ABSTRACT

Although virtual conferencing tools have been successfully used in executive meetings, current tools appear inadequate for the needs of software designers. As a result, while distributed code writing is becoming widespread, almost all software design meetings are still carried out face-to-face, incurring costs that undermine the potential of distributed software development.

Our research strives to build successful tools for supporting distributed software design meetings. To do so, we must first identify the unique activities of software design which must be mimicked in the virtual world. This paper does so with a detailed study of two colocated design meetings. We discuss issues that must be tackled in the transition to virtual settings, outline requirements for such tools, and propose strategies for meeting these requirements. In doing so, we also identify problems in existing colocated meetings which could be alleviated with these tools.

Note: Due to space restrictions, this version of the paper does not include the photographs which were taken during our observations. High resolution versions of these photos, along with a complete version of this paper, are available at:
http://www.uridekel.com/research/hsse05

1. INTRODUCTION

Distributed software development has become a necessity in many large-scale software projects. Globalization, outsourcing, increased telecommuting, and the open-source movement all demand a physical distribution of the software development process.

The notion of distributed development is mistaken by many for distributed coding. In truth, coding is a relatively decomposable process, and most distributed coding tools focus on preventing individuals from clashing and on reducing integration problems. Synchronous collaboration tools like instant messaging and shared editors are useful for distributed coding, but rarely involve more than two individuals, and are often a source of distraction which requires tools to mediate it. When more people are involved, asynchronous forms of communication are typically used; the individual nature of code development allows it to withstand the associated delays.

High-level software design is a different, yet crucial, part of the software development process. Its nature and overall effect on the project necessitates synchronous collaboration between multiple individuals. Even in large projects with rigorous use of documentation, mailing lists and other asynchronous forms of communication, there is usually a small group of individuals which conducts real-time collaboration throughout the lifetime of the project.1 At present, these collaborations almost always take the form of a face-to-face software design meeting (SWDM), whose outputs are typically informal diagrams, notes, and verbal agreements.

1.1 Motivation

Our long-term research goals are to develop collaboration tools which will allow SW engineers to conduct distributed design meetings, and to improve colocated meetings as a byproduct. To motivate this research, we must first see why it is necessary to enable distributed SWDMs to take place, and why they warrant special consideration.

Design meetings are a forum where critical decisions are made. Many of these decisions are undocumented mutual understandings, which participants later incorporate into their respective components or propagate to their teams. If crucial SWDMs are forgone due to cost or logistic constraints, the project faces significant risks from misunderstandings, which participants later incorporate into their respective components or propagate to their teams. Many of these decisions are undocumented mutual understandings, which participants later incorporate into their respective components or propagate to their teams. If crucial SWDMs are forgone due to cost or logistic constraints, the project faces significant risks from misunderstandings, which participants later incorporate into their respective components or propagate to their teams. Many of these decisions are undocumented mutual understandings, which participants later incorporate into their respective components or propagate to their teams. If crucial SWDMs are forgone due to cost or logistic constraints, the project faces significant risks from misunderstandings, which participants later incorporate into their respective components or propagate to their teams.

Executive meetings in the corporate environment have been successfully carried out in a distributed manner for a long time. Although not equivalent to physical meetings, such distributed meetings allow busy executives to successfully coordinate with many remote peers, with limited face-to-face contact and few travel expenses. Research continues to make the experience closer to that of real meetings [8].

We argue that the same tools will not allow SWDMs to be distributed in the same manner. Their adequacy for executive meetings arises from their ability to mirror the practices of such physical meetings: a group of individuals, taking turns in speaking or focusing on a person who is presenting

1This also applies to most open-source projects, as these often originate from the extension of either a traditionally developed system (e.g., Apache), the brainchild of a single individual (e.g., Linux), or the work of a corporate-supported team of architects (e.g., Eclipse).
materials or drawing on a whiteboard. To successfully distribute SWDMs, a Software-Design Conferencing Support Software (SDCSS) must allow the practices of design meetings in the Physical World (PW) to be mirrored in the Virtual World (VW).

Intuitively, SWDMs are different from executive meetings in many aspects, including heavier reliance on visual artifacts, loosely defined notations and protocols, more individual mobility, and different behavioral norms of the participants, etc. Our research attempts to concretely identify, validate, and handle these differences.

