Collaboration during Conceptual Design

Lara D. Catledge

and

Colin Potts

Georgia Institute of Technology

October 24, 1995

Abstract: The conceptual design of software involves the analysis of requirements, functional specification, and architectural design. Despite the recent interest in supporting the software development process, conceptual design remains a largely informal and little understood activity. 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 and findings, and the implications of the results 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 proceduralized development process and the urgency of delivering a working product. This tension manifested itself in the creation of ad hoc processes and various dysfunctional displacement activities. Although our findings are rooted in the specific context of a single project, we believe they are of general import, and discuss their implications in the light of comparable findings in other studies.

The conceptual design of software involves the analysis of requirements, functional specification, and architectural design. Despite the recent interest in supporting the software development process, conceptual design remains a largely informal and little understood activity. It is open-ended and often tackles ill-defined and poorly understood problems. In practice, "soft" factors, such as organizational culture, patterns of collaboration, and effectiveness of team communication, play as important a role in determining organizational culture as the "hard" engineering techniques, such as specification and validation methods, that have been more strongly emphasized by the software engineering research community.

In this paper, we describe a study that we undertook of 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 and findings, and the implications of the results for process and tool support. In particular, we discuss the effects that organizational structure and inter-team and intra-team communication practices have on design convergence as assessed by an analysis of how Centauri documents evolved and how ideas were expressed during project meetings.

1. Collaboration in Conceptual Design

1.1. Previous Research

A number of case studies exist that describe portions of the requirements process. By analyzing and cataloging the kinds of questions asked in writing requirements, Kuwana and Herbsleb [19] found that most of the questions asked related to how a particular feature would be implemented and very rarely why a feature had been included. Olson et. al. [25] developed a method for describing the percentage of time spent in Requirements meetings on various tasks and found that a large portion of time spent in meetings was to clarify points already discussed. Walz et. al. [41] found that the integration of knowledge and the depth of domain knowledge was critical to the success of early stages in the project. All of these studies indicate that while the details, the "what" and "how", are being discussed, the rationale, reasons and the larger vision are being neglected.

Another set of reseach focused on surveys. Curtis et. al. [7], by interviewing key personnel in 19 organizations found that one of the most significant and pervasive features that affected the success of large projects was the presence of what they called a "superdesigner," a person who was responsible for holding and supporting the project vision. Lubars, Potts and Richter [29] did a similar study of 23 organizations, but focused on the Requirements analysis process exclusively. That study clearly demonstrated that informal documentation, communication and coordination are all more important during what we now call the conceptual design phase, than conventional notational and analytic methodologies.

1.2. Research Objectives

We had a number of research issues that we wished to study at the start of the observation. We've integrated a lot of the aspects of earlier research in an ecolically valid environment. We've adopted a longitudinal approach,similar to waltz but opposed to surveys. Our data is more in-depth and comprehensive than the Olson or Kuwana work (also less focused). Focusing on the early stages of design, as opposed to SEI and other software improvement efforts. This is a real project with real deadlines and deliverables. Not a toy project.

Formation & facilitation of common technical vision

As Curtis, Krasner and Iscoe [7] discovered, successful projects, even very large ones, often have a single person who is recognized as the source and keeper of the project vision. This `super designer,' as Curtis et al named the role, is always a senior technical figure in the project who is intimately familiar with the problem domain, the requirements, and the design constraints imposed by the architecture, but is not necessarily an official project leader or manager. We are interested in ways a common technical vision is developed by teams doing original design work.

Keeping track of status & supporting project's "working memory"

It has become a commonplace to observe that effective teams and organizations `learn' from their experiences and that an `institutional memory' is an aid to rapid problem solving in familiar situations. Tools such as gIBIS [24] aim to provide teams with a structured record of their decisions so that they can be reconsidered, audited, or replayed later.

We are mainly interested in the remembering and forgetting of key design considerations and decisions during conceptual design. The duration of these memories (days or weeks, or at most months) is short by organizational standards and within the time course of many of the design activities that could use them. Thus these are `working' memories, not long-term memories.

Sharing and evolving informal information

During conceptual design, a design team may produce many ephemeral documents in addition to early versions of formal Requirements and design documents. These ephemeral documents include rough sketches of scenarios of use, system concept documents, discussion topics, white papers and memos. The relationship of these documents to the more formal documents that are the product of the conceptual design activities is not formally defined but they do play a vital role in the provenance of design decisions (i.e. recording where they came from and why).

Representing & analyzing concrete system properties

Exploratory conceptual design often proceeds from the concrete to the abstract as much as from the abstract to the concrete. In particular, many software engineering and human-computer interaction researchers have recently focused on the role of scenarios in defining system Requirements and in exploring design options [29]. This use of scenarios contrasts with their role in validating existing specifications or designs because it is aimed at generating such documents.

