Collaborative Conceptual Design: A Large Software Project Case Study

Colin Potts[1]

Lara Catledge[2]

Georgia Institute of Technology,

College of Computing, Atlanta, GA 30332-0280, USA

{[1]potts, [2]lara}@cc.gatech.edu

Abstract: During software development, the activities of requirements analysis, functional specification, and architectural design all require a team of developers to converge on a common vision of what they are developing. There have been remarkably few studies of conceptual design during real projects. In this paper, we describe a detailed field study of a large industrial software project. We observed the development team's conceptual design activities for three months with follow-up observations and discussions over the following eight months. In this paper, we emphasize the organization of the project and how patterns of collaboration affected the team's convergence on a common vision. Three observations stand out: First, convergence on a common vision was not only painfully slow but was punctuated by several reorientations of direction; second, the design process seemed to be inherently forgetful, involving repeated resurfacing of previously discussed issues; finally, a conflict of values persisted between team members responsible for system development and those responsible for overseeing the development process. These findings have clear implications for collaborative support tools and process interventions.

Keywords: Collaboration, Software Process, Conceputal Design

Industrial software design is always a collaborative activity. Although any collaborative design activity involves communication and coordination, the design of software has an additional complicating factor: The artifact being designed is a description, not a tangible object. Application designers can envisage software, which has a clear real-world function and external users, by writing prototypes and storyboards. But system software and "middleware" (types of support software that provide services for other programs) are inevitably more abstract and difficult to describe or portray.

Brooks (1986) has argued that the conceptual complexity inherent in software has most effect during the conceptual design activities early in the development process. It is therefore important for designers to communicate design proposals and issues effectively during these phases of development. Misconceptions that slip through to the implementation are known to be very expensive to correct (Boehm, 1983).

If we take Brooks's analysis of software design seriously, we must understand what design teams habitually do during conceptual design and what they are capable of, rather than basing our interventions on utopian, depersonalized theories of the "software process." However, surprisingly little is known about how conceptual design occurs in real projects, which means that it is difficult to predict the value of proposed process interventions or design tools.

We studied the conceptual design activities of a representative middleware development project, *Centauri*[i], for nearly one year . We observed the development team's conceptual design activities for three months, with follow-up observations and discussions over the following eight months and informal interactions for several months after that. Our goal was to understand how patterns of collaboration and communication in the project team affected its convergence toward a common vision and a documented architectural specification. In this paper, we present our research methodology and findings, and the implications of the results for process and tool support. In particular, we discuss the effects that organizational structure and between-team and within-team communication practices have on design convergence as assessed by an analysis of how Centauri documents evolved and how ideas were expressed during design meetings.

We start in Section 1 with a brief summary of the published literature about collaboration during conceptual design, concentrating on software projects, and the issues that the existing research leaves open. We then explain and justify our research methodology in Section 2. In Section 3, we briefly narrate the history of the Centauri project. The next four sections discuss our observations in terms of four significant themes: the nonmonotonic pattern of convergence among team members on a common vision (Section 4); the team's "working memory," what it forgot, and the effects of that forgetting (Section 5); key differences between the normative process defined for the project and the process that we observed taking place (Section 6); and, the aloofness of the project team's deliberations from the concrete contexts of use envisaged for the system (Section 7). In Section 8, we summarize some of the dysfunctional activities we observed, all the result of well-meaning but inappropriate goals. Finally, in Section 9, we summarize our findings and discuss their implications for CSCW design technology and collaborative process interventions.

1. Collaboration in software design

There are a variety of software engineering standards and prescribed processes that are designed to support teams engaged in software development. Most emphasize document traceability and deliverables. Two of them, IEEE's Standard 1074 and Software Engineering Institute's Capability Maturity Model (CMM), are particularly relevant in this setting. IEEE 1074 describes the activities of a team during conceptual exploration, the initial phase of the development lifecycle. The CMM is a more holistic approch, descibing various levels of organizational "maturity" on the basis of a team's ability to handle change. In addition, the corporate organization had developed it's own processes for conceptual exploration, combining the IEEE 1074 and SEI's CMM. None of these standards describe the kinds of collaboration required to produce these documents and deliverables; They describe the what, but not the how.

There are many types of design collaboration. For the sake of simplicity, we distinguish three forms: collaboration within a team (over weeks or months), collaboration within single meetings, and collaboration between design teams or between a design team and customer representatives.

1.1 Within-team collaboration

Some previous research has studied real projects. Others have emphasized some aspects of design, such as design rationale or sensemaking, or have emphasized the related but highly relevant subject of collaborative writing.

1.1.1 *Exploratory studies*

Exploratory studies of the conceptual design process have uncovered a number of themes that reoccur across organizations. Curtis et. al. (1988) interviewed key personnel in 19 organizations and found that application and technology domain knowledge is spread very thin in most projects. These writers concluded that a significant success factor for large projects is the presence of a "superdesigner," a person who is responsible for holding and supporting the project vision. In a later observational

study of a research and development project (an object server), Walz et. al. (1993) confirmed that the integration of knowledge and the depth of domain knowledge was critical to the success of early stages in a project. Lubars et al., (1993) surveyed 23 organizations, emphasizing requirements analysis exclusively. They found that informal documentation, communication and coordination are all more important during conceptual design than conventional notational and analytic methodologies.

Informal coordination mechanisms are vital in all types of software design. Formal methods for coordination, including impersonal (documentation) and interpersonal (meeting) methods, tend to be used more in large projects than in small projects and in projects that have finished their requirements gathering activities (Kraut & Streeter, 1995). In that study, however, developers reported that other people were the most valuable source of information when they needed help or advice. Interestingly, projects in which the developers had frequent technical contacts with people outside of their immediate project were associated with better project outcomes (customer satisfaction and staff perceptions of project success). This was especially the case in projects with uncertain requirements.

1.1.2 *Organizational memory and shared design knowledge*

Given that knowledge acquisition and dissemination is so important during design, there has been a lot of work in representing design knowledge. Much of it has been technology-driven and has emphasized the expressiveness of representations. For example, Lee (1991) proposes a design rationale representation that overcomes the expressive limitations of a representation (Potts and Bruns, 1988) that is, in turn, an extension of IBIS (Kunz & Rittel 1970). The inappropriateness of this representation-focused approach for supporting design teams is illustrated by the early stages of one project (Terveen et al. 1995) that sought to apply an AI-based classification scheme for a small area of middleware design knowledge: the design of fault-tolerant behaviors in a communications system. A design knowledge prototype was too structured for the designers to use and was ultimately watered down into one with far less expressive power. The deployed tool was "owned" by one of the members of the development group, not by the researchers and therefore was easier to diffuse into the design organization.

Other studies have started from a more ecologically valid viewpoint but have still run into the difficulties of structuring knowledge so that users can access it. For example, users of TeamInfo (Berlin et al. 1993) differ in how they classify information, and because most contributions must be classified in more than one way, a simple classification scheme was chosen with fewer than ten categories, but even this has led to disagreements.

All of this work deals with recurrent information, the assumption being that team members will be willing to bear the cost of encoding it for future use by others in other projects. Episodic organizational memory support systems are less ambitious in scope, addressing instead the recording of the rationale for design decisions on a given project. Usually, these systems involve less rich representations and therefore less encoding effort on the part of the designer. For example, less structure is apparent in the NYNEX Design Intent System (Atwood et al. 1995), a World-Wide Web community memory for two telephony projects. However, the absence of a standard structure means that it is not easy for users to find the information that they need and there is no way to prevent redundancy.

1.1.3 *Making sense of background information*

Another key idea in these studies is that much conceptual design, like much management planning, involves *sensemaking* (Russel et al. 1993). In a summary of four projects to write reports about research and new technologies, these authors state that data extraction is the most expensive constituent activity of the report-writing process. They argue that group idea-generation tools substantially facilitate the consideration of alternative organizations for the material and also improve the quality of the analysis by letting the team members consider more alternative organizations than they otherwise could.

