documenting the discussion. For each issue, participants in the design propose possible answers, called *positions*. Arguments for and against the positions are then given, along with arguments for and against other arguments. Finally, an issue is *resolved* by deciding which position to accept. Issues are linked to each other by various relationships to form a connected graph called an *issue map*. In Rittel's (1980; personal communication, 1975) version of IBIS, the inter-issue relationships included *logical-successor-of*, *temporal-successor-of*, *more-general-than*, *similar-to* and *replaces*.

IBIS provided no way of grouping issue-based discussions to represent higher levels of granularity in design processes. Thus, for example, the widely used description of design as being divided into larger-scale processes of analysis, synthesis, and evaluation (Lawson, 2005) could not be expressed in IBIS. This was no accident. Rittel (personal communication, 1975) was deeply suspicious of such higher levels of granularity. In particular, he argued that the belief in large-scale phases of design, such as analysis, synthesis, and evaluation, was the hallmark of the first-generation approach to design, which he judged a failure and sought to replace with a second-generation based on an argumentative approach (Rittel, 1972). He insisted that the only sensible level of description of design process was in terms of its *microstructure*—that is, the level of issue-based discourse (Rittel, personal communication, 1975).

Of course, it can be argued that analysis and synthesis might also be found at the microstructural level for the generation of positions on issues. And evaluation is certainly part of IBIS. Perhaps the generation of positions could be divided into processes of analysis and synthesis. Unfortunately, IBIS provided no account of any processes for devising positions. It may well be, therefore, that its picture of the microstructure of design is not complete.

PHI was meant to implement Rittel's argumentative approach more fully than IBIS by including noncontroversial issues and using a better structure for discussion. To accomplish the latter, PHI replaced the interissue relationships of IBIS with two types of *dependency relationships*: *serves* and *leads-to*. The former indicates that the resolution of one issue influences the resolution of another, while the latter indicates that the resolution of an issue influences the relevance of another. In PHI, a single *root issue* represents the project as a whole. Since all other issues are resolved in order to resolve the root issue, they serve the root issue directly or indirectly. PHI modeled design rationale as a quasi-hierarchy of issues connected by serves relationships, that is, a directed acyclic graph with some added cycles.

PHI showed the structure of discussion more completely than IBIS. In particular, its serve relationships provided a way of grouping issue discussions to represent higher levels of granularity of design process structure. These relationships also enabled representation of detailed processes by which positions on issues were devised—including processes of ideation—something not possible with IBIS. While PHI did not use terms such as *synthesis*, *analysis*, and *evaluation* to label its process structures, it did enable the representation of such processes at many different levels of granularity in issue-based discussion.

Because the quasi-hierarchical structure of PHI is far more orderly than the "spaghetti" structure of IBIS (Fischer et al., 1996), it enabled a substantial increase in the number of issues dealt with in a project. Rittel suggested that, for practical reasons, IBIS should deal

with no more than 35 issues (Rittel, personal communication, 1975). But most of the dozens of PHI projects undertaken since 1976 involved more than 250 issues.

The initial goal of the series of software projects described above was to extend the use of PHI to all aspects of design, thus demonstrating Rittel's point that the entire design process was nothing but argumentation. A virtue of attempting to create software that achieves such a grand goal is that the attempt can produce feedback from reality that challenges the assumptions on which the goal is based. This is precisely what happened.

A PRESCRIPTIVE AND PROCESS-ORIENTED APPROACH

The central topic of this paper is the way in which rationale processes promote creative software design. More specifically, this paper identifies processes of rationale generation that reflect software life cycle processes that lead to the generation of important, new ideas for software design. In addition, this paper aims both to analyze and to promote such processes. Doing these things is impossible without using a rationale modeling approach that can represent the processes of interest. In other words, it is necessary, to use a process-oriented approach to describe rationale in software creation.

Using a process-oriented approach to describe how rationale promotes creativity limits which rationale approaches can be used. This is because these approaches differ in the degree to which they model process. Most approaches can be broadly categorized as structure oriented or process oriented (Lee & Lai, 1996). Structure-oriented approaches make no attempt to record the temporal order in which rationale is generated in design. They only record the logical relationships between statements, for example, that one statement argues against another. Process-oriented approaches record the temporal order, meaning the history, of the rationale generation, for example, that an argument arose in response to another.

Many approaches to rationale are structure oriented. For example, the authors of the QOC approach (MacLean et al., 1996) are adamant that QOC in no way records the manner in which rationale statements arise during design. The proponents of DRL (Lee & Lai, 1996) generally make no claims about design processes, but they insist that DRL does not deal with processes by which solution ideas are generated, meaning ideation. Certain applications of IBIS and PHI have also been structure oriented (McCall, 1991). In particular, the domain-oriented issue bases created using PHI (McCall, Fischer et al., 1990) and used in JANUS and PHIDIAS give no indication of the processes in which rationale is generated.

Relatively few rationale approaches are explicitly process oriented. IBIS is process oriented in its original form (Kunz & Rittel, 1970; Rittel & Noble, 1989) and in the form used by Conklin, Begeman and Burgess-Yakemovic (Conklin & Begeman, 1988; Conkin & Burgess-Yakemovic, 1996). In addition, when PHI is used to document individual design projects, it typically is used in a process-oriented manner that records the history of rationale creation. Carroll and Rosson (1992) used a very different type of process-orientation. Their rationale approach centers on the processes represented in usage scenarios. More specifically, it documents "claims," that is, user evaluations of the pros and cons of system features, as the users go through such scenarios. I refer to this approach here as *scenario-claims analysis* (SCA).

While process-oriented rationale contains temporal information not found in structure-oriented rationale, structure-oriented rationale generally requires more work to create. The

reason is that process-oriented rationale is documented in the order and wording in which it is stated. Structure-oriented rationale must be edited to exhibit its logical structure and eliminate temporal information. Advocates of the structured approach, such as the authors of QOC, argue that it is worth spending the extra time to design the rationale statements and structure because it facilitates understanding (MacLean et al., 1996).

Since my analysis is process-oriented, it must employ process-oriented rationale methods. As is explained in the next section, the experiences that led to the understanding of how rationale relates to design creativity involved a series of projects that designed software supporting PHI and used it to document the software design. It seems only appropriate, therefore, to use PHI as the primary basis here for the analysis of rationale processes that support creativity in software design. But, since my analysis attempts to show how feedback from users promotes design creativity, SCA (Carroll & Rosson, 1992) also has a crucial role to play.

IDEATION AND EVALUATION: FROM SEPARATION TO INTERTWINING

Ideation and Evaluation in Design Rationale

In most approaches to design rationale—IBIS, QOC, and DRL being well-known examples—ideation takes the form of the generation of alternatives for decisions. In IBIS and its PHI variant, decision alternatives are *positions* and the things to be decided are *issues*. It should be noted, however, that not all issues in PHI deal with decisions about features of the artifact being designed. Any question arising in design is considered an issue, including questions about facts, goals, concept definitions, causes of problems, and effects of decisions. None of these other types of issues involve ideation as it is defined above.

QOC differs from IBIS in that it only deals with decisions about features of the artifact being designed, that is, decisions that involve ideation. In QOC the decision alternatives are called options and the things to be decided are called questions (MacLean et al., 1996). DRL is quite similar to QOC in many respects, but its decision alternatives are simply called alternatives, while things to be decided are called decision problems. From the examples that Lee (1991) gives, it appears that DRL's decision problems are identical to QOC's questions and thus deal exclusively with decisions about features of the artifact. As mentioned above, however, Lee and Lai (1996) make a point of stating that DRL does not represent ideation processes.

Evaluation in most rationale approaches is done by identifying pros and cons of decision alternatives. In IBIS and PHI this is done by stating arguments for or against the alternatives (positions), while both QOC and DRL perform evaluation by assessing how well the alternatives satisfy given criteria (called goals in DRL). In these and other approaches, the evaluation can be augmented by the stating of arguments that support or attack the statements of the pros and cons.

