A Social Collaborative Distributed Software Development Environment

A Dissertation

Presented in Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

with a

Major in Computer Science

in the

College of Graduate Studies

University of Idaho

By

Hani Ahmad Bani-Salameh

August 16, 2011

Major Professor: Dr. Clinton Jeffery

Authorization to Submit Dissertation

This dissertation of **Hani Ahmad Bani-Salameh**, submitted for the degree of Doctor of Philosophy with a major in Computer Science and entitled "**A Social Collaborative Distributed Software Development Environment,**" has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor: _____ Date: _____
Dr. Clinton Jeffery

Committee member: _____ Date: _____
Dr. Jim Alves-Foss

Committee member: _____ Date: _____
Dr. Robert Rinker

Committee member: _____ Date: _____
Dr. Barry Willis

Department Administrator: _____ Date: _____
Dr. Gregory W. Donohoe

Discipline's College Dean: _____ Date _____
Dr. Larry A. Stauffer

Final Approval and
Acceptance by the College
of Graduate Studies: _____ Date: _____
Dr. Jie Chen

Abstract

Software engineering is usually a team effort. Distributed software development needs real-time collaboration tools that help replicate the benefits of face-to-face meetings and support interaction among team members. Unfortunately, most of the tools that exist have limited capabilities, such as source code editing, and developers face collaboration and communication challenges in working with each other.

Over the past decade software researchers have invented various development tools that integrate collaborative features. Unfortunately, most software developers still lack the right means and level of communication to coordinate their work and perform their tasks effectively, particularly in distributed settings. Developers still lack a sense of remote team members' presence, and in some contexts, such as open source development, they still find it difficult to find appropriate project partners. This draws attention to the need for community integration inside collaborative development environment tools. This dissertation investigates collaboration and communication mechanisms for distributed software development. The research results include solutions to problems that are associated with providing online awareness and presence information, and determine the prospective gains from the use of these solutions.

This research contributes the core idea and novel design and implementation techniques for a real-time social collaborative integrated development environment (IDE). It describes the design and implementation of a social collaborative IDE inside a virtual environment named Collaborative Virtual Environment (CVE) where users can interact with each other within a 3D virtual world. A subsystem of CVE called Social Collaborative Integrated Development Environment (SCI) is presented that supports communication and collaboration within a distributed software development community, and integrates community and social features inside the development environment. SCI addresses the communication and collaboration needs in a variety of different phases in a team software development process, unifies the concepts of social networking and collaborative IDE, and integrates presence information and collaborative development tools into a single environment. In addition, this dissertation relates the contributions to previous research and describes directions for future work.

This dissertation discusses social networking features that offer a context, and provide activity, and presence information within a development community. A central component of these features is an awareness monitor that shows the team members' activity, and provides general awareness of projects' progress and development changes, and makes them easy to track.

Acknowledgment

Every dissertation is a joint effort and mine is no exception. I would like to express my gratitude to all the people who supported this work. It has been a long journey involved with major research challenges and overcoming critical developments obstacles.

I would like to thank my major professor, Dr. Clinton Jeffery, for his critical eye and foresight. It has been pleasure learning from his experience and insights for these past six years. Without his early guidance and encouragement, my years as a Ph.D. student would have been far less productive. I thank him for his encouragement, patience, support, and knowledge.

I would like to thank my committee members: Dr. Jim Alves-Foss, Dr. Robert Rinker, and Dr. Barry Willis, for their encouragement and constructive criticism about the presented research and the formatting of this dissertation. They have been a strong supporter of my work and they provided invaluable feedback.

I would like to thank Mr. Bruce Bolden and every member of the computer science I class from the Summer 2011 session who participated in the evaluation studies. This work would have been much more difficult without them.

Finally, thank you also to the corporate and governmental sponsors of this work. This research was supported in part by a grant from the National Science Foundation under agreement number DUE-0402572. In addition, this work was supported in part by the Specialized Information Services Division of the U.S. National Library of Medicine.

Hani Bani-Salameh

August 16, 2011

Table of contents

List of Figures

List of Tables

Part I

Introduction and Background Research

Chapter 1
Introduction

Software Development is a complicated task that is strongly based on the people that carry it out. -Orit Hazzan [1]

Software Development is a social activity. It is not just an engineering endeavor, it also essentially involves human interactions. Going back to common trends such as increasing distribution among software development teams, software development research faces an essential need to shift from personal productivity to "groupware" and "social" software engineering. Software development faces geographic and social boundaries, when many participants are engaged in the same development work. These barriers have a severe impact on communication, collaboration, interaction, and productivity issues.

Software development teams do not work in isolation from each other, and although they might be co-located or globally distributed, they need to continually interact. They communicate by meeting face-to-face, using various tools such as phones, emails, and text and audio chat, and through shared artifacts such as mailing lists, bug reports, shared software documents, and the project code itself. Despite being spread out in time and space, teams still have a deep need to interact with each other to improve their software development quality and productivity.

Software development teams perform many tasks working together. They ask their peers to help review code, design solutions, and accomplish projects' goals. To meet their goals, software development requires that teams be aware of other teams and their projects, activity, and interests. Unfortunately, while plenty of research has been undertaken to improve software development tools, relatively little has focused on supporting and integrating the social side of software development.

Software development projects present developers and team members with various communication and collaboration problems. As a result, other problems may arise in which team members duplicate work, overwrite changes, or write code that unfavorably affects other parts of the project. These problems occur because of a lack of awareness about what is happening elsewhere in the project. Current tools are focused around integrating collaborative features, with less focus on the presence and awareness of developers' activities, ignoring the fact that without better awareness information about team members, achieving smooth collaboration is challenging [2].

Gutwin et al [2], go on to observe that developers noticed that the lack of support for the social side is problematic for large-scale software development projects' success, because communication is a key to success. They recognize that there is a need for tools that integrate presence and activity awareness information, communication, and collaboration into a single environment. Using such an environment would provide tools that allow developers to collaborate and interactively share required information regardless of their geographic and/or time zone differences, and without having to juggle an assortment of disconnected tools.

Programmers use Integrated Development Environments (IDEs) in their everyday activities. They use IDEs in different editing and debugging activities, and often need to interact with people who share common interests to help solve particular technical problems. Consequently, IDEs are a primary venue for capturing and presenting presence and activity awareness information, and inventing new kinds of collaboration support for software development.

Social networks build online communities of people who share interests and/or activities, allowing users to explore the overlapping interests and activities of others. IDEs and social networks complement each other in collaborative software development, because software development communities *are* social networks.

To summarize, the evolution of programming environments from command-line tools, to IDEs, then finally to collaborative integrated development environments, along with the emergence of social networks are signs that embedding social activities, awareness of projects and team members (developers), and other varied social software features can help support distributed software development teams to perform their tasks and accomplish their goals.

1.1 Problem Description

For a long time, researchers have worked to create effective tools that help developers to collaborate, seamlessly switch between project artifacts, avoid coding conflicts, and reduce the total amount of time required for the coding, designing, and development. Many groupware systems are available for distributed developers; most target specific software development tasks, such as source code editing. Most synchronous groupware tools are aimed at teams and do not support distributed software development communities.

This work presents SCI, a real-time social collaborative integrated development environment. The dissertation covers both the technical issues in SCI's design and implementation, and its usability study. SCI is accessible over the internet and serves as a virtual collaborative development environment. The SCI system provides software development communities with

social activity, presence, and awareness information of team members, other teams, active projects, and current debugging and coding sessions. It also assists developers to find appropriate project partners from inside the development environment.

1.2 Research Hypotheses

The research project presented in this dissertation will test two hypotheses. The first hypothesis is that integrating social networking features inside a CDE makes the CDE an effective/usable classroom programming environment. This hypothesis is tested by the case study presented in Chapter 7.

The second hypothesis, the main focus of this dissertation, is that the combination of social networking with a virtual environment will provide the collaborative IDE with the online presence and awareness information that increases use and usefulness of collaboration tools. The virtual environment adds a collaborative social environment to the IDE and provides a "social presence"- the sense in which developers feel that other developers are present in the development environment. The test of this hypothesis utilizes the results from both the case study presented in Chapter 7 and the evaluation study presented in Chapter 8.

1.3 Contributions

The contributions of this dissertation are as follows:

- Augment ICI (Idaho Collaborative IDE) by the distributed development collaboration tools as part of the development environment (eg. real-time text editor, and real-time debugger), and to help reduce cognitive context switches between tools inside the development environment and between its tasks and virtual environment activities, allowing developers to share, in real-time, the process of editing, compiling, running, and debugging of their software projects. The complete tools design and implementation are presented in detail in Chapter 4.
- An awareness and presence augmentation for distributed software development projects and distributed communities support are presented along with a fully functional social collaborative development environment called SCI (Social Collaborative IDE). The awareness and presences requirements and features are presented in Chapter 5.
- A notifications' management approach is implemented inside the SCI system to manage incoming and outgoing invitations and requests. The implemented design is a medium fidelity design that aims to partially address the notification problems. The goal is to avoid making notifications interruptive, while future work directions address the issue of minimizing the

notifications' response time, and increasing the degree of intrusiveness (attention draw). This feature is presented in Chapter 6.

— A qualitative evaluation of the SCI system in a software engineering classroom. User observation, informal discussions and feedback via a questionnaire gave promising facts about the system. Students have reported that the tool eases communication between them and their project partners and that the tool presented them with passive awareness information that prevented them from affecting other's work and conflict changes while working on the project artifacts. This is presented in both Chapters 7 and 8.

1.4 Organization

This dissertation consists of three major parts; the dissertation structure is shown diagrammatically in Figure 1.1.

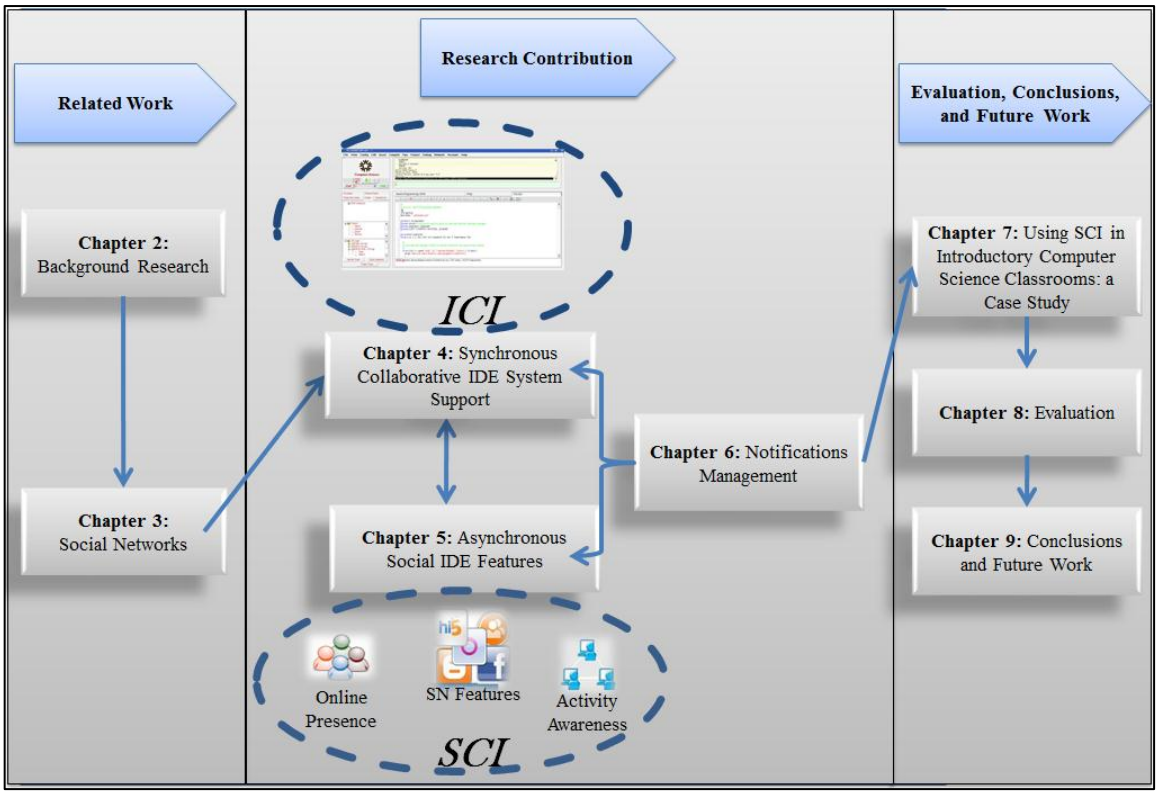

Figure 1.1 Dissertation Structure

The first part gives a research background and investigates previous related work. Following the introduction, related systems and research are discussed in Chapter 2. Chapter 3 defines social

networks and developers' social networks and discusses some examples of the existing social networks targeted at software development and developers.

