ClubDev: A System for Supporting Group-Based Software Development Activities

Jacob T. Biehl, William T. Baker and Brian P. Bailey
Department of Computer Science
University of Illinois
Urbana, IL 61801, USA
{jtbiehl, wtbaker, bpbailey}@cs.uiuc.edu

ABSTRACT
Existing tools do not adequately support the emerging practice of group-based software development, which is aimed at improving software quality. In this paper, we present a new, fully functional system called ClubDev. Our system allows developers to quickly share off-the-shelf applications between personal devices and available large displays, perform near simultaneous input with applications, and mediate which applications are shared and when. The benefit of using our system is that it better supports the types of activities that are desired, but often not able to be realized during group development. Our system was informed via a series of surveys and interviews with developers experienced in group development practice. Results from an initial evaluation showed that users want and are able to utilize our system during realistic group development activities. Practitioners can download and use our system, while researchers can leverage our system framework to create and test new interfaces for co-located collaborative work.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design, Human Factors.

Keywords: Co-located, Collaboration, Programming, Multiple Display Environments.

INTRODUCTION
Software developers are attempting to improve their work practices to meet the increasing demand for dependable systems [6]. A radical change is occurring in how the act of programming itself is being performed – it is rapidly transitioning from programmers working individually in their own offices to small teams working in co-located spaces.

As shown in Figure 1, these workspaces are typically configured with individual work areas but do not have physical barriers that prevent eye contact or inhibit verbal communication. The workspaces are also equipped with large displays, whiteboards, and other instruments to foster communication, coordination, and awareness. Early evidence suggests that group development could reduce defects in software and/or improve its overall design [19, 21, 25, 31, 32].

Though being situated within the same workspace allows for increased communication, it exacerbates the need for sharing and interacting with each other’s task artifacts (i.e. application windows) such as code editor windows, debug windows, and web browsers showing code examples. For example, a developer may seek advice on how to call a particular method, or two or more developers may want to work jointly to identify and correct a complex defect.

Unfortunately, existing practices are not sufficient for sharing task artifacts and managing input during group development. For example, when two or more group members want to resolve a bug together, they are forced to crowd around a single display and take turns interacting with the tools. While this strategy may be manageable for a pair, it does not scale to larger groups. Further, by funneling interaction on a single device, group members must work in a serialized fashion, preventing other important tasks from being performed in parallel (e.g. researching documentation about the error, searching forums for solutions, etc.).
In this work, we present ClubDev -- a new, fully functional system that facilitates group development activities. Our system was informed through a series of surveys and interviews with developers experienced in group-based development practices, as well as an iterative design process.

With our system, developers can quickly and easily replicate off-the-shelf applications between personal devices as well as available large displays. Group members can choose which applications are available to the group and the level of sharing allowed, and group members can decide when to instantiate replications of available windows. Group members can also interact with replicated windows without interrupting the input stream of the owner’s device. The overall benefit of using our system is that it better facilitates the types of activities that are desired, but are not able to be easily realized during group development. Also, the availability of systems like ClubDev may persuade more developers to engage in group development practices.

**RELATED WORK**

We describe group development practices and how our work supports it, review solutions that could be used to support group development and how our system overcomes many of their limitations, and describe how our work extends techniques for collaborating with multiple devices.

**Group Development Practices**

Group development is an emerging industry practice where teams, consisting of multiple (typically 2-8) programmers [20, 34], share the same workspace to design, develop, and test software [32]. The software is typically compartmentalized within the larger project, causing team members to interact mostly amongst themselves [20, 34]. Group programming is an essential component of newer methodologies such as Agile [25] and eXtreme Programming [13] that strive to reduce defects in software and improve its design.

The central impetus for group development is to improve coordination and reduce communication breakdowns, serious bottlenecks in the development process [14, 19, 34]. For example, programmers spend up to 40% of their time communicating with their team members [20], and often become “blocked” until needed assistance is gained [18].

