MAKING CUSTOMER-CENTERED DESIGN WORK FOR TEAMS

Karen Holtzblatt and Hugh Beyer

Building today's systems requires a more intimate understanding of users' work than ever before. Computers are smaller and more common and interfaces are more powerful. Today, many users of computers neither know nor wish to learn how computers operate—they merely wish to get their jobs done.

In addition, vendors are under increasing pressure to develop innovative products quickly. To be innovative means to address important needs in new ways, and since existing products cannot act as models, guidance must come from users themselves.

As the industry has recognized these challenges, practitioners are looking for new ways to involve the customer closely in design. This has resulted in such approaches as Joint Application Design (JAD) [23], user-centered requirements analysis [15], user-centered design [18], and many PD techniques [8, 20] including our own contextual design [1].

These *customer-centered design* approaches make the customer, and the understanding of the customer, the center of design activities. The two primary questions such approaches address are:

- *How do I understand the customer?*
- *How do I ensure this understanding is reflected in my system?*

Understanding the customer is difficult. Design teams need extensive, detailed information about customers and how they work to build systems that support them well. The first requirement on any customer-driven process is to build awareness of the customer into the design team, and continue providing customer feedback throughout the life cycle.

But even given customer data, we have found it is still difficult to build a system in response. It requires a series of conceptual leaps to go from facts about the customer to a system design. How can we turn facts into a system we know will be useful?

Finally, no system is built by a single individual, but the quality of the system is the result of individual actions. How do teams develop the *same* understanding of the customer, and the *same* vision for the system? How can we manage the interplay between people to that end?

Contextual design is our approach to bringing customer data into design through a well-defined sequence of activities. The foundations for contextual design were developed in 1988 [22]. We have since used and extended this process in developing hardware and software products, in both small groups and large, at multiple companies.

Here, we summarize our experience with customer-centered design. We describe the steps we have refined through our work on design problems. We describe the reasons for each step, and draw out the implications for managing the design process.

Our first concern is to incorporate valid, useful data about how people work into the engineering process. The system we provide will support and constrain the way people work [10]. We need to understand work in enough detail to know what the system must do to support it well, and what innovations will streamline the work.

Finding out about work is difficult: not only are developers building for users doing unfamiliar work, but users themselves have difficulty describing what they do. People are adaptable and resourceful creatures—they invent many workarounds and quick fixes to problems, and then forget they invented the workaround. Even the details of everyday work become second nature and invisible.

The users cannot describe what they really do because they are not conscious of it and do not reflect on it. The defined policy for an organization is no longer representative because it no longer reflects what is really going on.

Contextual Inquiry

How can we get detailed information about how people work when they cannot articulate it on their own? Holtzblatt's approach[1] was to adapt ethnographic research methods to fit the time and resource constraints of engineering. The result was the first step of our process, Contextual Inquiry.

Contextual Inquiry provides techniques to get data from users *in context:* while they work at real tasks in their workplace. In a contextual interview the interviewer observes the user at work and can interrupt at any time

[1]Contextual Inquiry was developed by Karen Holtzblatt in 1986. Sandy Jones assisted in developing the first course on Contextual Inquiry in 1988. Since then, Holtzblatt and Beyer have built on Contextual Inquiry to address the full design process.

and ask questions as an outsider: "What are you doing now?" "Isn't there a policy for this?" "Is that what you expected to happen?"

Confronted at the moment of doing the work, users can enter into a conversation about what is happening, why, and the implications for any supporting system. The user and interviewer discover together what was previously implicit in the user's mind. Talking about work as it happens, artifacts created previously, and specific past projects reveals the user's job beyond the work done on that day.

A contextual interview usually takes from two to three hours. Typically, several members of a design team interview several customers at the same site simultaneously, providing a view across a whole organization in half a day.

We recommend that a product's designers conduct interviews. Great product ideas are derived from a marriage of detailed understanding of a customer need with an in-depth understanding of technology. In our experience, the best products happen when the product's designers are involved in collecting and interpreting customer data.

This field-gathering technique has been extremely successful at collecting detailed data about work practice which is hard to elicit any other way.

☞ *Gather data through interviews with your customers in their workplace while they work.*

☞ *Put the people making design decisions in front of the user.*

Involving the customer. What is the best way to involve the customers in the design process? We certainly want to build the best system we can for them. But we also want to optimize both development time and the customers' own time. As outlined by Muller in [17], customer-centered techniques tend either toward having the designer participate in the users' world, or having the users participate in design activities. We find both approaches useful, but want to ensure the user is as effective as possible in both roles.

When we participate in the users' world, we want it shown to us so well that we know it—we want our feet to be sore where their shoes pinch. Working with users in their workplace helps provide this familiarity. When they are working on or describing their real problems, users are much more eloquent than when talking in generalities. The impact of the real situation is much greater.

Conversely, when users participate in design activities, we want to make them strong participants in the design process.[2]

In our experience, customers are at a disadvantage when brought into a design meeting. The users' unique contribution is their real work experience. Taken out of this context, they are much less able to represent real experience [22]. Worse, because we want them to represent the user community, we ask them to discount their own actual experience. Instead of allowing them to stand for "what I need," we ask them to stand for "what all users would want." They become just another designer among designers.

Customers are at a disadvantage when building data models or other specialized models with the design team. This requires that they learn an unfamiliar language and translate their experience into this unfamiliar language. Even if we work from the users' artifacts, the language represents an abstraction of what they do. The user must translate it back into specific instances to understand what it means [7, 9].

Customers are at a disadvantage when brought into our laboratory and asked to work on an unfamiliar problem. Once again we take them away from the context that ties them to reality. We ask them to imagine what their work is like without any of the reminders they use daily to do their work.

Instead, in contextual design we build on our users' strengths by doing all our work with them in their own context, on their own problem (or get as close to this ideal as possible).

If we wish to validate a model of how they work, we do not show and walk through the model with them. Instead, during an interview about their own work practice, we respond to their descriptions of their work by drawing a picture. This picture is one of our models which incorporates what they just said about their own experience. It is a conversation aid, not something to be learned.

If we wish to codesign with users, we take a previously developed prototype to their workplace (as described in a later section). We invite them to work through their immediate work problem using the prototype. Users respond directly to the prototype as though it were real and give much better feedback than would be possible in a meeting room [13].