It is important to note that while there are many studies on distributing and virtualizing various kinds of technical collaborations, including software-related activities (see [7] for a comprehensive review), this work is, to the best of our knowledge, the first comprehensive study to focus on the intricacies of individual behavior during SWDMs rather than on the actual contents of these meetings. One should also note that our focus is limited to meetings in which new software is designed: Meetings where existing artifacts are reviewed for maintenance or inspection have different properties which have been investigated by others.

1.2 Scope of this paper

Our research applies contextual design techniques [2] to software design meetings. We observe such meetings to gain an understanding of current practices and identify problems, in order to establish requirements for an SDCSS which can support distributed SWDMs. In particular, we identify physical and social cues which are lost in the transition to virtual settings. In the future we will examine these cues in the context of HCI, psychology, and software engineering literatures, and propose solutions whose efficacy we will then attempt to experimentally validate by implementing prototypes and conducting user studies.

While our primary objective is to support distributed SWDMs, our studies revealed many deficiencies in the current practices of colocated teams. Many of these problems could be addressed by the same software support we suggest for distributed meetings. This is not surprising, as many of these problems arise from the limitations of the physical world. Thus, our goal is not limited to allowing existing SWDMs practices to be translated as they are into the virtual world. We believe that virtual SWDMs offer many advantages and opportunities over current colocated meetings, and suggest that certain practices of the PW should perhaps be abandoned in favor of those of the virtual one, effectively creating a virtual design meeting. However, in the same way that virtual reality should be capable of providing a convincing illusion of the real world before simulating imaginary worlds, an SDCSS must support existing practices before it can be widely accepted in a manner that makes offering new capabilities worthwhile. This paper is therefore focused on these more "mundane" aspects of the transition.

This paper demonstrates our research approach by analyzing observations of two software design exercises at the ACM DesignFest® event [1]. In each of these exercises, a small group of experienced developers tackled a given design problem for several hours. While these sessions are not necessarily representative or exhaustive, they uncover many important issues that are likely to apply in actual conditions. We are conducting additional observations to validate these claims.

Outline: The rest of this paper is organized as follows: Sec. 2 describes the design meetings which we observed. Observations on the utilization of drawing surfaces are discussed in Sec. 3. We discuss the formation of teams in Sec. 4 and the problems of pointing and maintaining focus in Sec. 5. Finally, we present our conclusions and current directions of research in Sec. 6.

2. SETTINGS

The observations in this paper are based on two design meetings which took place during the annual ACM DesignFest® event at the Object Oriented Programming Systems Languages and Applications (OOPSLA) conference in October 2004. OOPSLA attendees come from both academia and industry, and the DesignFest event was created to give them an opportunity to learn and share ideas and practices with their peers. Interested participants register beforehand to work on one of several offered problems in a half-day or full-day session. During the session they study a requirements document and collaborate to come up with a design. The emphasis is on learning rather than on specific deliverables, although teams are encouraged to produce models and list their assumptions, problems, and lessons learned.

Our study focused on one half-day group and one full-day group, both working on the case management problem: a general OO framework for use in developing case management applications for industries like insurance, finance, and healthcare. The half-day group consisted of four experienced developers and two educators, and the full-day group consisted of five experienced developers. In accordance with the DesignFest rules, each group appointed a moderator and a recorder. Only the half-day group made use of the moderator role, and neither group used its recorder.

Both sessions were held in a large hotel banquet hall, and attendees were seated around large circular tables. Each team was provided with a single flipchart and a few posterboards for hanging materials; hotel notepads were provided to each attendee. The moderator of the half-day session also brought stacks of sticky notes for the use of his group. As we shall see, physical settings and resources had a significant impact on how teams performed their task.

Even though both teams worked on the same problem, each team used different processes, interaction styles, tools, and notations. At the guidance of its moderator, members of the half-day group first individually brainstormed entity ideas on sticky notes, proceeded to arrange them on the poster board, and then worked as a team on creating class diagrams. The full-day group, on the other hand, brainstormed use-cases as a team. It divided them into two groups, separating to work on them, and then regathered to discuss their results. With the session drawing to an end, they split again to create complete and finalized versions of their initial diagrams.

It is clear from the above description that while all teams are likely to share some steps or activities, there is no one process that fits all groups. We believe that as we continue to observe design teams, we will encounter many other variants.

