

Collaboration During Conceptual Design

Lara D. Catledge
Colin Potts
Georgia Institute of Technology
College of Computing

Abstract

Conceptual design involves requirements analysis, functional specification, and architectural design. It remains informal and poorly understood. We studied the conceptual design activities of a representative industrial software project, Centauri, for three months with follow-up observations and discussions over the following six months. Our goal was to understand how patterns of collaboration and communication in project teams affect the convergence of the project on a common vision and a documented specification. In this paper, we present our research methodology, our findings, and their implications for process and tool support. The following observations stand out. First, convergence on a common system vision was painfully slow. The major impediment to faster progress was the difficulty that the project team had in making critical allocation and interface design decisions. Second, Centauri project members repeatedly raised certain issues and failed to reach closure on key problems. Finally, we observed a persistent tension between the desire on behalf of nearly all project members to follow a prescriptive development process and the urgency of delivering a working product.

1. Introduction

Conceptual design involves requirements analysis, functional specification, and architectural design. Despite recent work in systematizing software development processes, conceptual design, which often tackles ill-defined and poorly understood problems, remains informal and open-ended. In practice, “soft” factors, such as organizational culture, patterns of collaboration, and team effectiveness, affect project success just as much as “hard” engineering methods.

Very little is known about how teams do conceptual design in commercial projects, so many requirements engineering approaches may be misdirected. However, research has addressed some pivotal issues that we investigated further in the study described in this paper.

Convergence on a common technical vision. Curtis et al. [3], by interviewing key personnel in 19 organizations, concluded that “superdesigners” enable large projects to converge. Superdesigners hold and support the project vision, are unusually good at relating customer desires to technical constraints, command the respect of team members, but are seldom managers. A later longitudinal study of a research project [14]

showed that knowledge acquisition becomes the major convergent activity in open-ended design, especially when there is no superdesigner. Analysis of design meetings in several cultures reveals that most discussion revolves around questions about how a feature could be implemented and only rarely about why it is needed [9], suggesting that team members either converge quickly on a common understanding or prefer not to question assumptions.

How, then, does a team converge on a common vision when working under commercial pressures? Previous studies have shown that design by individuals [6] and small teams [14] is not a straightforward process, but involves episodes of organized reasoning punctuated by periods of concrete information gathering and exploration. Software process models such as the spiral model are acknowledgments that design is inherently incremental and that initial directions must often be reconsidered once further concrete information has been obtained. But there is a large difference between using new analyses to help reconsider or revise design progress and sporadically abandoning and restarting the design process. The first is a monotonic process, albeit with plateaus in progress; the second is non-monotonic, because it includes reversals in progress.

Organizational working memory. Effective teams and organizations learn from experience. But what of the organization’s short-term, “working” memory? Requirements meetings typically spend much time clarifying points that have already been discussed [7]. Why does this “amnesia” arise and what are its effects?

Sharing and evolving informal information A survey of 23 organizations [11] found that informal documentation, communication and coordination are all more important during conceptual design than analytic methodologies. During conceptual design, a team may produce many ephemeral notes and sketches. What role do they play in the making and recording of design decisions?

Representing & analyzing concrete system properties. Exploratory design often proceeds from the concrete to the abstract, not the other way, and the recent interest in scenarios for exploring requirements and design options [13] reflects this. But do designers really use scenarios? Where, and to what ends?

In this paper, we describe a study of conceptual design in a typical industrial project, *Centauri*. We were on site for three months and conducted follow-up observations and discussions for six months after that.

We present our research methodology and findings and the implications for process and tool support.

Centauri was an ideal environment to study because of the open-endedness of the designers' task. Specifically, Centauri is a new development rather than a project to manage the evolution of an existing system; it is being developed for a speculative market rather than a specific client; and Centauri is "middleware," falling between operating system services and application software, thus making the requirements abstract and difficult to understand.