Situating themselves within the same physical workspace enables increased communication, but exacerbates the need for effective techniques for sharing and interacting with each other’s task artifacts such as source code windows, debugger windows, and browsers showing code examples. This is particularly needed to support informal, opportunistic collaboration, which is common in programming [19].

Our work provides a lightweight interface and supporting system that allows group members to share and interact with each other’s task artifacts. For example, this can be used to facilitate opportunistic collaboration, working in sub-groups, and maintaining finer-grained awareness.

**Tools for Supporting Group Development**

Several tools have been designed or could potentially be adapted for group development activities. For example, source code repositories like CVS [1] and SVN [4] can be used to help coordinate access to the shared source code. Since these systems typically embody a formal process, they would not provide an effective means for informally sharing task artifacts during group development activities.

Developers could use a file server or e-mail to share task artifacts. For group development, these approaches are not sufficient because they do not retain the interaction context of the applications in which the artifacts are loaded, do not allow group members to work on the artifacts together, and creates versioning issues. Using personal devices, with one driving a large display, is also insufficient because artifacts from multiple users cannot easily be shown in parallel.

Beyond helping group members share task artifacts, several tools have been developed for acquiring and maintaining awareness of each other’s activities. For example, Palanir [24] and Augur [15] provide visualizations of recent actions within a code repository, while FASTDash [10] extends this awareness to include developers’ actions within their local IDEs. Collaborative IDEs, such as Jazz [12], allow group members to see who is working within the shared code, receive updates of their actions, and be able to informally chat with fellow team members.

Our system combines advantages of many of these solutions while overcoming many of their limitations. For example, our system can be utilized to quickly share task artifacts between devices during opportunistic collaboration, allows a shared display to show artifacts from multiple users, and view the ongoing activities of group members.

**Techniques for Multi-Device Collaboration**

Many techniques have been developed for collaborating with multiple devices. One technique is to replicate the pixel data of application windows or the desktop to other devices, as in WinCuts [29], NetMeeting [3], LiveMeeting [2], Timbuktu [5] and Community Bar [30]. This provides a simple technique for sharing, but the limitations of the existing implementations are that users either cannot interact with the replicated windows or can only interact with a single window that must always be in focus.

CoWord [35] and the intelligent collaboration transparency framework [22] exemplify a second technique in which multiple instances of the same or similar application are synchronized via an underlying protocol. This allows for a looser coupling of the collaboration, but requires a new protocol to be developed on a per-application basis.

A third technique is to allow multi-input on a single display. This is provided by tools such as KidPad [8], PointRight [17] and Mighty Mouse [11]. This allows multiple users to interact with a single point of visual focus, but does not support individual work in parallel.

A fourth technique is to pass descriptors of content (e.g., a web URL) between devices that have the same or similar applications, as in iRoom [16]. If coupled with the ability to redirect input [17], it would allow users to relocate artifacts across devices and interact with them. Limitations of this technique include that it requires similar applications to
be installed, it does not maintain interaction context, and the user loses ownership of the relocated task artifact.

Our system differs from this corpus of work in that it allows more flexible modes of collaborative work. For example, it allows any number of application windows to be shared in parallel, near simultaneous input from multiple users (even when the source window is not in focus), and a large display to be utilized concurrently by group members. Finally, the focus of our work is on understanding how these types of interfaces and underlying systems should be designed to support group development activities.

UNDERSTANDING GROUP DEVELOPMENT
To better understand the emerging practices and needs of group development activities, we conducted a series of surveys and interviews with professional software developers.

Our survey was sent to a developer distribution list within a large software company, and 90 responses were received. Using reported experience with group development activities as the selection criteria, we then conducted face-to-face interviews with 13 of the respondents.

Our surveys and interviews were designed to gain a better understanding of programming-related tasks that developers collaborate on, how developers configure themselves in a collaborative workspace (and how often), and how tools could be improved to foster more efficient group work.