Even when we must use a laboratory for practical reasons, we recommend that users bring in their own work and try to do it in the lab. Even if we lose the context provided by their workplace, they are familiar with their own problem and it helps them reconstruct the missing context.

☞ *Use your users well. Let their own context strengthen them.*

Affinity Diagrams

As an interviewing process, Contextual Inquiry successfully extracts data about customers' work. However, one developer talking to one user is insufficient:

- The whole team needs to understand what happened with the customer.
- The whole team, including the interviewer, must understand the implications for the design.
- Different people have different perspectives and will see different implications in the data.
- Data from multiple users must be brought together.
- A working team will typically have many demands on their time. Not everyone will be able to go on every visit.

To bring the team together, share the data, and develop interpretations on which the team agrees, contextual design includes an affinity diagramming process [4].

The team, or a subset, sits down together and goes over the transcript or notes of each interview, writing

[2] We are indebted to the work of Pelle Ehn, Kim Madsen, and others at Aarhus University for inspiring our approach.

We take (customers) away from the context that **ties them to reality. We ask them to imagine what their work is like** *without any of the reminders they use daily.*

facts about the user, interpretations design ideas, and questions on Post-It notes. After the first round of interviewing is complete (usually 5 to 8 interviews or 400 to 600 notes), the team organizes the notes into clusters on a wall. These clusters are named and collected into higher-level groupings. (This entire process is fully described by Holtzblatt and Jones in [10].)

An effective affinity avoids using standard categories to cluster notes. We ban terms like "usability" or "quality," forcing the team to think deeply and creatively about the data, and making the name of the group represent what is really there.

For example, an affinity we built to understand object search mechanisms has a top-level note labeled "The user's purpose." The cluster names beneath it tell what the user's purpose in searching the object system might be: "Find a particular object," "Understand the structure of the system," and "Reuse existing objects." Under each of these headings is the cluster of individual notes defining the category. Later, we could read the affinity, understand these as the user's three primary motives, and ensure our design supported each well.

Group interpretation allows other members of the team to be brought back into the conversation. On real teams, it is rare that everyone can be in every meeting. By participating in interpretation sessions or in building the affinity, team members can be brought back into contact with the customer and can also provide their own unique perspectives on the data. When done, we 'walk' the affinity, saying what each part is about and brainstorming design ideas for that part. These ideas can be attached directly to the affinity itself. Later, when we pick up these ideas to develop, they will be directly tied to the customer data that sparked them.

An affinity captures our insight into the customers' work. The cluster names represent this insight and tie it back to data from individual customers through the notes in each cluster. The affinity organizes data across multiple customers and shows where the data is weak.

☞ *Interpret customer data together, as a team.*

The think tank. We prefer to dedicate a room to the team design effort. We are writing down an enormous amount of information about the customer. The affinity diagram and work models (described in the next section) represent everything the team has discovered, structured for easy understanding. Keeping them on the wall means the team is literally surrounded by its data about their customer.

Given the opportunity, the team will continually return to this data throughout the design process—it is common in our meetings for a team member to gesture or walk to a part of their affinity to support a design idea. It is difficult to achieve this kind of fidelity to the customer when the data about the customer is tucked away, out of sight.

The room also acts as a living record of the design process. A team member or manager who wants to catch up can browse the walls on their own, or another team member can use the walls to tell them what has happened. One manager told us he prefers to use the room to learn how the team is doing—he found it more immediate and more real than a status report or presentation.

☞ *If you want your team to be creative, give them a room.*

Work Modeling

The affinity organizes our data in a way that is easy to understand, and captures all the detail well. But to understand the structure customers put on their work, we also draw *work model* diagrams, showing the work of a single person or of an organization. They explicitly represent roles, flow of communication and information, work tasks, steps, motivation, and strategy of the work. Where there are problems in the work, they are shown directly on the model. Unlike a list of findings, requirements, or wishes, work models show how all aspects of work relate to one another.

We find four types of work models to be generally useful:

Context models (Figure 1) show how organizational culture, policies, and procedures constrain and create expectations about how people work and what they produce. Context work models represent standards, procedures, policies, directives, expectations, deliverables and other constraints. The context model shows what part of the work can be changed by introducing new technology, and what changes affect people or organizations who are not customers. Changing their work is always more difficult. Where an organization has standard procedures, we can design the system to support them directly, automating where possible.

Physical models (Figure 2) represent the physical environment as it impacts the work. To the extent they can, people structure their environment to support the work; then they work around any problems put in their way by limitations in the physical layout, location, hardware configuration, or technology. Physical work models show the physical space and systems that affect the work. Physical models reveal whether the work is split between locations and the sys-

tem could simplify work through direct communication. They reveal whether the work involves moving around, and whether the system must also move or must provide artifacts that move. They also show the range of hardware, software, and network platforms the system must support.

Flow models (Figure 3) represent the important roles people take on. A role is a set of responsibilities and associated tasks for the purpose of accomplishing a part of the work. Roles may be formal or informal, growing out of the work itself. One person usually fills several roles, and roles can be filled by several people. Each role represents a different type of customer of our system. When users interact with a system they are trying to meet the responsibilities their roles define. The flow model shows what is needed and what is supplied in filling a role. Flow models also show the communication and coordination between roles, and the flow of artifacts between roles. The flow model shows communication across a whole work domain, not only among current users of a system. This reveals new, unrecognized roles that could be supported by a system. It also shows the needs of people who will never be direct users, but depend on the system for information. With this knowledge, the team can build a system that better supports them.

Sequence models (Figure 4) represent the sequence in time of actions for specific important activities. A sequence model can be focused on the coordination of activities across individuals, on the thought steps and strategies of a single individual in doing one activity, or on the steps taken by a person in using a tool to accomplish an activity. (They are similar to flow charting and task analysis [5].) Sequence models show the specific tasks users perform. They define the work the system must support in detail. They show how the work can be simplified by removing, combining, or fixing steps.

Though these work models have proved most useful to us in design, we frequently develop new work models to represent important aspects of the design problem at hand. When we describe the work of customers with such explicit diagrams, we make our understanding of their work concrete. Without a picture, team members can hear the same words and understand them differently.