1.2 Collaboration in the here and now

Kuwana and Herbsleb (1993) have illuminated the types of questions and topics that most often arise during conceptual design meetings. They found that most of the questions that designers asked related to how a feature would work in practice and rarely why it had been specified as a requirement.

Even during the design phase, about half the questions were still about requirements. *What* questions (questions about external behavior or properties) were the most common. Interestingly, designers asked “what” questions *more* over time and “how” questions less, although naive process models of software development (in which “what” issues precede and are replaced by “how” issues) predict just the opposite. The authors do not make much of this apparent paradox and explain it in terms of the increasing frequency of questions about scenarios of use. However, they do emphasize the very low incidence of *why* questions (only four percent of the requirements questions and two percent of the design questions). About half of the questions raised were about more than one target. This low incidence, they argue, shows that design and requirements rationale capture tools are unlikely to help designers answer the questions that they most often raise.

1.3 Collaboration among teams and with customers

Conceptual design includes a large component of requirements gathering and analysis. Organizations use a variety of strategies for understanding requirements and communicating with customers. Because customers are not always members of the same organization, developer/customer interaction has a different flavor from collaboration within the design team.

Keil and Carmel (1995) surveyed managers in custom and package development organizations about their recollections of successful and unsuccessful projects. They found that more than one connection with the customer is essential, but that too many leads to marginal gains in quality. Methods for achieving this relationship with the customer included using facilitated teams, prototypes, interviews, support lines, user groups, etc. They also found that indirect communication through intermediaries or surrogates for the customer, such as professional systems analysts or marketing staff, are often misleading, even though they are usually associated with purported expertise and are widely used. Finally, they found that custom and package development projects tend to use different customer connections and that each could profitably adapt methods from the other. For example, custom projects use facilitated teams frequently but support lines hardly ever, whereas the opposite is true for package projects.

1.4 Research objectives

In contrast to most of these studies, we aimed to conduct a longitudinal study of conceptual design as it happened in a large, commercial project. Here are some of the questions that we set out to answer:

- *Convergence on a common technical vision*. As Waltz et al (1993) showed in their longitudinal study of a research project, knowledge acquisition plays a major role in open-ended design, especially in the absence of a "superdesigner." But how does a large team converge on a common vision when working under commercial pressure? Is the process monotonic and logical, or punctuated with conflict? Do design teams organize their ideas in depth before writing requirements documents, or do they commit early to one organization? What are the consequences of their choices?

- *Organizational working memory*. Effective teams and organizations learn from their experiences, but what of the "working memory" of the organization? Do team members remember key design considerations while work is in progress? What happens when they forget what they have decided? Do they question the rationale for requirements?

- *Sharing and evolving informal information.* 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 as much as from the abstract to the concrete. In particular, many researchers have become interested in using scenarios to help define requirements and explore design options (Potts, 1995). Do designers really use them, and if not, why not? And what are the consequences of not using them?

Centauri was an ideal environment for us 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 more abstract and difficult to understand than for most applications.

2. Methodology

We studied the Centauri conceptual design process for three months with follow-up observations and discussions over the following eight months. Because most of the data we would gather would be qualitative, we "triangulated" on key issues by gathering different kinds of qualitative and quantitative data and correlating it where possible. We identified a number of likely themes (see section 1.4) to direct our research. These were not taken to be hypotheses, merely guides to our study. We now describe our data gathering procedures.

2.1 Historiography

To get an overall picture of Centauri, it is important to understand the project's history (see Section 3). We wrote this history from interviews with project members, informal conversations, access to documentation, and our experience of participant observation on the project.

2.2 Meeting analysis

We taped all the meetings about requirements from June 30th to September 15th, 1994 (approximately 40 hours). We indexed the tapes with an outline of the meeting notes using Synthesis (Potts et al. 1993) and kept notes in a loose IBIS-like format. Key issues, assumptions, and questions were indicated. These were subsequently categorized by subject matter and scope. Subject matter categories included factual issues about existing systems, design issues about the proposed architecture and requirements, and issues about the Centauri development process. Scope categories were "high" (or broad), "medium," and "low" (or narrow). All categorization decisions were subjective and were based on our knowledge of the issues gathered through participant observation. It was impossible to have independent judges rate issues, because of the commercial sensitivity of the subject matter.

2.3 Documentation analysis

We collected and logged all requirements and architecture documents that the team produced between June 1994 and March 1995. We cataloged these documents regularly and cross-referenced their contents with our meeting notes. In addition, we had access to documents produced before and after this period, although we did not analyze them as thoroughly.

2.4 Developer interviews

We conducted in-depth interviews from November 1994 until March 1995. We used these interviews both to maintain contact with Centauri and to validate our observational findings from the summer months.

2.5 Participant observation

We were involved and participated in many of the meetings. One of us (Catledge) was on-site for three months as a participant observer.

3. Project chronology

3.1 Prehistory

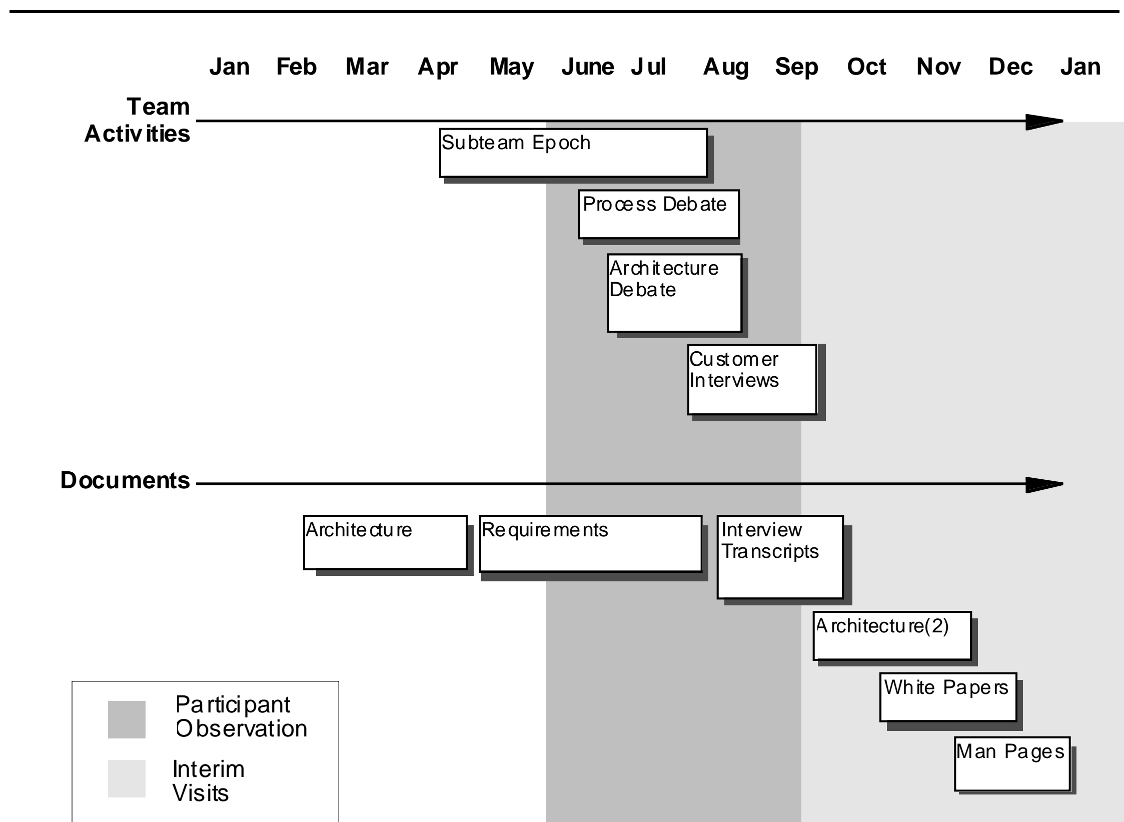
It is difficult to define the beginning of any project.. The roots of Centauri go back several years before the project was officially initiated. Senior technical staff in the company had argued for a generic middleware platform for some time, but it was difficult to justify the investment. An earlier attempt to develop a Centauri-like system had ended in that project developing a specific system for a customer.