The second part gives a detailed discussion of the dissertation's major preliminary research contributions. Chapter 4 describes a collaborative IDE (ICI) subsystem developed for this dissertation. It discusses the design and implementation and the interaction between its various components. The design and implementation of the social collaborative IDE (SCI) subsystem is the topic of Chapter 5. Chapter 6 covers in details the notifications' management inside the SCI and the effect of this in eliminating the distractions happen to users from the invitation pop-up windows while achieving their tasks, and its effect on SCI usability.

Finally, the third part includes the rest of the dissertation, and covers both the evaluation and conclusions. Chapter 7 describes a study that tests the usability of the system and contributes to a test of the hypotheses. An evaluation study is described in Chapter 8. Chapter 9 presents conclusions and future work.

This dissertation has six appendices:

- Appendix A: list of functional categories that represent a logical taxonomy for groupware tools.
- Appendix B: A list of both ICI and SCI network protocol messages.
- Appendix C: SCI's Real-Time Collaborative Editing.
- Appendix D: A BNF grammar that describes the structure for the network messages.
- Appendix E: Instructions Sheets (I - III).
- Appendix F: SCI's collaborative programming and usability surveys (I - III).

1.5 The state of this dissertation

In this dissertation all of the major proposed components are implemented, including: online presence, activity awareness, and social network features. The implementation of the SCI major components and its architecture are part of the CVE's source code distribution located at http://cve.sf.net.

Chapter 2
Background Research

This chapter reviews the current state of the art in collaborative development systems and establishes the context for this dissertation. The importance of collaboration within software engineering is addressed, followed by detailed information about collaborative systems for software development. Groupware architectures and groupware applications are discussed, and various research systems that fall within these classifications are described. Finally, Section 2.3 presents research related to collaborative software engineering.

2.1 The Importance of Collaboration

Collaboration is at the heart of software development, and it involves interactions between developers. Cook [3] defines collaboration as any form of interaction between distributed software developers or teams. To achieve a common objective, software development requires collaboration among developers within and occasionally outside their project teams. Previous research studies have shown that almost 70% of software development time is spent on collaborative activities [4].

In fact, research indicates that the way developers work together determines the success of the project [3]. Researchers and industry have produced a wide range of collaborative tools, and proposed many definitions of collaboration.

Collaboration is becoming more intertwined with daily aspects of software development processes, which require participation and interaction between team members. Consequently, software development processes are becoming social activities where key decisions are made in the context of collaborative groups.

Collaborative development environment tools cover several related disciplines of computer science. For a successful study and implementation of the development environment tools, knowledge of the related disciplines is required. The rest of this chapter presents related topics in the fields of *Groupware*, *Computer Supported Cooperative Work (CSCW)*, and *Collaborative Development Environments (CDE)*. Also, it presents the integration of source code control systems, awareness support, and online presence inside those fields. As illustrated in Figure 2.1, "Social CDE" forms an intersection of these overlapping fields.

Most of these tools focus on a few aspects of collaboration. To understand the functionalities of existing collaborative tools and how they compare with one another, several classification frameworks have been proposed. Appendix A provides a short discussion of some of these frameworks.

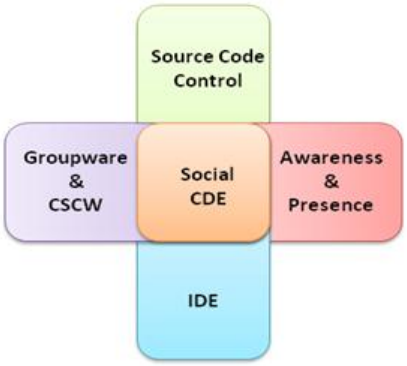

Figure 2.1 Disciplines Associated with Social CDE

2.2 Research Related to Collaborative Software Engineering

This section presents an overview of previous work towards Groupware and Collaborative Development Environment (CDE). CDE is a term coined by Grady Booch [5, 6] to describe applications that allow distributed developers to collaborate and work together.

2.2.1 Collaborative Software Systems (Groupware)

> *The reason I invent is to advance the evolution of society and its institutions. My crusade is to find much better ways for people to work together to make this world a better place. -Douglas Engelbart (father of groupware) [7]*

Before presenting details about groupware, it is appropriate to introduce the Computer-Supported Cooperative Work (CSCW) discipline. The term CSCW was coined in the mid-1980s and has many aliases including “computer-supported collaboration” and “workgroup computing”. According to Grudin [8], CSCW is the study of how people use tools and technology to work together in shared time and space.

CSCW examines the design, implementation, and use of groupware. CSCW is not just about “cooperation” or “work”, it also examines competition, and socialization. This field attracts those who are interested in software design and social behavior, including business professionals, computer scientists, psychologists, and communications researchers, among other specialties.

Groupware was first coined in 1984 by Peter and Trudy Johnson-Lenz at the New Jersey Institute of Technology. According to Ward [9], Trudy Johnson-Lenz defined groupware as an "intentional group processes plus software to support them". The term refers to a specific class of tools such as email, bulletin boards, asynchronous conferencing, group schedulers, group decision support systems, screen sharing software, whiteboards, video conferencing, newsgroups, and chat. Groupware is not just technology, it is also social. Groupware is collaborative activity that impacts the way people communicate with one another. In other words, groupware can be considered people as much as it is a tool that people use [10].

Groupware supports job functions that require people to work together, even though they might not be together, in either time or space. Groupware systems enhance the sharing of information and ideas between distributed team members and make their effort more efficient.

CSCW's primary concern is with aspects of the behavior of people and organizations. This is implicit in both of the "Groupware" and "CSCW" terms. CSCW and groupware research focuses either on utility and tools, or on usability.

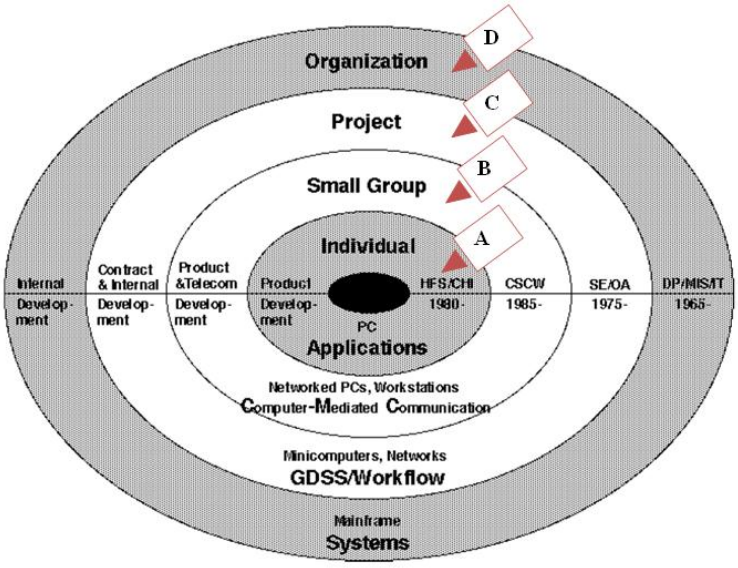

Figure 2.2 CSCW and Groupware in the Development and Research Contexts [8]

Figure 2.2 shows the classical representation of the US research and development contexts for CSCW and groupware, where each ring defines a work level (organization, project, group, or individual). "*The outer ring represents entire systems designed to serve organizational goals, such as MIS systems. The inner ring represents applications designed for individual users, such as word processors, spreadsheets, and games. The middle two rings represent* ***groupware****, designed with groups in mind* [8]*."* The reason why Figure 2.2 appears in this dissertation is to

show where its contributions fit as a CSCW and groupware, and that it fits somewhere between single user applications and information systems that supports organizations.

Various groupware systems have been developed to support particular forms of CSCW. This section provides an overview of these systems. Table 2.1 compares the groupware systems against specific criteria defined by Rama et al [11]. These criteria include functional, architectural, focus, time, and user involvement.

	System
Functional	Messaging
	Conferencing & Electronic Meeting System(EMS)
	Group Decision Support
	Document Management
	Document Collaboration
	Compound Document Management
Architectural	Central
	Replicated
	Hybrid
Focus	User Centered
	Artifact Centered
	Workspace Centered
Time	Synchronized
	Mixed
	Serial
	Unsynchronized
User Involvement	High
	Medium
	Low

Table 2-1 Common Groupware Criteria

According to Dover [12], Dix et al. divided groupware applications, based on the nature of the communication, into three groups.

— ***Computer-mediated communication*** aims to improve the communication between users. Such systems can be either asynchronous (such as email, bulletin board systems or news

feed), or synchronous systems that include types of computer assisted conferencing occurring in real-time (such as chat rooms, videoconferencing). They fall in the inner middle ring set of applications (**A**) shown in Figure 2.2.

— ***Meeting and decision support*** systems aims to help users establish common understanding about the task they support and generate ideas. Such systems can be asynchronous, for example, tools that record the discussions between users. Meeting rooms are usually synchronous and co-located; they support groups in face-to-face meetings. Shared drawing tools support synchronous meetings where users express their ideas in a shared drawing area. They fall in the outer middle ring set of applications (**B**) shown in Figure 2.2.

— ***Shared applications and artifacts support*** interaction of users with shared tools/artifacts. There are many systems available, including shared editors that allow n-users to create documents in a controlled synchronous manner, co-authoring systems that allow for asynchronous collaboration on documents, and shared calendars that allow for the collaboration of several users on a combined schedule. They fall in the inner middle ring set of applications (**A**) shown Figure 2.2.

Table 2.2 (adapted from [9]) shows a range of systems and applications specified as groupware tools.

Characteristics of system	Characteristics of environment	Author
System, group, processes, software		P. & T Johnson-Lenz 1978 in Ellis, Gibbs & Rein 1991
Applications		Grudin 1989
Applications	Networked computers, large databases	Ellis, Gibbs & Rein 1991
Any technology to support group productivity		Nunamaker, Briggs & Mittleman 1994
Hardware, and software	Groups working together	Dennis, Quek & Pootherie 1996
Applications	Networked environment	Byrne 1997
Networked based software, web pages, electronic bulletin boards, discussion lists, file sharing, synchronous or asynchronous	Shared hypermedia environment	Greenlaw 1999

Table 2-2 Systems Specified as Groupware

To summarize, within this dissertation, groupware is defined as: a research area that studies and supports the use of both synchronous and asynchronous software engineering tools for

different tasks, programming languages, and software development processes, to help groups of developers work together towards on a specific goal.

2.2.1.1 Asynchronous Systems

Asynchronous groupware supports communication and problem solving among groups of individuals who contribute at different times, and typically also are geographically dispersed. - (Baecker 1995, p. 743) [13]

Asynchronous groupware are group support systems that aim at supporting the distributed interaction occurring at different times. Such systems provide loosely-coupled interactions, so that users can modify the shared artifacts or code pieces without having any awareness or direct knowledge of the changes made by the other users, either because they work at different times or because they do not have access to each other's activities and actions [14].

2.2.1.2 Synchronous Systems

Synchronous groupware [15, 16], is a category of software application that facilitates real-time collaboration among co-located and geographically distributed group members. Synchronous groupware includes application sharing, voice and video conferencing, instant messaging, shared whiteboard, multi-player virtual games, shared editors, and group decision support systems. Day [17] defines synchronous groupware as *"the class of applications in which two or more people collaborate in what they perceive to be real time"*.