The remainder of the paper is structured as follows: in Section 2, we describe our methodology; followed in Section 3 by a summary of Centauri; then in Section 4 we present our findings and interpretations; and finally, in Section 5, we discuss their implications.

2. Methodology

Because the study was conducted longitudinally and in a commercial setting, qualitative, naturalistic research methods were a logical choice.

One of us (Catledge) was on-site for three months collecting data, performing interviews, attending meetings and keeping a general log of the activities of the project members. All documentation including formal requirements and architectures, informal notes and architecture diagrams from June '94 to March '95 were collected and logged. Changes in documentation were cataloged regularly and cross-referenced to the meeting notes.

In-depth, ethnographic interviews were conducted from November '94 until March '95. We used these interviews both to maintain contact with the issues Centauri was struggling with in the later term and to confirm our findings from the summer.

We video-taped all the meetings about requirements from June 30th to September 15th, 1994 (approximately 40 hours of meetings). We indexed the tapes with an outline of the meeting notes using Synthesis [12]. These notes were kept in a loose IBIS-like format; key issues, assumptions, and questions were indicated. These were subsequently categorized by subject matter and scope.

Discussion records, which summarized between a few seconds and a few minutes of issue-based discussion, were categorized as design, factual or process issues. Design issues were discussions prompted by questions about design strategies and approach. These can be paraphrased as "what should we do about...?" questions. Factual issues were discussions prompted by questions about the system's environment, such as "how does the OS handle...?" Process discussions, in contrast to the previous two categories, were not about the product of design but about the process of designing: They addressed project management, documentation formats, and the team's process and scope.

We used the standard procedure in qualitative research of 'triangulating,' [10] or using multiple sources of evidence to derive and validate various emerging themes. Specifically, this indicates that we analyzed the meeting notes, documentation and interview notes looking for common threads and contradictions. Al-

though we initially chose area of interest to study, these were not *a priori* hypotheses. As we collected and analyzed the data we used analytic induction and grounded theory [4] to develop more specific themes. These were then refined and changed in light of subsequent findings. Events were constantly compared to previous events so that relationships could be discovered. This also led to additional refinement and modification of the themes. The goal of this was to account for as much data as possible within our thematic framework.

Our interim findings have been read and checked by individuals in the group we studied. And some of our findings were consistent with other large, scaled studies of the requirement process, which is a kind of triangulation in itself.

3. Project Chronology

Centauri was staffed in Spring, 1994 to develop a common, extensible platform for a set of communications applications. During the early stages, team members wrote several documents to summarize the project's strategy. One of these stated the principal objective to be to "provide a small, low cost platform to target the small capacity cellular applications." This document also listed some internal customers for Centauri. The initial architecture document released in April, became the starting point for the initial requirements analysis effort.

3.1. First Team-based Organization

Our participation started in early summer when the principal teams were formed. For three months we worked closely with the Requirements Team. Other important teams were responsible for process¹ and API.

The company has a strong process quality culture, but the previously documented and adopted processes had been designed for evolutionary development of multi-version systems. Centauri was quite different: It was a first development, and it was not being developed under contract to an external customer. The Process Team therefore recommended that Centauri follow an external standard (IEEE 1074), with provision for a "conceptual exploration" phase prior to requirements specification.

During the summer, the Requirements Team did what it thought was conceptual exploration, and wrote a System Requirements Document (SRD), using the outline in the architecture document, experience, and the content of other SRDs as guides. Requirements were carefully labeled and cross-referenced. The team reviewed its progress extensively during bi-weekly meetings.

Teams were cross-staffed, and overlap between teams was considerable. They were also, for the most part, self-managed. Each team had a team leader who was responsible for guiding the group. This leader was

¹ Actually, two teams, one responsible for process, the other for development environments, but we will continue to refer to them as one.

responsible determining the activities, goals, deliverables and deadlines of the group. Project managers were responsible for collecting this information and converting it into group deadlines, but since their goals were often far-reaching, these deadlines had little affect on the day-to-day activities of the team. Unlike the vast majority of the group, the requirements team wasn't extensively cross-staffed.