2. Methodology

2.1. The Centauri Project Setting

We studied the conceptual design activities of a representative industrial software project, Centauri, within Motorola 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 and findings, and the implications of the results for process and tool support. In particular, we discuss the effects that organizational structure and inter-team and intra-team communication practices have on design convergence as assessed by an analysis of how Centauri documents evolved and how ideas were expressed during project meetings.

The Centari Project provided an ideal environment in which to study these issues because several of the frequent constraints on development projects were missing. First, Centauri is a new development rather than a project to manage the evolution of an existing system. Second, Centauri is not being developed for a specific client who could constrain the problem boundary. A third reason for the open-endedness of Centauri's conceptual design process is that Centauri is "middleware," not application software. That is, it is software that resides on and uses standard system software (operating system services), but it acts as a platform itself for application software. This makes the services provided by Centauri more abstract and difficult to understand than the services provided by an application to its end-users. The application software developers within Motorola who will be Centauri's direct users may use different terminology, and this compounds the problems of communication and development of a common understanding.

2.2. Observation and Summarization of Meetings

We taped all the requirements meetings from June 30th to September 15th, 1994. This totals approximately 40 hours videotape. These videotapes are indexed to an outline of the meeting minutes with Synthesis [REF]. The meeting notes were kept in a loose IBIS-like format. Key issues, assumtptions, and questions were indicated. These were subsequently catagorized by subject matter and scope.

2.3. Tracking Documentation Dependencies and Evolution

All Requirements/architecture documents from June '94 to March '95 have been collected and logged. This includes architecture documents and diagrams, the original requirements document, formal presentations as well as more informal white paper documents. These were cataloged on a regular basis and cross-referenced to the meeting notes.

2.4. Subjective Accounts of the Designers

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

3.Project Chronology and Criical Incidents

3.1. Background

Centauri was initially staffed in Spring 1994 with the goal to develop a common core platform that would be portable, scalable, and extensible for a set of Motorola products. The original staffing included a managerial team that would provide most of the managerial leadership throughout the Concept Exploration stage. The first managerial team consisted of three components: a Reuse Team, a Product Requirements Team and an Architecture Team.

This group developed several seminal documents that outline the basic strategy of the Centauri project during these early phase. One of these, the *Centauri Project Strategy and Concept Exploration Requirements*, listed the objective of the Centauri Project as to "provide a small, low cost platform to target the small capacity cellular applications" The document also listed an initial list of applications to be placed on top of Centauri. Centauri was initially thought to be producable in a very short amount of time. Estimates as early as one year from the initial staffing were given for deliverables. The primary deliverable of this group was the initial architecture document, finished in early April

3.2. First Team-based Organization

In early summer, the Centaui Team saw an influx of about 40 staff engineers from a cancelled project. This event roughly coincides with our involvement in the project. The team structure would change to include a Development Environments team, a expanded Reuse team, a Requirements team, a Core Platform Development team and an API team. Teams were fairly stable and they were cross-staffed extensively. A notable exception to this was the requirement team which tended to be smaller and more insular.

The Requirements team at this time was developing a System Requirements Document (SRD), unofficially forming the lynch-pin of the Centauri development. Generally this period is characterized by the Requirments team dividing up sections of the architecture document, going away and writing labelled requirements based on their previous exerience and other SRDs, and finally reviewing them within the team.

3.3. The Great Process Debate

Documentation and process was one of the large issues that continued to be a problem in the requirements team. It was considered at one point that certain sections of the SRD were complete enough to start spinning off development teams. In a group meeting with the development environments/process team to discuss this option it was determined that the requirements team was actually producing a functional specification, kicking off a huge debate over software lifecycles, process and documentation standards. In the end, the develoment environments spawned a subteam to examine the options, to make a decision on a lifecycle choice and to map that choice to the existing corporate process. Interestingly, the requirement team did not have representation on this team and, in the absense of a decision, continued to write and review sections of SRD.

3.4. The Architecture Debate

The issue of location of functionality continued to be a thorn in everyone's side through the middle of the summer. Finally, the Requirements team discovered that the API team was struggling with the architecture document's designation of the position of the API. This resulted in a series of discussions between the two teams in e-mail and face to face meetings. From 8/2 to 8/17 the Requirements and API teams met jointly occasionally to discuss these issues. Resulting from this collaboration was a document "This is Centari" that outlined all of the interfaces in great detail.