Teams argue when misunderstandings occur and when the only basis for decisions is opinion. A diagram makes disagreements explicit. Team members no longer have to argue with one another—they argue about the diagram and what is in it. It is clear where the diagram is incomplete and more data is needed. The result is to bring teams into a shared understanding more quickly.

A diagram takes a large amount of data and organizes it into a single image so it can be understood and used. Without such a way to incorporate the data, it is lost.

☞ *Use diagrams to capture your understanding of your customers' work.*

The use of language. Work models are a language for talking about work. People are not used to talking about work; what is there to say about it? What is important and what is not? A modeling language structures conversation by making certain concepts explicit, saying "talk about this." (See [16] for more on the use of diagrams as language.)

As a language, work flow models say: think about roles. Define what their responsibilities are. Define how each role communicates with others, and what they communicate. You must know these things to understand work. Someone building a flow model cannot help but ask questions about roles, their responsibilities, and how they communicate. The modeling language itself guides the designer in what to pay attention to. Conversely, anything the language cannot express is easy to ignore.

Other languages, such as data flow diagrams or object models, exist to support other conversations about systems. They make explicit the concepts needed to support these other conversations. For example, data-flow diagrams talk about the flow and transformation of data. These other languages do not support or guide the conversations we want to have about work.

This is why we introduce new modeling languages, despite the large number that exist. Thinking about work is difficult; thinking about how a system supports work is difficult. The languages we introduce in work models and in user environment design (UED) tell the designer what to pay attention to at each point in the process. No existing language does this for us.

We do not find that introducing new modeling languages confuses design teams. Our languages are simple—teams doing design work pick them up in a few minutes. We find it more powerful to introduce these languages than to make a mapping from an existing language to the concepts we are trying to express.

☞ *Let modeling languages help you. When you must, invent new ones to say exactly what you need to say.*

Work Redesign

Working with specific customers gives the team an understanding of the work of those customers. However, we want an innovative design that transforms work in new ways and is useful to all our customers. How do we invent such a transformation? How can we ensure we have transformed the work usefully?

This is a new conversation. Until now we have been talking about the work as it *is*; now we talk about the work as it *will be*, when our new system is in place. This is not a conversation you can avoid. *Every* system changes the work of its users. It is best to think about and design the effect you want your system to have explicitly.

We make this conversation explicit through *abstract work models* (Figure 5). We gather all the same kind of models together: all the flow models, all the physical models, all the context models, and the sequence models that address each task. Then we build new models of each type, removing the particular details of each customer's work and revealing its underlying structure. These abstract work models show the aspects of work our system will support. Anything the team chooses not to represent will not be supported by the system. This abstraction allows us

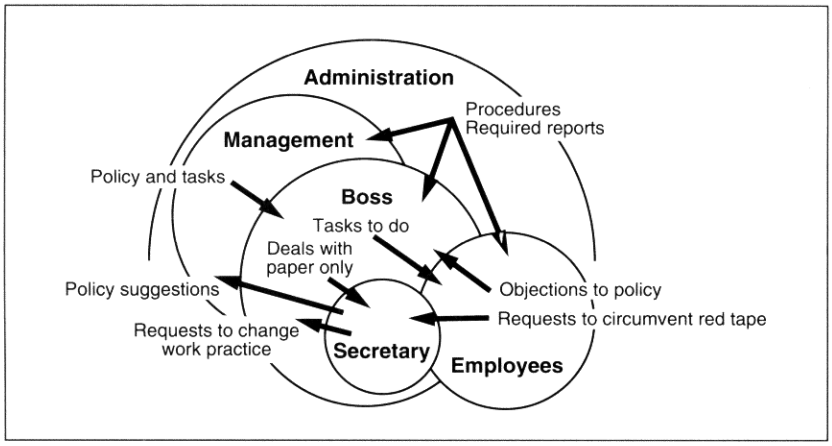


Figure 1. *Context model*
This and the following models are examples of work models describing the use of email. This partial model shows that the company's administrative groups constrain everyone by requiring certain reports and actions. The boss is also influenced by management requirements, and in turn sets requirements on the employees. The boss will not touch the computer, which affects what the secretary must do. The secretary asks the boss to change his work style to make the secretary's job easier; for example to keep the paper mail in the order in which it is given to him to make it easier for the secretary to enter his replies on-line.

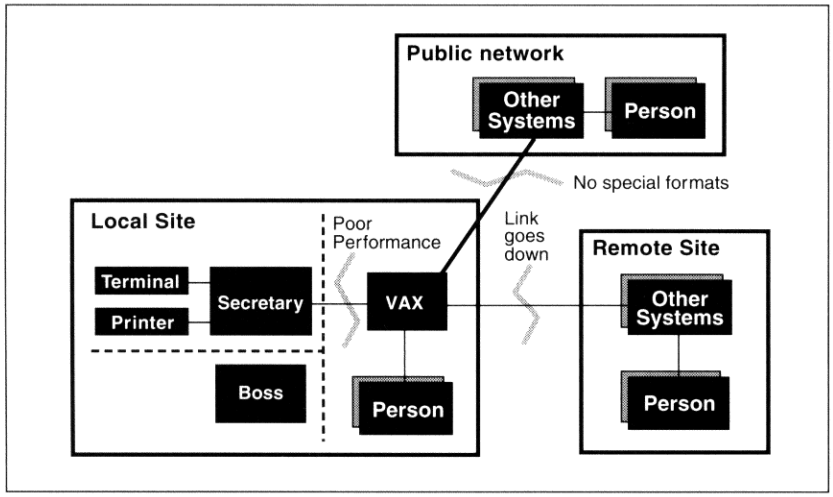


Figure 2. *Physical model*
This is the physical environment for our boss and secretary. Only the aspects relevant to our mail problem are shown, not the whole physical layout. The secretary has a printer in her own office. She shares a VAX with others. The VAX is overloaded and slow. The boss has no connection to the VAX—even if there is a terminal in his office, he never uses it, so it is not shown. The VAX is networked with other computers at remote sites. This network does go down, so the secretary uses a store-and-forward mail system. The VAX also has links to public networks which can only handle plain text messages.