According to Graham [14], "*the defining property of synchronous groupware is that it helps people to work together at the same time, allowing participants to immediately see the effects of other participants' actions. Synchronous groupware is meant to create group awareness, allowing people to work with the kind of direct communication they would have if they were all in the same room.*"

In recent years synchronous groupware has become widely used in at least the following three application areas [18]:

— ***Communication Tools****:* Tools such as MSN Messenger and Skype that allow people to communicate regardless of their distance (location), either by textual chat, or through voice over IP and video. They fall in the inner middle ring set of applications (**A**) shown Figure 2.2.

— ***Multiplayer 3D Games and Virtual Environments:*** Games allowing people to communicate, collaborate and compete have become enormously popular. Examples of such massively multiplayer games are World of Warcraft (www.worldofwarcraft.com) and Second Life

(http://secondlife.com) where people meet and socialize in a virtual world. They fall within both the middle rings set of applications (**A** and **B**) shown Figure 2.2.

— ***Electronic Meeting and Conferencing Tools:*** Tools that allow online meetings are increasingly used. Examples of such successful applications include GoToMeeting (www.gotomeeting.com), and WebArrow (www.namzak.com). They fall in the outer middle ring set of applications (**B**) shown Figure 2.2.

Developers in distributed development environments use many tools and applications to help them communicate and collaborate within the development community. The system presented in this dissertation integrates and merges such tools in a single environment in order to help reduce friction, by saving the developers from having to switch between those tools and manually dig for information. Also, the integration allows synergy where the interaction of those tools makes their combination greater than the sum of their separate effects, and reduces the steps required for finishing jobs than when tools are separated.

2.2.1.3 Groupware Architecture Analysis

Most groupware systems are inherently distributed [19]. A significant feature of groupware and CSCW is the distribution architecture that defines what part of the groupware application runs on a central server, and what parts run on decentralized sites, and how the decentralized sites are linked and cooperate with each other. The chosen architecture has an important influence on the way the groupware application is developed and used.

Choosing a distributed architecture affects the groupware application in several ways. For example, the application's response time and fault tolerance can be improved using decentralized architectures. In general, decentralized architectures scale much better than centralized. On the other hand, some runtime services such as storing documents can be developed much easier when using a central server or centralized distribution architecture.

Researchers in computer science usually consider three distributed architectures: centralized, replicated, and hybrid. The diagrams in Figures 2.3 and 2.4 (Figures adapted from [20]) illustrate the key components and communication paths between processes of a conceptual two-user collaborative system under fully centralized and replicated architectures [20].

Graham and others [18], identified five distribution architectures for synchronous groupware by examining the implementation design of existing systems. These architectures are (1) centralized core, thick client; (2) generic thin client; (3) centralized mixer with broadcaster; (4) replicated input broadcasting; (5) replicated state synchronization (see Figure 2.5).

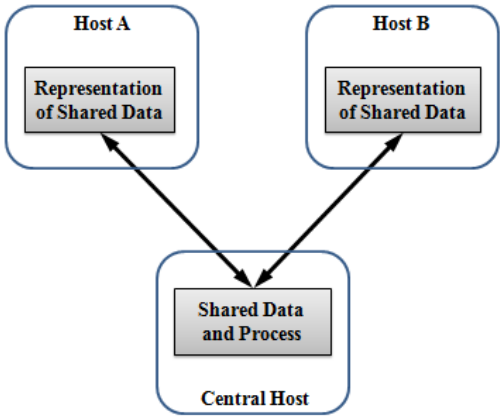

Figure 2.3 Centralized architecture.

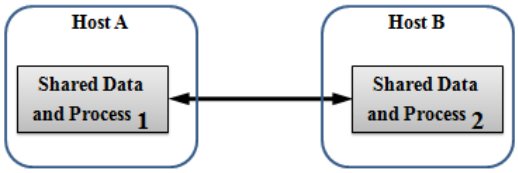

Figure 2.4 Replicated architecture.

The shared data, indicated by the shaded box are processed at a single host. A person on a remote host views and manipulates the data via a representation, indicated by a dashed box.

Each host contains and processes a full copy of the shared data. When the data changes on one host, all replicas must be made consistent.

Arrows indicate network traffic.

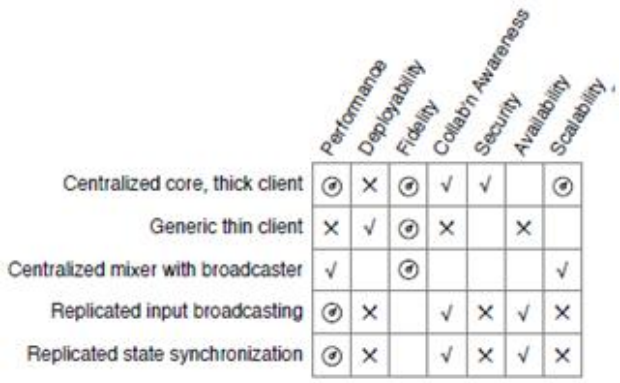

Figure 2.5 Distribution Architectures for the Development of Synchronous Groupware [18]

The work presented in this dissertation provides a synchronous groupware, which uses a replicated architecture that provides a low network bandwidth by distributing only changes to replicas compared with the centralized architecture that distribute graphical display information (eg. Microsoft NetMeeting) [20], and provides a fast response time to participants input. Although the presented groupware uses a "What You See Is What I See" (WYSIWIS) interaction mode, real-time individual work is supported by allowing participants to update their local replica and merge the changes with the original. Maintaining concurrency control among replicas is a challenge, SCI avoids this challenge by implementing a way to lock the developers who do not have permission.

2.2.2 Collaborative Development Environments

> *While traditional integrated development environments focus on improving the efficiencies of individual developers, collaborative development environments focus on improving the efficiencies of the entire development team.- (Grady Booch 2003) [5]*

A Collaborative Development Environment (CDE) describes an online meeting space where the developers can work together, regardless of their time-zone and region disparity, to discuss, edit, debug, solve, and produce project deliverables.

Research into collaborative software development environments is increasing rapidly. Factors that led to the rise of this research include mainly the advent of open source integrated development environments, the high need for distributed development, and big advances in the computer and programming fields. Software development encompasses a wide range of tasks, and collaboration tools exist for almost every conceivable task in software engineering (SE).

Tools of this nature, such as FIELD [21], Eclipse [22, 23, 24], Visual Studio [25], and CASE tools such as Rational Rose (http://www.rational.com) are often designed for large teams of software engineers, where numerous procedures and methodologies that require tool support are in place. The rest of this section highlights some of the tools categories.

The scope of this research studied tools that support collaboration during the implementation of the SE project artifacts. This category of tools focuses mainly on the code-level development [3]. Section 2.2.3 highlights some of the tools that fall under the development tools category.

Integrated Development Environments (IDEs) are one of the most heavily used tools in programmers' everyday activities. IDEs are where coding takes place and are the home of many different development tools. Developers may interact and collaborate with the other developers to obtain advice about the code, fix each other's bugs, and to get a general understanding of what is happening in the project artifacts. They use a variety of collaborative tools in their everyday activities. These include tools such as: source control, bug tracking systems, email, and instant messaging. Much collaboration in software development is accomplished using asynchronous technologies such as e-mail or revision control systems [26, 27].

Integrating such collaborative tools into the IDE, and enabling them with awareness of development processes and artifacts, may help reduce programmers' cognitive context switches between tools inside and outside the IDE and make the connection between development and collaboration more seamless [23]. Generic tools that provide shared views of an entire PC

desktop are bandwidth-intensive, have distracting session setup and teardown costs, and may provide more access and less control than is intended for a given collaboration.

Booch and Brown [5] point out that a rich collaborative development environment arises from the collection of many seemingly simple collaborative components to support coordination, collaboration, and community building. They also state that IDEs equipped with team-centric support and whose primary user experience focuses on the needs of the team, is a step up from those merely augmented with some collaborative support. The remainder of this section highlights some notable collaborative text editors (2.2.2.2) and more general CDE systems (2.2.2.3). Section 2.2.3 compares the listed CDE systems with SCI (Social Collaborative IDE) the proposed system in this dissertation.

2.2.2.1 Collaborative Editors

Collaborative editing is the practice of a group of individuals simultaneously editing a document. This section focuses on the synchronous collaborative editors. This section introduces two of those editing systems. A more detailed discussion of some of the existing systems is presented by Zafer [19] such as *ShrEdit* [28], *GROVE* [29], *SASSE* [30], *Calliope* [31], *Flexible JAMM* [32], *ITeach* [33], *MUSE* [34], and *NTE* (Network Text Editor) [35].

NetEdit [19, 36] is a collaborative text editor implemented in Java that uses a replicated architecture. It provides the user with centralized file and session management, flexible editing ability among groups, and chat session management. A user in a collaborative session can modify any part of the document, and can communicate with others using the chat room.

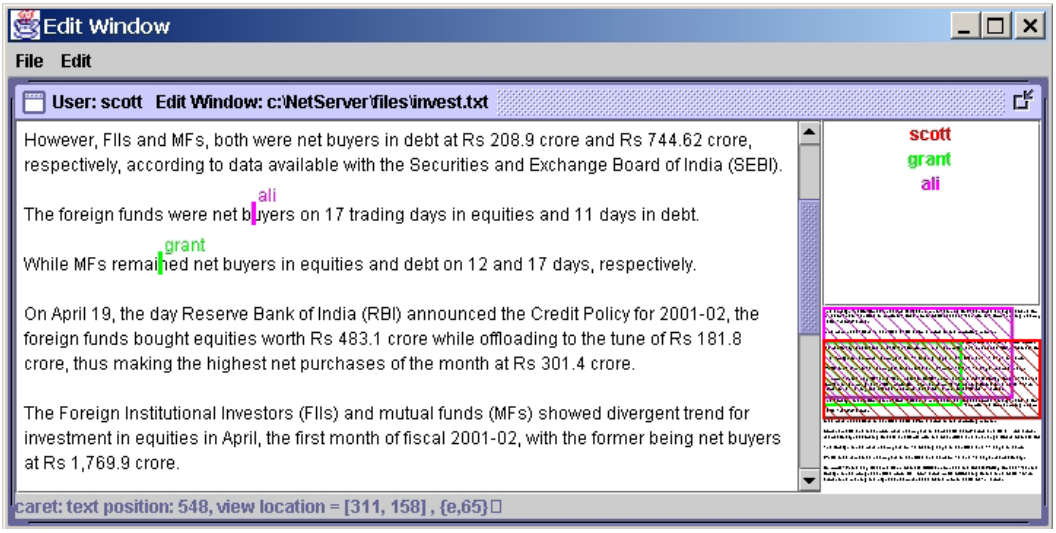

Figure 2.6 NetEdit's Main Editor Window [19]

NetEdit allows two or more users to remotely edit a document simultaneously (see Figure 2.6). It provides collaborative awareness functionality that allows following other users through the document and tracking changes. NetEdit handles the concurrency with an n-way consistency algorithm wherein each client has a state-space graph maintained by the server. Clients keep buffers for changes that have been made to its copy of the document, but have not yet been broadcasted to the other clients in the group.

GHT (Group Homework Tool) [37] is an educational groupware tool built to support synchronous, collaborative coding among novice programmers. In addition to the real-time editor, GHT provides an assignment definition and resource page, a chat system, and a shared whiteboard (see Figure 2.7). Also, GHT provides awareness information of users' work in the shared place.