Centauri was initially staffed in Spring, 1994 (see Figure 1) to develop a common core platform that would be portable, scalable, and extensible for a set of applications. At this stage, Centauri management was thinking in terms of a rapid development cycle of about one year. No one seems to have explicitly weighted the project objectives, but as time went on, Centauri took on greater potential strategic significance to the company, and cycle time increasingly took second place behind the goals of portability and generality.

During the Spring, three subteams were established: the Reuse Subteam, which was responsible for developing a software reuse strategy; a Product Requirements Subteam, which was responsible for defining the market and gathering requirements from potential customers; and an Architecture Subteam. The Product Requirements Subteam consisted of management from a collection of projects and was largely inactive. Defining requirements became the responsibility of the Architecture Subteam.

The Architecture Subteam wrote several documents to outline the basic strategy of the Centauri project. One of these, the *Centauri Project Strategy and Concept Exploration Requirements*, listed the objectives of the project and named several applications that could be reimplemented using Centauri. The next, and most important, document that the Architecture Subteam produced was the initial architecture document. This was finished in early April, 1994.

Figure 1: Project Chronology



3.2 The subteam epoch

Our participation in Centauri dates from June, 1994 when the architecture document had already gained widespread acceptance. About the same time, Centauri experienced an influx of about 40 engineers from a canceled project. Several new subteams were established. These included a Development Environments Subteam and a Process Subteam (which we will consider to be one subteam in the rest of this paper as their goals overlapped and the Process Subteam proper consisted of only one person), an expanded Reuse Subteam, a Core Platform Development Subteam and an API (Application Programmer Interface) Subteam. The Architecture Subteam was disbanded, but most of its members joined the newly formed Requirements Subteam.

Most of these subteams had little to do until the requirements were more fully defined. For example, the Reuse Subteam could define a software scavenging strategy and the API Subteam could define the way in which the API would be specified, but neither subteam could make much substantive progress until they knew what they were scavenging and what services they were specifying. We did not participate in these subteams directly, but team members told us that new and inexperienced engineers spent a lot of time attending courses on technologies that were new to them (especially operating systems and programming languages) rather than contributing directly to the project. This is consistent with the original staffing plan, which did not anticipate such a sudden growth in numbers.

During the summer, the membership of these subteams was quite stable. Subteams were set up to have overlapping membership. This cross-staffing was designed to facilitate inter-team communication. The Requirements and Process Subteams were more insular, however.

The Requirements Subteam produced a System Requirements Document (SRD). Beginning with the Architecture Document outline, team members wrote labeled requirements using their experience and knowledge of other SRDs. The subteam reviewed their requirements extensively in bi-weekly meetings.

3.3 Rumblings of change: The great process and architecture debates

As the summer drew on, documentation and process standardization became major issues for the requirements team. At one point, they suggested that parts of the SRD were complete enough for development teams to start prototype implementation. However, the Process Subteam and the Requirements Subteam concluded that the SRD was really a (too detailed) functional specification. This started a long debate about lifecycles and standards. Finally the Process Subteam recommended a lifecycle model for Centauri and mapped it onto the existing divisional process. Members of the Requirements Subteam were not represented in this discussion. Until a decision was reached, they kept writing and reviewing the SRD.

Function allocation also became a major issue. The Requirements Subteam discovered informally that the API Subteam was struggling with overlapping issues. This resulted in a series of discussions between the two subteams in e-mail and face-to-face meetings. During early August, the teams met occasionally several times and wrote "This is Centauri," a document that described the interfaces of architectural components in detail.

The most significant source of dissatisfaction among the Requirements Subteam, however, was the lack of direct customer contact. In late July, they adopted a divisional technical strategy team as their customers. These experts were surprised that the SRD did not emphasize implementation constraints. The Requirements Subteam immediately changed direction, and met with potential customers (application developers) individually. From mid-August until early September, they held eight such meetings, which they videotaped and transcribed. They went on to write another system architecture document, together with a wish list of features that resembled closely the list in the original architecture document.

3.4 The white papers epoch

In contrast to the SRD, which contained many placeholders for information to be added later, the customer interviews provided detailed accounts of how customers' systems operated and what

customers expected from a core platform. The interviews helped the Requirements Subteam to start answering the questions that had plagued them in the early part of the summer. The Requirements Subteam spun off four subcommittees to review specific technical issues. Each subcommittee was to publish a white paper "to facilitate a level of fundamental agreement on the requirements and functions of the core." When the white papers were compiled, they were expected to provide a final SRD, but this compilation had no direct traceability back to the SRD over which the Requirements Subteam had labored throughout the summer.

Centauri's team structure was virtually suspended in favor of this more task-driven approach. Whereas the SRD was reviewed incrementally and had been written in accordance with defined processes, the White Papers were not reviewed or controlled outside of the sub-teams until they were complete and compiled.

3.5 Recent history and current affairs

After the white papers were compiled, the API Subteam could finally specify the API, which it did in a collection of Unix-like manual pages. There was no clear traceability back from the manual pages to either the SRD or the white paper compilation. We learned later that another requirements document had been placed under configuration control in May, 1995, and had undergone three revisions by June. It had a different set of authors from the SRD and white papers, and we do not know what its relationship is to those documents.

3.6 Technology use by the project

Centauri used basic office technology to support conceptual design. Meetings were held at single sites with no provision (or need) for teleconferencing or videoconferencing. Notes were taken, if at all, on paper. Whiteboards in the meeting rooms were used to sketch designs and list issues and commitments. Whiteboards, however, served as external memories only during meetings, not between them. There were two reasons for this: Meeting rooms were in short supply, and so the whiteboard itself was a scarce resource; and, secondly, the company's security staff regularly checked that

company-confidential information was not on display outside normal working hours in unlocked rooms, and were empowered to notify senior management of violations. Anything written on a whiteboard was fair game.

Outside of meetings, documents were prepared using two incompatible document processors. Every designer seemed to prefer one or the other with almost religious fervor. Design issues were communicated by e-mail. Later in the project, a requirements traceability tool was adopted, but that tool requires that requirements documents be segmented into individually numbered and tracked requirements and is therefore suitable only for definitive documentation that has stopped evolving rapidly. Use of this tool during the early stages would probably have demanded too great an administrative overhead and would have encouraged premature commitment.

Although the Process Subteam evaluated several object-oriented analysis methods and support tools, these were not used during our involvement in Centauri, and were eventually only used by a parallel “shadow” project to analyze part of the architecture that was also being analyzed using traditional methods. No formal analysis method or tools were used.

4. Nonmonotonic convergence on a common vision

4.1 Convergence and project-historical epochs

The successive “epochs” of the conceptual design activity described in Section 3 indicate that the project team’s convergence toward a common vision was anything but monotonic. The process seems to have consisted of an oscillation between divergent, exploratory activities, and convergent documentation and review activities. During the first two cycles, the outcome of the convergent activities, while contributing to the background understanding of the team members, did not lead to substantive progress toward a common vision. Instead, the project dropped what it was doing each time and took a new tack. The artifacts shared a lot of common ideas, but they were still largely rewritten each time.

The retrenchments that punctuated the convergence on both occasions were caused by outside factors. The first epoch ended when external reviewers of the SRD commented that the emerging Centauri architecture would have to be redesigned to accommodate performance requirements that would surely be imposed by applications. Until then the Requirements Subteam had paid very little attention to performance requirements, largely because performance requirements would be application-specific and were therefore difficult to predict. By ignoring implementation constraints, the Requirements Subteam had successfully abstracted their requirements away from any specific application. In the process of standing back, however, the team had gone too far and had failed to consider the implications of certain requirements on the design architecture.

The second retrenchment occurred after our regular contact with the project ceased, but it seems to have been triggered by the arrival of a senior architect from another company.