3.5. Customer Involvement

The document review and subsequent changes to the document reflecting the discussion proceeded with pretty regularity until the week of July 25th. The requirements team decided that they needed more representation from a customer. So they decided to appoint a divisional strategy and evolution team as their customers. They commented that the requirements needed to include system level constraints. This derailed the Requirements team sufficiently that they decided to stop working on the Requirements document until they resolved some of the larger issues they were facing: including reuse, customer need identification and development cycle.

The requirements team decided to meet with representatives from likely aplication groups one-one one. This progressed from August 17th to September 9th. A list of questions was compiled. The meetings were video-taped and transcribed laboriously. On the basis of these interviews another system level architecture document was written. In addition, a wish list of features was created which bore a remarkable resemblance to the original architecture document.

3.6. Second Team-based Organization

From this wish list the Requirements team decided to spin off four sub-committees to review specific technical problems. Each team was to publish a "white paper." The purpose of the white papers is to "to facilitate a level of fundamental agreement on the requirements and functions of the core. The old team structure was virtually abandonned in favor of this more flexible new approach. These white papers, along with the others, were collected and compiled with a wrapper in what was intended to be the final System Requirments Document.

4. Conceptual Design in Centauri

In this section, we present our observations about the Centauri project. 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 issues.

4.1. Organizational Working Memory

4.1.1. Meetings

For the period of two months the requirements team reviewed and revised the large, 14,000 word system requirements document. The content of this original document was derived from a number of other documents. Only 2,000 words were added in the two months of our involvement during which the requirements document was also active.

Meeting notes collected over that period indicate that numerous topics were repeatedly discussed without reference to or awareness of previous discussions. In most cases the meetings were arranged as document reviews for particular sections of the requirements document. During the discussion of some of the smaller scaled items, larger issues would often present themselves. For instance, on 6/30, the issue of backwards compatibility with existing systems came up in a discussion of the first customer for the core platform. However, at the next meeting on 7/5 there was no mention of it. The discussion centered around what functionality belongs in the core software and what belongs above the API. While these issues are not unrelated, there was no mention of the previous week's discussion. Backwards compatibility did not come up in a meeting again until 7/12 during a discussion of the possibility of designing a platform that would support more than one product without any noticeable progress being made on an answer and apparently without any memory of the previous discussions.

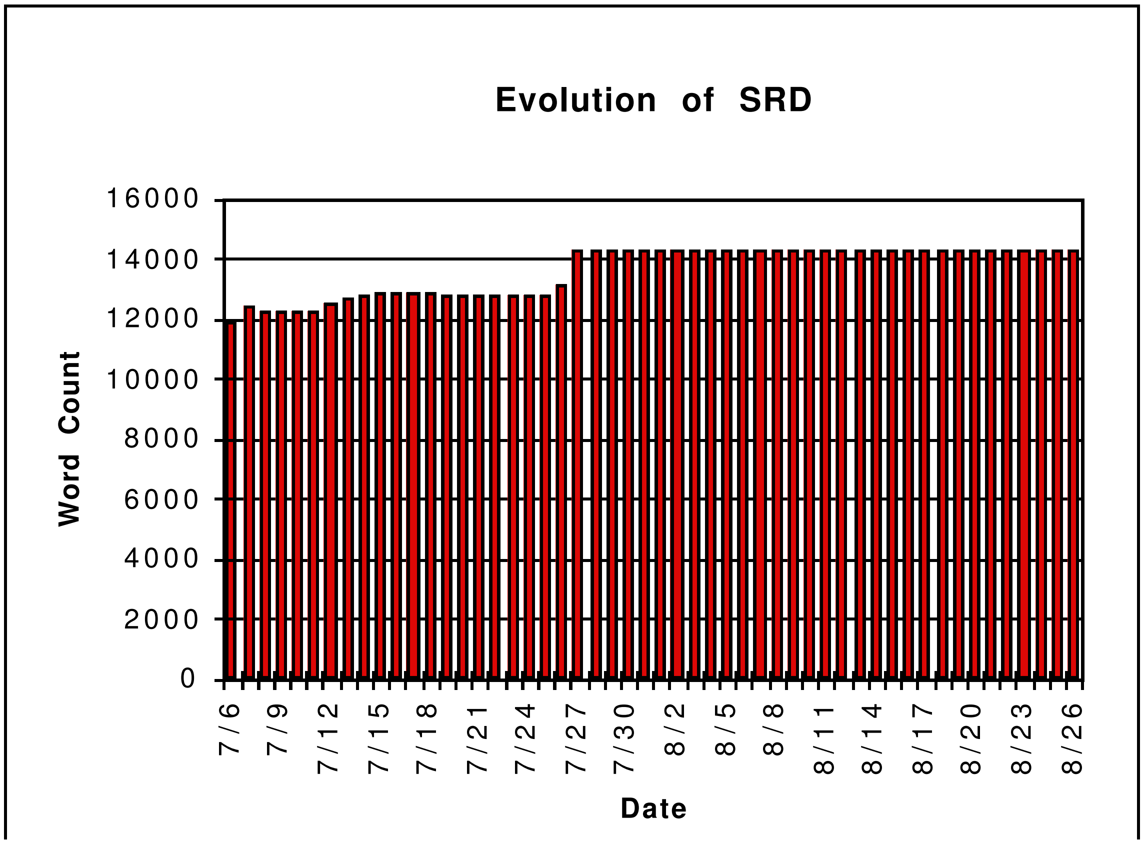
Other issues, such as the decision of what belongs in the core as opposed to what belongs in the core were omnipresent. In this case, however, continuity from meeting to meeting was still lacking. The difference in this case was that it was recognized as an large issue and eventually did see a sub-team staffed with members of the requirements team and members of the API team. The issue in this case was that it took two months for the group to determine that a different kind of action was called for.