3.2. The Great Process Debate

Documentation and process standardization were recurring issues. The original architecture document had many of the characteristics of an SRD although it was organized around a proposed system architecture, so the Requirements Team tried to specify the requirements very exactly and in great detail. The result, as they acknowledged, more closely met the company standard for a functional specification than it did the IEEE 1074 recommendations for conceptual exploration. This realization initiated a debate about processes and standards, although the Requirements Team kept writing and reviewing its SRD. But when the Process Team recommended a lifecycle process for Centauri, the Requirements Team was not represented.

3.3. The Architecture Debate

Function allocation was a major issue that the Requirements and API Teams started addressing separately. The teams then realized their overlap and held several discussions through e-mail and meetings. During early August, the teams met several times and wrote "This is Centauri," which defined the interfaces in detail.

3.4. Customer Involvement

In late July, the Requirements Team started looking for customer representatives and adopted a divisional technical strategy team that was responsible for present and future applications. When these experts decided that the SRD had to refer to system constraints, the Requirements Team changed direction, deciding to meet with application developers individually. From mid-August, they held eight such meetings, which they videotaped and transcribed. They wrote a second architecture document and a list of features that resembled the list in the original architecture document.

3.5. Second Team-based Organization

The Requirements Team formed four sub-teams to review technical problems raised by the feature list, in the process replacing the original team responsibilities by this more task-based approach. Each sub-team published a white paper, and these white papers were then compiled as the "final" SRD.

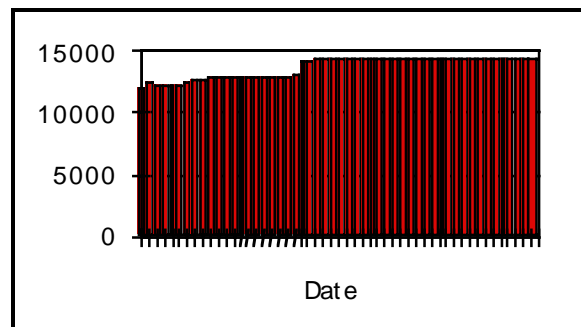


Figure 1: Evolution of SRD as a function of word count over time

4. Conceptual Design in Centauri

In this section, we present our observations. The first two subsections (on organizational working memory and convergence on a unified vision) deal with team collaboration; the second two (on concreteness of expression and architectural interfaces) deal with design.

4.1. Organizational Working Memory

4.1.1. Meetings

During the summer, when the requirements team reviewed and revised the 14,000 word SRD, they added only 2,000 words to it (see Figure 1). Many topics were discussed repeatedly without reference to or awareness of previous discussions. Most meetings reviewed short sections of the SRD, but broader issues often arose during these discussions. The review process was such that the team tended not to act on these more general issues when they revised the SRD, because they focused on short sections and fine details.

4.1.2. Documentation

During this review period very few changes were made to the SRD. Most were edits and the addition of narrative sections and illustrations. Significant changes were rare. Once the customer interviews started, the SRD became increasingly ignored and work on a unifying document was abandoned for several months.

4.1.3 White Papers

In contrast to the SRD, which contained many placeholders for information to be added later, the customer interview transcripts contained detailed accounts of how customer's systems operated and what customers expected from a platform. Unlike the SRD, the white papers that emerged were not reviewed incrementally, or controlled until they were compiled and released as a single document.

The first SRD was not referred to systematically during this second requirements process, and team members judged its contribution differently. Little text from the SRD was inherited by the white papers (providing ample opportunity to forget information), but many significant ideas reappeared in different words.

4.1.4. Conclusion: An amnesiac process

During these phases the project's process was amnesiac. Individuals took notes, but the team did not own them, and issues often re-surfaced during discussions as if they were new. The issues most often revisited concerned allocating functions and process issues.