4.2 Convergence and the primary design artifacts

Centauri’s history reflects an architecture-centric view and a desire to get down to engineering details as soon as possible. The project proper therefore started with an architecture proposal, not a

conceptual overview of the requirements that Centauri would meet. In an end-user application, this rush to architectural decisions before the requirements were clear might have been premature. The Centauri designers, in contrast, were developing a software infrastructure whose justification rested on the architectural benefits for future applications. The SRD was formed by taking the organization of and text from the original architecture document. By the time we started our observations, the SRD was already fairly stable.

The SRD was only modified slightly during the summer (see Figure 2). We classified the changes as minor edits, additions or deletions to labelled requirements, addition or deletion of narrative segments and examples, and organizational changes. The results indicate that there were just about as many changes made to narrative segments as there were to the labelled requirements, and most of the substantial changes reside in narrative segments (see Table I). This seems to indicate that the majority of the content in the SRD existed in the narrative segments, not in the labeled requirements.

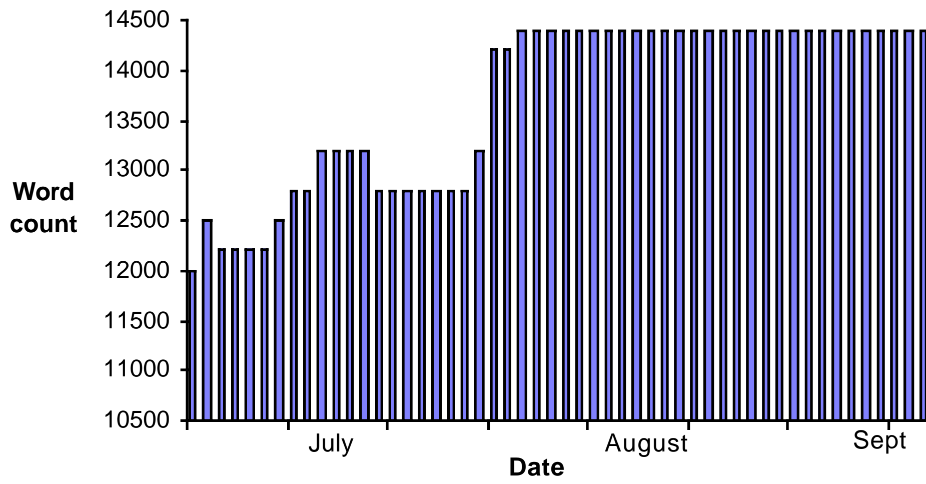


Figure 2: Size of SRD during the Summer.

Table I: Changes Made to SRD by Category

	Minor Edit	*Requirement*	*Narrative*	*Organization*	*Total*
Week 1 (7/4-7/8)	1	0	0	0	1
Week 2.(7/11-7/15)	5	4	5	1	15
Week 3 (7/18-7/22)	0	2	1	2	5
Week 4 (7/25-7/29)	0	3	4	4	11
Total	7	9	10	7	32

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 reconceptualize their architecture at a high level, team members continually questioned the direction of the project. Table II 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. The issues are also classified into three types: design issues (those that addressed the requirements for and design of the system itself), factual issues (those that addressed questions of fact about the environment in which the system would work or the platforms upon which it was planned to build it), and process issues (those that addressed the design process and the plans and objectives of the requirements team). Finally, we have divided the issues into three importance categories according to our judgment of the whether the significance to the project was high, medium or low.

Process issues dominated the review meetings. Moreover, the high-level process issues (of the type: “what are we trying to accomplish in this team?”) actually increased over time.

Table II: 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	

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."

In parallel with the formal documentation, the project produced a lot of informal documents, especially architecture diagrams. In general, team members tended to talk about the architecture in spatial terms. The layout of diagrams mattered, even though they had no formal semantics. Concepts of function allocation and responsibility were cast in terms of what box was on top in a diagram, what boxes were inside others, and what boxes were connected. As one engineer said "We could not figure out if this block should be touching that box, etc." Unfortunately, the diagrams were often embedded in documents, such as the architecture document or working papers, that became obsolete before the diagrams themselves did. Most team members therefore carried the architectures around in their heads and referred to them descriptively by such terms as "the cloud diagram" or the "bubble diagram," referring to the most salient (but superficial) visual properties of the diagrams.

4.3 Convergence and team collaboration

If we look at the discussions of the Requirements Subteam more carefully, we find two central issues that led to the first retrenchment. We refer to these as the *What goes where?* and *Who Is the Customer?* issues. The *What goes where?* issue addressed not only the allocation of functions to Centauri architectural components, but also whether a given function would be performed by Centauri at all, or assumed to be performed by the underlying operating system or the applications that would reside on Centauri. The *Who Is the Customer?* issue took two forms: the identity of the first few applications, and the identity of the customer representatives to whom the subteam members thought they should be listening.

The responsibility for resolving these issues was dispersed among the subteams and project management. The criticality of these issues was universally acknowledged, but because there was no single forum in which they could be highlighted and no single locus of responsibility for them, they were addressed in a piecemeal and partly inconsistent manner. Management wanted the designers closest to the problems to be responsible for technical decisions. Many issues, however, were neither technical nor strategic, but somewhere in between; subteam members often balked while waiting for management to commit to decisions (especially liaisons with application projects), while management waited for technical information (especially function allocation feasibility assessments) that they felt they needed to make such commitments.

In addition, the matrix organization led to parallel attacks on same problems. Cross-staffing among teams reduced this redundancy of effort to some extent, but teams still worked on the same problems without knowledge of the other's activities. The emphasis on completed products, rather than on informal documentation, tended further to isolate teams since documents were not controlled until reviewed; informal notes and annotations were seldom shared.

The *What goes where?* issue is interesting in this regard. It is natural to think of middleware as the filling in a sandwich. Above are the applications; underneath are the platform and 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. There remained significant issues about where functions such as application-specific device handling belonged. Many of these issues arose from different conceptions of what architecture diagrams denoted. One team member noted that:

> 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 promote extensibility and portability, so it had to run on a number of platforms and provide a common interface. One aspect of the *What goes where?* issue was whether this

standardization was to be accomplished by Centauri itself (in a multi-version OS interface, one version for each platform type) 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 a question that the Requirements Subteam could not answer. In the absence of a first application that would have forced the team to take a stand one way or the other, this issue resurfaced continually.

In response to their lack of progress, several subteams joined forces as a "tiger team," with the goal that their recommendations would be elevated to a strategic level. It was not until the tiger team was formed that significant progress was made. However, the process involved much thrashing and loss of time between organizational levels while the decisionmaking responsibility migrated from the subteams, to the tiger team, to upper management, and finally back down to subteams again.

Although the primary subteams remained a fairly constant set of eight, task-related tiger teams were spawned on a regular basis: Over an eight month period, 11 such teams were formed to address issues in the process arena and to write the white papers. Given that there were approximately 40 people working on Centauri at any one time, 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. Management generally approved and helped 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, but Centauri differed from most of the company's projects. One of the managers commented:

> We normally develop products not cores, so there's a paradigm shift there. Also, we were trying to organize informally. So we have these teams that come and go, and we're letting this happen. [We] had to grow it like a plant. You can't rush it. Water it and let it grow.

Effective inter-team communication made great demands on the engineers, and led to a 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.

Eventually, sub-teaming became such a common method for tackling difficult issues that a process was written on how to be a sub-team.

The absence of a “superdesigner” during this phase of the project undoubtedly 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. What was lacking during the conceptual design of Centauri was a common vision of the way the system worked in its environment and how its components would work together.

4.4 The dynamics of convergence

Convergence toward a common vision occurred, then, through two kinds of process: slow, gradual convergence on a set of common ideas, during which the team concentrated primarily on revising a document that expressed their design commitments; and occasional, discontinuous reevaluations of direction, partly determined by outside forces. The dynamics of the process depended on the balance between two opposing tendencies: the need to make demonstrable progress in the short term, and the need to take account of design information and knowledge that would lead to a higher-quality system in the long term. The goal to make progress encourages continuance of whatever design paths to which the team has committed. A vicious cycle then takes hold: the more work done along a given path, the less the team wants to abandon it. However, the countervailing tendency to take account of design knowledge, and especially to take heed of warnings, may trigger a reevaluation; As the team makes progress, the opportunity for more precise self-evaluation arises.