Based on lessons learned from the surveys and interviews, we created and evaluated several paper prototypes. Evaluations were performed with 7 developers (one at a time) at two local software companies. Developers were screened to ensure they had experience with the types of collaborative activities that we were most interested in. The developer was first asked to explain his collaborative practices. Using his response as context, we introduced our prototype and asked him to work through several tasks with it (e.g., sharing an application window with a team member, moving a window from a personal device to large display, etc.). The developer also commented on why he would or would not use a fully functional version of our prototype in practice.

Lessons Learned
The most important lessons learned from our surveys and interviews (L1-L4) and prototype sessions (L5-L7) were:

L1. The use of ‘bullpen’ configurations is a rapidly emerging practice. Developers engage in many forms of collaboration, but we found that use of co-located workspaces (‘bullpens’) was gaining increasing acceptance. For example, some developers described using bullpens on an as needed basis to fix complex or critical defects, whereas others reported using them as part of their daily work practice. A primary impetus for using bullpens was to encourage multiple perspectives during development and to reduce communication overhead.

L2. Multiple modes of collaboration are employed. For example, we found that developers would work individually in parallel (e.g., each working on their own code), in subgroups (e.g., to correct a complex bug), or collectively (e.g., to review design decisions or assess project status). In addition, transitioning between these work modes was frequent and spontaneous; driven by demands of the ongoing programming activity. Thus, any new tools created for these types of workspaces must not preclude or inhibit any of these work modes.

L3. Large displays such as LCD projectors are present, but often underutilized. Most developers acknowledged the potential value of using a large display to share information of interest to the group (e.g., source code, documentation, examples, etc.). However, few developers reported utilizing these displays because most of their collaborations were short-lived and opportunistic, causing the relative overhead of configuring these displays to be too high. For example, several developers said they would not “hassle” with configuring a large display simply to receive assistance on a syntax error or compile problem, even though this initial request might evolve into a deeper and lengthier collaboration.

L4. The opportunistic collaborations require more effective tools for sharing task artifacts. For example, if a group member wants assistance to resolve a compile error, the other group members must physically move to her work area, crowd the screen, and token pass the input devices. Moving to the screen where the problem is occurring is necessary because developers need to see the context of the problem (e.g., what parameters are being passed, where breakpoints are set, etc.), and any potential solutions that have been attempted. Always having to gather around a single personal device was disliked because only one person could interact with the system and the task could not be easily split into sub-parts.

L5. At least two levels of sharing are needed. For example, in some situations, a developer would want to show a group member a particular method so that it could be called appropriately (view-only). In other cases, the developer would want to share his code editor window so that a group member could modify the method call for him (view/write). Developers stressed that they would always want to be in control of whether other group members could modify or only view their task artifacts.

L6. Any solution for sharing information must work with the tools that developers already use. Developers use a wide variety of complex tools, including code editors, debuggers, source repositories, bug databases, etc., and have very deep convictions about their use. Developers were adamant that they would not adopt any solution that required them to abandon their current tools.

L7. Sharing of information must allow social negotiation. Developers recognize many potential benefits of collaborating within co-located workspaces, but this does not mean that they want others to be able to push application windows onto their device at any time. There must be some form of negotiation, e.g., a developer can signal when she needs assistance or has information that is needed by others; while other developers can determine when to suspend their current task to offer assistance or choose when to view the information.
Though many other lessons were learned, we felt that these lessons were the ones that would most influence the design of our functional system, and would be most useful to other researchers investigating similar systems.

CLUBDEV
ClubDev is a new, fully functional system that allows a group of developers (e.g., 2-8) to quickly and easily share and interact with task artifacts between personal and shared devices. Benefits include fostering opportunistic collaboration, working jointly (or in subgroups) on tasks, and maintaining better awareness of each other’s ongoing work. The system is comprised primarily of three components: the user interface, system framework, and coordination server.