One could argue that these issues were remembered, they just weren't acted upon. Another possibility is that the issues needed to achieve some sort of critical mass before the issue could be acted on. A third speculation is that this team did not have the resources (whether power, knowledge, or manpower) to solve the particular problem at hand. The answer to all of these is that we do not have any indication that the team even noticed how often the issues came up.

During this period, there was also a tendency for the requirements team to spin off issues like the great process debate. While this tended to make work in other areas of the project, the requirements team itself was not often involved in the nitty-gritty details of solving these roadblocks and they seemed perfectly happy to keep working on the reviews and meetings as if the issue had not come up.

4.1.2. Documentation

Proving that items were forgotten from meeting to meeting in the absence of any physical artifact is difficult. After all, our meeting notes are selective and subject to interpretation. However, we can indicate where and when changes were made to the primary artifact, the System Requirements Document, and how these are related to the content of the discussions. It is interesting to note that during this document review period, very few changes were actually made to the SRD. The changes that were made generally fall under minor edits to incorporate suggestions made in meetings (often word changes), narrative sections and illustrations that back-filled the detail like illustrations of how a core platform would work. Very rarely were significant changes to requirements made after the group review. There is a notable exception to this in the case where reorganization of the SRD would take place. Overall, however, these changes amounted to less that a 14% change in the size of the document over two months.



In the beginning of the summer, sections were being reviewed every week. Common User Interfaces, Resource Management, Timer Management and State Table Support were all reviewed by 7/7. There continued to be small revisions made to these documents through the next week, usually 3-5 small revisions over that time period. Major ideas, like the suggestion presented at one of the meetings that the method for extending core should mimic the Macintosh operating system, seemed to be ignored completely. As the summer wore on, edits to documentation dwindled until the point where no edits were made to the Operating System Interfaces section or to the Message Management section after they were reviewed.

Even though the team continued to have bi-weekly meetings, there were no further changes made to the System Requirements Document after August 1st. The requirements team started a series of two “tiger team” meetings with the API team to determine the life between the core and applications. Documents were written after this meeting, but not by the requirements team. The SRD was as a document, for all practical purposes, abandoned. The project would not start working on unifying document again for several months.

4.1.3 Comparison

In contrast to the period described above, the next month was characterized by a profusion of smaller, informal documents that were not reviewed or controlled. Overall, there were six customer interviews, two system architecture documents, and four white papers written in the same amount of time as the SRD review. The customer interviews were transcribed laboriously. Each transcript entailed hours of reviewing the video-tapes. In contrast to the System Requirements Document which tended to have a large number of place holders 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 core platform. The content in these documents helped the requirements team to start answering the questions that had plagued them in the early part of the summer.

Based on these interviews, the requirements team was able to specify a number of troublesome areas and recommend subteams to study these areas. Each sub team was to produce a white paper examining the options. Once again, these white papers were not vacuous. But they were also not reviewed or controlled until they were combined with a wrapper document that tied all the white papers together. This conglomerate document was intended to be the next requirements document.

Interestingly, the original SRD was not referred to in this second requirements process, and different members of the team perceived the contribution of the SRD differently. Most of the engineers felt that the effort on the first SRD had been spinning their wheels. As one engineer said: "We didn't use much of what was in the requirements document." Management, however, reported that large bodies of text had been cut from the original SRD and pasted into the later document. Our analysis of the documentation shows that the engineers were right about the text, but wrong about the ideas. Very little text actually got carried over, providing ample opportunity for loss of information, but many of the significant ideas explored during the writing of the first document show up transformed in the second. The effort that went into the first document was certainly not wasted.