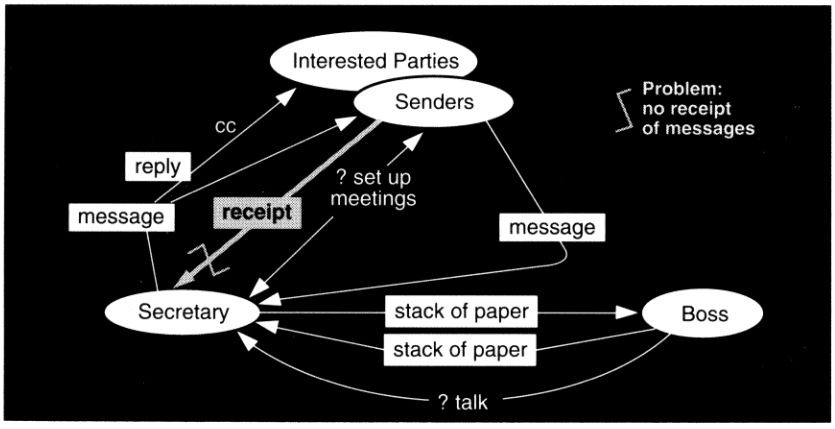


Figure 3. *Flow model*
This model shows the communication between people in the organization. Messages sent to the boss are intercepted by the secretary, who prints them and passes them to the boss on paper. The boss writes replies and gives them back to the secretary who sends the replies and other messages to the original sender and sometimes to others. Because the secretary uses store-and-forward mail, she has no way of knowing if the replies ever get through. We know the secretary is communicating with people by phone, but we do not know why—perhaps to set up meetings. We also do not know how she coordinates with the boss off-line.

to meet the needs of a whole market by building on what we discovered from individuals.

Our best ideas for improving the work often come from seeing how a particularly thoughtful person or group has solved their own problems. We build this solution into our abstract work models and our system, so all customers can take advantage of it. Once we have this consolidated model, we study it for problems and inefficiencies. We develop an abstract work model that brings together data from all customers, keeping good ideas, fixing problems, and using technology to combine and remove steps. When done, we have a statement of how our users will work, if we can implement the

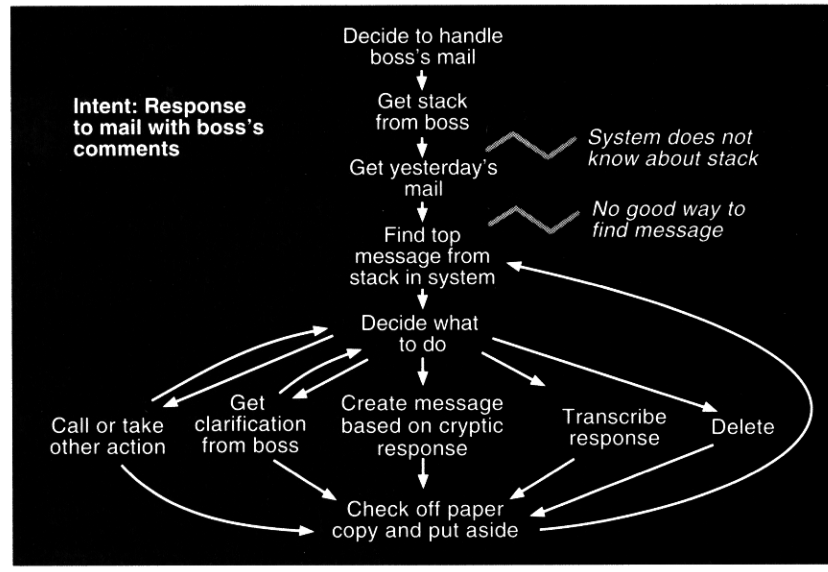


Figure 4. *Sequence model* This model shows the steps the secretary takes to answer the boss's mail from his handwritten replies. The secretary gets the stack of printed messages with the boss's replies written on them and works through the stack. The secretary may write a reply from the boss's written reply, get further clarification from him, take other action such as calling the sender, or delete the mail without doing anything. When the secretary has dealt with a message she marks the paper copy so she knows it is done.

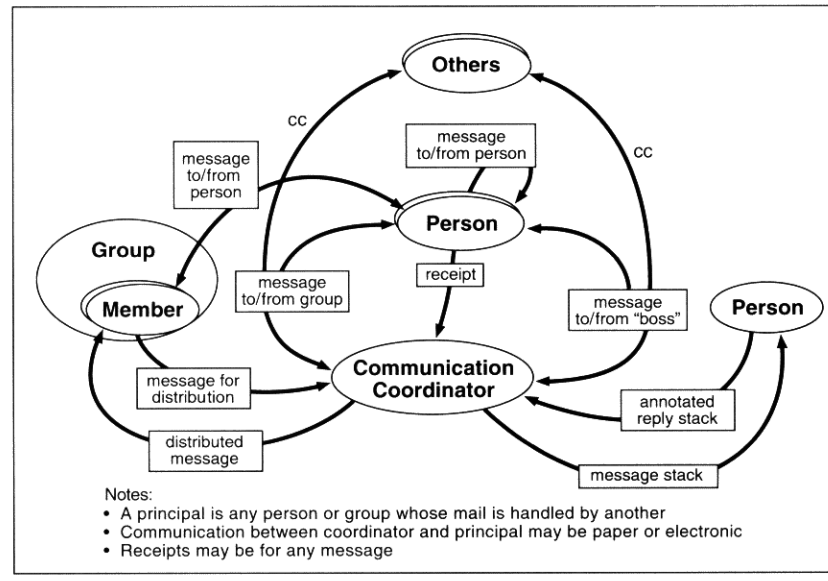


Figure 5. *Abstract flow model* This is an abstraction of the work of communicating, incorporating the boss and secretary as well as other data. The secretary's role in helping the boss communicate has been named "communication coordinator." Looking at other customers, we discovered group communication can break down when no one is handling it for the group. We borrow the idea of a communication coordinator from the secretary, and use it to solve the group's problem. The coordinator can manage a group's communications in the same way that secretaries manage their boss's. When supporting a group, the coordinator intercepts messages sent to the group as a whole and distributes them to individual members. Messages can still be sent to specific group members. The coordinator may be a group member playing both roles.

system to support it.