The Separation of Ideation from Evaluation

The Separation of Ideation and Evaluation in Approaches to Creativity

Literature on creativity frequently emphasizes the value of completing ideation before evaluation begins. The main argument for this phased approach is as follows. Criticizing ideas as they are

generated inhibits the elicitation of new ideas, especially innovative ideas, which can sound risky and are often vulnerable to attack as first stated. Fear of being attacked can make people reluctant to propose creative ideas; so evaluation should be postponed until after ideas are generated.

The well-known creativity-enhancing methods known as brainstorming (Osborn, 1963) and lateral thinking (de Bono, 1973) focus on ideation. In both cases, it is treated as separate from evaluation. In fact, both methods have explicit prohibitions on evaluation during ideation, so as not to inhibit the free flow of ideas. In brainstorming, this prohibition is called "suspension of judgment" (Michalko, 2006) or "withholding criticism" (Osborn 1963). In defending this prohibition in lateral thinking, de Bono (1973, p. 7) explains, "One is not looking for the *best* approach but for as many *different* approaches as possible." He even adds, "In the lateral search for alternatives these do not have to be reasonable" (p. 7). Both approaches emphasize quantity over quality, in the belief that quantity leads to novelty. The writings of Osborn and de Bono have been very influential; thus many other creativity techniques come with warnings about not evaluating ideas as they are generated.

The Separation of Ideation and Evaluation in Rationale Research

Rittel's (Kunz & Rittel, 1970) work on IBIS has also been influential. Conklin and his colleagues have done extensive work with IBIS (Conklin & Begeman, 1988; Conklin & Burgess-Yakemovic, 1996). And PHI (McCall, 1979), of course, is a revision of IBIS. In addition, the Potts and Bruns (1988) approach to rationale is a revision of IBIS with the goal of fitting it better to software engineering. DRL is a revision of Potts and Bruns (Lee, 1991) and RatSpeak (Burge & Brown, 2006) is revision of DRL for software engineering—ironically, one that restores some features of IBIS. QOC (MacLean et al., 1996) was devised entirely separately from IBIS yet strongly resembles DRL. While there are many deviations from Rittel's approach, few of them stray far from it.

Because of Rittel's influence, it is important to understand his ideas about the relationship between ideation and evaluation in design. Simply put, Rittel saw no need to intertwine them. This is reflected in the following statement in which he briefly describes a phased model of how designers attack a decision task:

> A designer first tries to develop a set of alternative courses of action, then to figure out their potential outcomes and their likelihood, and then to evaluate them, finally to decide in favor of one of them. (Rittel, 1966, p. 13)

In this statement, the ideation part corresponds to the phrase, "to develop a set of alternative courses of action." Evaluation corresponds to the phrase, "to figure out their potential outcomes and their likelihood, and then to evaluate them."

Rittel further states that he sees design as "an alternating sequence of two kinds of basic mental activities" (Rittel, 1966, p. 17), the first kind being ideation, which he describes as follows:

> Initially, a phase of "generating variety": the search for a set of relevant possibilities which might solve the problem at hand. (This is the process of developing ideas. It ends with a set of alternatives which contain at least one element.) (Rittel, 1966, p. 17)

The second kind consists of evaluation and selection, which he describes as follows:

> This is followed by a phase of "reducing variety": the alternatives are evaluated for their feasibility and desirability, and a decision is made in favor of the most desirable, feasible alternative …. (Rittel, 1966, p. 17)

Because of these statements, from an article published 4 years before his first paper on IBIS, it should not be surprising that ideation and evaluation became incorporated into IBIS as separate processes: first, generation of positions, and then argumentation to evaluate the already-generated positions.

Rittel's commitment to separating ideation and evaluation appears to be mirrored in other rationale approaches that, like IBIS, center on the evaluation of alternatives for design decisions. Thus, for example, none of these other approaches contains a type of link that could be used to indicate that an alternative was suggested by an evaluation of another alternative or that any alternative is an improvement on another alternative. The latter is important for the simple reason that the notion of improvement implies evaluation. In short, there is no sign of any connection between ideation and evaluation in any of the major approaches for modeling rationale about design decisions. Whether intentional or not, all of these approaches, like IBIS, give the impression that ideation and evaluation are in no way intertwined. This similarity might not be entirely due to Rittel's influence, however, because many early theories of design (Alexander, 1964; Jones, 1970; Simon, 1969) exhibited a similar separation of ideation and judgment.

The Intertwining of Ideation and Evaluation in Design Discussion

MIKROPLIS (McCall, 1989; McCall et al., 1984) was the first PHI project to reveal the intertwining of ideation and evaluation in design discussion. Whereas its predecessor, the PROTOCOL project (McCall, 1979), had only a single designer, MIKROPLIS had a team of people involved in its design. Much of their discussion was documented. Because users of PROTOCOL had complained about not having control over the order in which it elicited rationale, MIKROPLIS was aimed at giving users control over display and input. This led to discussion of many issues of user interaction.

While MIKROPLIS team membership changed over its 5-year history, it included at various points people with solid knowledge of IBIS theory and applications. These included Wolfgang Schuler (Schuler & Smith, 1990), Barbara Lutes-Schaab (Lutes-Schaab, McCall, Schuler, & Werner, 1985), Harald Werner (Reuter & Werner, 1984), and Wolf Reuter (1983). Reuter, in particular, had a decade of IBIS experience when he joined the project.

As we documented discussions of the MIKROPLIS design team, differences emerged between our rationale and the adversarial rationale that Rittel (1980, pp. 7, 8) wrote about. Discussions in our team had a fundamentally different character from the clash of worldviews that IBIS was meant to deal with. Rather than being adversarial, our discussions were generally cooperative and collaborative. This is not to say that proposed ideas were not subjected to strong criticism, but the thrust of this criticism was constructive and there was a general openness to it by the group. This was also characteristic of teams in the later PHI projects.

One strong pattern that emerged in group discussion was that new ideas often arose out of evaluations of proposed ideas. While the response to criticism of (arguments against) a proposed idea (position) was sometimes to argue against it, often the response was to accept the criticism and propose a new or modified position. The adversarial argumentation that

Rittel wrote of featured an uncompromising defense of positions; the collaborative argumentation in our teams featured a general willingness to rethink positions. Where adversarial argumentation responded to criticism with rebuttal, our collaborative argumentation responded with creative ideation. Thus, while the former tended to separate ideation from evaluation, the latter intertwined them.

One of the forms that the intertwining commonly took was arguments that proposed better positions. Such arguments would typically identify an undesirable consequence of a proposed position and then immediately suggest a new or revised position that avoided that consequence. In fact, it seemed that the inclusion of the new position at the end of an argument was, in effect, a demonstration that its criticism was constructive. Thus, new positions were contained within arguments on old positions. Unfortunately, neither IBIS nor PHI recognized such combined utterances, because neither recognized intertwining. The following simple example, taken from a recent project, shows how a new position, indicated in italics, arose in an argument critical of an existing position:

> ISSUE: What programming technology should we use to create our 3D, Web-based, educational game for Mars exploration?
>
> POSITION: Flash CS4, using open-source Papervision3D for the 3D graphics.
>
> ARGUMENT FOR: Flash has 98% browser penetration. The new version of ActionScript runs up to 10 times faster, and Papervision3D looks promising.
>
> ARGUMENT AGAINST: The problem is that existing approaches to Flash 3D, such as Papervision3D, cannot make use of the GPU. This will prevent us from creating the complex graphics we need for the game. *It would be better to use a technology that doesn't have these limitations—such as Java. That way we could use Java3D or JOGL for the 3D graphics.*

Intertwining took many other forms as well. Sometimes complex negotiations would take place between the person who proposed an idea and those who criticized it. These sometimes turned into mini design projects, each with the goal of devising ways of overcoming negative consequences of a proposed idea. Often these discussions were aimed at "rescuing" a flawed proposal by figuring out how to defuse its undesirable consequences.

It was not just criticism of an idea that produced new ideas. Some arguments approved of the basic idea behind a position but advocated taking it further. Such arguments often had the form, "If you're going to do that, why not go all the way and do X."

Design ideas often went through considerable evolution as a result of many iterations of critical argumentation and revision. These tended to be long, critical conversations among the team members. Sometimes there were creative breakthroughs during meetings. Sometimes discussions dead-ended but breakthroughs occurred between meetings.