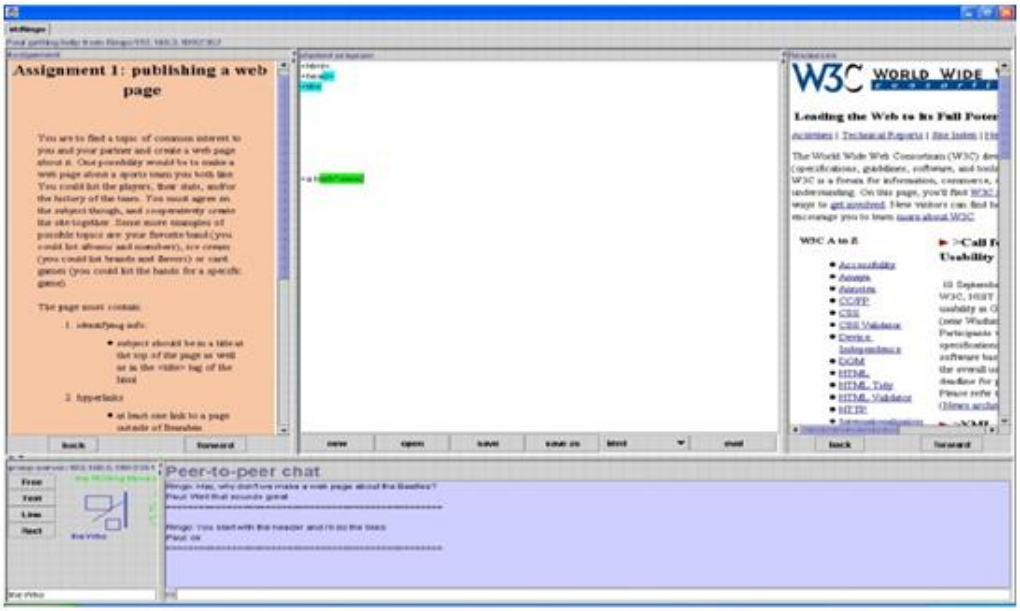

Figure 2.7 Main GHT Window [37]

2.2.2.2 Eclipse-based CDE's

Eclipse [23] is an open source development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle [22]. Eclipse itself does not support code-level collaboration, but a new Eclipse Communication Framework project [3] aims to support the development of distributed Eclipse-based tools and applications and to allow the Eclipse code repository and project model to be shared and collaboratively edited. Eclipse provides an API to perform basic sharing, along with some prototype client

applications. One such application is presented in Figure 2.8 where a shared graph editing tool is hosted within the Eclipse IDE [24, 38].

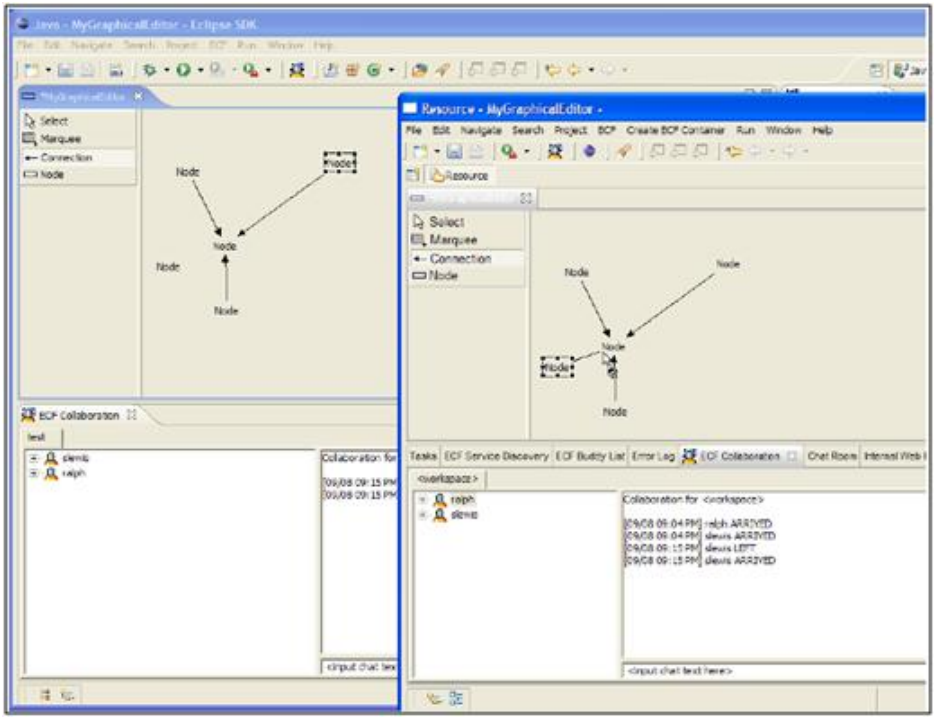

Figure 2.8 A Graph Editing Tool within the Eclipse Communication Framework [3]

A number of other Eclipse-based projects focus on the integration of collaborative features into IDEs. ***GILD*** is an example of a project that provides cognitive support for novice programmers and support for instructor activities [39].

CodeBeamer is a commercial product that has plug-ins for integrating collaborative capabilities into IDEs such as chatting, messaging, project management, and shared data [40]. Another example is Sangam, a plug-in for the Eclipse platform that features a shared editor and chat for pair programming [41].

Stellation (http://www.eclipse.org/stellation) is an open source effort (led by IBM Research) that introduces a fine-grained source control that supports the notion of activities and aims to simplify collaboration and provide awareness of changes to team members [38]. It enables developers to manage relevant work, notify the team of their current work, be informed of changes pertaining to their own activities, and provides a context for persistent conversations.

Palant´ır [42, 43] is another Eclipse plug-in that supports awareness features by showing which artifacts have been changed by which developers and by how much. Palant´ır provides

workspace awareness such that developers can monitor other teams' activities while working on their current task and without the need for switching contexts.

Jazz [26, 27] extends Eclipse with collaborative capabilities to support coordination, communication, and awareness among a small close-knit team of developers. As illustrated in Figure 2.9, the Jazz environment includes: a) Jazz Band showing teams, members, and status icons, b) menu offering communication options, c) decorators and tooltips on resources, d) anchored chat marker, e) code modification indicator, f) team member's status message [26].

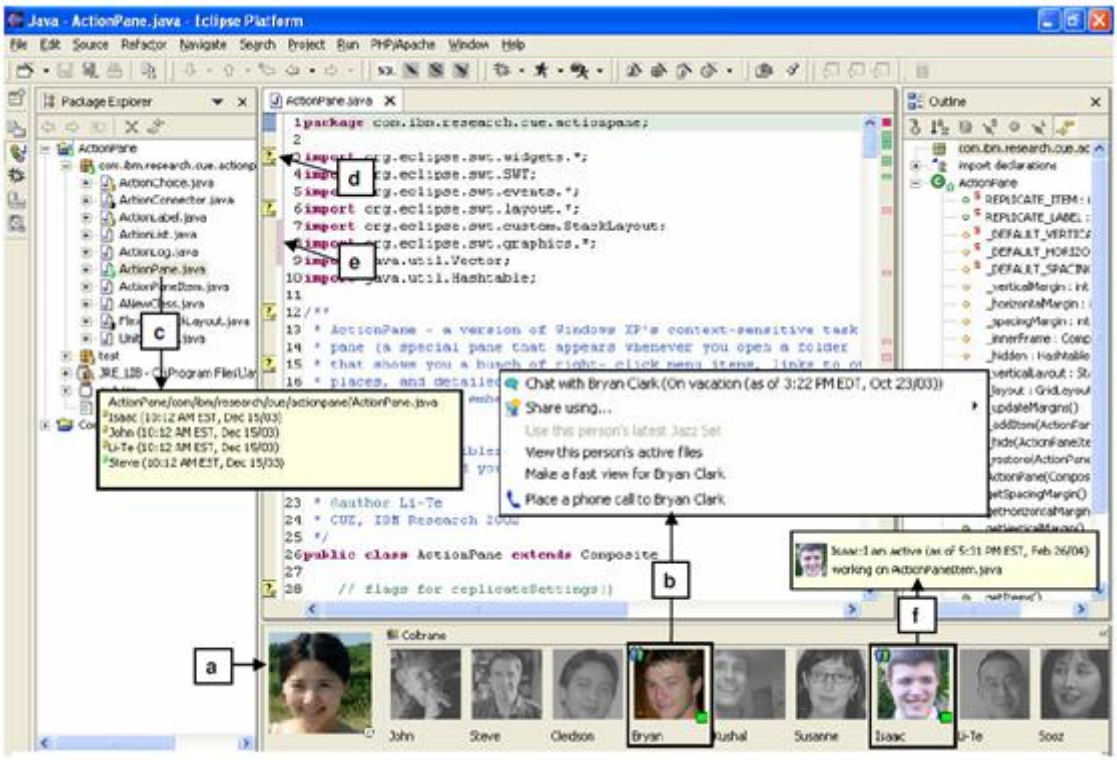

Figure 2.9 The Jazz Project Scene [26]

Jazz focuses on developing better team-building strategies, managerial processes, architectural designs, collaborative coding techniques, and software development practices. The Jazz goal is to provide a platform that allows integrating tasks across the software life cycle. However, Jazz also seeks to increase teams and teams' productivity [44, 45].

The objective is to raise the feeling of the other developers' activities as a social team, while capturing the team's artifacts to provide a better communication and collaboration environment. Jazz provides a facility similar to an IM buddy list to monitor who is online and coding or not. The IM status message shows the other developers what file the developer is currently working on. Developers can initiate chats, which can be saved as code annotations or into a discussion forum, or use screen sharing and VoIP (voice-over-IP) telephony, without the need for any

additional setup effort. Jazz also provides resource-centered awareness. Files and other resources in the file viewer are decorated with colored icons to indicate what other developers are doing with their local copies of the files (e.g., indicating that a file is in focus and being edited at this very moment or that a file has been locally saved but not checked back into the code repository). Also it shows who is responsible for these changes. This awareness information reveals cues normally associated with files and users, such as file size, last modified date, and who is responsible for the changes that incorporate traceability [26, 27].

Borland's JBuilder 2008 [46] is an enterprise-class Java IDE based on Eclipse (Figure 2.10). JBuilder is based on Eclipse 3.3 and Eclipse Web Tools Platform (WTP) 2.0. It supports real-time

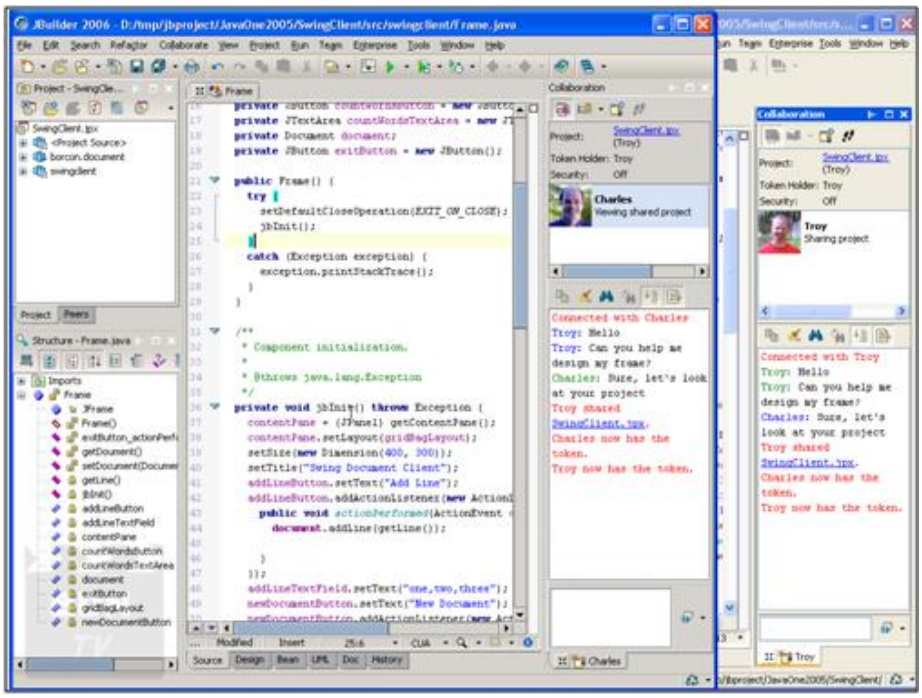

Figure 2.10 Borland's JBuilder IDE with Pair Programming Capabilities [46]

remote refactoring, distributed views of UML diagrams, and chat channels. It includes a collaborative development environment with integrated tracking, source code management, project planning, and continuous builds, allowing team members to easily monitor and manage project activity and progress.

JBuilder incorporates a shared pair-programming code editor and collaborative debugging capabilities [24, 46]. JBuilder provides useful support for version management using a CVS repository. JBuilder shows the history of the checked projects in a history tab that allows developers to view different versions of an application, revert to an older version or examine differences between versions, show the revision information and work with merge conflicts [47].