User Interface
The user interface provides a visual representation of the group members and large displays, and the application windows that have been made available to the group and those that have been placed on the large displays. See Figure 2. The user interface is comprised of the Collaboration Control, Collaborator Bar, and Shared Screen Dock(s).

Collaboration control. This control allows a user to configure whether an application window is available to other group members, and if they are allowed to modify or only view the content of the window. See Figure 3. The control is automatically displayed on the title bar of every top-level application window. This location was chosen to reinforce the concept that this is a window-level operation, provide quick access to the functionality, and provide a persistent indicator of the window’s sharing state.

Selecting the control reveals three sharing options (Fig. 3):

- **Do not show or share.** The window is not available to group members, and this is the default value.
- **Show.** The window is available to the group, but in a view-only mode. A live thumbnail of the window is displayed within the show area of the user’s representation in each group member’s Collaborator Bar.
- **Share.** The window is available to the group and anyone can modify its content. A thumbnail of the window is displayed within the share area of the user’s representation in each group members’ Collaborator Bar.

Offering both show and share is necessary as we found that developers typically have a strong sense of ownership over source code and related artifacts. For example, a user can set a window to show to allow others to maintain awareness of her activity in relation to that window, but not be able to interact with it. Whereas with share, group members could edit a source code file or other document together, or a user could pass control temporarily to another group member.

Collaborator bar. This component provides a representation of each user participating in the collaborative session and the application windows that each user has made available to the group. When a user joins a session, her personal photo (or other selected image) appears within the Collaborator Bar, located on either side of the screen. See Figure 2 (A).

Each user’s representation in the Collaborator Bar has a drawer with two rows. The top row displays thumbnails of...
application windows that have been set to share while the bottom row displays thumbnails of windows that have been set to show. Moving the cursor over a user’s image causes the corresponding drawer to animate out. See Figure 4.

From a group member’s drawer, a user can drag the desired thumbnail representation and drop it onto the desktop, causing a replication of the source window to be rendered. For example, in Figure 2, a developer has created a replicated view of a team member’s Visual Studio window. If the owner sets the window to share, the user could edit the code while the owner switches to another task, or the two could edit the content near simultaneously. Within a replicated window, a tele-pointer is rendered whenever the owning user’s cursor is within the source. This helps establish presence, provide awareness, and improve coordination [7].

The interaction needed to replicate an application window embodies a desired negotiation process. For example, it is the owner of an application window who determines if it is available to the group, while it is each group member who decides if and when to create the replication of a window.

Shared screen dock. This component allows a user to place a replication of an application window on a large (shared) display, organize the windows remotely, and redirect input to the display to interact with replicated windows. Our system can support multiple large displays, and each display would be represented by its own dock. Any user can place any number of replicated windows onto the displays.

The dock is minimized by default, and opens when the user moves the cursor over it. When opened, the dock shows thumbnails of all of the application windows on the corresponding large display. In this view, the thumbnails are shown left-to-right, as this allows all of them to be seen at once without occlusion (Figure 5a).

When the expand button (bottom of an opened dock) is selected, it expands and displays a miniature representation of the windows on the large display (Figure 5b). By interacting with the thumbnails in this view, any user can adjust the position and z-order of the remote windows.

Any application window that has been made available (set to show or share) can be placed on a large display. From any group member’s representation in the Collaborator Bar (including her own), a user drags the representation of the desired window and drops it onto the appropriate screen dock. The dock expands and the user can position the replicated window as desired. A large display can contain replicated windows from multiple users at the same time. A replicated window is removed from a large display by selecting the close icon at the top right of its thumbnail.

Users can also redirect their local input to a large display. For example, this would allow group members to collectively interact with a replicated window and share the same visual focus. To redirect input, the user selects the redirection button on the screen dock. Redirection is stopped when a special key sequence is performed.