We validate our redesign of the work by checking it against the data from customers we have visited and through Contextual Inquiry with new customers. When interviewing new customers, we look for aspects of their work our redesigned work model cannot account for. These refine and extend the redesigned work model.

Making the work redesign conversation explicit ensures we do not do silly things unintentionally. For example, in creating a presentation, ideas move from slides to handout notes and back again as the creator tries different approaches to presenting the ideas. So a presentation system should support modifying slides and notes in parallel. Providing a notes facility that does not allow the slide to be changed, as some commercial systems do, is not enough.

We verify any design idea against the redesigned work model to ensure that it fits into the users' jobs well. We use it to see that the new work practice our system will support hangs together. We anticipate new problems our changes may cause, and prevent them.

Taken together, abstract work models are a coherent statement of who our customer is. We use this statement throughout the rest of the design process.

☞ *Design the way you want to change your users' work on purpose, or you will do it by accident.*

☞ *Your customers are your best source of ideas. 'Borrow' from them where you can.*

· ***Separating conversations.*** In our discussion of work redesign, we have an example of *separating conversations*. We will continue to do this throughout the design process. One major reason design meetings are difficult, contentious, or lack clarity is because the team is unknowingly mixing two concerns. Members of the team are not even arguing the same issue—it is no wonder they cannot agree.

For example, I might argue for a return receipt feature in a mail product, and you might argue against it as wasteful of system resources. As long as we conduct the argument as though it were about the system feature, we have no basis for decision. In fact, this argument is about the work, and how we will change it through our new system. Once we recognize this, we can easily discover whether the idea solves a real work problem. If so, we can have a separate conversation about whether we can implement it technically.

In this case we are separating the work redesign conversation from the implementation conversation. In the introduction of work redesign, we showed how we can separate the conversation about the work as it is now from the work as it will change.

Each time we wish to separate conversations, we provide separate physical places to have the conversations. We describe the work as it is now as a work model on one part of the wall. We describe the work as redesigned in a separate work model, on a different part of the wall. We define the system on yet another part of the wall.

It is always clear what conversation we are having because we are always referring to and modifying one model or the other. The wall space defines the conversations in progress. We can conduct several conversations in parallel, switching between them as necessary. This speeds up the process. (Running out of wall space is a problem. One team went on to use the ceiling. You can also start another layer, or start taking down conversations which are not immediately relevant.)

☞ *Always be aware what conversation you are having. Use the wall to keep conversations separate.*

Design of the System

Up to this point, we have thought exclusively about the customers and their work. It is rare for an engineering team to think so deeply or so clearly about the work of their users. But these conversations give us a solid foundation for designing the system.

We would expect a design team with other demands on its time to get to this point in about a month. Spending longer to gather data from more customers is not useful—the team should begin to see the overall structure of the work and have initial ideas for a system. These should be captured as they develop by starting an iteration of system design.

Our initial and primary concern in system design is to ensure that the structure of the system we deliver supports the work as we redesigned it. We want to be sure the system delivers the right functions and organizes them to let people work efficiently.

We could use prototypes, mockups, or sketches to represent this system structure. But we find they focus the team on the user interface (UI). They hide the structure of the system behind UI details, making it easier to talk about menus, icons, word choice, and screen layout than about whether the structure and organization are right.

User Environment Design

We must design the structure and function of the system, as experienced by the user. We must create this design from our knowledge about the customer. Surprisingly, most development approaches do not provide any support for this design step. (In this we agree with others who have recognized this missing step [11, 21].) Without it, we have no way to consider how the system hangs together as a whole.

To separate the conversation about system structure from the conversation about UIs, we introduce a new language for representing the system. We call this *user environment design*, because it defines the environment we give our users to work in (Figure 6).

The user environment design is a language for defining how the system will structure the user's work. What do we need to consider in designing the structure of work? Look at how people structure their own work when they get a chance:

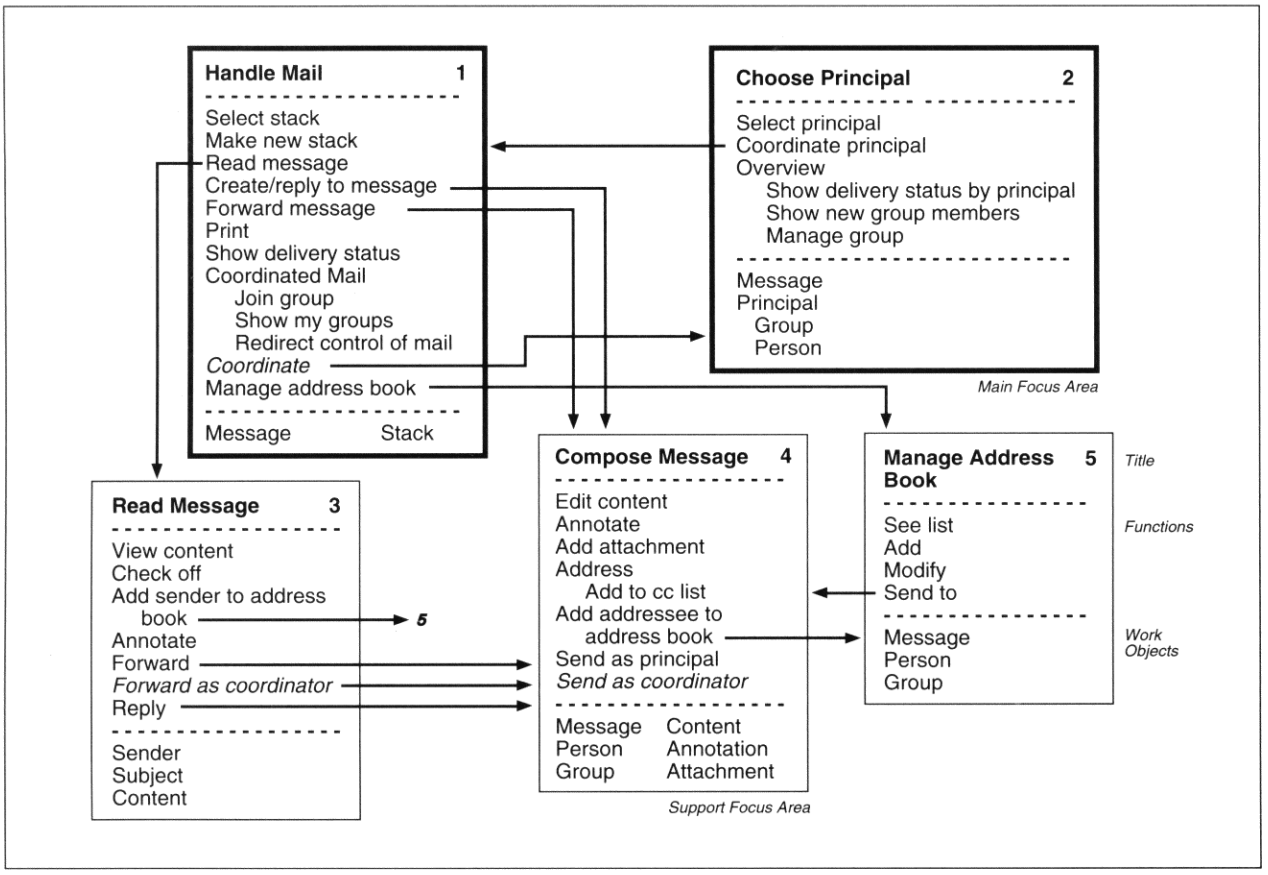
In designing a house, we dedicate rooms to different purposes. The kitchen, for example, is for cooking. We put the tools we need for cooking there: knives, cutting boards, an oven. We also put food there, because food is what we work on when we cook. We do not put unrelated things in the kitchen—they only get in the way. Even a related but distinct purpose such as eating is given its own place, a dining room. We just ensure that the dining room is handy to the kitchen.