2.2.2.3 Other CDE's

CollabVS [48] extends the Visual Studio programming environment by adding text and audio-video chat, browsing of remote unchecked versions of files, and notification of presence in elements inside a file. Although CollabVS targets collaboration among developers, it still relies on the classical "*check in*"/ "*check out*" model and treats files as the lowest level of granularity. CollabVS provides two kinds of presence. *Real-time Presence* information lets the user know what the other team members are currently doing. It shows what users are online and whether they are editing, debugging, engaged in an instant messaging session, etc. *Contextual Presence* information facilitates finding relevant information and people quickly.

Collab.net [6] is a commercial CDE that has both a public and a private face. Collab.net's public face is SourceForge, an open source CDE that focuses on the development of open source software. SourceForge serves as a host to approximately 230,000 projects such as CVE (http://cve.sf.net). Collab.net's private face is SourceCast (http://sourcecast.org/) a CDE that supports a number of important features not in SourceForge, such as greater security that is not a big issue for open source development.

A user may enter this web-based application using a specific project or from a personal web page. Within a project, Collab.net supports features for artifact storage, simple configuration management via CVS, bug tracking, task management, and discussion boards. SourceForge offers other features such as the ability to self-publish, change tracking, and project membership management.

2.2.3 Comparison to SCI Tool

Before presenting the SCI tool, it is worthwhile comparing the features and abilities of existing collaborative development environment (CDE) tools. The CDE tools presented previously in this section are categorized in Table 2.3. Table 2.3 shows that CDE tools vary in the number of supported features; most CDE tools are designed for specific purposes and in general do not need to support all software development tasks.

In the design of the SCI framework, a major goal is to provide many CDE features for application developers to utilize. However, instead of writing tools for all purposes of SCI, the research presented in this dissertation provides a framework that supports key categories listed in the features table, while focusing on the awareness, social presence, and social network features, and mainly the 3D virtual environment integration with the gains from the interaction among users' avatars and their online presence.

		GILD	CodeBeamer	Stellation	Palant'ir	Jazz	JBuilder 2008	CollabVS	SCI
	Synchronous Features	_	Partial	Partial	Partial	X	X	X	X
	Asynchronous Features	_	Partial	Partial	Partial	X	_	X	X
	Artifact Management	X	_			X	X	_	X
	Purpose	EDU	COM	DEV	DEV	COM	GEN	_	GEN
	Multiple Language Support	_	_	_	_	X	_	_	X
	3D Virtual Environment	_	_	_	_	_	_	_	X
Awareness	Social/Presence	_	_		_	X	_	X	X
Awareness	Historical	_	_	X	_	X	_	X	X
Awareness	Group-Structural	_	_	X	_	X	_	X	X
Awareness	Workspace (Artifacts)	_	_	X	_	X	_	X	X
SN Features	Groups	_	_	_	_	X	_	_	X
SN Features	Friends	_	_	_	_	X	_	_	X
SN Features	Help/Support	_	_	_	_	X	_	_	X
	Purpose: EDU (Education), COM(Commercial), DEV(Development), GEN(General)								

Table 2-3 Features Table of Existing CDE Systems

Over the past few years, there have been advances in the development and adoption of social awareness inside IDEs, which allow developers to communicate, collaborate, and form successful online communities. This dissertation does not cite them all but instead highlights various existing systems and research that provide interactive collaboration and awareness for multiple phases of program development.

To our knowledge, no existing tool supports the full "social networked IDE" features of SCI. Jazz, the closest relative, supports many elements of social networking. Like SCI, Jazz focuses on increasing the user's awareness of people, resources, and activities, and on fostering communication among team members. Both Jazz and SCI support synchronous chat discussions. Also, Jazz provides team-centric discussion boards that compare to the asynchronous news feed supported by SCI. User profiles are not supported by Jazz, where in SCI developers can view another developer's profile where they can see their friends, groups, projects, and activities. Jazz supports awareness of the committed code changes with respect to the code repository. In contrast, SCI provides the developer awareness information of the committed and uncommitted code changes, of the currently edited files, and indicates who is responsible for the changes. Jazz has a sophisticated setup and teardown process required to install and start both the server and the client, and occupies more than 500 MB of disk space. In contrast, SCI takes ~42 MB of disk space, and saves the setup time.

Two major differences between SCI and almost all of the related work cited in this section are 1) the integration of social network features inside the software development environment; a recent exception is that Jazz integrated social network features by bridging with SharePoint (details in section 3.4), and 2) the integration of the collaboration IDE within a 3D virtual

environment. Most of the cited projects have some social network-like features, but not others such as user profiles and news feed. Open source development platforms, such as SourceForge [49], Google Code (http://code.google.com/hosting/), and CodePlex (http://www.codeplex.com/), provide simple awareness mechanisms along with configuration management (CM) functionality. SourceForge has limitations in terms of what and when information is shared, and how the information is presented to the developers. It is difficult to maintain awareness across SourceForge's multiple communication channels. Perhaps the limitation appears where most open source projects inform developers only of conflicts related to specific project artifacts and ignore the other developer activities [50]. In contrast, SCI integrates multiple social network information sources and provides the user with better awareness about the other developers and the project artifacts. It provides what most other open source projects are missing: the overall view of other developer's workspace activities.

2.3 Summary

The relevant literature for this dissertation comes from the groupware and CDE fields. Many tools discussed here help to identify the research issues. The review showed that many tools lack support for the awareness and online presence. The review also demonstrated an absence of the main social networking features support.

Supporting CDE has been challenging to achieve since most conventional development environment tools are inherently single user tools, and cannot easily be augmented by Groupware to solve all CDE problems. Even though specific purpose CDE tools provide limited facilities to support collaboration among the distributed developers' community, a more general purpose solutions may be needed for a cohesive development teams.

A CDE solution, named SCI (Social Collaborative IDE), is supported that allows developers to be aware of the actions of others in real time, aware of artifacts changes, avoiding coding errors, and potentially improves the collaboration during the software development process. A CDE that combines: communication, collaboration, awareness, and social networking and online presence features from inside a single environment.

Chapter 3
Social Networks

Social networks have received great interest not just from scientists and researchers in various domains (eg. psychology, philosophy, education, and computer science), but also from business people. This chapter defines social networks and developers' social networks, outlines social networks' evolution, and gives examples of existing social networks targeted at software development and developers.

In order to implement the SCI development environment, and to augment it with presence and awareness information, identification of the social features that programmers need, in general, and the core social features that developers will require when using SCI, specifically, is essential. This chapter reviews the features of social networks and the issues relevant to their integration inside development environments.

3.1 Definition of Social Network

Most social networks are web-based applications that provide users with a variety of methods to interact and build their community, through email, instant messaging, posting comments, and friend search, and so on. Social networks have triggered new ways to communicate and share information for millions of people all around the globe. Social networks become an enduring part of those people's daily practice. Boyd and Ellison [51], offer the following definition for today's social networks:

> *"Web-based services that allow individuals to*
> *(1) Construct a public or semi-public profile within a bounded system,*
> *(2) Articulate a list of other users with whom they share a connection, and*
> *(3) View and traverse their list of connections and those made by others within the system."*

According to [52] and [53], social networks are a social structure of nodes that represent individuals or organizations, the relations that connect them within a specific domain or environment, and the interactions among them.

3.2 Social Network Research Advances and Applications

A social network is a social structure made of sets of linkages among nodes (generally individuals or communities). Nodes are linked by one or more specific types of interdependency or characteristics, such as ideas, interest, or friendship [54, 55, 56].

Social networks emerged as part of the Web 2.0 revolution that aimed to support communication, information sharing, collaboration, and functionality of the Web [56]. There are many social network sites, which are used by people to connect with friends, former classmates, and business people. However, social networks have been around since the early days of the internet. Early social networks started as generalized online communities such as the Well "Whole Earth 'Lectronic Link" (1985) that can be considered as one of the oldest communities. It's best known for its forums, and also provided email, shell accounts, and web profiles. According to Boyd and Ellison [51], SixDegrees.com is the first recognizable social network website; it launched in 1997.

This section presents a brief list of popular social networks. More detailed information about the evolution of the most popular social networks such as MySpace, Facebook, LinkedIn, Twitter, Friendster, LiveJournal, Orkut, Bebo, Hi5, and Classmates.com is available at [57,58]. A historical record of the evolution of social networks was made by Boyd and Ellison [51]. The following are examples of popular social networks:

MySpace (http://www.myspace.com) offers an interactive, user-submitted network of friends, personal profiles, blogs and groups, commonly used for sharing photos, music and videos.

Facebook (http://www.facebook.com) allows people to communicate with their friends and exchange information. Facebook launched a platform that provides a framework for developers to create applications that interact with core Facebook features.

OpenSocial [59] is a set of common APIs for Web-based social network applications. OpenSocial is commonly described as a more open cross-platform alternative to the Facebook Platform although its impact to date seems limited. Applications implementing the OpenSocial APIs are interoperable with other social networks that support them, including MySpace[60], NetLog [61], Friendster[62].

Twitter [63] is an online application tool that serves as blog, social network, and cell phone/IM, designed to let users answer questions and point to resources and interesting information by posting tweets. In a tweet, users can describe their current status in short posts (usually less than 140 characters) distributed by instant messages, mobile phones, email or the Web. Like other social networks, for example Facebook, Twitter lets users create "friendships".

Twitter works with cell phones and make it easier for mobile users to stay in touch virtually anywhere [64].

3.3 Developers' Social Networks

This dissertation defines developers' social networks as graphs of nodes that support the tracking of software developers' activities, preferences, and behavior. They give distributed developers the ability to draw upon implicit or explicit interactions and connections with other developers to do something useful.

There are few social networks that are targeted at software developers. As mentioned earlier, developers use many collaboration tools. If all these tools were supported in a single environment, that would be a step in the right direction. A tool that merges services such as wikis, blogs, workspaces, and basic group management is the start for social networking to support software development, but ideally, real-time software engineering collaboration tools should be integrated. A tool that combines features from social networks such as Facebook (http://www.facebook.com/), SourceForge (http://sourceforge.net/), and remote meeting service is a significant contribution to the software development community.

A few professional social networking sites such as Twitter, blogs (the Linux developers' network http://ldn.linuxfoundation.org/blog), or Stackoverflow (http://stackoverflow.com/) are available to help developers. Assembla (http://www.assembla.com/) is a social network of development teams that helps maintaining distributed teams and their projects. It provides an online workspace with tools and services (eg. code repositories, collaboration, and management tools) for accelerating software development. Such sites have potential because of the following two reasons:

First, they allow developers to ask, and hopefully find answers to their technical questions. Developers can ask the right person and save on wasted time looking the wrong direction such as searching the web and reading blogs which are still a huge resource and important sources of information, but often fail to deliver the needed information in a timely manner. From a technical social network, developers can get almost real-time response to their technical-questions (eg. Stackoverflow and Slashdot (http://www.slashdot.org)). This can result directly in a cost benefit for software development.

Second, they help developers keep up with new trends in technology when they simply ask questions and listen to the answers without engaging in social networking. Thus they can realize what other developers are doing and generally feel more aware.

Social networking can help the developer community increase productivity and efficiency. According to [65], Lesser and Storck mention that: *"Communities appear to be an effective way for organizations to handle unstructured problems and to share knowledge outside of the traditional structural boundaries. In addition, the community concept is acknowledged to be a means of developing and maintaining long-term organizational memory. These outcomes are an important, yet often unrecognized, supplement to the value that individual members of a community obtain in the form of enriched learning and higher motivation to apply what they learn".*

"*Social networking also assists in the extinction of information and capability silos by identifying subject matter experts and resources"* [66]. Information workers seeking these resources can use a social networking technology platform to discover individuals and teams that are related to their initiative or effort. This process, supported by the right technology, circumvents a traditionally elongated and arduous discovery and resource-seeking process [53]. The following section describes a few quality social networks for software developers.

3.4 Social Network Examples for Software Development

MydeveloperWorks: IBM introduced MydeveloperWorks social networking service with the motto: "social networking is the development process". MydeveloperWorks is a new way for distributed developers to connect and interact with their fellow developers. Using the MydeveloperWorks environment, developers can create their own personal profile and customize their home page to get instant access to the people, feeds, tags, bookmarks, blogs, groups, forums, etc. that they care about, and search through user profiles for those with like-minded interests. IBM's goal with MydeveloperWorks is to connect the global community of software developers and make it easier for them to create new technologies based on open standards such as Java, Linux and XML [67, 68].