4.2. Convergence on a Unified Vision

Not surprisingly, the issues that tended to be forgotten were also the ones that exhibited slow convergence. The Centauri team used a number of methods to achieve faster convergence once an issue was recognized as a stumbling block.

4.2.1 Responsibility for decision making

Decision making was dispersed among the teams. Management wanted the designers closest to the problems to be responsible for technical decisions. Many issues, however, were both technical and strategic; designers delayed decisions until management committed to customer projects, while management delayed those very decisions until the technical constraints were better understood.

The absence of a "superdesigner" during this phase of the project further compounded the slow convergence toward a unified vision, especially regarding the scope of the system and the nature of the interface between Centauri and its applications. Everyone recognized that the failure to allocate functionality was significantly blocking progress. It was not until the sub-teams were formed that significant progress was made. However, communication among several teams, sub-teams and management led to further delays.

4.2.2. Team Membership

Although the primary teams remained a fairly constant set of eight, sub-teams and tiger-teams were spawned on a regular basis: Over an eight month period, 11 such teams were formed to address issues and to write the white papers. Given that there were approximately 40 people working on Centauri and that most of the team leaders were each members of only one team, this total figure of 19 teams means that the average project member served on more than two teams. Management generally approved and helped to organize these efforts, viewing its role as coordination, rather than executive authority. This "organic" view of the project (as one manager called it) diverged from the company's normal procedures, because Centauri differed from most of the company's projects.

Eventually, sub-teaming became such a common method for tackling difficult issues that the Process Team even defined a process for forming sub-teams.

Effective inter-team communication made great demands on the engineers, and led to a piecemeal and reactive approach to problems that one designer said was reminiscent of "a flock of fishes." The teams organized around a particular issue, and once they had solved that one they moved on to the next en masse. They lacked a common vision of the big picture and a sense of continuity from one problem to the next.

4.2.3. Conclusion: Diffused responsibility and diluted vision

Overall, then, the project did not converge on a unified vision as quickly as the developers and management thought it would. No single person emerged as the principal architect until much later in the project. Instead, the spawning of sub-teams led to a diffusion of responsibility for the core ideas and a dilution of the resulting vision.

4.3. Concreteness and the Effects of Vagueness

4.3.1. Early commitment to detail

Centauri's history reflects an architecture-centric view and a desire to get down to engineering details as soon as possible. But detail is not the same as concreteness. Ironically, the urge to stay detailed coexisted with a persistent unease among the team that they were not getting to grips with the real problems. Despite not going back to review or rethink their architecture at a high level, team members continually questioned the direction of the project. Table 1 shows the distribution of issues raised in the meetings during this period. The data are broken into three groups: the first 10 review meetings (up to mid-August, 1994), the next six meetings, and the eight customer interviews. Process dominated the reviews. Moreover, the high-level process issues (of the type: "what are we trying to accomplish in this team?") actually increased in the review meetings at the very time that the document stopped being changed. The team's attention to detail let it ignore its concerns about direction while making apparent progress. Several months later, one member of the team blamed this growing sense that the team was addressing minutiae and not getting to grips with the bigger issues as follows: "We really didn't know what we were doing. We were inventing requirements without any idea of who the customer was."

4.3.2. Envisioning applications

Why was convergence slow? We think it was not because there was no customer (and therefore no definitive source of requirements), but because Centauri is a platform and there were no clear requirements distinguishing the applications that would use it. Team members wanted to anticipate the first applications so that they could accurately understand the requirements. They were acutely aware that a commitment from an application project to use their architecture would be shot in the arm for the project. Several potential first applications came and, and several times we were told that the "First Application" had been chosen, only for that decision to turn out to be tentative.

4.3.3. Scenarios

We expected the team to communicate ideas through concrete scenarios because descriptions of the interactions between Centauri and the proposed applications would otherwise be very general and abstract. However, scenarios were rare. Not only were the requirements stated generally but the discussions about the requirements were driven by general questions, not

Table 1: Percentage of issues and discussion topics, categorized by theme and importance.