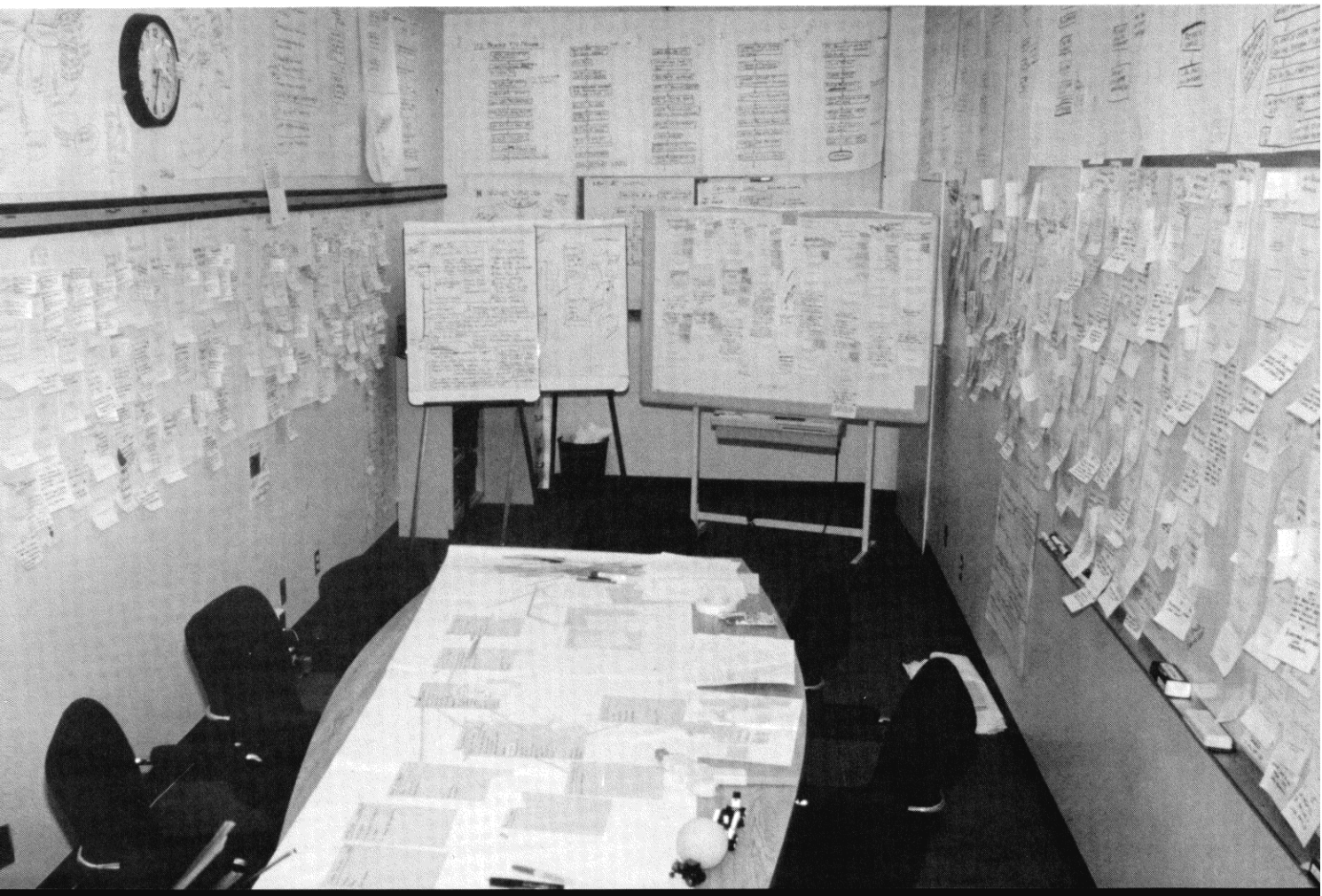
We see this same pattern in every area of life. We make a place to perform each activity. We collect the items we will work on, and we clear away anything unrelated. Related but distinct activities are kept nearby, but are not allowed to interfere with one another.

It is the same for a software system. We want to create a place for each coherent activity. We want to provide the functions needed to do this work. We want to clear away any functions not necessary to this work—they clutter the UI and confuse our user. And we want to provide convenient access to parts of the system supporting related activities.

Our redesigned work models define the coherent activities our system must support. In the user environment design we define *focus areas*, places in the system for focusing on each activity. Each focus area provides the functions and work objects necessary to do one activity.

We build user environment designs in paper, on a table. Each focus area goes on a half-size piece of paper; the flows are paper strips held down with tape. The size and physical nature of the design allow the whole team to gather around and manipulate it together. Paper is still the best medium we have found for quick change in a meeting.