The implication of this variance for an SDCSS is that it should provide an open collection of tools and capabilities which enable different working modes, rather than forcing a fixed process. Developers focused on a design problem are not likely to adopt a system that requires them to change their habits or forces them to wrestle with it to accomplish their goals. It is likely that with time, teams will find ways to realize the inherent potential of the VW, but they must do so at their own pace.
3. OBSERVATIONS ON CANVAS USE

Virtual conferencing has been most successful for executive meetings, where most interaction is verbal rather than visual. Software design activities, on the other hand, involve complex visual artifacts, including: requirements, use cases, UI mockups, class diagrams, statecharts, etc. An SDCSS must support these artifacts and activities.

One might ask why we cannot simply use a collaborative CASE tool or UML editor, as UML strives to serve as a standard notation. Undoubtedly, while an SDCSS should certainly provide equivalent capabilities, it must also support less formal styles of work. One of the deficiencies of UML is that despite its daunting variety of features, it does not support the way designers think about their systems, such as combining structure and behavior and using metaphors [5]. Indeed, our study participants used free-form notations with ad-hoc semantics to capture ideas while brainstorming. UML notation was only used at a later stage to document the design, and even then, very few features were used.

Note that the specifics of the formal and informal notations used in SWDMs are outside the scope of this paper, although their implications must be taken into consideration in designing an SDCSS.

Of no lesser importance, however, are the surfaces upon which these notations are rendered. One of Brooks arguments in “No silver bullet” [3, p. 195] for the infeasibility of visual programming is that the workstations of the time could not try to match the freedom of the real world. Clearly, the same argument also applied to high-level software design. But in the thirty years that have since passed, technological advances brought us closer to tackling this challenge.

To better understand this challenge and the implications for an SDCSS, this section (and most of this paper) focuses on the surfaces upon which this creativity takes place. Throughout this paper, these surfaces will be referred to as physical canvases, and their VW equivalents will be virtual canvases.

3.1 Canvas types

Most canvases can contain other canvases, creating a containment hierarchy. For example, a sticky note can be placed on a sheet of paper (Fig. 1(a)), which can be posted on a posterboard (Fig.1(b)). As we shall see, designers interact heavily with this hierarchy of canvases, implying that usable SDCSS must provide a notion of grouping and containment.

While most canvases serve as containers, some have special symbolic meanings or properties that hint at how they should be used. For example, sticky notes are small, and prompt people to fill them with concise and limited text. Their color helps us distinguish them from notepads, and remind us of the adhesive material on their back, which hints that they should be attached to other canvases. Similarly, large sheets of paper from the flipchart “invite” complex diagrams to be drawn upon, and that sticky notes should not be attached to it. Rather, it suggests the use of pins to attach other materials, such as paper sheets from the flipchart.

To allow existing processes to be translated to the VW, an SDCSS must preserve these canvas types. For example, instead of offering generic canvases with “simple” grouping capabilities like most drawing programs², it should provide recognizable canvas primitives. These primitives could be offered in several palettes, similar to stencils in Visio, providing convenient access without forcing their exclusive use. In fact, the meeting moderator could guide participants towards a certain working mode by setting up a palette of recommended canvas primitives, and indicate a shift in the design phase by changing this palette.

Even when the type of a canvas suggests how it should be used, it might occasionally be used differently. For example, Fig. 2(a) shows a member of the half-day team using a small sticky-note for a complex diagram. Fig. 2(b), on the other hand, shows a member of the full-day team using notepaper for brainstorming in the way that one would use a sticky: note how only a small portion of each sheet is used. In many cases, such misuse is accidental, but it can also result from a sudden change in plans or simply because a more appropriate canvas is unavailable. The full-day team, for example, did not have stickynotes or any smaller notepads.

An SDCSS can reduce such mistakes by offering flexibility in canvas size and a variety of canvas primitives. It can also mediate such mistakes by allowing the types of existing canvases to be changed, in the same way that the types of existing widgets can be changed in a Visual Basic form.

In other cases, however, the misuse could also be intentional, perhaps owing to individual style. For example, one might choose to draw a diagram on a small sticky note because that drawing might represent a single brainstormed idea which is more difficult to describe in words. Again, the system should not prevent such use.