(a) Requirements review meetings (first group)

Importance	Design issues	Factual issues	Process issues	Total percentage
High	5.01	3.17	4.75	12.93
Medium	10.82	6.33	14.78	31.93
Low	21.90	14.25	19.00	55.15
Total percentage	37.73	23.75	38.52	

(b) Requirements review meetings (second group)

Importance	Design issues	Factual issues	Process issues	Total percentage
High	14.38	8.22	11.64	34.25
Medium	9.59	6.85	32.88	49.32
Low	4.11	1.37	10.96	16.44
Total percentage	28.08	16.44	55.48	

(c) Customer interviews (contemporaneous with second group of requirements review meetings).

Importance	Design issues	Factual issues	Process issues	Total percentage
High	10.88	7.16	3.71	21.75
Medium	19.89	15.65	7.43	42.97
Low	14.85	15.92	4.51	35.28
Total percentage	45.62	38.73	15.65	

imagined usage. The scenarios that arose, were not named or referred to later. For example, during one meeting the team discussed the need for applications to set timers. The discussion included a scenario that explored issues of clock resolution and how event notification would affect the architecture. Apart from temporarily clarifying some hitherto cloudy issues, this discussion was never revisited and did not affect the SRD because the insights arrived at, being too concrete, did not seem belong in it.

4.3.4. Process improvement

The Requirements Team spent much time discussing the type of document they were writing and what the process dictated. Throughout the summer, about half the issues the team discussed were about process, documentation and team goals. This extraordinary focus on process bears the hallmarks of a displacement activity². Designing and keeping to standardized processes plays an essential role in the company's drive to reduce product defects and accelerate the engineering process. Nevertheless, on occasions the team took refuge in process discussions rather than trying to resolve issues requiring risks and commitments.

Some team members concluded that process standardization did not substitute for design progress:

"Whenever anybody tells me to read 1074 I tell them to forget it. I see a lot of people reading it and thinking they know

what they're doing. I see that as academic knowledge, which misses another key point of knowledge: experience.... We have a lot of academic domain knowledge."

It is important not to confuse this concern with a cavalier desire to dispense with planning. Centauri designers wanted process guidance; but some felt that the emphasis on process was disproportionate.

"Half of [Centauri] is working on process and procedure, which is incredible in my opinion. We do not have a good balance between product and process right now. Most engineering is based on business. I don't hear many people asking how does this fit in with our customers' needs."

Even in as process-conscious culture as Centauri, the Process Team was seen by many designers as external advisors who were unaware of the constraints of a real design process. Much of the work of this team to acquire and customize tools and to develop policies for tool use affected Centauri only after the conceptual exploration phase and for activities where careful document management and review processes were part of the company culture. The process definition effort had little effect during conceptual design.

4.3.5. Conclusion: Illusory precision

In some ways the Centauri project plunged into engineering details too soon, while in others it remained aloof from concrete commitments for too long. There is no contradiction here as long as we distinguish between concreteness and detail. On several scores, the Centauri team appeared to make quick progress toward

² This is the term used by ethologists to refer to functional behavior applied dysfunctionally, particularly in response to stressful or constraining situations (for example, an animal grooming when trapped in a headlight beam, or a computer user compulsively procrastinating by responding to e-mail).

detail, only to find that it had not developed sufficiently the goals and abstractions on which these details depended. For example, the Requirements Team submerged itself in detailed documentation before the goals for the system were really understood; and similarly the Process Team's work was detached from the activities of the other teams. Many of the design and process issues were spuriously detailed because they were not grounded in a context of use (how applications would use Centauri and how developers would use the process). These tendencies were accentuated by the need for Centauri to be generic and by the company's strong quality-conscious culture, but we suspect they are inevitable during conceptual design when the need to structure and accelerate an engineering process clashes with the need to explore system goals and options.