4.1.4. Conclusion: An amnesic process

Discussions on certain problematic issues recurred, often without team members' recognition. Although groups met regularly (in the case of the requirements team twice a week), they kept no meeting notes that would have helped them recover previous discussions. In addition, the matrix organization led to parallel attacks on same problem. Cross-staffing among teams reduced this redundancy of effort to some extent, but teams often worked on the same problems without knowledge of the other's activities. The emphasis on completed products, rather than on informal documentation, tended to further isolate teams since documents were not controlled until reviewed; informal notes and annotations were seldom shared. The issues most affected by the absence of an effective organiational working memory included the allocation of functionality between applications and Centauri, process issues such as documentation standards, team responsibilities and leadership, and customer identification.

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. These are each described below.

4.2.1 .Responsibility for decision making

Decision making in Centauri was dispersed among the subteams. The matrix organization and cross-staffing of the subteams arose out of a desire to make the designers closest to the problems responsible for technical decisions. Many issues, however, fell somewhere between being technical and strategic; subteam members often waited in vain for management to make up its mind, while management waited for the subteams to make faster progress.

The absence of a "superdesigner" during this phase of the project further compounded the lack of convergence on 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 decide where key functionality was to reside (in Centauri or in the applications) was significantly blocking progress. In response, several teams spawned subteams to discuss that issue and hold joint meetings as a "tiger team," with the goal that their recommendations would be elevated to a strategic level. It was not until the sub-teams were formed that significant progress was made. However, the process involved much thrashing and loss of time between organizational levels while the decision making responsibility migrated from the internal teams, to their sub-teams, to upper management, and finally back down to sub-teams again.

4.2.2. Team Membership

The staffing and team allocation on Centauri was fluid and encouraged the informal cross-fertilization of design ideas. 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 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 teams. Management generally approved and help 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.

Eventually, sub-teaming became such a common method for solving difficult issues that a process was written on how to be a sub-team. It included recommendations that the goals of the sub-team be clearly stated up front, that exit criteria be set and an evolution for the outputs be managed. 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 in a swarm. 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 multiple subteams led to a diffusion of responsibility for the core ideas of the system and a dilution of the resulting vision.

4.3. Concreteness and the Effects of Vagueness

4.3.1. Early commitment to detail

The history of the project reflects an architecture-centric view of Centauri and a desire by all concerned 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.

Next, the requirements document was incrementally reviewed, and, as we showed in Section 4.1.2, it was only modified slightly during this process. 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 reconceptualize 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 over the categories listed in Section 2. The data are broken into three groups: the first 10 review meetings (up to mid-August, 1994), the next six meetings, and the eight meetings of the architecture sub-team with application experts (contemporaneous with the second group of six review meetings). Issues about the process being followed and the documentation standards dominated the review meetings, amounting to more than one third of those raised in the earlier set and over one half of the later set. 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 much as a result of the substantive issues also raised.

The team’s attention to detail acted almost like a shield, letting the team ignore its concerns about overall 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. Envisionment of applications

Recall that system was middleware, and so had no direct end-user, but rather should be generic for a range of applications. How broad the application coverage should be and what functions should be provided in the Centauri core and what would be continued to be supplied and reimplemented in the applications were recurrent issues (see Section 4.4). One of the pervasive impediments to the Centauri conceptual design activity was not so much that there was no customer, and therefore no definitive source of requirements, but that Centauri is an infrastructure and there were no clear requirements distinguishing the applications that would use it.

It became important to the team to anticipate the first few applications so that it could accurately understand the requirements for the infrastructure. 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 Centauri architecture, they very soon started talking about the “First Application” as if it held secrets that they needed revealed before they could make any 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 offline 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.

Members of the team had eagerly dived into detail regarding the requirements 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 management made a decision. The result was a mild paralysis and a replacement of design work with process busywork. When the first application eventually committed to use Centauri, its specific needs might easily have been anticipated earlier by studying any of several existing systems as if they were to be Centauri clients.

4.3.3. Scenarios

A form of concrete design thinking that we expected to observe was the use of scenarios to refine and validate general design ideas. 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, the use of scenarios was uncommon.

Not only was the documentation general, a property encouraged by the standard adopted, but the discussions of the requirements did not tend to occur at the level of concrete cases or exceptions. Most of the review discussions we recorded were driven by general questions, not imagined use. When specific scenarios came up, they were not named or referred back to later. For example, during one meeting the team discussed the need for applications to set timers. The discussion included a scenario that further explored the issues of clock resolution and the effect that the event notification mechanism would have on the system's architecture. Apart from temporarily clarifying some hitherto cloudy issues, this discussion was not returned to and did not materially affect the contents of the requirements document because the insights arrived at, being too concrete, did not belong there.