3.2 Canvas size

The physical dimensions of a canvas affect the amount, style, and layout of contents that it can accommodate. Most physical canvases are rigid in their sizes, deterring people from an optimal use, and requiring a mental effort to place contents in the limited space.

If space in a particular canvas is likely to run out, one might decide not to record certain details, risking information loss. Alternatively, one might try to force the available area to increase or force excessive amounts of contents into the available area. For example, Figs. 3(a)-3(c) show how additional sticky-notes had been forced into a limited space by hanging them outside the canvas boundaries or by letting them invade into other logical regions. Figs. 4(a) and 4(b) show complex diagrams squeezed into an available space, rendering them confusing and unreadable.

Virtual canvases should therefore not be rigid in their sizes, and resizing controls similar to those found in a drawing program should be available. Since in-place resizing of a canvas might interfere with its neighbors, the software must try to preserve the layout, and provide convenient means to minimize, reshape or push aside nearby canvases.

3.3 Scaling

Designers often choose the kind of canvas to use based on their intended audience. For example, small notepads are convenient for individual notes but difficult to share with others, whereas larger sheets are easier to share but are bulky and space consuming. As a result, when a previ-

²Note that even providing “simple” grouping in a collaborative editor is far from trivial [6].
ously personal artifact has to be shared or collaborated on, it must first be reproduced on a larger scale. To eliminate this hassle, an SDCSS should provide effective rescaling capabilities. It should therefore use vectors rather than rasters for most internal representations.

There is a danger, however, in providing automatic rescaling capabilities, since these might cause designers to miss an important byproduct of the rescaling process: Recreating the artifact in its larger form is not simply a matter of duplicating the original, but is often an opportunity to rethink, improve, and polish. For example, Fig. 5(a) shows a designer copying a complex object diagram from a cluttered note into a large sheet of paper. In doing so, he is elaborating each entity, listing its properties and drawing it in straight lines instead of freehand curves. Meanwhile, others are examining both drawings, offering suggestions and approving each element as it is copied (Fig. 5(b)).

### 3.4 Partitioning and merging canvases

A canvas representing a single entity is often partitioned in order to allow properties and other information to be organized into categories. For example, Fig. 3(c) shows the partition of a canvas representing the event abstraction of the case management system; no sub-canvas was intended to stand on its own. CRC cards are another example of this behavior. Occasionally, a certain partition is only relevant in the context of another, as in Fig. 6(b) which shows an area cordoned off for notes. An SDCSS should allow canvases to be placed inside other canvases, allowing the notes to be exposed only when focus is on the main part.

Sometimes, a partitioned canvas contains independent but somewhat related artifacts. Such partition can serve to contrast design alternatives, as in Fig. 6(a). In other cases, however, it seems that this partitioning is an alternative to artifact organizations not possible in the VW. For example, in the lower sheet of Fig. 4(a), the two diagrams are only weakly related (by involving the case abstraction). These would be better off on separate canvases, with the SDCSS providing some hyperlinking capability which will allow links between related artifacts to be traversed even if their containing canvases are not adjacent or visible at the same time.

Occasionally, unrelated artifacts might be placed on the same canvas to optimize the use of space, as in Fig. 6(c) which contains lists of assumptions and synchronization points. Supporting flexible canvas sizes in the SDCSS would eliminate this problem. It should also be possible to cut virtual canvases into independent units rather than simply partition them on the same canvas, in the same manner that a physical canvas could be cut with scissors.

In the VW, canvases could be merged just as they can be partitioned. The half-day team had to place several numbered flipchart sheets next to each other to simulate one large sheet with all the related sticky notes. An SDCSS could allow these canvases to actually be merged, but since they could be different in size and scale, it should also provide means for normalizing them before the merge.

### 3.5 External documents

Although content is usually rendered on a previously blank canvas, there are cases when an existing pre-printed document is further annotated or drawn upon. For example, most participants made initial annotations on the requirements document provided by the organizers. In some cases they even drew small diagrams next to specific requirements, rather then draw them on separate sheets of papers where they would be more difficult to access associatively. Other teams might want to use preprinted accessories such as CRC cards.

An SDCSS should provide means to import external documents and use them as canvases. To allow quality rescaling, it should perform de-rasterization or OCR if possible. One must note, however, that the use of external documents raises issues about replication and consistency in group work, as we shall discuss later.