Also, IBM added enterprise social networking features to the Rational Team Concert (RTC) development environment in their release of Mainsoft Document Collaboration for Rational Jazz mentioned on SPTechBlog [69]. Developers are able to: (1) Access SharePoint My Sites, with links to blogs and wikis, (2) View SharePoint Personal Profiles, and (3) Use SharePoint People Search. These features are available on top of the existing integration of SharePoint document libraries and workflows with the Jazz development process. Developers can view their other team members SharePoint My Sites from the RTC's team artifacts view (see Figure 3.1). Blogs and

wikis linked from My Sites are also listed in the Team Artifacts view, and they can be opened directly from RTC [70].

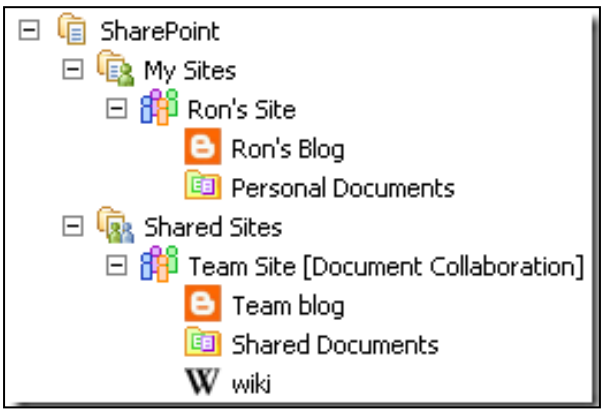

Figure 3.1 SharePoint **My Sites** View [69]

SourceForge.net: Recently SourceForge, one of the oldest and best-known code hosting services for open source software projects [71], acquired Ohloh, a social network for open source software communities. Ohloh analyzes the source code in version control systems in order to collect data about software and developers. Developers of Ohloh can set up profiles at the site which aggregates all of the information collected by Ohloh about their participation in open source software projects. The site aims to provide useful metrics about a huge number of open source software projects and makes it easier for developers to find each other and interact [72].

GitHub (http://github.com/) is a social network for programmers. It is a web-based hosting service for projects that use the Git revision control system. Git is a fast, efficient, decentralized version control system. Because Git makes it effortless to fork and merge projects' code, Git is ideally suited for the collaborative development of software. GitHub is a hosted Git repository service. It allows developers to participate and take part in collaboration: forking projects, sending and pulling requests, and monitoring development, all with ease [49]. GitHub provides social networking functionalities such as feeds, followers, and the network graph to display how developers work on their versions of repositories [73]. GitHub combines standard features of social networking sites with distributed source-control Git. Developers can follow or message a person, watch or fork projects and activity streams, and share their behaviors. Users are able to easily fork projects and create their own versions that can then be merged back to the original or take on a life of their own [74]. Every user and project has a profile which tracks progress and participation. Users and projects also have public activity feeds which display activity on public

projects such as commits, comments, forks, etc [75]. When using GitHub users can host their public and private projects, they can use it to collaborate on projects in a truly distributed way, and expand their social coding network [76, 77].

Zembly [78] is a socially-networked development environment from Sun. Zembly supported the development of cloud applications on platforms including Facebook, OpenSocial, iPhone and other platforms. With Zembly users can do social programming, and develop applications with other people using social networking-type features. Not only can developers reuse pieces and parts of other developers' projects (a work that they previously implemented to construct new applications), but also inviting friends and colleagues for collaboration. Also, they can see what colleagues are working on via news feeds, and keep up with what others publish and even with what changes they make to their projects' artifacts. It is a browser-based environment where all activities such as editing, testing, and documenting happen within the browser with the collaboration of other developers.

3.5 Summary

The work in this dissertation allows developers to find other developers and project partners, and ask for help. They can check the other developers' commits, their feeds, their activity in the project, their availability schedule, and get help in solving a particular problem or issue.

The supported system provides the user with features that are essential to the programmers' community, with a great focus on interactive presentation for the online presence and awareness of the project artifacts, possible project partners, and current collaboration sessions (chatting, editing, and debugging). In general, the supported features provides the developer with a better awareness and appropriate online presence by gathering, processing and analyzing data from multiple sources such as the collaborative virtual environment (CVE), the collaborative IDE (ICI), the versioning control systems (eg. SVN and CVS), and any other data that is either entered by the developer or by other developers' feedback and evaluation.

As mentioned earlier Jazz provides access to social network feature from inside their environment by bridging to SharePoint. Jazz users have direct access to forums, wikis, blogs, file sharing, and bookmark sharing, from within the Rational Team Concert (RTC) web interface. Hovering over a Jazz user in Rational Team Concert, a business card shows up with information imported from both the enterprise social network features and from Jazz. From the card, developers can switch to the RTC web interface to view their partners' profiles and view their

social network activities. In contrast, developers can view other members' profiles from inside the SCI environment without switching to another application. Also, within SCI users get most of the awareness information, friends list, groups list, projects list, and other supported features "for free" while concentrating on their project tasks.

On most social networks, user profiles are structured in a typical way; those profiles focus on personal information. In this work, the presented tool supports user profiles that focus on professional information, and present topics related to development and project artifacts.

Part II

Primary Contributions (Synchronous and Asynchronous Features)

Chapter 4
Synchronous Collaborative IDE System Support

> *While traditional integrated development environments focus on improving the efficiencies of individual developers, collaborative development environments focus on improving the efficiencies of the entire development team .- (Grady Booch 2003) [5]*

A collaborative integrated development environment enables developers to share programming-related tasks. This chapter presents the design and implementation of a synchronous collaborative IDE named ICI (Idaho Collaborative IDE) [79] that is integrated inside a collaborative virtual environment named CVE. ICI enables developers in different locations to collaborate on a variety of software development activities in real-time. In Section 4.1, a brief introduction about the CVE virtual environment is presented. An overview of ICI is presented in Section 4.2. The architectural design of ICI is detailed in Section 4.3. Finally, Section 4.4 concludes the chapter with a detailed discussion about ICI's implementation. Portions of this Chapter are adapted from [79].

The research contribution of this part is to augment ICI by the distributed development required collaboration tools as part of the development environment (eg. real-time text editor, and real-time debugger), and to help reduce cognitive context switches between tools inside the development environment and between its tasks and virtual environment activities, allowing developers to share, in real-time, the process of editing, compiling, running, and debugging of their software projects.

4.1 Overview of CVE

3D virtual environments (VEs) have been used for a variety of contexts including teaching in classrooms, distance learning, business, and e-commerce. As seen in today's computer games and virtual world simulations, VEs provide the user with the amazing experience of moving around and interacting with a simulated world [37, 80]. As mentioned in [81], virtual environments can be defined as *"computer-generated, three-dimensional representation of a setting in which the user of the technology perceives themselves to be and within which interaction takes place."* As the technological barriers to creating VEs have decreased, researchers have created many collaborative virtual environments (CVEs) to serve various domains. A CVE [82] can be defined

as an environment that "*actively supports human-human communication in addition to human-machine communication and which uses a Virtual Environment (including textually based environments such as MUDs/MOOs) as the user interface."* The CVE used for this dissertation is (rather uncreatively) named *CVE* (http://cve.sourceforge.net/), an educational platform that was built primarily to support two uses: (1) distance learning within by collegiate and computer science students; and (2) software development and group collaboration.

CVE is a multi-platform collaborative virtual environment where users can interact with each other within a 3D virtual world. Figure 4.1 shows an example scene that users (who are primarily software developers) might see in this environment. The collaborative virtual environment provides developers with a general view of other users and what they are doing. It allows developers to chat via text or VoIP with other team members and with developers from other teams in real time.

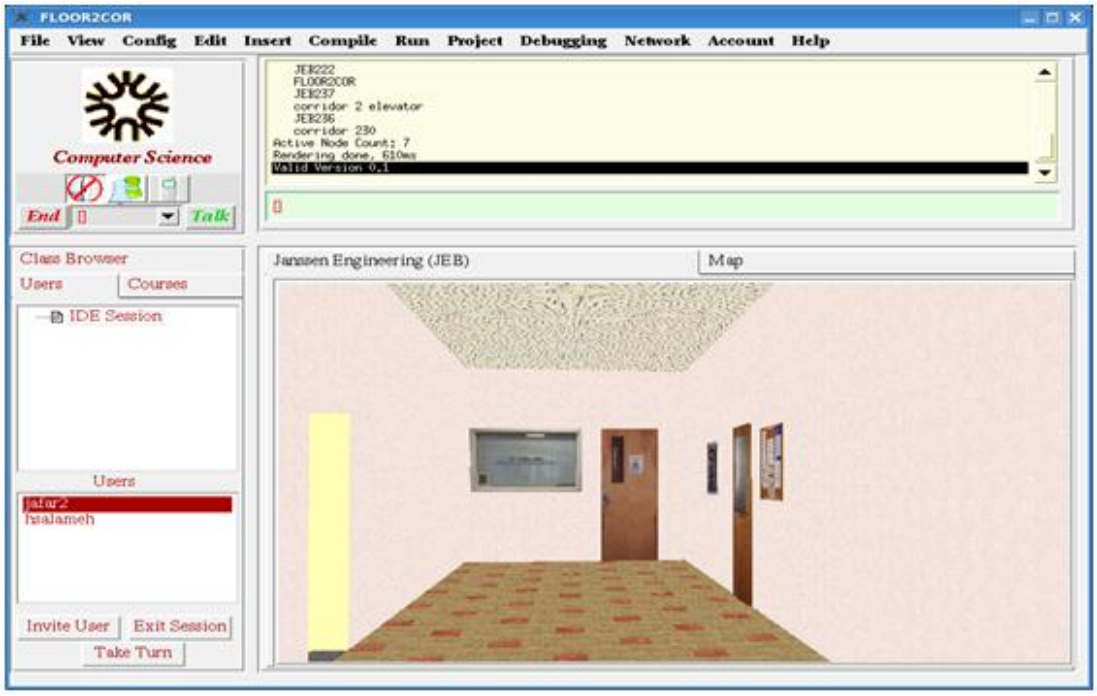

Figure 4.1 3D Environment Scene

4.2 Overview of ICI

Contexts such as computer science distance education need CDEs (see Section 2.2.2) that are not limited to a specific software development task, and 1) support real-time collaborative compiling, linking, running, and debugging sessions, and 2) provide an environment where developers can communicate easily; all from within the same tool [3, 83]. ICI supports software development in

C, C++, Java, and Unicon. ICI combines a synchronous collaborative program editor and a real-time collaborative debugger within a 3D multi-user virtual environment. It allows developers to share, in real-time, the process of editing, compiling, running, and debugging of their software projects. Figure 4.2 shows ICI inside CVE. Tabs allow easy switching between the virtual world and IDE tasks.