4.3.3. Process improvement

The company has a strong process quality culture. Engineers have a tradition of following standard processes. They will follow willingly processes that constrain their creative freedom provided that there is a strong management commitment to quality measurements and there is some evidence that the process they are asked to follow will have the desired result. Accordingly, the Centauri project set up both a process subteam and a development environments subteam that were responsible for defining development processes and tool requirements. All the previously documented and adopted processes in the organization 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 process subteam in consultation with project management therefore recommended that Centauri follow an external standard (IEEE 1074), with provision for a "conceptual exploration" phase prior to requirements specification.

The original architecture document (which predated the subteam epoch) had many of the characteristics of a software requirements document, 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 periodically acknowledged, more closely met the company standard for a software functional specification than it did the IEEE 1074 recommendations for conceptual exploration.

The requirements subteam spent a lot of time during their meetings discussing what type of document they were writing and what the standard meant. Throughout the summer, about half the issues they discussed were about their process, document structure and goals. In contrast, the architecture subteam formed later only spent about 20 percent of its time on these process-related issues.

The extraordinary focus on process design that we observed in most subteams of the project and throughout our period of involvement 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 responding to 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.

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 process they were supposed to follow.

A look at the staffing of the process and development environments subteams bears out the skepticism that many designers expressed about the relevance of the Centauri development process. Although management encouraged inter-team cross-fertilization by assigning the same people to several subteams, in the case of the two process-related subteams.

> *"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."*

In conclusion, even in as process-conscious culture as Centauri, the process and development environments subteams were seen by many designers as external advisors who unaware of the constraints of the real design process. Much of the work of these teams 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 widely acknowledged to be necessary and thus already part of the company culture. The process definition effort had little effect during the conceptual design activities, and so the project floundered initially.

4.3.4. 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 detail and precision, only to find that it had not developed sufficiently the goals and abstractions on which these details would depend. For example, the requirements team submerged itself in detailed documentation before the goals for the system were really understood; and similarly the process teams' effort to standardize the project's process 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 Centauri to be a generic system 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 broadly system goals and options.

4.4. Architectural Interfaces

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

4.4.1. Architectural vision

Architecture and requirements were inextricably intertwined because the scope of the requirements for Centauri directly affected the thickness of the filling of the architectural sandwich and the nature of the interfaces on both sides of the filling. Conversely, constraints afforded by the available operating systems and preexisting applications would affect the requirements. At one extreme, Centauri could be seen as a grandiose virtual machine for its application domain, so that applications would instantiate it and plug in extra, application-specific functionality. At the other extreme, Centauri could be seen as a toolkit for application developers, consisting of loosely connected services that they might or might not use. In the absence of a clear technical leader for the project 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.

During our involvement in the project, no single person or team was responsible for drawing up a single architectural vision, and at no time were competing architectures set side by side and analyzed. Later, a series of meetings were held that 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."*

Earlier in the project there had been a prevailing background discussion about the architecture, but our analysis of the issues that arose in meetings obscures this issue in the details (this is a good example of the need for triangulation methods in this type of empirical study). The issue was essentially whether Centauri should be big and fat or clean and lean. Ultimately, this issue was decided in favor of the clean and lean alternative, but only when external architecture experts intervened. The eventual architecture was prefigured in one of the requirements meetings when one of the participants (probably the quietest) had the insight that Centauri should be for its application what the Macintosh interface was to end users. This view was unheralded at the time.

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 function allocation (for example, application-specific device handling) to the core and the API and OS interface components. Many of these issues arose from different conceptions of what architecture diagrams denoted:

> *"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. An important requirements 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 OSs, 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 one way or the other, 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 the general and detailed, the factual and the optative, and the substantive and the meta-level was very typical.

4.4.3. Architectual reasoning

Many different looking architectures were proposed during the conceptual design phase. These were documented as informal (though frequently elaborate and colorful) diagrams. Unfortunately, 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 by such terms as "the cloud diagram" or the "bubble diagram," referring to the most salient (but superficial) visual properties of the diagrams.

In general team members tended to talk about the architecture in spatial terms. The layout of diagrams really mattered even though the diagrams had no formal meaning and the people invented notation as they went. Their 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."

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

4.4.4. Conclusion: Architecture as vision

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. 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 (often spanning requirements, implementation and process issues).

5. Discussion and Implications

We now summarise the results and conclusions from this study, state what the implications are 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 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 in the process. We suspect, however, that punctuated and occasionally nonmonotonic progress is inevitable in the early design process and should be planned for.