The MIKROPLIS project showed me that critical conversations promoted creativity in design. Since then I have seen this pattern of creative argumentation in a wide variety of design discussions, both in PHI-based projects and in other projects that made no use of rationale methods. It seems that the hallmark of successful collaborative discourse is the revision of ideas based on feedback from argumentative evaluation.

In retrospect, it is clear that our documentation of such creative discussions was inadequate. When a new position on an issue was generated in response to an argument, we simply connected the argument to the position with an argument-for link. When an argument contained a new position, we would extract the position and record it separately as a position linked to a revised version of the argument that omitted the statement of the position. The problem with this approach was that inspection of the documented rationale revealed no evidence of the intertwined processes by which ideas had in fact been generated. While we were in theory using a process-oriented approach to rationale, in fact we were misrepresenting the processes involved. This was because PHI had unwittingly inherited IBIS's built-in separation of ideation from evaluation—in the form of link types that treated arguments only as *responses to* rather than *generators of* positions. As a consequence, the impression that our documented rationale gave was that positions were generated intuitively and immediately as direct responses to stated issues and that the only role of arguments was to evaluate previously generated positions. There was no real indication that argumentation had played a crucial role in ideation.

The intertwining of ideation and evaluation in discussions among designers turned out to be merely one of a number of ways in which such intertwining promotes creative design. Discovery of other ways was made possible by a profound change in our understanding of the nature of design. The change was from Rittel's (1966) view of *design as planning* to Schön's (1983) view of *design as situated cognition.* This change in perspective solved major problems we encountered in creating the PHIDIAS software (McCall, Bennett et al., 1990; McCall et al. 1994; McCall, Ostwald et al., 1990). The following section begins by looking at the differences between these two views and their implications for the role of rationale in design. It then describes the problems we encountered and explains how these led us to adopt Schön's point of view.

DESIGN: FROM PLANNING TO SITUATED COGNITION

The term *situated cognition* is used with a number of different meanings. It is used here in the behavioral sense of "a transactional process of transforming and interpreting materials in the world" (Clancey, 1997, p. 23). It is in this sense of the term that we can say that both Suchman (1987) and Schön (1983) have written about situated cognition.

Two Views of Design

There are two fundamentally different views of design: as planning and as situated cognition. The former sees design as reasoning that precedes action, the latter as reasoning intertwined with and informed by action. The implication of the former is that design rationale is the documentation of the thinking and discussion of designers preparing for the actions of implementation and use. The implication of the latter is that design rationale is the documentation not only of planning by designers but also of (a) the feedback from actions that challenges design decisions, and (b) the creative thinking of designers in response to such challenges. The situated cognition viewpoint thus sees design as an intertwining of ideation and action-based evaluation. To date, the literature on all rationale methods except SCA (Carroll & Rosson, 1992) has dealt exclusively with rationale as planning.

Rittel's View of Design as Planning

Rittel clearly viewed design as planning, not as situated cognition. He declared, "Designing means thinking before acting," and he described design as a process of devising a plan (Rittel, 1966, p. 13). In fact, Rittel used the terms *designing* and *planning* interchangeably and saw design as a phase that is completed before feedback from action is available:

> The distinctive property of designing lies in the—frequently very long—interval between the design process (i.e., the construction of the plan) and the "feedbacks"—the effects of the execution of the plan. (Rittel, 1966, p. 14)

This lack of feedback implies that designers cannot test their ideas in real-world settings:

> …there is not the opportunity to approach solutions by trial and error; there is nothing like experimentation with real situations. (Rittel, 1966, p. 14)

Therefore, designers must rely solely on their imaginations to determine the consequences of their ideas:

> As a result of these characteristics, the designer operates in a world of imagination. He has to anticipate, to guess, to judge what *might* happen if a certain contemplated action will be carried out. (Rittel, 1966, p. 14)

The picture that Rittel paints is of design as speculative reasoning aimed at the production of a plan. In other words, Rittel's notion of design as purely a process of argumentation is a direct consequence of his view of design as planning.

Schön's View of Design as Situated Cognition

Schön's (1983) theory of design as reflective practice provides a fundamentally different view. Schön saw design as an alternation between an intuitive process he called *knowing-in-action* and a type of reasoning he called *reflection-in-action*. With knowing-in-action, the designer is engaged in performing a task without conscious reflection. With reflection-in-action, the designer stops acting and instead reflects on how to perform the task at hand. A designer cannot simultaneously engage in both knowing-in-action and reflection-in-action.

Knowing-in-action proceeds until a breakdown occurs. This happens when intuitive performance produces unexpected feedback from the situation at hand. In other words, there is a breakdown in the designer's expectations. Schön describes this by saying "the situation talks back" (1983, p. 131). A breakdown results when something goes wrong, but it also results when something unexpectedly good happens. Breakdowns occur when intuitive action produces either problems or opportunities that intuition cannot deal with. At this point, the designer switches to reflection-in-action to reason about how to deal with the unexpected results. If and when reflection is successful, the designer resumes knowing-in-action.

Reflective practice is repeated alternation between knowing-in-action and reflection-in-action. Schön describes the designer as engaging in an ongoing "conversation with the situation" (1983, p. 76). This is a view of design as a type of situated cognition, in that it sees design reasoning as intertwined with and informed by action.

Reflective practice models design as an intertwining of ideation and evaluation. When the situation "talks back," the "backtalk" is evaluative feedback that reveals consequences of the actions taken. The purpose of the resulting reflection-in-action is to devise new ideas for how to act; in other words, the purpose of reflective practice is ideation. Putting new ideas into action with knowing-in-action is how the designer resumes "talking to the situation." This eventually results in more backtalk that again triggers reflection that results in further ideation—and so forth.

Implications of the Two Views

To Rittel (1966), design is nothing but explicit reasoning, that is, argumentation; to Schön (1983), design is both explicit reasoning and intuitive action. Rittel's view implies that rationale can represent all design processes; Schön's view implies that it cannot. For Rittel design is reasoning in preparation for action in an external environment; for Schön design is reasoning triggered and motivated by action in an external environment. Rittel portrays design as a conversation among designers, Schön as a "conversation" between designers and a situation. As my colleagues, students, and I implemented Rittel's view of design in software, experiences in implementing and using prototypes ultimately led to rejecting Rittel's view of design as planning, in favor of Schön's view of design as situated cognition.

From Viewing Design as Planning to Viewing It as Situated Cognition

Limitations of MIKROPLIS

Towards the end of the MIKROPLIS project (McCall, 1989; McCall et al., 1984) in 1984-1985, user testing revealed two major shortcomings. One was that it did not solve the *rationale capture problem*, that is, the reluctance of designers to document their rationale. We originally thought this problem resulted from the copious and tedious secretarial work involved in documenting rationale. MIKROPLIS successfully eliminated most such work. Unfortunately, this merely revealed the enormity of the cognitive overhead in rationale capture. The other shortcoming was that when MIKROPLIS was used to design buildings, its users created rationale that failed to deal with decisions about the forms of the buildings. Without representing and editing these forms graphically, there was apparently no way for users to make decisions about them.

Ideas for PHIDIAS

In 1985 my colleagues and I began designing PHIDIAS (PHI-based Design Intelligence Augmentation System; McCall, Bennett et al., 1990; McCall et al., 1994; McCall, Ostwald et al., 1990) by extending MIKROPLIS. The new functionality supported design ideas aimed at overcoming the two major limitations of MIKROPLIS.

The first idea was for PHIDIAS to use *domain-oriented issue bases* to mitigate the capture problem (McCall, Bennett et al., 1990). Such an issue base is a collection of the issues, positions and arguments that commonly occur in a design domain—for instance, the design of a given type of building. The main goal was to reduce the work of creating a project issue base by "priming the pump" with a generic issue base for a domain—for example, design of lunar habitats—that could be tailored to a specific project, such as the design of a

specific lunar habitat for four astronauts. In addition, to alleviate the capture problem, domain-oriented issues bases could help designers by providing useful design information.

The second idea was to have PHIDIAS enable decision making about building forms by adding functionality for CAD graphics. We created this functionality but failed to foresee that attempting to incorporate form-making into PHI would lead us to abandon Rittel's (1966) view of design as nothing but argumentative planning.