4.4. Architectural Interfaces

It is natural to think of platform software as the filling in a sandwich. Above are the applications; underneath are the operating system. Most of the design issues raised during the Centauri conceptual exploration phase concerned the thickness or scope of these three layers and were expressed in spatial or visual terms.

4.4.1. Architectural vision

Architecture and requirements were inextricably intertwined because the scope of the requirements affected the thickness of the filling and the nature of the interfaces on both sides. Conversely, properties of available operating systems and preexisting applications constrained the requirements. At one extreme, Centauri could be seen as a grandiose virtual machine for its applications, while at the other it could be seen as a loosely connected toolkit for application developers. Without a superdesigner who might have advocated a single approach, team members adopted widely differing views. This made it difficult for the Requirements and API Teams to coordinate their work.

4.4.2. Function allocation

Very early, it was decided that some functions fell in the Centauri "core" while others were provided in the API. Most functions could not be allocated this easily. There remained significant issues about where functions such as application-specific device handling belonged (core, API, or OS interface). Many of these issues arose from different conceptions of what diagrams meant:

"The way people look at it is different so the boundary is different. What different people mean by the application differs: they could mean [company] applications or services specific to a class of products. It's made it difficult to discuss the system because of this."

Centauri was intended to make applications extensible and portable, so it had to run on a number of systems and provide a common interface. An important issue was whether this standardization was to be ac-

complished by Centauri itself (in a multi-version OS interface, one version for each OS) or by designing Centauri to run on only one standardized virtual platform (such as POSIX). Many of the applications of the kind that Centauri would support ran on proprietary operating systems, and whether they would eventually migrate to an industry standard OS was not a question that the Requirements Team could answer. In the absence of a first application that would have forced the team to take a stand, this issue resurfaced continually.

Although we discuss these issues as if they were separate, there were times when it seemed to the designers that everything was related to everything else. For example, during one meeting the application question: "What are the applications' timing requirements likely to be?" could not be discussed without considering the requirements question: "Should applications be given an interface to set the system clock(s)?" and the architecture question "Can there be multiple system clocks?" Inevitably, these substantive issues led to a process question: "Are we making premature implementation decisions?" Such bouncing back and forth between general and detailed, factual and proposed, product and process issues was very typical.

4.4.3. Architectural reasoning

Many different looking architectures were proposed during conceptual design. These were documented as informal diagrams. But the diagrams were embedded in documents that often became obsolete before the diagrams themselves did. Most team members therefore carried the architectures around in their heads and referred to them descriptively as "the cloud diagram" or the "bubble diagram," referring to their most salient (but superficial) visual properties.

In general, team members talked about architecture spatially. The layout of diagrams really mattered, even though the diagrams had no formal meaning. Concepts of function allocation and responsibility were cast in terms of what box was above, inside, or connected to another. As one engineer said "We could not figure out if this block should be touching that box, etc."

In contrast, dynamics (such as when an architectural component was invoked, or which could be running and interacting asynchronously) played a minor role in discussions, a finding consistent with the low incidence of scenarios during the reviews.

4.4.4. Conclusion: Architecture as vision

What was lacking during conceptual design was a common vision of the way Centauri worked in its environment and how its components would work together. The need for such a common vision of Centauri's architecture, in the broadest sense, was manifested in the prevalence of box and arrow models. While they had no strict meaning, these diagrams acted as a focus for wide-ranging discussions.

5. Discussion and Implications

We now summarize the results and conclusions from this study, state the implications for practice, and relate our findings to previous research.

5.1. Summary of Results and Conclusions

We integrate our findings in terms of three general phenomena: nonmonotonic convergence, amnesic teamwork, and contextual aloofness.

5.1.1. Nonmonotonic convergence

Not only was the convergence on a common vision among Centauri team members slower than they wanted it to be, it was also nonmonotonic. Progress, or apparent progress, was punctuated regularly by a stepping back to reconsider what had been done. The project's goals and architecture were rethought several times. With hindsight, it is tempting to say that convergence was inefficient and could have been accelerated. We suspect, however, that punctuated and nonmonotonic progress is inevitable during conceptual design and should be anticipated.