### 3.6 Modifications and deletions

Physical canvases differ in the ways their contents can be changed or erased. Contents on paper, for example, are most difficult to change: As we can see in the upper diagram of Fig. 7, contents sketched in ink can only be removed in an untidy way which does not recover the lost space. As more and more items are erased, the document becomes too cluttered and must be redrawn, although this is an opportunity for cleanup. Matters would have been further complicated if different ink colors had been used. Using correction-fluid or erasing contents written in pencil does not completely clear the erased area and affects its future use. Dry erase boards are more flexible when it comes to erasing material, but they require the use of an inaccurate thick-felt pen and can be erased accidentally.

The main problem with both mediums, however, is that entire areas, rather than specific objects, are erased. Thus, even if one could remove a specific object, all overlapping objects, including fragments of connectors and text, will have to be redrawn. Dependent objects, such as connectors to other objects, will need to be explicitly removed as well. A clear advantage of the VW is that specific elements can be erased or modified without harming overlapping elements. An SDCSS could even automatically rearrange the connectors and other elements following the change.

### 3.7 Versioning and rollbacks

A critical observation is that canvases are typically used in discrete bursts of activity rather than continuously. An individual working on a diagram often does so in cycles, alternating between thinking or consulting other material and drawing or making modifications to the canvas. These cycles are much longer when several people are working together, partially because of the additional time required for discussion. Larger teams often abandoned canvases for a while, working on another before coming back.

For example, Figures 8(a) to 8(d) show a progression of changes to an artifact. Fig. 8(b) shows how the original rectangle is expanded into a diagram, in a continuous work interrupted by frequent discussions and modifications. In the transition to Fig. 8(c), a change in ink color indicates that the new artifact at the bottom has been added as a separate burst. Finally, we see in Fig. 8(d) that a comment in blue has been added to the existing list of comments in black.

The implication from this mode of work is that periods of quiescence with respect to a particular canvas or artifact may implicitly represent a version. An SDCSS could thus provide version-tracking and rollback capabilities at the canvas level, without requiring checkpoints to be explicitly marked. To understand why this could be useful, consider our observation that during discussions, engineers often needed to temporarily illustrate their ideas on top of existing materials. For example, they needed to show that certain entities were connected, or draw connections to objects outside the current diagram. The hassle of erasing such temporary modifications often caused participants to
demonstrate their intentions with gestures rather than an actual drawing. An SDCSS can easily alleviate this problem with roll-backs, although suitable mechanisms would be needed to distinguish temporary illustrations from the unrelated concurrent activities of others.

The identification of activity periods at the canvas level carries additional benefits. A complete version history implicitly stores design knowledge which would otherwise be lost. For example, rather than study a finished artifact, an engineer could explore its development, and infer the design rational by studying branches which were abandoned and rolled back. In addition, activity periods could be used to create an implicit heuristic for object locking, whereas an entity or an entire canvas is implicitly locked if it had been recently modified. A complete record of timing information for every entity could be used to provide meaningful visualizations of activity throughout the project. For example, we could paint using a luminous virtual ink color which “dries up” into the regular ink color as time passes. This helps others see where the most recent activity took place and avoid interfering with ongoing work.

### 3.8 Rapid access

Even though the abundance of artifacts created during SWDMs prevents more than a few from being in focus at any time, a certain portion could still be physically positioned for rapid access. For example, every DesignFest team was furnished with two or three poster boards, each capable of displaying up to three sheets of paper from the flipchart. Once a specific sheet was located on the board, its contents could be read effortlessly. To locate a particular canvas or artifact, however, people employ a variety of techniques.

The most efficient technique in the PW is based on recalling the spatial location of the canvas. This location could be relative to other artifacts, such as “the class diagram on the sheet to the right of my current canvas”, or relative to the environment, such as “on the left-side of the poster-board by the window”.

Preserving such spatial cues in the transition to the VW is difficult. Since sizes and locations of virtual canvases are easier (and thus more likely) to change, the relative positions of objects would quickly become outdated. Environment-relative artifacts could also be difficult to locate, since most 3D conferencing systems rarely attempt to provide a sophisticated background environment beyond the focus of the interaction. In such systems, posterboards are likely to be “floating in space”, making orientation difficult. This implies that the SDCSS should provide an elaborate enough background environment, not for its aesthetic benefits but rather for improved orientation.