Handle Mail 1
Select stack
Make new stack
Read message
Create/reply to message
Forward message
Print
Show delivery status
Coordinated Mail
Join group
Show my groups
Redirect control of mail
Coordinate
Manage address book
Message Stack
Choose Principal 2
Select principal
Coordinate principal
Overview
Show delivery status by principal
Show new group members
Manage group
Message
Principal
Group
Person
Main Focus Area
Read Message 3
View content
Check off
Add sender to address book
5
Annotate
Forward
Forward as coordinator
Reply
Sender
Subject
Content
Compose Message 4
Edit content
Annotate
Add attachment
Address
Add to cc list
Add addressee to address book
Send as principal
Send as coordinator
Message Content
Person Annotation
Group Attachment
Support Focus Area
Manage Address Book 5
See list
Add
Modify
Send to
Message
Person
Group
Title
Functions
Work Objects



Ideas not chosen are not lost—*in developing each idea,* **the groups are expected to pull together** *the best parts of all the ideas.*

Building a user environment design focuses the team on the appropriate level of detail. It allows us to put off low-level decisions about look and layout until after we have made fundamental decisions about structure. It defines the requirements on the implementation, provides initial objects for the system data model, and defines the structure of the UI.

Just as a floor plan allows the architect to see all the parts of a house and how they relate to one another, a user environment design allows the team to see all the parts of a system and how they relate. It shows the system as a whole.

Making the system structure explicit in a user environment design gives the team something concrete to make the conversation real. The team can test this structure by walking scenarios through it and fix usability problems before user interface design is even begun.

Figure 6. *Mail user environment design (UED)*
This is a partial user environment design supporting the abstract work model in Figure 5. Two main activities are defined: *Handle Mail* and *Coordinate Communication*. *Coordinate Communication* is the main activity of the coordinator. The coordinator must decide whose mail to handle and then start looking at it. *Handle Mail* is the common activity of sending and receiving mail, printing messages, forwarding messages and seeing the delivery status of messages sent. We have provided function to manage stacks directly, since this was part of the work of our users. Some work, such as composing a message to send, is sufficiently complex that we created a separate focus area so the user can focus on it alone.

A typical think tank. An affinity diagram is on the walls; work models are on the back wall. A user environment diagram is being built on the table.

☞ *Design your system as a system. Make sure it hangs together before designing the parts.*

Divergent thinking. Throughout the process we work to converge on a single, shared understanding of the customer. This is good for building team consensus, but can be bad for creative design. All this agreement and harmony can limit the range of solutions we consider.

To counteract this tendency, we include activities designed to encourage divergent thinking—thinking as widely and creatively about the problem as possible. The first time we do this is after the affinity, when we walk the affinity and brainstorm design ideas for each part. We do this again here, before starting the UI design. Our purpose is to bring the unconnected design ideas together into a coherent solution, and to investigate a range of such solutions before settling on one.

The team brainstorms ideas. As each idea is suggested, we write it on a flip chart, and the team develops it until everyone can see the possibilities. This takes 5 to 15 minutes for each idea.

The team then divides into groups of two to four people. Each group takes one of the ideas to develop. We choose the ideas by voting for those most likely to make a good base for a system design. We usually choose the top vote getters, but might choose an idea just because it is especially unconventional. Ideas not chosen are not lost—in developing each idea, the groups are expected to pull together the best parts of all the ideas. (We borrowed aspects of future workshops [12], Pugh matrices [19], and traditional brainstorming for this process.)

When the teams have developed their ideas for a half to a whole day, we come back together and each team presents their designs. We examine them for completeness and fit with the user environment.

Where designs overlap, we consider ways to merge them. Where designs suggest interesting alternatives, we do not try to merge them. Instead, we take them to customers and test them, providing better feedback for developing a single design after the visits.

This brainstorming process is good in several ways. First, it is fun. We develop a lot of ideas very quickly, and make lots of progress. It breaks us out of the mold we have allowed ourselves to set in. It allows us to quickly test several ideas for our system. And it allows us to think about the system as a whole before designing each part.

☞ *Stir things up now and again. Let your team go wide, then bring them back together.*

Mapping to the UI

The UI is the presentation of the user environment design on a given platform. Our primary concern in designing the UI is having it realize the user environment design and make working within the system convenient.

Each focus area describes a coherent activity and is represented as a coherent part of the interface. In a windowing system this often means a window, but may also mean a pane or area of a window. The UI should say to the user by its look and organization, "here is the place to do this kind of work."

We test the design with paper prototypes, inspired by Kyng [7, 14]. These are rough mockups of the UI which we take to the user's workplace and ask them to pretend it is a system and to work with it. They are trying out their real work using the prototype, so they can react as they would

to a real product. We observe and probe in the same way as a contextual interview.

We do not have to tell our users what level of detail they should respond to—the roughness of the prototype does that. If we present a prototype running on a computer, they respond to details of the look and the layout. If we present a hand-drawn prototype on paper, they respond to the structure and function in the system.

We start with very rough prototypes and encourage users to explore, trying to accomplish a task of their own. When they ask if the system does something, we design on the spot: "Yes. How would you expect it to work? Show me." The user sees the design is incomplete and open to change, and is drawn into the design conversation. (This requires designers to run the interview, to respond appropriately and to design with the user.)

This kind of rough prototyping tests our user environment design. We can see whether the structure and function we provide is useful. Users can respond to the prototype without learning the user environment language. The user environment design successfully predicts how users will react to a given interface. Where an interface is unfaithful to the design we have found that users reject it. For example, one interface we tested merged focus areas in the user environment design. The users' comments indicated they were rejecting it because the interface mixed unrelated work, just as predicted by the user environment design: "I don't want to know all that—take it away!"

As the user environment design stabilizes, we start to care more about the user interface. We build more careful prototypes, and in our customer interviews ask our users to live with the limitations of the system we designed. Finally, it becomes useful to build and test running prototypes that can evolve into the real system.

☞ *Structure your system first. Then make it real in the user interface.*

Iteration with customers. Customer iteration is a powerful team design technique. When we can produce an idea, develop it, prototype it, test it with customers, and validate, modify or discard it within 48 hours, we can stabilize a design very quickly.