Our depiction of the convergence process as a nonmonotonic form of progress in which gradualistic activities were punctuated by occasional dramatic retrenchments invites obvious comparisons with Kuhn’s (1970) analysis of scientific revolutions. We should not make too much of what could be only a weak analogy. However, one important parallel is that just as Kuhn drew attention to the important

social cohesiveness of "normal science" and the correspondingly threatening nature of paradigm shifts, so we must acknowledge the value of the review-and-modify process that occurred through most of the summer. It met management's and the team's expectation of demonstrable progress, and gave the team a strong direction. Until the retrenchment occurred, the only regularly exercised alternative to making this gradualistic progress was the increasingly frequent sounding off that occurred during meetings concerning project direction, the allocation of functionality, and the identity (and existence) of Centauri's customers. In the absence of constructive alternatives, dwelling on such issues seemed to be negative and defeatist.

5. Team working memory and forgetting

We turn now to a lower-level analysis of how the team made progress, concentrating again on the activities of the Requirements Subteam during the summer of 1994.

5.1 Progress as a cycle of inquiry

A major determinant to making progress in any activity is the ability to retain context over time. In the psychology of human skilled performance, this faculty is referred to as *working memory*. Teams have a similar need for working memory mechanisms (e.g., meeting minutes, interim documents, the recollections of participants) to retain context during the performance of collaborative tasks. How did the Requirements Subteam retain context during its work?

To analyze the subteam's working memory mechanism and how well it worked, we use the Inquiry Cycle model (Potts & Takahashi 1993; Potts et al. 1994), in which the conceptual design process is viewed as the repeated iteration of a basic cycle of three activities: the expression of design ideas (e.g. in written documentation), the discussion of these designs (e.g. in informal review meetings), and the refinement of the expressed designs as a result of decisions reached during the discussions. This is portrayed in Figure 3.

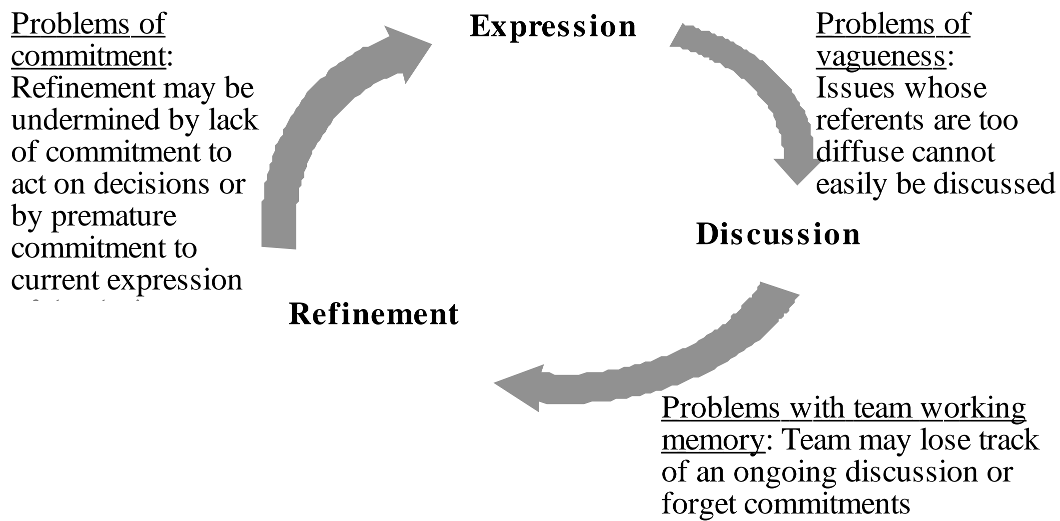


Figure 3: The Inquiry Cycle. Conceptual design is viewed as three intertwined activities: expression, discussion and refinement. The link between each activity and its successor may be blocked by three factors: problems with vagueness, problems with the team's working memory, and problems of commitment.

For the Requirements Subteam, the SRD provided a shared expression of design ideas. Sections were assigned to individual subteam members who brought them back to the subteam for periodic review in their bi-weekly two-hour meetings. Subteam members marked up their copies of the SRD with comments that they brought to the attention to other subteam members. There was some e-mail interaction, and subteam members often visited each other's cubicles to discuss the SRD. Refinements were made offline by the person responsible for the section in question.

5.2 Meeting notes

The team kept no meeting notes for review or distribution among themselves. Although the Requirements Subteam was aware that its working memory leaked badly, they did not realize how often many issues came up and how much time they spent in meetings discussing them. They took no measures to alleviate this situation. Early on, they seriously considered using Synthesis, the same video-annotating note taking tool we were using (see section 2.2), but they did not want to rely on a research prototype[ii]. Later, we proposed some simple strategies for keeping joint meeting notes and suggested devoting the closing minutes of each meeting to summarize decisions reached during the meeting and to list the outstanding issues that had arisen and which required further investigation. One subteam member wrote a memo that defined a design rationale recording process, but the subteam believed that management approval would be necessary to implement it. It is not clear to us whether management approval was sought, and, if it was, why it was not forthcoming. Months later, the subteam was still not taking any sort of meeting notes.

5.3 Artifacts as external memories

The Requirements Subteam used the SRD itself as an external working memory by discussing and quickly refining it. However, all three of the problems shown in Figure 3, occurred here. First, many diffuse issues were raised that were not easy to associate with fragments of text in the SRD (see Table II).

Numerous topics were repeatedly discussed without reference to or awareness of previous discussions. For instance, on June 30, the issue of backwards compatibility with existing systems came up, but at the next meeting five days later there was no mention of this issue, even though the topic (allocation of functions to the core and API) was closely related to backwards compatibility. Backwards compatibility came up again in the July 12 meeting during a discussion about the possibility of designing a platform that would support more than one product. No resolution was reached, and, once again, the previous discussions were not referred to. (The issue was resolved two months later.)

Not all issues that could not be resolved on the spot were forgotten in this way, of course. The *What goes where?* issue was recognized as pervasive and eventually a subteam was formed from members of the Requirements and API Subteams. Even then, it took two months for the two subteams to determine that action was required.

The final type of problem occurred when decisions started not to be acted on. During August, two sections were discussed in detail, but the SRD was never edited as a result of the discussion. By this time, the SRD had become more an impediment than anything. It embedded too many premature commitments, and could not be modified without major rewriting. Once the customer interviews started, the Requirements Subteam started to ignore the SRD, and it was eventually abandoned. The last edit to this document was made on July 28, the day after a meeting in which a member of the divisional strategy team reviewed the SRD and dropped the bombshell that it should have been organized around nonfunctional requirements.

The SRD was not referred to systematically during the White Papers epoch. Very little text was carried over from the SRD to the white papers (providing ample opportunity to forget information), but many of the significant ideas explored during the writing of the SRD show up transformed in the white papers. The stimulus for carrying over this information seems to have been the understanding of the authors of the white papers, rather than what had already been written.

These problems seem to be intrinsic to the writing process. The desire to commit to an organization and get ideas down on paper for scrutiny by others conflicts with the goal of maximizing flexibility and the capacity to rework vague ideas when they are shown to require refinement. In retrospect, the Centauri team probably committed prematurely to a document structure (Sharples, 1993), but they could hardly avoid doing so given the document standards in force in the company.

6. Differences between actual and normative processes

6.1 Process definition

The company has a strong process quality culture, and every division has standard development processes that its projects are expected to follow. However, all the previously documented and adopted processes in Centauri's division had been designed for evolutionary development of long-lived, 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 standard development process therefore probably would be too restrictive.

Accordingly, the Centauri project set up a Process Subteam to define an appropriate development process and its tool requirements. The internal process specification was derived from IEEE 1074 and SEI's Capability Maturity Model. However, the process was largely untried for new development, problems occurred. The Process Subteam, at this time, recommended that the IEEE 1074 be followed.