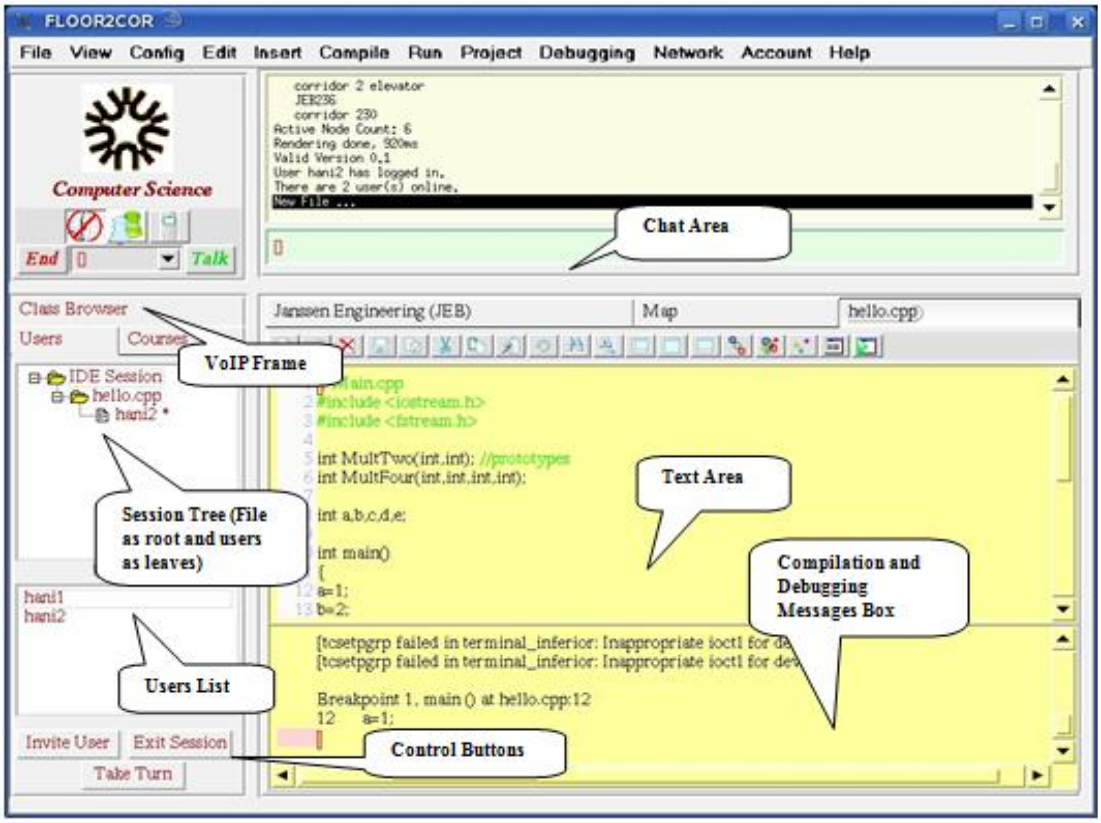

Figure 4.2 An ICI Session

Users may invite one another into collaborative IDE sessions, where they work together on tasks such as coding or debugging (see Figure 4.3). The intent of integration the IDE within a virtual environment is to make searching for collaborators, seeing who is available, or queuing for the attention of an expert less difficult and less intrusive, especially for the busy developers who serve as architects, chief surgeons, or instructor/mentors. These users are often on the receiving end of a large proportion of collaboration or assistance requests.

While logged in to the collaborative virtual environment, developers can use ICI for their normal IDE tasks, in between collaborative sessions. The fact that it is online, videogame-like, and chat-enabled is a potential distraction, but enriching the sense of online presence while

programming is what allows ICI to make it easier to support multiple collaborations or to switch between tasks.

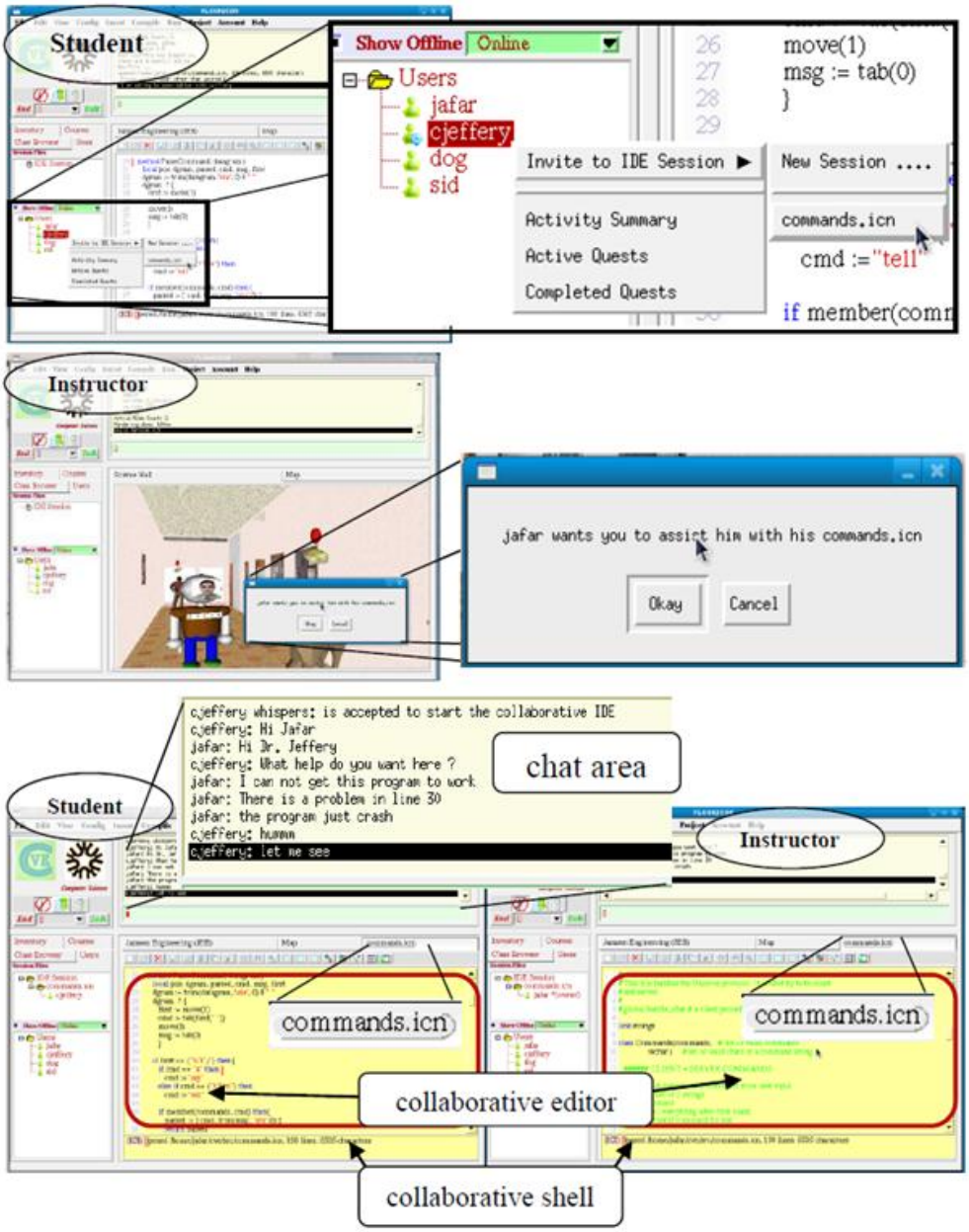

Figure 4.3 Collaborative IDE Session

ICI supports general collaborative software development tasks. However, it was built to serve the specific needs of computer science and software engineering education, particularly distance education. Typical requirements scenarios were computer science teaching environments where an instructor and/or a small team of students are interacting during a software development task. For example, one scenario is a virtual office-hour visit, in which a teacher assists students on their

assignments without having to be in the same physical location. Similarly, ICI may be used in a virtual lab to help in teaching a group of students on remote computers. A third example is a distributed team environment, where team members use a collaborative session to work on a shared task, each from his/her own location [80]. Section 4.2.1 shows some possible use scenarios.

The tool's suitability for software engineering distance education also makes it highly suitable for distributed software engineering teams. In addition to its focus on interactive collaboration, ICI supports asynchronous features such as: online presence information, awareness information, and social network features, all inside a single framework called Social Collaborative IDE (SCI). Detailed information about SCI and the supported features are discussed in Chapter 5 (Asynchronous Social IDE Features).

4.2.1 Use Scenarios

ICI can facilitate the education of novice programmers. It can help students taking introductory computer programming to collaborate and solve their programming problems, and to improve the student-instructor and student-tutor interaction. ICI can be used in the following scenarios for software engineering educational purposes:

- First, in the classroom the instructor can start a collaborative session and teach programming and the students can watch his editor, and ask questions.
- Second, personal use where a student can use SCI as a stand-alone IDE and benefit from its features; students can create, open, or edit projects.
- Third, group and peer use, where students and their peers or teaching assistants can collaborate on their assignment; also project groups can use it to solve their long-duration group projects.

4.3 Design

ICI's architecture is composed of four major components: 1) a collaborative editor, where developers can share source code editing and navigation, 2) a collaborative shell, where developers can share compilation, program runs, and real-time debugging sessions, 3) a set of communication tools such as text chat, provided by the surrounding virtual environment context, and 4) an interface for collaboration control, which allows users to invite other developers, take turns at the IDE controls, and enter and leave collaborative sessions.

Users meet in a collaboration session that is an interactive work session. The session owner, the person who started the session, is responsible for inviting the other participants. ICI uses a client-server architecture. The clients communicate with a collaboration server that implements shared collaborative services. The collaboration server forwards messages to the appropriate clients based on the message type. Appendix B shows a list of the system's network protocol messages.

4.3.1 Collaborative Sessions

An ICI collaborative session allows *n* developers to see and cooperatively control the same view of the code and the execution or debugging session (see Figure 4.4).

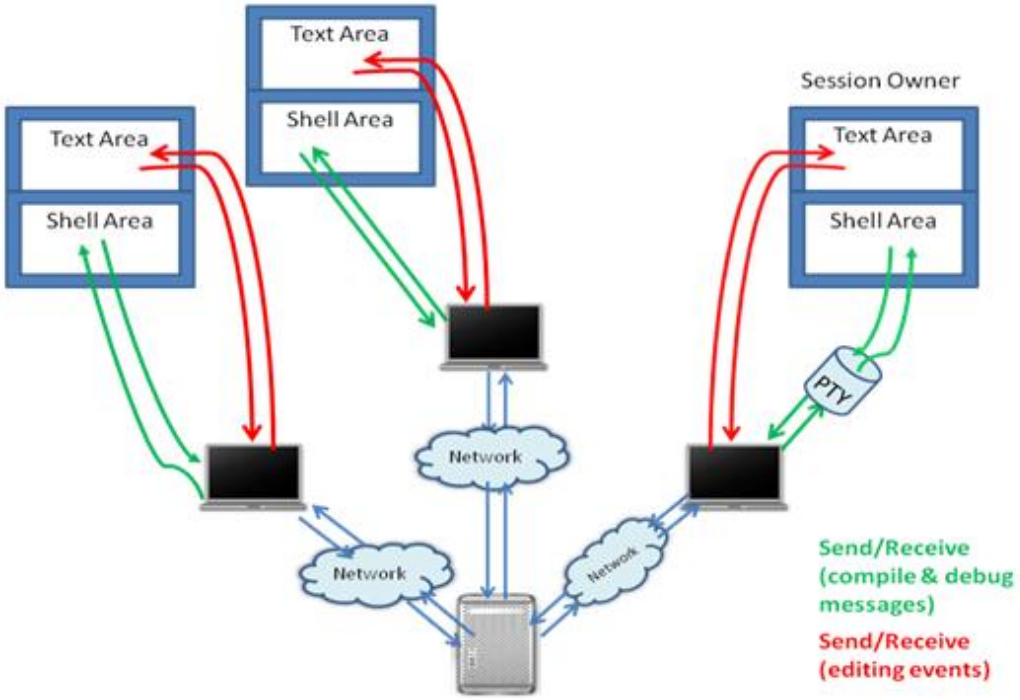

Figure 4.4 Architectural View of a Collaborative IDE/Debugging Session

ICI's collaborative IDE sessions are conducted within the context of the virtual environment and developers can enter and leave a session as needed. The collaborative session is owned by its creator. The system supports two kinds of sessions: 1) real-time editing, and 2) real-time debugging.

A real-time collaborative session begins when a participant opens a file and invites a others for a collaborative editing session. Section 4.3.2 presents a brief overview of the collaborative editor and collaborative editing sessions.

A real-time collaborative debugging session begins when one of the participants in the session starts a debugger such as gdb, jdb, or udb [85]. During a collaborative debugging session, developers can take turns controlling the debugger, and act as if they were the owners during the collaborative session, while other developers watch and discuss the debugging commands and messages. Section 4.3.3 describes the collaborative shell.

4.3.2 Real-Time Collaborative Editor

ICI provides a programmer's editor, in which a user can edit files privately, and then invite others into a shared session on the fly when consultation is needed. Unlike an ordinary text editor widget, the collaborative editor widget must send and receive network messages for all editing actions to the appropriate session on the collaboration server. The design of the collaborative editor is kept as simple as possible. ICI's editor has two modes: watch and edit. During a collaborative session, only one of the participants may edit; the rest of the participants are in watch mode. In watch mode users can ask questions, provide suggestions, or ask for permission to take a turn at the controls. A participant in watch mode sees and automatically follows any code modification or navigation made by the edit mode user in the collaborative session. Appendix C shows the system's real-time collaborative editing session scenario.