When team members advocate different design solutions and the best is not clear from the customer data, it is often more efficient to prototype and test the alternatives than to try to reach consensus in the team. Team members let go of ideas more easily when they see users react badly to them than when another team member rejects them.

We use customer iteration from work modeling through system delivery. We visit new customers after building abstract work models to ensure the abstraction holds for them. We build rough prototypes of user environment designs to test that our system structure works. And we prototype the UI and early system versions to ensure we are being true to our design and have not broken it in the implementation.

Furthermore, the development process itself is iterative (as recognized by Boehm [2], Booch [3], and others). The insights we gain from working with users on prototypes cause us to modify our understanding of their work and our redesigned work models. We get quickly to an initial system design for a small part of the problem, but return to earlier steps to incorporate new information and to expand the system to new areas. The quick design of a part of the system gives the team a sense of accomplishment.

☞ *Iterate with your customers. Iterate, iterate, iterate.*

Conclusion

One participant in our design process said to us afterward, "It was cool, but it was also structured. I always knew what to do." Along with producing good results, this should be the test of any design process: it should make people feel they can be creative and move rapidly, but also that, at every point, they know what to do.

Too often, a methodology feels like a straightjacket. Structure need not conflict with creativity—in providing a clear path forward, the right structure should set people free to be creative. Too often, this does not happen. When we combine customer-centered design with creative team processes, we avoid this restrictiveness and good results can be achieved. ◘

References

1. Beyer, H. and Holtzblatt, K. Contextual design: Toward a customer-centered development process. *Softw. Dev. '93 Spring Proceedings* (Santa Clara, Calif. Feb. 1993).
2. Boehm, B. A spiral model of software development and enhancement. *IEEE Comput. 21* 5 (1986), 61–72.
3. Booch, G. Object-oriented development. *IEEE Trans. Softw. Eng. SE-12* (1986).
4. Brassard, M. *Memory Jogger Plus*, GOAL/QPC, Methuen, Mass., 1989.
5. Carter, J. Jr. Combining task analysis with software engineering for designing interactive systems. In *Taking Software Design Seriously*. J. Karat, Ed, Academic Press, New York, 1991, p. 209.
6. Ehn, P. and Kyng, M. Cardboard computers: Mocking-it-up or hands-on the future. In *Design at Work*. J. Greenbaum and M. Kyng, Eds., Erlbaum, Hillsdale, N.J., 1991, p. 169.
7. Ehn, P. and Sjögren, D. From system descriptions to scripts for action. In *Design at Work*. J. Greenbaum and M. Kyng, Eds., Erlbaum, Hillsdale, N.J., 1991, p. 241.
8. Greenbaum, J. and Kyng, M. Eds. *Design at Work: Cooperative Design of Computer Systems*. Erlbaum, Hillsdale, N.J., 1991.
9. Holmqvist, B. and Andersen, P.B. Language perspectives and design. In *Design at Work*. J. Greenbaum and M. Kyng, Eds., Erlbaum, Hillsdale, N.J., 1991, p. 155.
10. Holtzblatt, K. and Jones, S. Contextual Inquiry: A participatory technique for system design. *Participatory Design: Principles and Practice*. A. Namioka and D. Schuler, Eds. Erlbaum, Hillsdale, N.J., 1993.
11. Keller, M. and Shumate, K. *Software Specification and Design*. Wiley and Sons, New York, 1992.
12. Kensing, F. and Madsen, K.H. Generating visions: Future workshops and metaphorical design. In *Design at Work*. J. Greenbaum and M. Kyng, Eds., Erlbaum, Hillsdale, N.J., 1991, p. 155.
13. Knox, S., Bailey, W. and Lynch, E. Directed dialog protocols: Verbal data for user interface design. In *Human Factors in Computing Systems*

CHI '89 Conference Proceedings (Austin, Tex. May 1989), p. 283.
14. Kyng, M. Designing for a dollar a day. In *Proceedings of CSCW'88: Conference of Computer-Supported Cooperative Work* (Portland Oreg.) ACM, New York, pp. 178–188.
15. Martin, C. *User-Centered Requirements Analysis*. Prentice-Hall, Englewood Cliffs, N.J., 1988.
16. Martin, J. and Odell, J. *Object-Oriented Analysis and Design*. Prentice-Hall, Englewood Cliffs, N.J., 1992, 121.
17. Muller, M., Wildman, D. and White, D. Taxonomy of PD practices: A brief practitioner's guide. *Commun. ACM 36*, 4 (June 1993).
18. Norman, D.A. and Draper, S.W., Eds. *User Centered System Design*. Erlbaum, Hillsdale, New Jersey, 1986.
19. Pugh, S. *Total Design*. Addison-Wesley, Reading, Mass., 1991.
20. Schuler, D. and Namioka, A. Eds. *Participatory Design: Perspectives on Systems Design*. Erlbaum, Hillsdale, N.J., 1993.
21. Seaton, P. and Stewart, T. Evolving task oriented systems. *Human Factors in Computing Systems CHI '92 Conference Proceedings* (Monterey, Calif., May 1992).
22. Whiteside, J., Bennett, J. and Holtzblatt, K. Usability engineering: Our experience and evolution. *Handbook of Human Computer Interaction*. M. Helander, Ed., North Holland, New York, 1988.
23. Wood, J. and Silver, D. *Joint Application Design*. Wiley and Sons, New York, 1989.

CR Categories and Subject Descriptors: K.6.1 [**Management of Computing and Information Systems**]: Project and People Management—*training*; K.6.3 [**Management of Computing and Information Systems**]: Software Development; K.7.2 [**The Computing Profession**]: Organizations

General Terms: Management
Additional Key Words and Phrases: Contextual design, Contextual Inquiry

About the Authors:

KAREN HOLTZBLATT is co-founder of InContext Enterprises Inc., a firm specializing in coaching teams in customer-centered product design. Current interests include the creation of methods for design from customer data appropriate to varied organizations, products, and team structures.

HUGH BEYER is co-founder of InContext Enterprises, Inc. Current interests include the use of explicit customer-centered systems design to drive object-oriented methods. **Authors' Present Address:** InContext Enterprises, Inc., 30 Magnolia Road, Sudbury, MA 01776; email: {karen, beyer}@acm.org