If many artifacts are graphical rather than textual in nature, their shapes can be identified in a rapid sweep over the entire workspace. A quick glance is often enough to locate a UML sequence diagram or a “spider shaped diagram”. In other cases, people title their canvases, making them easier to locate. An SDCSS can highlight these titles while the user is panning the viewport, to aid in the search and counter the disorientation caused by the movement.

Unfortunately, the space available for rapid access in the manner described above is limited by the available physical space and the effective access speed. Important canvases, such as to-do lists, will not be removed. Others, however, may be removed in a least-recently-used manner to make space for others. In the PW, organizing and finding space for these removed sheets is difficult. Many teams, for example, placed these sheets on the floor (Fig. 9(a)), where they were more difficult to access later (Fig. 9(b)). As we can see in Fig. 9(c), participants individually faced similar problems when maintaining large numbers of canvases.

An SDCSS must provide tools for accessing and organizing material, but should do so with less ceremony than long-term data organization tools. Since most artifacts are graphical, it might be possible to borrow techniques from tools for organizing digital images.

### 4. OBSERVATIONS ON TEAM STRUCTURE

So far we only investigated how software engineers interact with a canvas. Of course, SWDMs are a group activity, and an SDCSS must support the interaction patterns of design teams. In this section we discuss the formation of teams; the next section will discuss how the team maintains its focus on an artifact. Note that we will refer to all participants in a particular session as a group, and to all smaller units as teams.

In the course of the DesignFest sessions, group members repeatedly split into teams, regathered, and split into different teams; occasionally they worked individually. Much of this behavior was ad-hoc: Rather than having clearly defined roles and memberships throughout the session, teams simply coalesced around specific tasks and artifacts, and later dissipated without ceremony. As a result, designers might face a difficulty specifying what they are looking for when they are searching for artifacts from a specific team activity. For example, an engineer might look for “the class diagram I worked on with John and James before lunch”. An SDCSS should thus offer the possibility of searching artifacts via a timeline, or based on collaborators.

As ad-hoc discussions often form around specific artifacts, the SDCSS should provide convenient means for starting a collaboration session with a peer in the context of a particular artifact. This will require a way to draw the focus of our peer to a specific artifact without causing disorientation.

Since most of the participants did not know each other beforehand, it is interesting to investigate which factors determined the structure of teams. While social or cultural familiarity, as well as interest or skill, obviously played some role, the physical layout of the meeting area had a surprisingly significant effect. The random seating order around the round table shaped the first division into teams, perhaps because of the convenience as well as the newly-gained familiarity between neighbors.

Furthermore, when gaps existed in the physical layout, contiguous chains of individuals tended to become teams. For example, the initial seating order of the full-day group was random: seated from left to right were: Leif, Paul, Benton, Jesse, and Jody. To avoid obstructing the flipchart and posterboard, Leif and Paul shifted left, leaving a gap. Although Leif occasionally moved across this gap to see the board (Fig. 10(a)), this gap dictated the later separation of Paul and Leif into a separate team (Fig. 10(b)). As the session progressed, however, both formed teams with other participants.

Spatial location also played an important part in the organization of the half-day group. Since the positions of its poster-boards and flipchart formed a wide angle, it was difficult to work on one board while maintaining awareness of the other (e.g., Fig. 11(a)). As a result, ad-hoc interaction was often limited to people working on the same area of the same board; the perceived distance of others was far greater than their actual distance. Surprisingly, even though people moved about, they tended to work more with posterboards...
which were physically closer to their original seats. Ad-hoc interactions sometimes occurred when a person who wasn’t part of any team was drawn to the activity of another team and joined it. For example, Fig. 11(b) shows one participant, returning from a coffee break, joining a team which was discussing an artifact.

The effects of physical layout on team formation and interaction raises many issues for the design of an SDCSS. Location-based interaction can be mimicked in a 3D SDCSS which supports avatars. Unfortunately, reproducing physical avatars in the VW also introduces blocked views and a need to circumnavigate around avatars. Many would find this unacceptable given the cumbersome navigation in current 3D systems. A 2D system, or a 3D system with no avatars will lack many of the spatial cues which promote interaction. To compensate, it is important that such SDCSS will visualize the viewpoints of other individuals. Such viewpoints are not avatars in the strict sense, but will provide some degree of peer-awareness and a chance for location-based interaction, especially around specific artifacts.