In contrast to other interfaces for multi-device environments [9, 26, 31], our interface does not provide a strict spatial representation of the workspace or a portal view of applications on personal devices. Developers in our initial studies expressed that being able to view how applications are arranged on a group member’s personal device provides little value and they wanted the interface to emphasize the people they were working with rather than the relative location of their devices.

System Framework

A central goal of our system is to allow any off-the-shelf application to be shared and interacted with across devices. This goal is important because it would allow developers (and other users) to continue using the applications that they prefer and need for their daily work activities.

Our approach is to use a replication model. In this model, the pixel data associated with an application window in the frame buffer is captured, and is then available to be sent (replicated) to other devices in response to user requests. Interaction with a replicated window is enabled by capturing input within the replicated window and sending it to the corresponding source. Advantages of our approach include:

- Any off-the-shelf application can be replicated across devices, and its interaction context is maintained.
- Only one device needs to have the application installed in order for it to be utilized by the entire group.
- The owner of the application window maintains control over its content. For example, the owner can choose to discontinue sharing the application window at any time.
Figure 6: The system architecture of ClubDev. Any user can replicate any application window that has been set to show or share, and interact with windows (and their content) that have been set to share.

- Input originating from replicated windows does not interrupt the input stream on the owner's device.
- Users can interact with a replicated window even when the owner of the application does not have it in focus (e.g., it is minimized or occluded from view).

These advantages are particularly important for collaborative programming, where users often utilize different tools, applications typically reflect a rich interaction context (e.g., breakpoints in a debug window), and users have a strong sense of ownership over content. As illustrated in Figure 6, our system is comprised of three main components: the dispatcher, the host provider, and the input concentrator.

**Dispatcher.** The dispatcher is responsible for establishing and maintaining replicated windows between devices. The dispatcher executes on every device participating in the session, and listens for replication initiation events from the Coordination Server. When an event is received, the dispatcher on the source device establishes a network connection with the dispatcher on the destination device. New threads are then spawned to coordinate the point-to-point flow of the window’s pixel data and input between devices.

**Host Provider.** When an application window is replicated, an instance of the host provider is executed on both the source and destination device. At the source, the host provider is responsible for capturing and streaming the pixel data of the application window. Capturing frames of the application is performed using PaintWindow and BitBlt calls within the Windows API. These calls allow a window’s pixel data to be captured even when the window is not in focus or at the top of the z-order.

A special case is when an application window is minimized, as it is no longer redrawn. To handle this case, the host provider overrides the minimize operation such that the window is actually moved into an off-screen buffer. This allows the window to continue to be available to remote users, yet is completely transparent to the owning user (they still perceive that the window has been minimized).

On the destination device, the host provider is responsible for rendering the replicated window. As illustrated in Figure 2, replicated windows are drawn with a different colored title bar and border to differentiate them from locally running applications. One color (green) is used for replicated windows that are in share mode, whereas another color (purple) is used for windows that are in show mode.

**Input Concentrator.** A common technique for allowing multiple users to interact with an off-the-shelf application is to multiplex the hardware cursor [8, 11, 17]. A known limitation of this approach is that other users’ interaction with the application causes temporary interruption to the local user’s input stream. For example, two users would be unable to each simultaneously interact with different applications because they would always be competing for input focus.

As a user interacts with applications on the local machine, the host provider intercepts each event and checks if it corresponds to a replicated window. If not, it is ignored. If it does, the host provider removes the event from the queue and forwards it to the Concentrator on the owning device.

When a forwarded event is received, the Concentrator identifies the source window, enumerates its child controls, identifies the control that contains the event, transforms the coordinates of the event relative to the control, and ‘sends’

Figure 7: Control logic of the Input Concentrator. Other users are able to interact with replicated windows without interrupting the input stream of the owner. Users can interact with replications even if the source window is minimized or not in focus.
the event directly to the control. This technique has proven to be extremely flexible. For example, it allows other users to interact with a replicated window even when the source window is not in focus or has been minimized. It also allows two or more users to each be interacting with a different application hosted on the same device without conflict.