Unexpected Problems in Creating PHIDIAS

We had no difficulty creating domain-oriented issue bases and integrating them into PHIDIAS, and these issue bases greatly reduced the work of creating a project-specific issue base. Unfortunately, they were not effective in providing student designers with useful information. Since students did not know what information was and was not in the system, they did not know whether searching for information would pay off. As a consequence, they often searched for information that was not in the system, got frustrated and then stopped searching for any information. This was especially unfortunate, because the system had information that could have saved them from many of the mistakes they made in design.

We also successfully implemented basic CAD functionality, but we ran into profound difficulties in attempting to integrate CAD graphic editing into the interface for rationale creation. The problem was conceptual, not technical. It resulted from apparent conflicts between the activities of form making and verbal reasoning. To solve this problem we attempted to study how student designers reasoned about form making. This attempt was repeatedly frustrated. Asking students to document their own reasoning while they drew building forms produced little or no plausible rationale. Sending others in to document the rationale of designers also produced no significant results. They would explain their rationale right up to the moment they started drawing, at which point they would not talk about what they were doing. We did succeed in getting one talented student to record a think-aloud protocol about his form making over six weeks. Unfortunately, he felt that reasoning aloud had interfered with his ability to design; so he redid the entire design over a weekend without recording any rationale. So, while we made excellent progress on implementing CAD functionality in PHIDIAS, we made no real progress integrating form-making into rationale. This prevented us from completing the PHIDIAS interface.

CRACK

The solutions to the problems that PHIDIAS had encountered became obvious when I saw a demo of the CRACK (CRitiquing Approach to Cooperative Kitchen design) system created by Anders Morch under the supervision of Gerhard Fischer (Fischer & Morch, 1988). Fischer had been investigating the use of domain-oriented construction kits for design (Fischer, 1987; Fischer & Lemke 1988). A *construction kit* is a set of graphical building blocks that can be dragged and dropped into a workspace. He found that while such kits greatly facilitated the creation of designs, these designs were often functionally flawed. He concluded that construction kits had to be supplemented with some way of avoiding design mistakes. For this purpose, Fischer proposed using what he termed *knowledge-based critics* to guide design with construction kits. Morch's master's thesis implemented Fischer's ideas in the kitchen design domain, in which Morch had previously worked.

CRACK featured a CAD graphics editor for creating kitchen floor plans using a kitchen construction kit featuring such domain-level building blocks as walls, windows, doors, counters, stoves and sinks. This kit provided a direct and intuitive way for users to construct kitchen floor plans. Since each building block had an assigned domain-level meaning**,** knowledge-based critics could determine whether a constructed floor plan satisfied or violated rules of kitchen design. If rules were violated during the construction of a layout, critiquing messages popped up on the screen to tell the user which rules had been broken. For example, if a stove were placed where pans could be hit by an opening door, then the designer got a message saying that the stove should not be located next to a door.

CRACK was intended not to enforce its rules, for example, as an expert system would, but rather to empower the user to decide whether to accept or reject them. Unfortunately, it was often difficult for users to decide whether to break rules. I suggested that this was because such decisions required knowledge of the rationale underlying the rules. I therefore proposed the addition of a hypertext subsystem containing rationale for the rules of kitchen design in the form of a PHI-based, domain-oriented issue base. The decision was made to create a successor to CRACK that did just that. The successor was called JANUS (McCall, Fischer et al., 1990; Fischer et al., 1996), after the Roman god with two faces, because it had both a form-construction interface and an argumentation interface.

JANUS and PHIDIAS

From the perspective of the PHIDIAS project, the notion of coupling PHI hypertext to a CRACK-type interface was a revelation. It offered in one stroke a solution to two problems plaguing the PHIDIAS project. First of all, it showed how users could be alerted to the existence of useful information in a PHI issue base while they worked on a design problem. Secondly, it suggested that rather than attempting to integrate the editing of CAD graphics into the editing of a PHI hyperdocument, the solution was to have two separate interfaces—a form construction interface and an argumentation interface—and switch between these using critics. So while Morch and others constructed JANUS, my programming team constructed a similar coupling of CAD form-construction and argumentation in PHIDIAS. User testing showed that both systems successfully supported use of rationale to inform construction of floor plans.

From Argumentative Planning to Reflective Practice

It was not immediately clear that the new systems challenged Rittel's (1966) theory of design as argumentative planning. Awareness of that challenge first surfaced when Morch wrote a working paper proposing that JANUS supported two different modes of designing: *constructive design* and *argumentative design*. At first, I balked at that distinction, which was heresy from the Rittelian perspective. But the failed attempts to integrate form-construction into PHI ultimately led me to abandon the notion that form making is purely an argumentative process. Morch's names for the two design modes were therefore put in the title of our first paper on the new type of system (Fischer, McCall, & Morch, 1989).

Not long after this it became clear that the failures in integrating form-construction into PHI and the success of our dual-interface approach both fit Schön's (1983) ideas about reflective practice. Constructive design with construction kits corresponded to knowing-in-

action, critiquing corresponded to breakdowns, and argumentative design with PHI hypermedia corresponded to reflection-in-action. So we came to see JANUS and PHIDIAS as unintended demonstrations of the correctness of Schön's theory of design—a theory fundamentally incompatible with Rittel's.

While at first Schön's theory was merely a retrospective explanation for the success of our systems, later it became the central driving principle behind the design of PHIDIAS and HyperSketch (McCall et al., 1997; McCall et al., 2001). PHIDIAS implemented a variety of additional ways in which the existence of breakdowns could be detected by the system (McCall & Johnson, 1997) or volunteered by users of the system (McCall, 1998). An example of the former is shown in Figures 1 and 2. Here, knowledge-based agents are created by system users