It is also important to note that team members tended to maintain a peripheral awareness of the activities of other teams. Reproducing these cues is difficult without complete virtual-reality capabilities, but it might be possible to partially compensate for them by means such as spatial audio or the use of fisheye lens. Awareness is important, because when teams were physically separated, they exchanged less information with one another and their composition changed less frequently. For example, due to lack of space one team in the full-day group used a second table hidden by a poster-board. To compensate for the loss of awareness, members of each team occasionally ventured into the space of the other to check on its progress, affecting the performance of both teams.

A significant problem regarding awareness arises when several people are working from one physical location, such as a conference room, in a distributed meeting. It might be difficult for those in other locations to keep track of all the parallel activity going on in that location from a single viewpoint. One approach would be to use automated means to monitor, digitize, and provide after-the-fact record of the activity.

5. OBSERVATIONS ON FOCUS

As discussed in the previous section, designers constantly shift between working with the entire group, working with a smaller team, and working individually. While they were working with others, we often observed their focus shifting away from their team for short periods of time. A person might, for example, review some notes or change some diagram, as can be seen in Fig. 11(a) where he turns around to examine an old artifact. Yet even when his attention is not on his team, he maintains awareness of its activities and is able to rapidly locate its current focus.

An obvious implication for an SDCSS is that each user must have a personal, controllable viewpoint. Viewpoint sharing might be adequate for executive meetings but not for the need of designers: Since most software engineers seem to enjoy the creative part of their work, giving up control would be like observing a game rather than participating, or giving up the TV remote. Nevertheless, the use of individual viewpoints poses new problems in the transition to the VW, since many of the cues which provided awareness and allowed a user to quickly rejoin the team focus in the PW are lost.

In this section we examine how designers maintain their focus on nearby and remote artifacts and how they draw the focus of others. We then discuss the problem of locating the current focus of the group after diverging from it.

5.1 Maintaining individual focus on artifacts

The complexity of the task and the distractions of group work cause individuals to intermittently lose their focus on an artifact. To maintain it, an individual who studies a nearby canvas will often use physical means, such as placing a finger or a pen at immediate proximity or in actual contact with the canvas surface (Figs. 12(a) and 12(b)). In fact, we observed designers moving towards a distant canvas in order to maintain physical contact, even if they could see it clearly from afar.

Supporting this physical contact with an SDCSS is difficult: While people are conditioned to avoid contact with a monitor and can point at specific locations without touching, tracing complex paths at proximity is difficult. An accidental touch might trigger unexpected interaction in a touch-sensitive devices. One solution for such devices would be to ignore single touches which can be considered accidental if they they occur during a period of quintessence. Note that the mouse, if available, is unlikely to be of use here since it is inaccurate and since it will probably be used for control tasks.

The problem is further aggravated when trying to concurrently maintain focus on two nearby items. Such a scenario is in fact very common: For example, figs. 7, 13(a), and 13(b) respectively show a designer creating a new diagram based on another diagram, a requirements document, and personal notes. Figs. 5(a), 3(a), and 8(a) show designers trying to maintain focus on small and large canvases at the same time. An SDCSS on a touch-sensitive screen would be bombarded with input from multiple areas of the surface and would have to distinguish between them. This is especially important if a drawing tool is activated, as we would like to use it on one canvas but not on the other.

Of course, the difficulty of maintaining or reestablishing focus on an artifact depends on its size and on the flexibility of our head and eyes, as dictated by Fitt’s law. Since it is unlikely that their efficiency could be mirrored in the VW without VR equipment, we can expect to see some effect on working habits. Focusing on a small and remote artifact, however, may be more effective in the VW since spatial movement requires less effort.

5.2 Drawing the focus of others to an artifact

When designers are working at close proximity to the canvas and to each other, they often use similar means to point at specific objects (Fig. 14(a)). Since they often think in metaphors [5], they might use special gestures to indicate certain behaviors such as components that “talk” or “search for each other” (Fig. 14(c)). This implies that providing a simple telepointer in an SDCSS is not enough: a variety of pointers should be available to convey the range of possible meanings. Of course, the system must also accommodate gestures that involve multiple artifacts.

One advantage of the transition to the VW is the reduction of clashes of the type evident in Fig. 14(d), where one designer points at an artifact inside the personal space of another.