Coordination Server
The coordination server maintains the state of the collaborative session, including IP addresses of participating machines, sharing status of application windows, source/target pairs for replicated windows, etc.

The coordination server broadcasts updates to all participating devices whenever the state of the session changes (e.g., a user joins or leaves the session, or the sharing status of an application window has changed). The coordination server is also responsible for sending replication dispatchers’ requests to instantiate a replication of an application window.

The coordination server is a publicly accessible service that manages all of the collaborative sessions. When ClubDev is first launched, the user is presented with a configuration interface. The interface is used to initiate or join a collaborative session, specify whether the device is a personal device or large display, and configure personal settings such as the user’s icon that appears in the collaborator bar.

CLUBDEV IMPLEMENTATION
ClubDev is fully functional and was written mostly in managed C#, though some of the lower-level components were written in unmanaged C++. The interface runtime is built using the Windows Presentation Foundation libraries that are part of the Microsoft .NET 3.0 framework. The replication services are built on top of the Win32 API 6.0.

For reliability and scalability, the coordination server is implemented as a set of tables, stored procedures, and notification services loaded on a SQL Database Server. While our system currently works with Windows Vista, it is also compatible with legacy Windows-based operating system such as Windows XP. It could also be ported to other operating systems as the techniques that we developed could be mapped onto other commonly used windowing systems.

USER STUDY
We performed a user study to understand how users would utilize our framework during group programming activities, better understand the framework’s strengths and weaknesses, and identify opportunities for improvement.

Users and Task
We recruited 4 project groups of 2-3 individuals from a senior-level software engineering course in our department. Since the course spans multiple semesters, we were able to recruit groups that had been working together on a project for several months. These teams thus resembled many professional software teams in that they were already comfortable working together and that they had some understanding of how to subdivide and coordinate labor on programming tasks. Additionally, we felt these groups would be interested in trying out new tools for software development.

For the task, each team was asked to build a personal bookmark manager with the following functionality:

- Add bookmarks, which were composed of URLs, titles, user-defined comments and last access date.
- Remove bookmarks.
- Present users with a table of bookmark entries.
- Allow users to search comments (string matching).
- Allow users to open bookmarks in a web browser directly from the bookmark manager.

The task was carefully designed to be sufficiently challenging for the group, but not so complex that meaningful progress could not be made in the time allotted.

Though groups were informed that they could perform the task however they preferred (e.g., by working jointly), we recommended that the task be subdivided into functional components, and each person be assigned one of the components. For example, one person could create the component that parsed the input data, another could develop the graphical interface, and third could handle the search. The group was asked to produce a single integrated demo.

Because ClubDev had already been tested with MS Visual Studio, we instructed the group to use this particular application for generating the source code. However, the group could use any language supported by Visual Studio (e.g., C, C++, or Visual Basic). Groups were also informed that they could use any application desired for other parts of the task. Each user received $15 for participating in the study.

Procedure and Workspace
The experimenter met with recruited groups to go through an informed consent process and schedule a time for the programming task. The day before the scheduled session, the description of the task was e-mailed to the group. This allowed group members to begin thinking about possible solutions and how the task could be divided prior to the scheduled session, thereby maximizing the amount of time available for writing, testing, and integrating code.

The task was performed in our department’s HCI lab. The workspace was configured such that users were sitting around a large conference table with HP tc4400 laptops as their personal devices and a large NEC 61” plasma display positioned nearby. The large display was driven by an independent PC. This configuration is representative of those commonly used in professional settings (see Figure 1).

The experimenter demonstrated the functionality of ClubDev and provided time for the group to practice using it and ask any questions about the system or task. Users reported that they had read the task description prior to the study.