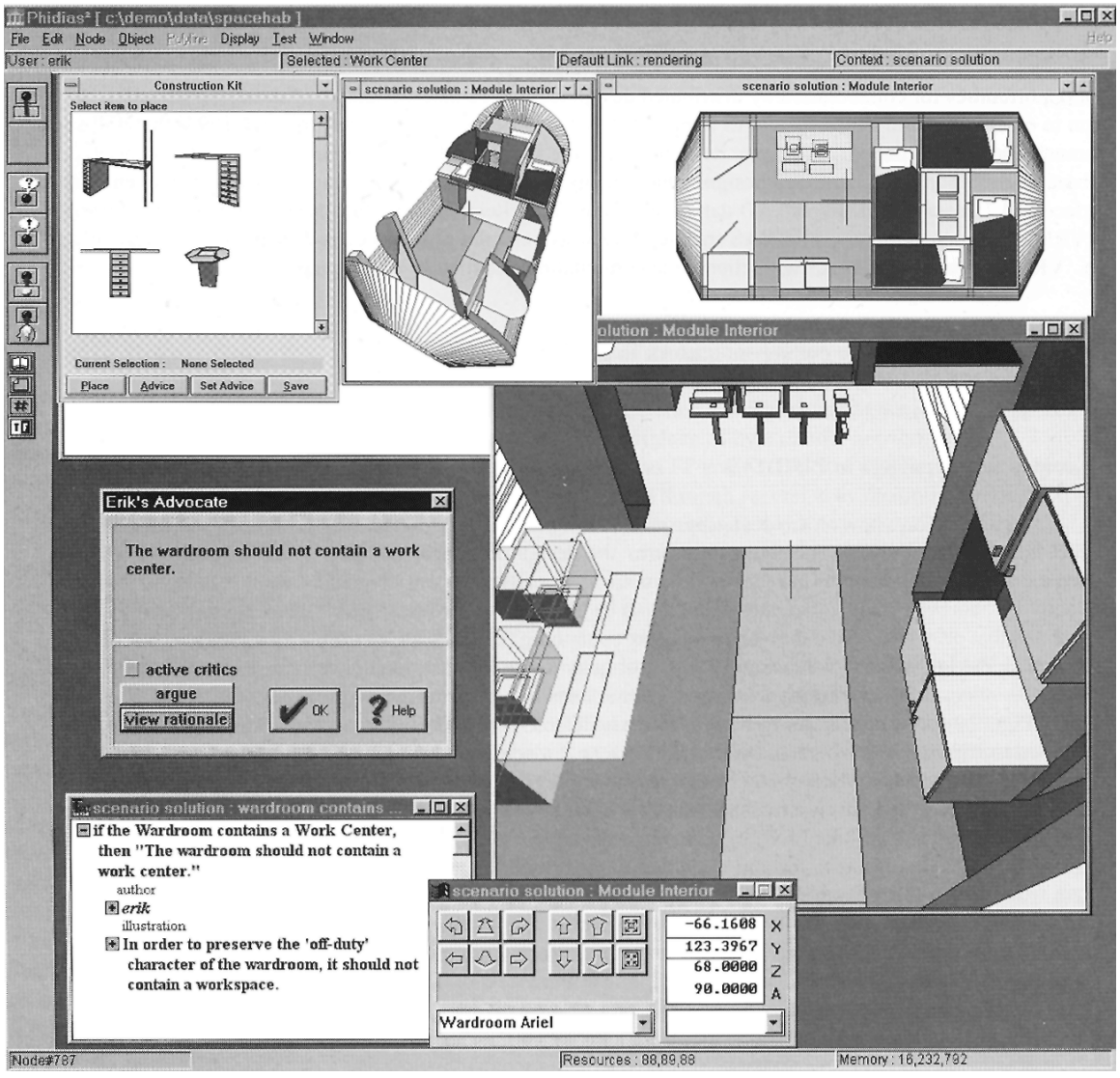

Figure 1. In PHIDIAS, designers working on the same project can create knowledge-based agents called *advocates*, which are critics that lobby for design principles that they believe in. In this figure, Patrick violated an advocate created by Erik, and thus received a critiquing message. Patrick has opted to view Erik's rationale for the advocate.

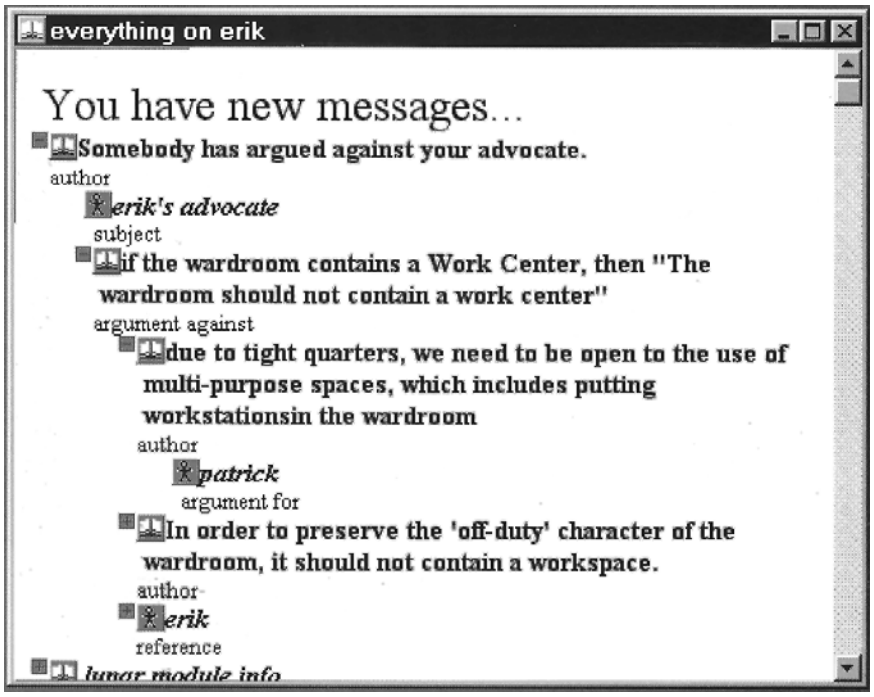

Figure 2. Patrick argued against the rationale for Erik's advocate, so Erik was notified and sent the argument. He was then given the option to participate in an issue-based discussion with Patrick about whether the advocate should be violated.

as advocates of their opinions in a collaborative design environment. Then when other designers use the system, they are alerted if they construct design features that conflict with any of these advocate agents, as in Figure 1. They also have the opportunity to view and argue with the rationale for the advocate. If a designer argues against the advocate agent, the designer who created it is alerted to this fact and offered the chance to discuss this situation with the designer who disagreed with the advocate. The resulting online discussion can be recorded in the form of issue-based argumentation, as in Figure 2.

Other research driven by Schön's ideas inquired into what sorts of interfaces were needed for intuitive knowing-in-action. HyperSketch (McCall et al., 1997; McCall et al., 2001) explored intuitive form construction through computer-supported sketching. This was in response to architecture students who complained that construction kits inhibited their intuitive exploration of building forms.

Schön's (1983) theory of design as situated cognition shows another way in which ideation and evaluation are intertwined. Previously we saw this only in argumentative design discussion; now we see it when argumentation is coupled with action. Furthermore, this intertwining can be seen as promoting creative ideation. When a critic reveals that something is wrong with the design, the designer rethinks a design decision and devises new solution ideas.

It should be noted that the criticisms here of Rittel's (1966) ideas about feedback and argumentative planning in no way imply a rejection of his theories in toto. Instead, this criticism is meant as a necessary corrective if design rationale, the field that Rittel pioneered, is to be successful. Nor does this criticism imply an unqualified endorsement of Schön. In fact, it is argued below that Schön's notions of reflective practice are too limited to account

for several important ways in which creative design involves situated cognition. Accounting for these additional ways involves extending Schön's notions by bringing into the picture Rittel's ideas about collaborative and participatory design.

Software Design as Situated Cognition

How Our Software Design Experiences Differ from Rittel's Description of Design

Our experiences of software design and Rittel's description of design differ in the role of feedback from implementation and from use in informing design. For Rittel (1966), a distinguishing feature of all design is that it cannot be informed by such feedback. Yet our experiences provide numerous counterexamples to this claim.

Was Rittel completely wrong? Or was he simply referring to a different kind of design than we engaged in? His arguments against learning from feedback suggest the latter. Consider the following statement from the article that he wrote with Webber about wicked problems:

> One cannot build a freeway to see how it works, and then easily correct it after unsatisfactory performance. Large public-works are effectively irreversible, and the consequences they generate have long half-lives. Many people's lives will have been irreversibly influenced, and large amounts of money will have been spent—another irreversible act. (Rittel & Webber, 1973, p. 163)

Rittel (1966) claims that his theory applies to all types of design, yet the above-stated argument depends on properties found in some types of design but not others. In particular, the argument applies to large-scale design projects with large costs and large consequences. The specific example used, a freeway, represents an infrastructural level of design, meaning a very low-level of structure—*infra* meaning *below* in Latin (Hoad, 1996). Designing such a large-scale physical artifact might indeed be, as Rittel claimed, a one-shot operation in which feedback from implementation and use plays no role. Nevertheless, this does not imply that it plays no role in other levels of design.

If one substitutes a "high-level" artifact, such as a piece of furniture, into the Rittel-Webber argument, the credibility of that argument collapses. For example, an industrial designer can in fact build a chair to see how it works. If its performance is unsatisfactory, for example, if it is uncomfortable or structurally unsound, the designer can easily correct the bad design. Furthermore, its consequences are unlikely to have long half-lives. If any consequences are irreversible, they are unlikely to be severe and can be restricted to a small group of users who test the chair before it is made available to the public. The costs of redesigning and re-implementing the chair are likely to be small compared to profits made from selling thousands of well-designed chairs. In other words, feedback from implementation and use can play a significant role in the design of chairs and other high-level artifacts.

Difference in level, however, cannot explain all the differences between the design of software and the design of the sorts of low-level, large-scale artifacts that Rittel focused on. The design of new buildings and freeways generally might not involve learning from feedback about implementation and use, yet it is hard to find any level of software design that cannot learn from such feedback. The implementation and use of working prototypes and early versions play crucial roles in shaping the design of new operating systems, new browsers, and

new rich Internet applications—three very different levels of software design. There are no comparable roles for usable prototypes or early versions of buildings or freeways.