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 amnesic 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 amnesic 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.

5.1.3. Aloofness from context

Perhaps the most pernicious general phenomenon of the three 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 team was developing. This failure to get down to brass tacks may seem ironic in an engineering community, but it is not really that odd given that Centauri was conceived as a middleware system without direct end-users and that many of the developers were inexperienced at starting a completely new project. Once more, we need to stress that this was not a inadequacy of the Centauri project, 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 commitments) 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).

5.3. Recommendations

Several procedures and tools could have helped Centauri and by extension should be of value in similar design settings. 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, and so we concentrate on low-tech interventions.

The first type of improvement would be to externalise the team's working memory and thereby reduce the likelihood and impact of amnesic teamwork practices. One way to accomplish this might seem to be the use of design rationale tools, such as gIBIS [REF], or structured notetaking and multimedia indexing systems, such as Synthesis [REF] and Where Were We [REF]. We have tried to "get designers to use" design rationale tools before, and attempted briefly to encourage the Centauri team to use Synthesis. (Actually, they did in a very limited way, but that is another story). But these tools never seem to work as well as they should; in part because of theoretically uninteresting but practically significant limitations of any research prototype, but in part because such tools (design rationale tools especially) defer the benefit of their use [REF] and force people to structure their ideas and collaborative processes in ways that they may think artificial [REF]. A better first stab at externalizing the team's working memory would be some simple organizational and meeting management practices. Centauri team members often took notes at meetings, but these were very sketchy and the team did not collate their notes as a joint product. Simply having one team member act as scribe and have a summing-up 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.

Nonmonotonicity of progress was first reported by Guindon [REF] 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 the general to the concrete, these designers were making slow, painful and frequently punctuated progress. Really, however, they were coming to understand the problem better through their nonmonotonic process. However, what happens in the mind of an individual designer during a two-hour experimental session cannot parallel too closely what happens in a multi-person team over the course of several months.

The absence of a single architectural visionary, already referred to as a “superdesigner,” echoes Curtis et al’s. [REF] finding that successful projects usually have such a person in the team. When Centauri floundered most it did so because it lacked such a person.

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

Understanding the customer is important. As Carmel and Keil [REF] show, there is no substitute for direct access to the customer; intermediaries and analysts may introduce noise. Centauri’s potential customers were future application projects, and the Centauri team had no mechanism for reliably tapping the application expertise, even though such expertise existed within the company and did not relate sufficiently well the requirements they were developing to contexts of use. Having application experts on the team, as recommended in the previous paragraph, would be one way to address this issue.

A complementary strategy would be to organize the acquisition of customer requirements and application expertise around concrete scenarios or use cases [REFS]. As reported in Section 4.3, the detail of the SRD was largely illusory and the document did not survive. The requirements team became hidebound by the formality and structure of the document and were unable to step back and ask the questions about the requirements that they were writing: why is this required, and how would it work in practice? We believe that being able to refer to a number of standard usage scenarios would have made their work more effective and convergent.

Our final recommendation is perhaps the most controversial. Although we believe that product quality can be improved by standardizing design processes, most process improvements concentrate on regularizing clerical activities and responsibilities (such as documentation standards, configuration management, and formal reviews). For conceptual design, these types of control are heavy-handed and counterproductive. What are needed instead are lightweight process enhancements that emphasize the production, dissemination, and timely destruction of the interim results of *work in progress*. The place to look for these ideas is not the software engineering literature, but the popular creative design literature, and the technologies likely to be most useful are large whiteboards and post-it notes.

Acknowledgements

References

References

[1] Baecker, Ronald M., Groupware and Computer-Supported Cooperative Work: Assisting Human-Human Collaboration.

[2] Bergland et. al. "Improving the Front End of the Software-Development Process for Large-Scale Systems", AT&T Technical Journal, March/April 1990.

[3] Blomberg et al " Ethnographic Field Methods and their Relation to Design", 1993.

[4] Boehm, Barry W. "A Spiral Model of Software Development and Enhancement", IEEE Software, May 1989.

[5] Boehm, Barry W. Software Engineering Economics. Prentice-Hall, 1981.

[6] Christel et. al. "AMORE: The Advanced Multimedia Organizer for Requirements Elicitation", Technical Report, CMU/SEI-93-TR-12 , June 1993.

[7] Curtis, B., H. Krasner, and N. Iscoe (1988), "A Field Study of the Software Design Process for Large Teams," Comm. ACM, 31(11): 1268-1287.

[8] Dahlbom and Mathiassen, Computers in Context: The Philosophy and Practice of Systems Design. NCC Blackwell, Cambridge: 19??.