The limitations of the PW are also obvious when a remote person gestures at a nearby canvas. Whereas it is easy to identify the specific artifact being pointed to on a large-scale canvas (Fig. 15(a)), doing so for a small canvas is difficult.
The canvas may be too far away (Fig. 15(b)), or might be difficult to examine from up close due to the presence of others (Fig. 15(c)).

An SDCSS can take several approaches to this problem. First, it could allow a user to “bookmark” the current location, and then navigate to the relevant canvas or to our peer’s point of view. While movement in the VW requires little effort, it does not solve the problem if user avatars cannot occupy the same space or can block each other’s view, unless we make them transparent. Another option is to provide convenient zooming facilities, perhaps using the mouse’s scrolling wheel. This is, of course, impossible with touch-screens, and does not solve the problem of viewing artifacts at inconvenient angles in 3D systems. Perhaps the solution is to allow items to be replicated; thus, we would click the target item, and receive a temporary replica conveniently floating in front of our viewpoint. This suggests supporting “private artifacts”: items that are only available and known to specific users.

This last solution brings us to the related problem of pointing in documents for which multiple copies or base documents exist. For example, Fig. 11(b) shows several people trying to locate the same spot in their respective copies of the requirements document. In a virtual setting, people may rearrange, scale, or annotate their documents in a way that makes finding the common location difficult. An SDCSS would therefore have to maintain connections between different copies of the same document to allow such synchronized browsing.

5.3 Inferring peer focus

Even while breaking themselves from the focus of the team, designers continue to maintain peripheral awareness of its activities, and are able to rapidly rejoin its focus. Consider, for example, the person standing with his back to the group in Fig. 11(a). When he turns around, he will quickly find the artifact everyone is looking at, even if nobody is specifically pointing at it.

Consider Figs. 16(a), 16(b), and 16(c), and note how it is possible to follow or intersect the gaze of others to determine the artifacts they are looking at. In fact, it seems that we can do this almost instantaneously in our daily lives, and even identify those who are looking at other objects and ignore them. The ability to follow the gaze of others has been shown to improve performance in different tasks, but is not readily available in the VW.

To compensate, many techniques provide awareness of the focus of other participants, with varying costs in usability and distraction [4]. Telepointer techniques are inappropriate for touch-sensitive devices; using gaze-tracking requires, at present, prohibitively cumbersome equipment. A better approach is to visualize the current viewports of other participants. Since separate high-level radar views consume precious design space, the solution is likely to involve overlaying the viewports of others on a person’s own view. The problem is how to minimize distraction: overlaying semi-transparent rectangles representing viewports on top of a class diagram which naturally contains rectangular shapes is confusing.

A possible solution which we are currently experimenting with uses spotlights: Rather than overlay a uniform rectangular shape on top of the current canvas, a spot light is projected onto the canvas through a portal which represents our peer’s viewport. As a result, only the area which our peer is capable of seeing is significantly illuminated, with a stronger intensity towards the center and much lower intensity towards the edges. We believe that this circular varying intensity would be less distracting than a uniform shading. The main advantage of this approach, though, is that illumination is additive. Therefore, if many viewports are overlapping, that region will be illuminated at high intensity, helping other users identify and join it faster. We are currently working on a prototype of this technique and planning a user study to validate its effectiveness.

6. CONCLUSIONS

In this paper we presented issues that a system for supporting distributed and colocated software design meetings would have to tackle. We identified many requirements and suggested some solutions.

Although these findings are based on only two observed teams, the commonalities substantiate many of our claims. Nevertheless, we are continuing to gather information, and are planning a series of observations, interviews, and surveys in industrial settings. In addition, we are building and experimenting with prototypes of the different solutions presented here, and investigating frameworks which will allow us to build an experimental prototype of a complete SDCSS that will satisfy all of the presented requirements.

Acknowledgements

This work was supported by an IBM faculty award to James Herbsleb. The authors would like to thank the organizers of the ACM DesignFest® event at OOPSLA’04 for assisting us in conducting this research.

Special thanks go to the following participants for allowing us to observe and document their work: Jesse Eichar, Robert Fewster, Shane Fruch, Jody Garnett, Leif Geiger, Judy Mullins, Paul Stoxen, Eric Theriault, Sharon Vest, and Brenton Webster.

7. REFERENCES