Another limitation of Rittel's theory is that it ignores the redesign of artifacts. It is a truism that buildings and cities evolve over decades through many episodes of redesign (Brand, 1994). Such redesign is often informed by implementation and use. Successful software at all levels also goes through many episodes of redesign that are informed by feedback from implementation and use of previously released versions.

Our PHI-based software projects contained many cases where the design was changed in the middle of being implemented. The design of the PHIDIAS interface between PHI rationale and CAD graphic construction of form is the most conspicuous example of this. Current work by software engineers on iterative and incremental design also has this character. To be sure, software engineering for years militated against changes in decisions about requirements and design, because they were so costly. But in recent years, software engineers have become increasingly open to such changes.

How Feedback from Implementation Led to New Design Ideas

Over the history of the MIKROPLIS and PHIDIAS projects, a single type of phenomenon dominated the generation of design ideas: the repeated discovery of new affordances that arose as unplanned side-effects of implementing required design features. These discoveries influenced the design of the software in two ways. One was in suggesting ideas for the architecture of the system; the other was in leading us to re-evaluate and revise the requirements for the system.

Over the 18 years of the projects, the system architecture that emerged was a radically simple and integrated hyperbase management system (HBMS) with an operator-algebraic, functional language called PHIQL (PHIDIAS Hypermedia Inference and Query Language). This HBMS was coupled with subsystems for display of a wide range of multimedia data, including text, vector graphics, images, and video, together with subsystems for editing text and vector graphics. We came to call this a *hyperCAD architecture*.

The way in which ideas for PHIDIAS' architectural features emerged shows how feedback from implementation can shape the design of system architecture. For example, when we decided we needed to represent and edit vector graphics, the obvious approach was to buy or build a separate 3D graphics system and add it to the architecture. I started to do just that, but my knowledge of the implementation details of the graph-handling functionality of MIKROPLIS led to the insight that it could be used for scene graphs as well as textual networks. Once this new affordance of the MIKROPLIS system was discovered, it became clear that utilizing this affordance would make it possible to link any text to any vector graphic object in the system—thus enabling PHI-based discussion of all graphical objects and configurations. In other words, knowledge of implementation details led to discovery of an unplanned affordance of an existing system, which in turn led to the insight that exploiting this affordance served the goals of the larger project in ways that had not been foreseen. Here, both knowledge of implementation details and the affordances of those details provided feedback from implementation that led to the generation of design ideas.

As it turned out, once we had designed a system architecture that implemented scene graphs in the HBMS, additional unplanned affordances emerged as direct consequences of

this decision. For example, since PHIQL could now construct arbitrary displays of linked text and vector graphics, it became trivial to construct in PHIDIAS the catalogues of completed designs that existed in JANUS—something which had previously been of interest to us but too far down on our priorities to appear in our system requirements. Using PHIQL and scene graphs also made it possible and easy to create a catalogue of reusable subassemblies, something that did not exist in JANUS. Though we had never before thought of creating such a catalogue, we quickly realized it would be a very useful feature for a designer. So we added this and a catalogue of completed designs to our list of system requirements.

The integrated hyperCAD architecture emerged as a consequence of repeated discovery of unplanned affordances. Over the history of the PHIDIAS project, we frequently found that desired new functionality could best be implemented by exploiting affordances of the existing system rather than by adding new code that implemented the functionality from scratch. It was a more efficient use of our time and knowledge, and it tended in turn to produce still more affordances. We kept discovering that we were able to generate valuable new functionality almost for free. We began talking not only of what we wanted the system to be but also of what the system itself "wanted to be"—a metaphorical way of referring to new affordances produced as side effects of implementation. This sort of metaphor, which anthropomorphizes the artifact being designed and treats it is if it were a partner in discussion, has been used by a number of well-known (building) architects, most famously Louis Kahn (Twombly, 2003). It is closely related to Schön's (1983) reference to the situation "talking back."

There are dozens of other examples of how feedback from implementation shaped the architecture of PHIDIAS and led to the addition of new system requirements, far more than there is room here to describe. While this sort of feedback was the most frequent source of new design ideas, many of the more profound ideas emerged in feedback about system use.

How Feedback from Use Led to New Design Ideas

Our PHI-based software projects contained a number of important cases where feedback from use led to new design ideas. These included the following:

- Users of PROTOCOL complained about lack of control of the order in which issues were dealt with. This led to the design of MIKROPLIS as a system where users had complete control over the order of rationale input and display.
- Use of MIKROPLIS indicated that it had not solved the rationale capture problem. This led to the use of domain-oriented issue bases in JANUS and PHIDIAS
- Tests of MIKROPLIS users attempting building design determined that they failed to deal with decisions about the building form. This led to the inclusion of CAD graphics in the redesigned version of MIKROPLIS that came to be called PHIDIAS.
- Tests of users of domain-oriented issue bases in PHIDIAS showed that they had difficulty finding useful information in these issue bases. This contributed to the use of critics in PHIDIAS to identify and retrieve useful issue-based information.

There were numerous other examples during all of our software projects. One early example of this happened in 1982 with the very first MIKROPLIS prototype. MIKROPLIS had originally been designed as a query-based retrieval system, but tests with users revealed

this approach to be inadequate. In particular, almost all users of the system kept pointing to individual texts displayed on the screen and saying something like, "How do I find the information about this?" We repeatedly showed users how to use queries to find such information, but they continued to have difficulties. I finally got the idea of enabling them to place the cursor on the desired text and instruct the computer to traverse a link associated with that text—something roughly comparable to clicking on a link in a Web page. At the time we had no graphical user interfaces, so I had the user move the cursor to the text with the arrow keys and then press the Enter key to signal the computer to perform link traversal. Once we had implemented this feature, all users rapidly adopted this as the favored mode of interacting with the system. This was the first inkling we had of what was to become the future of interacting with hyperdocuments: clicking on links. The crucial point is that without feedback from users, we would not have come to this idea on our own.

Another example came from having design students use PHIDIAS to construct building forms. Many complained that construction kits were too restrictive and not sufficiently intuitive, especially since using construction kits in realistic projects requires browsing through many menus and panels of information to find the objects the designer wants to place in the scene. In response to these complaints, we created functionality for pen-based drawing and creating hyperdocuments of linked drawings (McCall et al., 1997).

Extending Schön's View to Account for Feedback from Implementation and Use

Schön's (1983) theory of reflective practice does not cover the sort of situated cognition in which feedback from implementation and use challenges a designer to revise the design of software. This is because Schön's theory only deals with action in the sense of the purely intuitive process he calls knowing-in-action. According to reflective practice the designer is in this process when feedback occurs that produces a breakdown and a switch to reflection-in-action. There are a number of features of this account that do not fit crucial cases of situated cognition in software design. First of all, actions do not have to be intuitive to produce feedback that leads designers to rethink the design of the system. The actions of implementation and use may well involve complex combinations of knowing-in-action and reflection-in-action. In any case, the mental states of the implementers or the users are not relevant here. Nor is it relevant what mental state the designer is in when feedback arrives; the designer could be acting, reflecting, just browsing the web, or eating a sandwich. The only thing that matters is that the feedback produces surprises and that these constitute a breakdown of the designer's expectations about the consequences of design ideas—either in the form of unexpected problems or unexpected opportunities. In such cases, the breakdowns will challenge the designer to rethink the design of the system and come up with new ideas that solve the problems or exploit the opportunities.

If we simplify Schön's model of reflective practice, we can make it general enough to cover all the cases. Rather than talking of knowing-in-action and reflection-in-action, we can talk simply of action and reflection. We can then say that in all cases of design as situated cognition *action* produces *feedback* that results in a *breakdown of expectations*, and that this promotes *reflection* aimed at the generation of new design ideas (ideation) to deal with the source of the feedback.

We can further modify Schön's model to account for critical conversations in argumentation among designers. Here a designer proposes an idea to a group of participants and gets feedback from them in the form of critiques of the idea. These critiques are only based on *speculations about the consequences* of the proposed design idea but are still capable of causing a breakdown in the expectations of the designer who proposed it. Such a breakdown then leads that designer—and others participating in the discussion—to reflect on how to revise the proposed idea or to devise a new idea. Here we have feedback, breakdown, reflection and the generation of new ideas without any actions of any kind. And yet this type of critical conversation bears a clear resemblance to reflective practice.