The Centauri engineers found the 1074 standard to be very vague. Because they had no examples to follow, they inexorably reverted to what they knew. The original architecture document had many of the characteristics of an SRD although it was organized around a proposed system architecture, so the requirements subteam 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.

6.2 Overemphasis on process issues

The requirements subteam spent a lot of time discussing what type of document they were writing and what the standard meant. About half the issues they discussed during their review meetings were about their process, document structure and goals (see Table II). In contrast, during the meetings with customer representatives, they only spent about 20 percent of the time on process-related issues.

The process definition effort spanned the entire lifecycle of Centauri, not just conceptual development and functional specification. However, the definition of the process for even the first phase of the project lagged the phase itself. The Requirements Subteam could not wait until its process had been defined for it, but had to make progress as best it could. Here we see a conflict between the goal to define and impose high quality processes and the goal to reduce development cycle time. The more discussion that occurred about the process, the less time there was to apply it.

In general, the process developed by the Process Subteam and the document standards that the Requirements Subteam was following, seemed to be too heavyweight for conceptual design, because they strongly emphasized the layout, control and formal inspection of documentation at the expense of activities that tapped the creativity of individuals and provided external working memory mechanisms for the team.

6.3 **Process obsession as a displacement activity.**

This extraordinary focus on process issues that we observed throughout Centauri bears many of the hallmarks of a displacement activity. 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 reading e-mail). Designing and keeping to standardized processes is an important component of any quality improvement program, and it plays an essential role in the company's drive to reduce product defects and accelerate the engineering process. Nevertheless, we observed many occasions when the team took refuge in process discussions rather than attempting to resolve substantive issues that required commitments or risky assumptions.

6.4 **Marginalization of the process definition effort**

Some team members came to the conclusion that much of this emphasis on process standardization was a poor substitute for making 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 and get on with the job. The Centauri designers wanted as much process guidance as they could get; they were just skeptical about the appropriateness of the process they were supposed to follow.

> 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 a process-conscious culture like Centauri, the process and development environments subteams were seen by many designers as external advisors who were unaware of the constraints of the real design process. Much of the work to acquire and customize tools and to develop policies for tool use affected Centauri only after the conceptual exploration phase or in activities where careful document management and review processes were already part of the company culture. The process definition effort had little effect during the conceptual design activities, and so the project floundered initially.

The emphasis on process issues persisted after our involvement in Centauri. Much later (August, 1995), we were given the log of documents that had come under configuration management and therefore had been formally reviewed and were being treated as definitive. There were at that time 31 documents, five of which had several versions under configuration management. Of these 31, 19 (61 percent) were about the development process or support tools, and only twelve were about Centauri itself.

7. Aloofness from use context

It was inevitable that the team would find it difficult to stay in touch with the possible contexts of use of the system: Centauri was a middleware platform on which other developers were to build, so an extra level of indirection was automatically introduced between the Centauri team and the end users of the company's products. The test of Centauri's functionality, once it was implemented, would be how well it supported the development of external customer applications.

7.1 Predicting future applications

It became important to the team to anticipate the first few applications so that it could accurately understand the requirements. Despite the team's appreciation that too great an emphasis on the specific details of applications could tempt them to compromise the generality of the architecture, they very soon started talking about the "First Application" as if it held secrets that they needed revealed before they could make significant progress. Certainly, there were differences in emphasis among the applications that could have rendered any concrete work premature if the team were to have anticipated the wrong application. For example, one candidate application would have required mainly off-line data access, whereas another would have required real-time access to distributed copies of data - a complex set of requirements that the team wanted to defer if possible. Team members were acutely aware that a commitment from an application project to use or even to consider using their architecture would be shot in the arm for the project as a whole, and they did not want to make too many architectural decisions before Centauri had a commitment from an application project. Several potential first applications came and went during the months of our involvement, and on several occasions we were told that a decision about the "First Application" had been made, only for that decision to turn out to be tentative.

In deciding to defer choices, the team was in effect trying to balance the consequences of predicting wrongly that a given customer-directed system would be one of their applications (a Type I error) against the consequences of failing to predict an application (a Type II error). Centauri teams and

management consistently chose the conservative option of not risking Type I errors and risking Type II errors. If there was any doubt whether a system would become a Centauri application, it was treated as if it was not.

The team had eagerly delved into requirements details without waiting for a commitment from an application project and without having access to detailed customer requirements, but they were reticent to make significant architectural decisions that they thought would depend on the "First Application" before a project made a commitment to Centauri. When the first application eventually committed to use Centauri (albeit tentatively), its specific needs might easily have been anticipated earlier by studying any of several existing systems as if they were to be Centauri clients.

This inability to commit to a direction created a distance between Centauri and its potential user applications. When feedback did come from the outside, it had a suddenness and shocking effect that was described in Section 4.4.

7.2 Scenarios vs. general descriptions

We expected the team to communicate many ideas through concrete scenarios, because descriptions of the interactions between Centauri and the proposed applications would otherwise be very general and abstract. But scenarios would have required some commitment, however tentative, to consider a given system as an application. In keeping with the team's conservative refusal to make such assumptions, they rarely discussed system use in terms of concrete scenarios.

When specific scenarios came up, they were ephemeral discussion topics that were not named and could not easily be recalled or referred to later. For example, during one meeting, the Requirements Subteam discussed the need for applications to set timers (for example, to time out on a communications failure). The discussion featured a scenario in which the subteam members explored issues of clock resolution and the effect that decisions about the event notification mechanism would have on the architecture. This was a detailed and incisive discussion, and it helped to clarify some previously cloudy issues. Remarkably, however, the discussion was never revisited during the

following few weeks and it had no effect on the SRD, because the insights, having been arrived at and couched in system-specific terms, did not belong there.

One result of this conservatism was that the architecture diagrams that designers sketched on white boards and carried around in their heads always featured static relationships among components. Dynamic relationships, such as when an architectural component could be invoked, patterns of communication among components, or when interacting components could run concurrently, did not feature in these diagrams. Dynamic behavior consequently played a minor role in discussions.

8. Nobody's perfect: misguided strivings

With the benefit of hindsight, we can see several ways in which Centauri team members and management were striving after the wrong (or too strongly stated) goals.

8.1 Striving for process perfection

The process culture of the company undoubtedly contributed to the widespread attitude that a project could not be making "real" progress until it had defined a process to follow. However, Centauri was breaking new ground and clearly had to make up some rules for itself as it went along. Our observations of the team, and the Requirements Subteam in particular, show that this happened. Nevertheless, the attitude persisted that any such process innovations were stopgap measures until "the" process could be defined and documented. This attitude stultified the creativity of the team and its willingness to take a stand on technical issues.

There is a fine line to be drawn here between the paralysis that arises when everyone thinks that no activities can be legitimized until a formal process is in place and the sloppiness that inevitably results from everyone working however they want. However, Centauri was nowhere near that line. In our opinion, the team failed to converge on a common vision of Centauri as quickly as it could because the project culture and professional expectations of the team members militated against taking risks and accepting responsibility for decisions. Too strong an emphasis on process purity may be a symptom that some team members are projecting their authority and responsibility onto an impersonal process, and thereby diffusing their personal sense of responsibility and involvement.

8.2 Striving for consensus

During our involvement in the project, no person or team was responsible for drawing up a single architectural vision. A pair of meetings were held between the API and Requirements Subteams to discuss differences in architecture diagrams. This eventually led to "This is Centauri," an architecture overview document. One manager described the process as follows:

> Over the last few months [i.e. Fall, 1994] people have been putting out their own proposals about how it should work... Everyone with different ideas put down their ideas and had some meetings to build consensus.... "This is [Centauri]" is pretty much the conclusion of these meetings.