The degree of nonmonotonicity that we observed, and which we suspect is typical of the exploratory nature of conceptual design, goes far beyond the mid-course corrections of incremental process models, because it may require discarding work and restarting. Ironically, too strong a commitment to the software process improvement mindset may militate against such broad rethinking of project objectives until the need for a new approach becomes undeniable.

5.1.2. Amnesic teamwork

One reason for slow and nonmonotonic convergence (but not the only one) is that issues are allowed to linger unresolved for too long. This in turn can happen because the team is unable or unwilling to commit to decisions that fall outside its area of competence, about which it lacks crucial information, or in areas where it is not empowered to act. All of these factors were at work in Centauri, and they contributed to an amnesiac set of work practices. So many threads were continually left hanging that the individual team members could not remember where they had got to, what issues depended on others, what they had decided, and what they needed to do or find out. As a team, they lacked tools and procedures that could compensate for these limitations. We hypothesize that the degree of amnesia observed in Centauri is typical and stems from fundamental memory limitations, system complexity, and the immaturity of processes to manage intellectual teamwork.

5.1.3. Aloofness from context

Perhaps the most pernicious problem was the team's failure to relate abstractions to concrete contexts. Because it lacked clear commitments from other projects, the team could not easily ground its explorations of desired features and architectural constraints in the concrete needs and details of customer applications. Because the organization usually worked on evolving systems, team members were unable to relate the new conceptual exploration standards to their experience. Again, these were not deficiencies of the team, but arose out of an inevitable conflict during conceptual design between the need to stay abstract and general

(and not to plunge prematurely into implementation) and yet to ground one's ideas in the concrete contexts in which they will be applied (and not to remain in an ivory tower).

5.2. Recommendations

Several procedures and tools could have helped Centauri and should be valuable in similar settings. However, our experience working with Centauri convinces us that the most effective interventions are likely to be modifications of current practice rather than radical new technologies.

One improvement would be to externalize the team's working memory and thereby reduce the likelihood and impact of amnesiac teamwork practices. One way to accomplish this might seem to be design rationale tools, such as gIBIS [2], or structured note-taking and multimedia indexing systems, such as Synthesis [12]. We have tried to "get designers to use" design rationale tools before, and attempted briefly to encourage the Centauri team to use Synthesis. But these tools never work as well as they should; partly because of the limitations of research prototypes, but mainly because they defer benefits [5] and force people to structure their ideas and collaborative processes artificially. A better approach to externalizing the team's working memory would be simple organizational and meeting management practices. Simply having one team member be scribe and having a period at the end of every meeting to summarize what had been decided, what issues were raised and left pending, who was responsible for finding out information, and whose advice was to be sought would be promising first steps.

Nonmonotonicity of progress was first reported by Guindon [6] in her study of the verbal protocols of experienced software designers solving realistic problems. She observed that constant oscillation between abstraction and detail was not only common but useful: It was frequently only after thinking about a relevant detail (including implementation possibilities) that a designer had a general insight at the abstract level. According to traditional measures of design progress, in which design is a linear refinement from general to concrete, these designers were making slow progress. Really, however, they were coming to understand the problem better through their nonmonotonic process.

We believe that the slow and nonmonotonic convergence on a project vision is best accelerated through organizational change. A project like Centauri should have a superdesigner or small core team of very experienced people. The knowledge that is needed, and was lacking in Centauri, spans several areas, including the application domain and knowledge of architectures of similar or related systems. Bright, professional people are not enough.

Understanding the customer is also important. As Keil and Carmel [8] show, there is no substitute for direct access; intermediaries and analysts may introduce noise. Centauri's potential customers were future application projects, and the team had no reliable way to gather application expertise. They also did not relate

requirements to contexts of use. Having application experts on the team is one way to address this issue.