IMPLICATIONS FOR RATIONALE THAT PROMOTES CREATIVITY IN SOFTWARE DESIGN

Critical Conversations That Promote Creativity in Software Design

This paper has identified three processes in which the intertwining of ideation and evaluation promotes creativity in software design. When design ideas are evaluated, this evaluation can produce feedback that challenges designers to generate new ideas that improve the quality of the design. The three processes are as follows:

- The intertwining of ideation and evaluative argumentation in design discussion,
- The intertwining of the action of software implementation with reflection on the feedback from implementation, and
- The intertwining of the action of use with reflection on the feedback from use.

The first process involves purely argumentative conversation. The second and third involve types of situated cognition that do not precisely fit Schön's (1983) model of reflective practice but which, nevertheless, can be described as designers' conversations with situations.

Rittel's (1972) idea about the importance of involving implementers and users in participatory collaboration with designers comes into play in creative design of software—but in a way that Rittel did not anticipate. While he envisioned participation as taking the form of argumentative discussion, understanding design as situated cognition leads to us to extend this participation to the provision of feedback by implementers and users about the actual consequences of implementation and use of the software being designed.

What Rationale Needs to Do to Support the Critical Conversations

Critical conversations are rationale processes that help designers to be more creative. Since they are processes, a rationale approach that recognizes and promotes them is by definition process oriented. Since these processes are for the purpose of improving design, the rationale approach is by definition prescriptive. To support the evaluation that promotes ideation, a rationale approach must represent how evaluations promote ideation. It must represent the evaluations and the ideas they lead to. It must also provide links that show which ideas were generated in response to which evaluations. Any approach to rationale that aims to support the full range of design creativity must encourage and document the generation of evaluative

feedback from (a) design discussion, (b) implementation, and (c) use. To do this, it must capture rationale containing this feedback from designers, implementers, and users. It must also support the communication of this rationale to designers. If feedback from action conflicts with feedback from the pure argumentation, it is likely that the former should trump the latter—since evidence and experience trump speculation. Because of this, documented feedback should always indicate whether its source is argumentative discussion, implementation, or use. In addition, the author of the feedback should always be indicated so that follow-up conversations can be established.

Decision-centric approaches to rationale, such as IBIS (Kunz & Rittel, 1970) and PHI (McCall, 1979, 1986, 1991), are unlikely to be sufficient for collecting feedback from implementation and use, because such methods only model the design process as a coherent whole. A rationale method, such as SCA (Carroll & Rosson, 1992), is highly preferable for collecting feedback from use, because it models use processes as coherent wholes. It can thus systematically enumerate use situations and the feedback resulting from them in a way that decision-centric approaches simply cannot match. However, what needs to be done is to more closely integrate approaches like SCA with decision-centric rationale.

An open question is how the feedback from implementation should be collected. Should it be treated as a decision-centric rationale process or should a special method be developed? Whatever is done needs to be capable of systematically enumerating the feedback from implementation and it needs to be integrated with the decision-centric rationale for design.

CONCLUSIONS AND FUTURE WORK

Methods for rationale elicitation and documentation can promote creativity in software design by recognizing and promoting feedback-driven critical conversations in software projects. Critical conversations are rationale discussions in which the ideation, meaning the generation of design ideas, is intertwined with evaluation of those ideas in the sense that feedback from evaluation challenges designers to devise new ideas. There are three main types of such conversations:

- purely argumentative design discussions where designers get feedback from the speculative reasoning of other design participants,
- discussions where designers get feedback from implementers about the consequences of implementation of the software being designed, and
- discussions where designers get feedback from users about the consequences of use of the software being designed.

The first of these corresponds to Rittel's (1972) view of design as purely a process of argumentation, but it goes beyond the argumentative discussions that IBIS supports. The other two view design as a process in which argumentation is situated in the context of action that motivates and informs it. To maximize the potential of rationale to promote creative software design, we must move beyond Rittel's view of design rationale as pure argumentation and see it also as situated argumentation.

Considerable work needs to be done in revising approaches to rationale to support the critical conversations described above. Decision-centric rationale methods, such as IBIS and PHI, have

to be revised to represent the intertwining of argumentative evaluation and idea generation. The changes made to represent this intertwining in pure argumentation will provide a basis for further changes needed to support situated argumentation in the context of implementation and use. In addition to modifying decision-centric approaches, usage-centric approaches to rationale such as SCA (Carroll & Rosson, 1992) need to be utilized as ways of systematically obtaining feedback from use situations. Research also needs to be done to determine how best to support the capture and communication of feedback from implementation. Finally, work needs to be done on integrating these various approaches to rationale.

A crucial lesson of the JANUS (McCall, Fischer et al., 1990; Fischer et al., 1996) and PHIDIAS (McCall, Bennett et al., 1990; McCall et al., 1994; McCall, Ostwald et al., 1990) projects is that both delivery and capture of rationale need to be integrated into the software that supports action. This means that rationale functionality should be integrated into the tools for modeling and implementing software. It also suggests that rationale capture may need to be integrated into the software artifacts being designed to enable feedback from actual use.

REFERENCES

Alexander, C. (1964). *Notes on the synthesis of form*. Cambridge, MA, USA: Harvard University Press.

Brand, S. (1994). *How buildings learn: What happens after they're built*. London: Phoenix Illustrated.

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

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

Clancey, W. J. (1997). *Situated cognition: Human knowledge and computer representations.* Cambridge, UK: Cambridge University Press.

Conklin, E. J., & Begeman, M. (1988). gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions on Information Systems, 6*, 303–331.

Conklin, E. J., & Burgess-Yakemovic, K. C. (1996). A process-oriented to design rationale. In T. P. Moran & J. M. Carroll (Eds.), *Design rationale: Concepts, techniques, and use* (pp. 393–427). Mahwah, NJ, USA: Lawrence Erlbaum and Associates.

de Bono, E. (1973). *Lateral thinking: Creativity step by step*. New York: Harper Colophon.

Dutoit, A. H., McCall, R., Mistrik, I., & Paech, B. (2006). Rationale management in software engineering: Concepts and techniques. In A. H. Dutoit, R. McCall, I. Mistrik, & B. Paech (Eds.), *Rationale management in software engineering* (pp. 1–43). Heidelberg, Germany: Springer Verlag.

Fischer, G. (1987). An object-oriented construction and tool kit for human-computer communication. *Computer Graphics*, *21*, 105–109.

Fischer, G., & Lemke, A. (1988). Construction kits and design environments: Steps toward human problem-domain communication. *Human-Computer Interaction*, *3*, 179–222.

Fischer, G., Lemke, A. C., McCall, R., & Morch, A. I. (1996). Making argumentation serve design. In T.P. Moran & J. M. Carroll (Eds.), *Design rationale: Concepts, techniques, and use* (pp. 267– 321). Mahwah, NJ, USA: Lawrence Erlbaum Associates.

Fischer, G., & Morch, A. (1988). CRACK: A critiquing approach to cooperative kitchen design. In *Proceedings of the International Conference on Intelligent Tutoring Systems* (pp. 176–185). New York: ACM.

Fischer, G., McCall, R., & Morch, A. (1989). Design environments for constructive and argumentative design. In K. Bice & C. Lewis (Eds.), *Proceedings of the SIGCHI Conference on Human Factors in Computing System* (CHI'89; pp. 269–275). New York: ACM.

Hoad, T. F. (Ed.). (1996). *Oxford concise dictionary of English etymology*. New York: Oxford University Press.

Jones, J. C. (1970). *Design methods: Seeds of human futures*. New York: Wiley-Interscience.

Kunz, W., & Rittel, H. W. J. (1970). Issues as elements of information systems (Working Paper 131). Institute for Urban and Regional Development, University of California, Berkeley. Available at http://iurd.berkeley.edu/sites/default/files/wp/131.pdf

Lawson, B. (2005). *How designers think: The design process demystified* (4th ed.). Burlington, MA, USA: Architectural Press (Elsevier).