Thus, during the summer of 1994 and for several months afterward, there was a desire to achieve a technical consensus. Early on in the project, a need for technical leadership was recognized. However, difficulty was encountered when the Centauri management attempted to fill that role; No one was available to them. Possibly in response to this, management perceived its role as "organic" and not directive. In accepting that an authoritarian style of management might not be desirable for this type of project, however, management and subteam leaders initially were unable or unwilling to exploit the use of strong technical leadership and decision making mechanisms. In retrospect, the Centauri team culture can be seen to be too consensus-seeking and communitarian. This was yet another way in which technical responsibility was diffused from individuals to the group.

8.3 Striving for 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 detail, only to find that it had not developed sufficiently the goals and abstractions on which these details depended. For example, the Requirements Subteam submerged itself in detailed documentation before the goals for the system were really understood; and similarly the Process Subteam's effort to standardize processes was well intentioned but too detached from the development effort to affect it strongly. Many of the design and process issues that the team resolved were spuriously detailed because they were not grounded in the context of use (how applications would use Centauri functions and how developers would use the process).

Undoubtedly these tendencies were accentuated in Centauri by the need for it to be a generic system and by the company's strong quality-conscious culture, but we suspect these tendencies are inevitable

during conceptual design when the need to structure and accelerate an engineering process clashes with the need to explore broadly system goals and options.

8.4 Striving after gestalt knowledge

Architecture and requirements issues were inextricably intertwined because the scope of the requirements affected the “thickness” of the filling in the three-layer architecture (OS, Centauri, and applications) 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 application domain, 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 subteams, especially the Requirements and API Subteams, to coordinate their work.

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 then led to a process question: “Are we making premature implementation decisions?” Such bouncing back and forth between the general and detailed, the factual and the optative, and the substantive and the meta-level was very typical.

As with some of the previous strivings, here too there is a fine line to be drawn between not seeing the forest for the trees and not seeing the trees for the forest. In retrospect, Centauri designers tried too hard to get the big picture. Ironically, they may have made much faster progress and developed a superior understanding of the system’s requirements had they restricted their focus, arbitrarily if need be. For example, they could have looked at just one or two potential applications in greater detail in the hope that the insights this investigation yielded would be of general use whether or not the systems became Centauri applications. They could have artificially restricted their investigations by

fixing certain architectural assumptions (e.g. one global clock) and working through the implications of this commitment when analyzing requirements.

We speculate that this tendency toward Gestalt thinking is endemic in teams that design general-purpose products like Centauri.

9. Summary and implications

In the previous four sections, we have summarized our findings in terms of four general phenomena: nonmonotonic convergence, forgetful teamwork, overemphasis on process issues, and contextual aloofness. We now recap the major points and the implications of each of them for software design practice. In many cases, specific interventions would likely reap benefits in several of these four areas. We discuss these last, relating our findings and recommendations about teamwork to current software engineering practice and research.

9.1 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 roughly every few weeks or month by a stepping back to reconsider what had been done. During the first year of the project, its goals and architecture were rethought fundamentally several times. With the benefit of hindsight, we can say that this convergence process was inefficient and could have been accelerated if only the designers had considered this factor or that at critical junctures. We suspect, however, that punctuated and nonmonotonic progress is inevitable during conceptual design and should be planned for.

Slow and nonmonotonic convergence on a project vision is probably best addressed organizationally. A project like Centauri should have a "superdesigner" or small core team of very experienced people (Curtis et al. 1988; Lubars et al. 1993). The knowledge that is needed (and was lacking in Centauri) spans several areas, including the application domain and knowledge of architectures of similar systems. Bright, professional-minded people are not enough.

Some general-purpose productivity tools could also have been used effectively to support convergence. The project did not have a common repository of architectural ideas, in which sketches were set side-by-side and could be scrutinized. Instead, architecture diagrams were embedded in longer

and often obsolete documents. Even having common access to a web browser with a standard diagramming tool could have surfaced key architectural issues. The company had strict documentation standards for post-specification design, so there was to be no lack of a project long-term memory once the project got beyond the conceptual design phase. What was lacking was a means of recording and disseminating tentative ideas that were being worked on by members of the team and which were to act as common references for convergent discussions.

9.2. Forgetful 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 a forgetful set of work practices. So many threads were continually left hanging that the individual team members could not easily remember where they were, what issues depended on others, what they had already decided, and what they needed to do or find out. As a team, they lacked tools, procedures and support mechanisms that could compensate for these inabilities. Far from regarding this phenomenon as a limitation of this particular team of designers, we regard forgetful teamwork as an inevitable side-effect of the cognitive limitations of individual designers, the complexity of large software systems, and the immaturity of processes for intellectual teamwork.

A structured approach to making team processes less forgetful would be to adopt design rationale methods and tools such as gIBIS (Conklin & Begeman 1988), or structured note-taking and multimedia indexing systems, such as Synthesis (Potts et al. 1993). 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. In part this is because of the limitations of research prototypes. The main reasons, however, are that the people expending the effort to record the information are not the ones who benefit directly (Grudin, 1988) and the tools force people to structure their ideas and collaborative processes artificially and in isolation from the artifact that is

being discussed (Potts et al, 1995). A less intrusive form of tool support would be to allow freely structured annotations to be attached to the shared repository of tentative architectural sketches proposed above. If paired with a semi-automatic notification mechanism modeled after the Object Lens (Lai et al. 1988), in which users were notified or could filter broadcast notifications according to subject matter, this mechanism would support a forum for focused, asynchronous discussion about substantive design issues. The technology for this is, of course, widely available today. More experimental notification-based technologies specifically developed to keep track of requirements issues and trade-offs also deserve examination. The "Win-Win" System (Boehm et al., 1996) is the most well-developed example of such a tool.

An even simpler strategy to externalizing the team's working memory is provided by simple organizational and meeting management practices. What we have in mind is a procedure that would be for the retention of team working memory what the Delphi procedure is for consensual forecasting or the inspection (Fagan 1976) is for formal reviews. Such artificial structuring of design meetings would be compatible with most design cultures. Simply having one team member act as scribe and have 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 a good first step.

9.3. Overemphasis on process issues

So poor are the development processes in many software organizations that recent years have seen concerted industry-wide efforts to standardize and evaluate development processes. The best known and most influential effort of this type in the U.S.A. is the Software Engineering Institute's "Capability Maturity Model" (CMM), a five-level scale of process excellence measured by answering a detailed set of questions (Paulk et al., 1995). The organization we studied has a commitment to process excellence, has a CMM score that most organizations would envy, and has won prestigious awards for its achievements. There is no question therefore that the company and the Centauri project members had a genuine commitment to follow professional processes in all aspects of their work.

But it was precisely this attempt to specify and adhere to process standards that gave rise to some of the Centauri team's worst problems. Because the team was attempting to formulate processes for solving an inherently ill-structured and poorly formulated problem, the only processes that were available as exemplars were inappropriate. This led to a divergence between the sub-team responsible for process supervision and the sub-teams responsible for developing the product, a divergence that both sides recognized as undesirable.

We can only conclude that it is unproductive to overemphasize adherence to an espoused process if that process has not been worked out in advance, experimented with and adopted voluntarily by team members. The contrast between espoused, preplanned practices and actual work patterns has a long history - see, for example, Schon's (1983) distinction between professionals' espoused theories and theories-in-use, Suchman's (1983) critique of expertise as planning, and more recently Sachs's (1995) distinction between "organizational, explicit" and "activity-oriented, tacit" views of work. Our findings lead us to agree with these authors' perspectives, and we suspect that the current frenzy of interest in process excellence among members of the software community underestimates the importance of tacit, informal work practices in determining which individuals and teams develop excellent products.

9.4. Aloofness from context

Perhaps the most pernicious general phenomenon of the four was the failure of the team to relate the abstractions with which it was dealing - whether general or specific, product-related or process-related - to concrete contexts that gave these abstractions their meaning. Because they lacked clear commitments from other projects, they could not easily ground their explorations of desired features and architectural constraints in the concrete needs and details of applications. Because they usually worked on evolving systems, where new features are grafted onto an existing system with detailed, formal design documentation, they were not able to relate to their actual experience the conceptual exploration standards that the Process Subteam was developing. This failure to get down to brass tacks may seem ironic in an engineering community, but it is unsurprising given that Centauri had no direct end-users

and that many developers lacked experience at starting a completely new project. Once more, we need to stress that this was not a inadequacy of Centauri, but an intrinsic difficulty in conceptual design. There is 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).