The group was given an hour to complete as much of the task as possible. Users then completed a questionnaire and participated in an open-ended discussion about ClubDev. The session was videotaped, and the tapes were later analyzed to understand how the framework was utilized.
Results
We discuss results from our study, focusing on how groups utilized our framework, users’ reactions to the features and functionality of the framework, and notable opportunities for improvement.

Use of the framework. Groups made extensive use of our framework throughout the task. For the hour that a group was engaged in the task, each group member had made available a total of 4 windows on average and typically had more than 1 window available at any given time. Also, each group member created a total of about 9 replications on average. Typically, replications were created in response to a new collaborative sub-task and were closed once the sub-task was complete. Each group member replicated at least one of their local application windows to the large display.

The framework was primarily used in these contexts:

• To request and receive assistance. There were several instances during the task where a group member would request help or advice on their part of the code, e.g., to understand and correct a compiler error. In these cases, the framework was used to replicate the source code editor window between their personal devices, allowing the code and potential solutions to be more easily discussed. For example, it was common for each group member to interact with the editor window to reference specific lines of code, propose ideas, or capture agreed upon solutions.

• To create a repository of reference information for the task. A group often found it necessary to research documentation, sample code, or existing code in order to provide necessary understanding for proceeding in the task. Since this reference information was generally beneficial to the entire group, our framework was utilized to place the relevant task artifacts on the large display. Also, multiple users contributed task artifacts to this repository.

• To integrate individual contributions. Once group members completed their individual tasks, they would attempt to integrate and test their pieces. In these cases, the group members would place their source code editors onto the large display and work together to perform the integration. For example, group members would advise each other on how to make appropriate calls into their respective parts of the code. In these cases, the ability for each group member to place their local application windows onto the large display was particularly beneficial.

• To maintain awareness of group members’ progress. As group members worked on their individual assignments, it was often necessary to see how another group member was structuring her code. For example, one user wanted to ensure that he was using compatible data types while another wanted to verify cross-functionality between components. Our framework was utilized to provide this awareness in two ways. The first was for each user to place a replication of his or her code editor window onto the large display. This allowed the users to glance at the display to extract information they needed without interrupting group members. A second method was for each user to show her code editor so that other group members could replicate it as needed to extract information.

Though not exhaustive, these examples demonstrate that development teams want and are able to utilize our system to quickly share task artifacts, transition between individual and (sub)group work, and maintain awareness. We felt this was a very positive result given the relatively short duration of the task and that none of the users had previously used our system or had prior knowledge of it.

Many of these scenarios (especially their sum) could not have been easily realized with existing solutions. For example, if the group had token passed a VGA cable to the large display (or used an input switch), multiple users would not have been able to place artifacts on it at the same time. In addition, working jointly on a single display would not have allowed them to divide the labor on the task or work in subgroups to solve specific problems.

User reactions. When asked about the framework’s features and functionality in the questionnaires and ensuing discussion, users provided many insightful comments about what aspects of the framework worked well and what didn’t.

Users were very enthusiastic about the framework’s support for different modes of collaboration. For example, one user stated that our framework allowed him to “work independently but easily and quickly be able to see [other] code that is being worked on and join in when needed.” Another user stated, “It’s very useful for big projects where each programmer likes to work on their own computer but at the same time they can give their code to their teammates and be able to [get] comments and help.”

The ability to place task artifacts on the large display was also appreciated. For example, one user stated, “I found it extremely useful to be able to quickly glance up and see what my partner was doing and what files he was modifying”. While another stated, “It’s good for highlighting items and for clear communication. It provided a sense of security because I know exactly what they can see, there is no guess work”.

Users also found value in being able to quickly replicate windows between personal devices. As one user stated, “It’s definitely useful for looking at other’s code — especially because I’m rusty with C++. I could easily see what [a team member] was doing without having to look over his shoulder.” Others commented that this type of replication would be especially useful when a large display was not available or when group members were working in separate offices.