A complementary strategy would be to organize the acquisition of requirements and application expertise around concrete scenarios [13]. By concentrating on the SRD's illusory precision, the team became unable to step back and ask questions about the requirements such as: "Why is this required, and how would it work in practice?" Being able to refer to standard scenarios might have made them more incisive.

Our final recommendation, that process improvement strategies be modified for conceptual design only with caution, is probably the most controversial. Although product quality can undoubtedly be improved by standardizing processes later in the lifecycle, most process improvements achieve this by standardizing form over content. For conceptual design, however, where systematization has been less successful, this substitution of form for content can lead to counterproductive measures of progress and heavy-handed management. One of the sources of difficulty in Centauri was a self-imposed desire among team members to make progress as measured by the review of the original, prematurely detailed, SRD. Yet, the project teams were self-managed, and the Requirements Team was far closer to the content of the system than project management would have been. We predict that successful process improvement efforts for conceptual design will not closely resemble those for later design processes, but will incorporate lightweight and self-managed processes that emphasize production, dissemination, and timely destruction of the interim results of work in progress.

Based on our observations, we predict that the place to look for these ideas may not be the software quality literature so much as the popular literature on creative design. And the most useful technologies will probably not be requirements documentation and modeling systems, but simple whiteboards and post-it notes.

Acknowledgements

This project was supported by Motorola Cellular Infrastructure Group. It would not have been possible without the openness and wholehearted support of the Centauri designers and management, and especially the members of the Requirements Team.

We also wish to thank Laurette Bradley and the anonymous reviewers for their recommendations on earlier versions of the paper.

References

- [1] Boehm, Barry W. "A Spiral Model of Software Development and Enhancement", IEEE Software, May 1989.
- [2] Conklin, J. & Begeman, M. gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Trans. Office Inf. Sys.* 6(4): 303-331, 1988.
- [3] Curtis, B., H. Krasner and N. Iscoe. A Field Study of the Software Design Process for Large Teams. *Communications of the ACM* 31(11): 1268-1287. 1988.
- [4] Glaser, B.G. and Strauss, A.L., *The discovery of grounded theory: Strategies for qualitative research*, Chicago: Aldine, 1967.
- [5] Grudin, J. Why Groupware Applications Fail: Problems in Design and Evaluation. *Office: Technology and People*, 4(3): 245-264, 1994.
- [6] Guindon, R., *Designing the Design Process: Exploiting Opportunistic Thoughts*. *Human-Computer Interaction*, 5: 305-344, 1990.
- [7] Herbsleb, J. Object-Oriented Analysis and Design in Software Project Teams. *Human-Computer Interaction*, 10, 1995.
- [8] Keil, M. and Carmel, E. Customer-Developer Links in Software Development. *Communications ACM* 38(5): 33-44, 1995.
- [9] Kuwana E. and Herbsleb, J. Representing Knowledge in Requirements Engineering: An Empirical Study of what Software Engineers Need to Know, *IEEE Software*, 1992.
- [10] Lincoln and Guba, *Naturalistic Inquiry* Sage Publications: Beverly Hills, CA: 1985.
- [11] Lubars, M., Potts C. & Richter, R. A Review of the State of the Art in Requirements Modeling. *Proc. RE'93: Int. Symp. Requirements Engineering*, IEEE Computer Society Press. 1993.
- [12] Potts, C., Badre, A. & Bolter, J.D. *Synthesis: A Computer System for the Rich Recording and Indexing of Collaborative Ideas*. Georgia Tech. GVU Technical Report. 1993.
- [13] Potts, C. Invented Requirements and Imagined Customers: Requirements Engineering for Off-the-Shelf Software. *Proc. 2nd IEEE International Symposium on Requirements Engineering*, IEEE Computer Society Press, 1995. p. 128.
- [14] Waltz, E., Elam, J. & Curtis, B. Inside a Software Design Team: Knowledge Acquisition, Sharing, and Integration. *Comm. ACM*, 36(10): 1994.