Understanding the customer is probably the most significant success factor here. As Keil and Carmel (1995) 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 would be one way to address this issue. A complementary strategy would be to organize the acquisition of requirements and application expertise around concrete scenarios (Potts, 1995). By concentrating on the SRD's illusory precision, the team became unable to step back and ask fundamental 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.

9.5. Common factors and implications

The generality of these findings cannot be assured from a single case study, however detailed. But we are confident that our use of triangulation methods and the broad agreement between our results and those of earlier, smaller-scale studies or retrospective surveys points to generally applicable conclusions. For example, the effects of adopting a consensual decision making style and the absence of a single source of application knowledge within the project agrees with the finding from previous studies of the importance of having a "superdesigner" in the team and the importance of knowledge acquisition channels for the rate of convergence (Walz et al. 1994). And our finding that most discussions revolved around process issues and architectural consequences agrees with those of Kuwana & Herbsleb (1993), that discussing requirements rationale does not play a major role in conceptual design.

We end as we began by reminding the reader of Brooks's (1986) claim that software design is a problem of conceptual complexity, and his warning not to believe that a single "silver bullet" can solve many of the problems that occur in practical projects. He had in mind technical proposals for formalizing the development process with new notations, languages, and support environments. But the same conclusion can also be drawn for collaborative technologies. Sophisticated meeting room technologies or semi-structured design rationale systems would probably not have helped Centauri designers converge on a common understanding of their system any faster, and the support activities needed to make these technologies work effectively would probably have only added to the conceptual complexity of their design task. Complex methods and defined processes are not the answer either if they do not respect the talents and limitations of the people concerned and the contingencies of the design situation. Formal administrative controls and documentation standards may be misplaced during conceptual design and may actually contribute to the freezing of poor design decisions. Our experience working with this project and others like it convinces us that the most effective interventions are likely to be minor modifications of current practice rather than radical new proposals.

// 10. Acknowledgments

The project would not have been possible without the openness and wholehearted support of the Centauri designers and management, and especially the members of the Requirements Subteam.

11. References

Atwood, M.E., Burns, B., Gairing, D., Girgensohn, A., Lee, A., Turner, S., Alteras-Webb, S. and Zimmermann, B. (1995): Facilitating Communication in Software Development. *Proceedings DIS'95: Symposium on Designing Interactive Systems*, Ann Arbor, MI, August 23-25. New York: ACM Press, pp. 65-73.

Berlin, L.M., Jeffries, R., O'Day, V.L. Paepcke, A. and Wharton, C. (1993): Where Did You Put It? Issues in the Design and Use of a Group Memory, *Proc. InterCHI*, Amsterdam, Netherlands. New York: ACM Press, pp. 23-30.

Boehm, B. (1983): *Software Engineering Economics*, Engelwood Cliffs, NJ: Prentice-Hall.

Brooks, F. (1986): No Silver Bullet. *Proceedings of the IFIP Tenth World Computing Congress*, Dublin, Ireland. Amsterdam: Elsevier, pp. 1069-1076.

Conklin, J. and Begeman (1989): gIBIS: A Tool for all Reasons, *J. Am. Soc. Inf. Sci*., May, pp. 200-213.

Curtis, B., H. Krasner and N. Iscoe (1988): A Field Study of the Software Design Process for Large Teams, *Comm. ACM, Vol. 31, No. 11*, pp.1268-1287.

Fagan, M. (1976) Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal 15(3*), pp. 182-211.

Grudin, J. (1988): Why Groupware Applications Fail: Problems in design and evaluation, *Office: Technology and People, 4(3),* pp. 245-264.

Keil, M. and Carmel, E. (1995): Customer-Developer Links in Software Development. *Comm. ACM 38(5)*, pp. 33-44.

Kraut, R.F. and Streeter, L.A. (1995): Coordination in Software Development. *Comm. ACM 38(3)*, pp. 69-81.

Kuhn, T. (1970): *The Structure of Scientific Revolutions*, 2nd Ed. Chicago: Univ. Chicago Press.

Kunz, W. and H. Rittel (1970): *Issues as Elements of Information Systems*, Working Paper 131, Inst. Urban and Regional Development, Univ. California at Berkeley.

Kuwana, E. and J. D. Herbsleb, (1993): Representing Knowledge in Requirements Engineering: An Empirical Study of What Software Engineers Need to Know, *Proc. RE 93: IEEE Int. Symp. Requirements Eng*., San Diego, CA Jan 4-6. IEEE Comp. Soc. Press, pp. 273-276.

Lai, K.-Y., T. W. Malone & K.-C. Yu (1988): Object Lens: A "Spreadsheet" for Cooperative Work. *ACM Trans. Office Inf. Sys.* 6(4), pp. 332-353.

Lee, J. (1991): Extending the Potts and Bruns Model for Recording Design Rationale*, Proc. 13th Int. Conf. Software Eng*., IEEE Comp. Soc. Press, 1991, pp. 114-127.

Lubars, M., C. Potts and C. Richter, (1993): A review of the state of practice in requirements modeling*, Proc. IEEE Symp. Requirements Eng. (RE'93),* San Diego, CA, January 4-6. IEEE Computer Society Press, pp. 2-14.

Paulk, M. B. Curtis, M. Chrissis and C. Weber (1995): *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, MA: Addison-Wesley.

Potts, C. (1995): Using Schematic Scenarios to Understand User Needs*. Proceedings DIS'95: Symposium on Designing Interactive Systems*, Ann Arbor, MI, August 23-25. New York: ACM Press, pp. 247-256.

Potts, C., J. D. Bolter, and A. Badre, (1993*)*: *Collaborative Pre-Writing with a Video-Based Group Working Memory*, Georgia Institute of Technology, GVU Technical Report, 93-35.

Potts, C. and G. Bruns (1988): Recording the reasons for design decisions*, Proc. 10th Int. Conf. Software Eng*., Singapore (May), IEEE Computer Society Press, pp. 418-427.

Potts, C. and K. Takahashi, (1993): An Active Hypertext Model for System Requirements*, Proc. 7th Int. Workshop Software Specification and Design*, Redondo Beach (December), IEEE Computer Society Press, pp. 62-67.

Potts, C., K. Takahashi and A. Anton, (1994): Inquiry-Based Requirements Analysis, *IEEE Software, 11(2)*, March, 21-32.

Russell, D.M., Stefik, M.J., Pirolli, P. and Card, S.K. (1993): The Cost Structure of Sensemaking, *Proc. InterCHI*, Amsterdam, Netherlands. New York: ACM Press, pp. 269-276.

Schon, D. A. (1983): *The Reflective Practitioner: How Professionals Think in Action*, New York: Basic Books.

Sharples, M. (1993): Adding a Little Structure to Collaborative Writing. In D.Diaper and C. Sanger (eds.) *CSCW in Practice: An Introduction and Case Studies*, London: Springer.

Suchman, L. (1983): Office procedures as practical action: Models of work and system design, *ACM Trans. Office Systems, 1*(4), pp. 320-328.

Terveen, L.G., Selfridge, P.G. and Long, M.D. (1995): Living Design Memory: Framework, Implementation, Lessons Learned. *HCI 10(1)*, pp. 1-38.

Walz, D., B. Curtis and J. Elam (1993): Inside a Software Design Team: Knowledge acquisition, sharing and integration , Comm. ACM, 36(10), pp. 62-77.

---

[i] *Centauri* is a pseudonym. Information about the Centauri design process is reported faithfully, but details about the Centauri system have been sanitized to protect the company's trade secrets.

[ii] The team videotaped and transcribed their later customer interviews, but they did not index the videotapes using Synthesis.