Lee, J. (1991). Extending the Potts and Bruns model for recording design rationale. In *Proceedings of the 13th International Conference on Software Engineering* (ICSE 13; pp. 114–125). New York: ACM.

Lee, J., & Lai, K.-Y. (1996). What's in design rationale? In T. P. Moran & J. M. Carroll (Eds.), *Design rationale: Concepts, techniques, and use* (pp. 21–51). Mahwah, NJ, USA: Lawrence Erlbaum Associates.

Lutes-Schaab, B., McCall, R., Schuler, W., & Werner, H., (1985). *MICROPLIS – ein Textbank-Management-system* [MICROPLIS: A textbase management system; Final report]. Heidelberg, Germany: Gesellshaft für Information und Dokumentation (GID), Sektion für Systementwicklung (SfS), [Society for Information and Documentation, Section for Systems Development].

MacLean, A., Young, R. M., Belotti, V. M. E., & Moran, T. P. (1996). Questions, options and criteria. In T. P. Moran & J. M. Carroll (Eds.), *Design rationale: Concepts, techniques, and use* (pp. 53–106). Mahwah, NJ, USA: Lawrence Erlbaum and Associates.

McCall, R. (1979). *On the structure and use of issue-systems in design*. Unpublished doctoral dissertation, University of California, Berkeley. (Available from University Microfilms International, Ann Arbor Michigan)

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

McCall, R. (1989). MIKROPLIS: A Hypertext system for design, *Design Studies*, *10*, 228–238.

McCall, R. (1991). PHI: A conceptual foundation for design hypermedia. *Design Studies*, *12*, 30–41.

McCall, R. (1998). World wide presentation and critiquing of design proposals with the Web PHIDIAS System. In S. van Wyck & S. Seebohm (Eds.), *Digital Design Studios: Do Computers Make a Difference? Proceedings of the 1998 Conference of the Association of Computer Aided Design in Architecture* (ACADIA 98; pp. 254–265). Quebec City, Quebec, Canada: ACADIA.

McCall, R., Bennett, P., & Johnson, E. (1994). An overview of the PHIDIAS hyperCAD system. In A.C. Harfmann & M. Fraser (Eds.), *Proceedings of the 1994 Conference of the Association for Computer Aided Design in Architecture* (ACADIA '94; pp. 63–76). St. Louis, MO, USA: ACADIA.

McCall, R., Bennett, P., d'Oronzio, P., Ostwald, J. L., Shipman, F. M., & Wallace, N. F. (1990). PHIDIAS: A PHI-based design environment integrating CAD graphics into dynamic hypertext. In R. A. Streitz & J. Andre (Eds.), *Hypertext: Concepts, systems, and applications* (pp. 152–165). Cambridge, UK: Cambridge University Press.

McCall, R., Fischer, G., & Morch, A. (1990). Supporting reflection-in-action in the JANUS design environment. In M. McCullough, W. Mitchell, & P. Purcell (Eds.), *The electronic design studio: Architectural education in the computer era* (pp. 247–259). Cambridge, MA, USA: MIT Press.

McCall, R., & Johnson, E. (1997). Using argumentative agents to catalyze and support collaboration in design. *Automation in Construction*, *6*, 299–309.

McCall, R., Johnson, E., & Smith, M. (1997). HyperSketching: Design as creating a graphical hyperdocument. In R. Junge (Ed.), *Proceedings of the 7th International Conference of Computer Aided Architectural Design Futures* (CAAD Futures 1997; pp. 849–854). Dordrecht, Netherlands: Kluwer Academic Publishers.

McCall, R., Lutes-Schaab, B., & Schuler, W. (1984). An information station for the problem solver: System concepts. In *Proceedings of the First International Conference on Application of Mini- and Microcomputers in Information, Retrieval and Libaries* (pp. 111-118). Amsterdam: North-Holland.

McCall, R., Ostwald, J. L., Shipman, F. M., & Wallace N. F. (1990). The PHIDIAS hyperCAD system: Extending CAD with hypermedia. In *From research to practice: Proceedings of the 1990 Conference of the Association for Computer Aided Design in Architecture* (ACADIA 90; pp. 145–156). Big Sky, MT, USA: ACADIA.

McCall, R., Vlahos, E., & Zabel, J. (2001). Conceptual design and hyperSketching: Theory and Java prototype. In B. de Vries, J. van Leeuwen, & H. Achten (Eds.), *Computer-aided architectural design futures 2001* (CAAD Futures 2001; pp. 285–297). Dordrecht, Netherlands: Kluwer Academic Publishers.

Michalko, M. (2006). *Thinkertoys: A handbook of creative-thinking techniques* (2nd ed.). Berkeley, CA, USA: Ten Speed Press.

Osborn, A. F. (1963). *Applied imagination: Principles and procedures of creative problem solving*. New York: Scribner.

Potts, C., & Bruns, G. (1988). Recording the reasons for design decisions. In *Proceedings of the 10th International Conference on Software Engineering* (pp. 418–427). Washington, DC, USA: IEEE Computer Society Press.

Reuter, W. D. (1983). *Thesen und Empfehlungen zur Anwedung von Argumentativen Informationssystemen* [Theses and Recommendations for the Application of Argumentative Information Systems; internal report; Working Paper A-83-1]. Institut für Grundlagen der Planung, Universität Stuttgart, Germany.

Reuter, W. D., & Werner, H. (1984). *Zusammenstellung und Beschreibung von Anwendungsfaellen Argumentativer Informationssysteme* [Compilation and Description of Applications of Argumentative Information Systems; internal report; Working Paper A-84-4]. Institut für Grundlagen der Planung, Universität Stuttgart, Germany.

Rittel, H. W. J. (1966). Some principles for the design of an educational system for design. Available as Reprint 54, Institute of Urban and Regional Development, University of California, Berkeley, http://iurd.berkeley.edu/sites/default/files/pubs/RP54.pdf

Rittel, H. W. J. (1972). On the planning crisis: Systems analysis of the "first and second generations" In *Bedrifts Okonomen, 8*, 390–396. Also available as Reprint 107, Institute of Urban and Regional Development, University of California, Berkeley, http://iurd.berkeley.edu/sites/default/files/pubs/RP107.pdf

Rittel, H. W. J. (1980). *APIS: A concept for an argumentative planning information system*. Working Paper 324, Institute of Urban and Regional Development, University of California, Berkeley, http://iurd.berkeley.edu/sites/default/files/wp/324.pdf

Rittel, H. W. J., & Noble, D. (1989). *Issue-based information systems for design*. Working Paper 492, Institute of Urban and Regional Development, University of California, Berkeley, http://iurd.berkeley.edu/sites/default/files/wp/492.pdf

Rittel, H. W. J., & Webber, M. (1973). Dilemmas in a general theory of planning. *Policy Sciences, 4*, 155–169. Also available as Reprint 86, Institute of Urban and Regional Development, University of California, Berkeley, http://iurd.berkeley.edu/sites/default/files/pubs/RP86.pdf

Schön, D. (1983). *The reflective practitioner: How professionals think in action*. New York: Basic Books.

Schuler, W., & Smith, J. B. (1990). Author's argumentation assistant (AAA): A hypertext-based authoring tool for argumentative texts. In A. Rizk, N. Streitz, & J. Andre (Eds.), *Hypertext: Concepts, systems and applications* (pp. 137–151). Cambridge, UK: Cambridge University Press.

Simon, H. A. (1969). *The sciences of the artificial*. Cambridge, MA, USA: The MIT Press.

Simon, H. A. (1973). The structure of ill-stuctured problems. *Artificial Intelligence*, *4*, 181–202.

Suchman, L. (1987). *Plans and situated actions: The problem of human–machine communication*. Cambridge, UK: Cambridge University Press.

Twombly, R. (2003). *Louis Kahn: Essential writings.* New York: W.W. Norton & Co.

Author's Note

All correspondence should be addressed to:
Raymond McCall
Department of Planning and Design
University of Colorado, Denver
Boulder, CO 80309
USA
mccall@colorado.edu

Human Technology: An Interdisciplinary Journal on Humans in ICT Environments
ISSN 1795-6889
www.humantechnology.jyu.fi