[9] Dobson and Strens, "Organisational Requirements Definition for Information Technology Systems", IEEE Computer, 1994.

[10] Eason, Ken, "Information Technology and Organisational Change." Taylor & Francis, New York, 1998.

[11] Fafchampts, Danielle. "Organizational Factors and Reuse", IEEE Software, Sept. 1994.

[12] Fagen, M.E., “Design and Code Inspections to Reduce Errors in Program Development,” IBM Systems Journal, Vol. 15, No. 3, 1976. pp. 182-221

[13] Goguen and Linde "Techniques for Requirements Elicitation", IEEE Software, 1992.

[14] Grudin, Jonathan, "Computer-Supported Cooperative Work: History and Focus", IEEE Computer, May 1994.

[15] Grudin, Jonathan, "Eight Challenges for Developers", Communications of teh ACM, Jan. 1994.

[16] Heninger, Kathryn L. "Specifying Software Requirements for Complex Systems: New Techniques and Their Application", IEEE Transactions on Software Engineering, Vol. SE-6, No. 1, January 1980.

[17] Herbsleb, James D. et. al. "Object-Oriented Analysis and Design in Software Project Teams" [unpublished].

[18] Holtzblatt and Jones, "Comtextual Inquiry: A Participatory Technique for System Design", 1993

[19] Hsi, Idris and Colin Potts, "Integrating Rationalistic and Ecological Design Methods for Interactive Systems", [unpublished].

[20] Kuwana and Herbsleb, "Representing Knowledge in Requirements Engineering: An Empirical Study of what Software Engineers Need to Know", IEEE Software, 1992.

[21] Lubars, Mike, Colin Potts and Charles Richter. “Developing Initial OOA Models.” The 15th International Conference on Software Engineering, Los Alamitos, CA, IEEE Computer Society Press, 1993. p. 255

[22] Luff et al "Work, Interaction and Technology: The Naturalistic Analysis of Human Conduct and Requirements Analysis", 1994.

[23] Markus and Keil, "If We Build It, They Will Come: Desiging Information Systems That People Want to Use", Sloan Management Review, Summer 1994.

[24] Moran, Thomas P and John M. Carroll. Design Rationale: Comcepts, Techniques and Use. Lawrence Earlbaum Associates, 1995. [in press]

[25] Olson, McGuffin, Kuwanna and Olson, "Designing Software For A Group's Needs: A Functional Analysis of Synchronous Groupware," User Interface Software, 1993.

[26] Onoe and Kuwana, "Communication analysis of Argument Structure-based Collaborative Design", NTT Technical Report.

[27] Perry, D. E. and Michael Evangelist, "An empirical study of software interface errors," Proc. International Symposium on New Directions in Computing, Trondheim, Norway, 1985, 32-38.

[28] Perry, D. E. and Michael Evangelist, "An empirical study of software interface errors: An update," Proc. Twentieth Hawaii International Conference on System Sciences, Kailua-Kona, 1987, 113-126.

[29] Potts et. al. "Inquiry-Based Requirements Analysis" IEEE Software, March 1994.

[30] Potts et. al., "An Evaluation of Inquiry-Based Requirements Analysis for an Internet Service," [forth-coming in REI '95]

[31] Potts, Colin "Software Engineering Research Revisited", IEEE Software 1993.

[32] Potts, Colin, Al Badre and Jay David Bolter. “Synthesis: A Computer System for the Rich Recording and Indexing of Collaborative Ideas.” 1993.

[33] Potts, Colin. “Invented Requirements and Imagined Customers: Requirements Engineering for Off-the-Shelf Software.” Second IEEE International Symposium on Requirements Engineering, Los Alamitos, CA, IEEE Computer Society Press, 1995. p. 128

[34] Randall et al "Steps towards a Partnership: Ethnography and System Design", 1994.

[35] Sakamoto and Kuwana, "Toward integrated support of synchronous and asynchronous communicatino in cooperative work: an empirical study of real group communication", ACM COOCS 1993.

[36] Schon, Donald A. The Reflective Practitioner: How Professionals Think in Action. Harper Collins Publishers, 1983.

[37] Simon, Herbert A.,

[38] Sommerville et al "Integrating Ethnography into the Requirements Engineering Process" 1993.

[39] Truex, Duane P. and Heinz K. Klein, "A Rejection of Structure as a Basis for Information Systems Development", Collaborative Work, Social Communications and Information Systems, 1991.

[40] Waltz, Elam and Curtis, "Inside a Software Design Team: Knowledge Acquisition, Sharing, and Integration." CACM, Oct. 1993. Vol. 36, No. 10.