Opportunities for improvement. Results from the study revealed several opportunities for improving the design of ClubDev and similar systems:

• Support cut-copy-paste operations involving replicated windows. On several occasions, we observed users attempting to copy/paste information between replicated windows or between a replicated and local window. Because the windows do not share a common buffer, the operation could not be performed, and was frustrating to users. A partial solution would be to maintain shared
A goal of our system is to foster sharing of task artifacts and input during group-based software development activities. Overall, results from our study strongly suggest that the use of our system could indeed have a positive impact. For example, as one user stated “It made me more useful, and I was better able to help my team” while another stated “it was extremely useful to just be able to quickly see what it was my partner was doing; I could quickly help check for bugs or answer a question.” Though targeted for group-based software development, our system may have value in other task domains, e.g., co-located collaborative writing.

Through the process of designing and developing our system, we encountered several challenges introduced by the use of a replication model. One challenge concerns the potential for someone to use a replicated window to gain access to another’s private information. For example, consider a team manager who is sharing a document editor that contains non-sensitive information. Any user who creates a replication of that window could select File Open and gain access to confidential documents on the owner’s local device (e.g., performance reviews). To protect against this type of security breach, our system prevents Open and Save dialogs from being rendered in replications. Future work should investigate alternative levels of security (e.g., allow the open dialog to be shown, but only allow files from a group share to be listed).

A second challenge is that the replication model does not explicitly support relaxed WYSIWIS [28]. For example, when sharing a replicated view of a code editor window, two or more users cannot view different sections of the code in parallel. Utilizing functionality already available within some applications could provide a partial solution to this issue. For example, applications such as Emacs and MS Excel allow the view of the content to be split. If the view within the code editor window could be split in this fashion, it would allow users to independently view different parts of the code, as their input streams would now be sent to separate controls within the source application.

A third challenge is keeping the network load to a manageable level when many replications have been created. Our system currently tests for a difference from the last frame before grabbing and sending a source window update, applies a simple compression scheme, and uses only a modest frame rate (10 fps). Future work should seek to further reduce the network load by applying more advanced compression as well as by pausing updates when a replicated window has been minimized or is not currently in view.

ClubeDev was described as a single system, but the interface is decoupled from the underlying system framework. This would allow other researchers to create and test new interfaces and interaction techniques on top of our existing system framework, thus saving a large implementation effort.

CONCLUSION AND FUTURE WORK
Existing tools do not adequately support the emerging practice of group-based software development activities. Our work has made several contributions addressing this challenge. First, we conducted a series of surveys and interviews with developers engaged in this emerging practice, and produced lessons that can guide the design of systems that seek to support it.

Second, we produced a new system framework that enables any off-the-shelf application to be shared and interacted with across devices. We also produced a new interface that allows group members to interactively control what application windows are available to the group and when replications of those applications are created. Results from an initial evaluation showed that users want and are able to utilize our system during group development activities.

Finally, practitioners can download and use our system for their own group development activities, while researchers can utilize our system framework to create and test new interfaces for co-located collaborative work. ClubDev is available at http://orchid.cs.uiuc.edu/projects/ClubeDev.

Beyond fixing known usability issues, the primary direction of our future work is to conduct a field study to better understand how our system is used for and impacts group-based software development.

ACKNOWLEDGMENTS
We would like to thank the developers who participated in this work.

REFERENCES
1. Concurrent Versions System (CVS).
   http://www.nongnu.org/cvs/
2. Microsoft LiveMeeting.
   http://office.microsoft.com/livemeeting
   http://www.microsoft.com/windows/netmeeting/
4. Subversion (SVN).
   http://subversion.tigris.org/
5. Timbuktu.
   http://www.netopia.com/software/products/tb2/


8. Benford, S., et.al., Designing Storytelling Technologies to Encourage Collaboration Between Young Children. in Proc. CHI, 2000, 556-563.