ICI's collaborative editor sends network messages with physical pixel coordinates instead of logical (row, column) coordinates within the buffer being edited. It causes the editor to get out of sync when running on different platforms such as Mac, and Windows because these platforms do not have same fonts or font engines. Options to solve this issue include providing a portable font engine like freetype or finding fonts that are logically interchangeable. Practical forces latter choice, fixed-width fonts, and a dynamic runtime check for a small set of likely-compatible fonts. Fixing this problem to use logical coordinates requires an extensive code re-engineering.

Our simple fix for the problem used a font information reader, and comparison tool written in Unicon. It is part of the session creation/initiation code in ICI. To make use of this fix, ICI changed the session invitation process to include the following steps:

1) Hardwire the driver's client window font size for collaborative editing
2) Pass the font size information through the network invite, and accept commands.
3) Force the watcher to use the driver's font.
4) The users are allowed after that to change the font while on collaboration.

Figure 4.5 shows the used algorithm for font portability implementation.

```
#
# Change the Session driver's client font
# to a compatible font
#
fh := WAttrib(ca, "fheight") #Return the font height
fw := WAttrib(ca, "fwidth")  #Return the font width
#
# Checks the nearest compatible font size
# (small, medium, large, or huge) by
# comparing the font height (fh) with a
# hardwired font heights.
#
mycompfont := nearestfont(fh)

# Checks the compatible font and
# forces the driver's client to use it
fnt  := compfontdlg.mpfont(mycompfont)
CurrentEditBox().set_attribs("font=" || fnt)

# Sends an invitation to the watcher to
# join the collaborative editing session.
session.Write("\\CETLOpen "||mycompfont||…….)

# Once accepted the invitation the watcher's
# client window will be forced to use the
# compatible font
```

Figure 4.5 Font's Portability Implementation Algorithm

4.3.3 Real-Time Collaborative Shell

In order to collaboratively compile, run, and debug a shared program, ICI implements a collaborative shell. The collaborative shell allows developers to see the compilation messages of the target program and to share the inputs and outputs of the running program. A real-time collaborative debugging session can be started by any developer as a private IDE activity and subsequently shared on demand. The collaborative shell uses a simple multiplatform virtual executor facility to interact with the execution or debugger session, and a network protocol to share shell I/O with the rest of the participating developers in the collaborative debugging session.

The virtual executor funnels bytes from one process to another. On Windows it is implemented as a bi-directional pair of pipes; on UNIX platforms it is a pseudo-tty which behaves like a pair of

pipes with the additional property that to the child process, one end of it looks like a conventional *TTY* terminal [86]. A *PTY* (pseudo-terminal) denotes a pair of virtual character devices that provide a bidirectional communication channel to an external application such as a compiler or debugger.

When a user starts a debugging session, a call to open a virtual executor runs the debugger appropriate for the shared program. Once the debugging session starts, a prompt appears at the collaborative shell allowing the owner of the session to start entering commands to the debugger, while the other clients can observe the text commands and the debugging messages simultaneously. The collaborative shell uses the same protocol used by the collaborative editor, but with a different set of network protocol messages (see Appendix B).

4.3.4 Communication, Control, and Activity Awareness

The ICI design perspective is that what happens in between collaboration sessions is just as important as exactly how the shared session works. Much of what happens between sessions is ordinary, non-collaborative IDE work, and the primary emphasis in design was to minimize the transition effort between individual work, collaborative session, and back to individual work, so that this can easily occur dozens of times during the course of a work period. In such cases, users might get a lot of invitations and requests. The management of pending notifications is discussed in Chapter 6.

Group awareness is an important factor for a successful collaboration, providing an understanding of other developers' activities. ICI provides support for group awareness. In addition to the current collaboration sessions which are temporary in nature, there are persistent groups, modeled after those found in Massively Multiuser Online (MMO) games. Persistent groups (often called "guilds" in games) provide both chat and wiki-style collaboration aids that remain across work sessions. Unlike most MMO "guilds", in ICI one may be a member of as many groups as needed.

4.4 Implementation

As mentioned earlier, ICI is implemented as a part of the open source CVE virtual environment (cve.sourceforge.net). CVE is written in Unicon, a very high level object-oriented programming language [86, 87]. Unicon provides a simple interface to the standard internet protocols, TCP and UDP, as well as several higher level communications and messaging protocols [86].

Each set of ICI clients that are working together is associated with a session object on the collaboration server which allows clients to broadcast messages to all members of the group. ICI uses two kinds of event messages to ease the collaboration during the collaborative sessions: *SHL* and *CETL*. Figure 4.6 shows how events are captured and rendered within the ICI environment. The shared editor is implemented using an approach similar to the one used by GHT [37]. In ICI, insertions and deletions are executed locally on the client before they are sent to the server. The other clients then apply the modifications to the text.

The concept of a collaborative IDE session appears in both the server and the client. The server manages the sessions using a table that contains all the needed information about each collaborative IDE session: the owner, current edit mode user, file, and list of users in the session. On the client, there is another session table that contains all the information for this client about its sessions (session id, owner of the session, reference to the user interface component for the session, file, and list of users in the session).

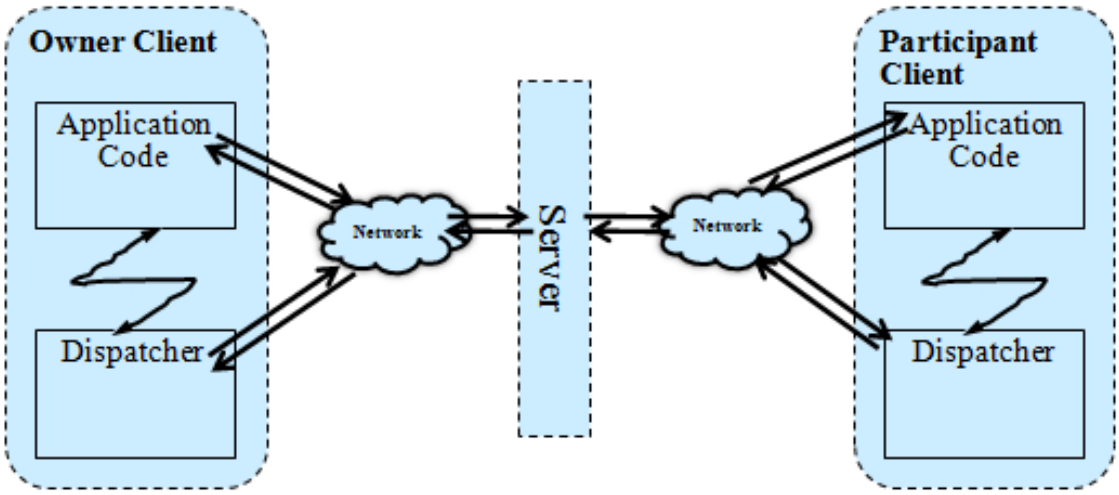

Figure 4.6 Events' Capturing and Rendering

4.4.1 Network Protocol

The ICI network protocol messages are strings consisting of a message name followed by arguments which are often a data payload or a list of users. Messages are divided into different categories (for detailed information see Appendix B). Appendix D presents a BNF grammar that describes the structure for the network messages.

4.4.2 Source Code

ICI's source code is organized into four different groups of classes. Figure 4.7 shows the UML diagram for the classes related to ICI.

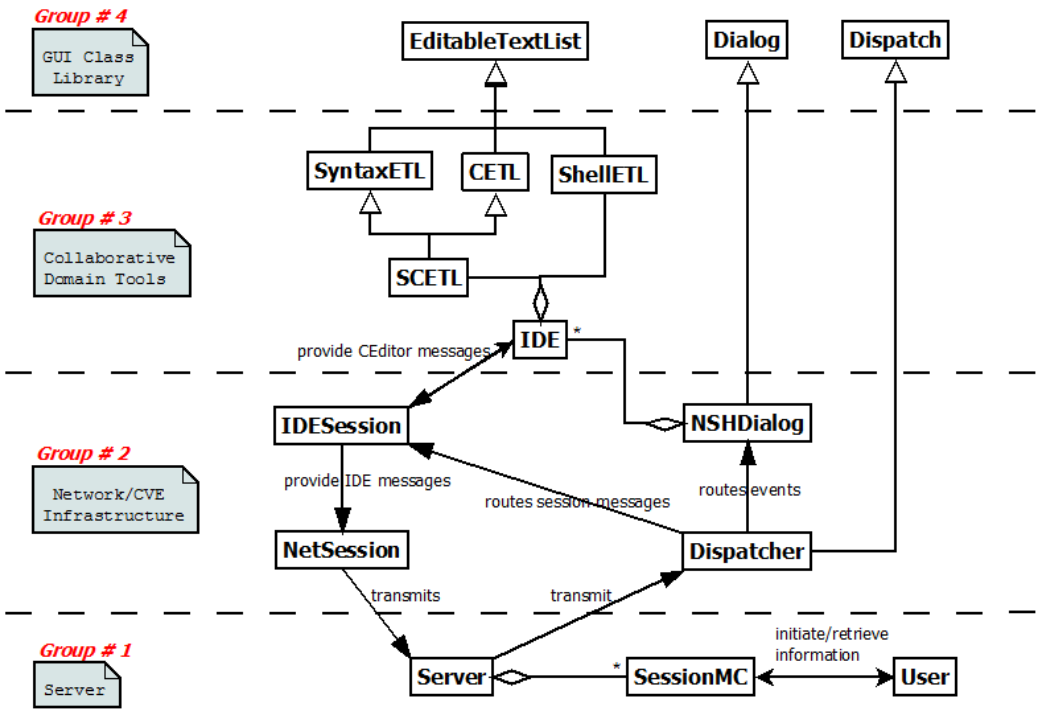

Figure 4.7 Collaborative IDE Class Diagram

The first group is ***Server Classes***. This group includes three major classes: 1) ***Server*** as the main collaborative virtual environment server class which has methods for managing the virtual environment. This class is the manager for the collaborative IDE sessions. It creates a session entry there when a user invites another user. Also it adds another user into the users list when additional users are invited into the session. Once a user exits the session or logs out from the virtual environment, they are removed from the users list; 2) ***SessionMC*** a class that acts as a protocol manager for the collaboration sessions; and 3) ***User*** class that describes the behavior and properties of the user entity in the CVE.

The second group of classes is ***Network/CVE Infrastructure Classes***. This group includes the CVE virtual environment classes that provide the context in which SCI executes. The SCI design did not have to establish communications capabilities or create its own window; instead it interfaced with an existing infrastructure. This group includes classes such as 1) ***IDESession*** a class responsible for managing the collaborative IDE session (create new sessions; receive events from collaborative IDE, etc.); 2) ***NSHDialog*** a class which has methods

related to the GUI (buttons, trees, etc.) that are used by the virtual environment and by the collaborative IDE; 3) ***N3Dispatcher*** as the client's message reader. The collaborative IDE uses this class to synchronize different events between the clients and the server. This is done by sending different types of messages between them (invite user to the session, remove user from a session and fire event in the editor which is currently in a collaborative IDE session); and finally 4) ***NetSession.***

The third group of classes is ***Collaborative Domain Tools***. This group includes the main IDE classes such as 1) ***SyntaxETL*** a subclass of the Unicon standard library EditableTextList class provides a multi-language syntax-coloring collaborative editor widget; 2) ***CETL*** the main class of the IDE. It provides a scrollable editable text area. This class issues "CETL events" to send the changes through the network to the collaborating clients; 3) ***ShellETL***: this class executes a simple command shell within a collaborative editable textlist widget, in order to fulfill the requirements of the compiling and debugging procedures; and 4) ***IDE***: this class is the home for almost all the collaborative IDE functions and methods.

The fourth group is the ***GUI Class Library***. This group includes Unicon standard library classes such as ***EditableTextList*** and ***Dialog***, and other files such as ***Dispatch***.

Figure 4.8 depicts event transmission during a typical collaborative session in which a GUI operation is sent to the server and forwarded to other participating clients.

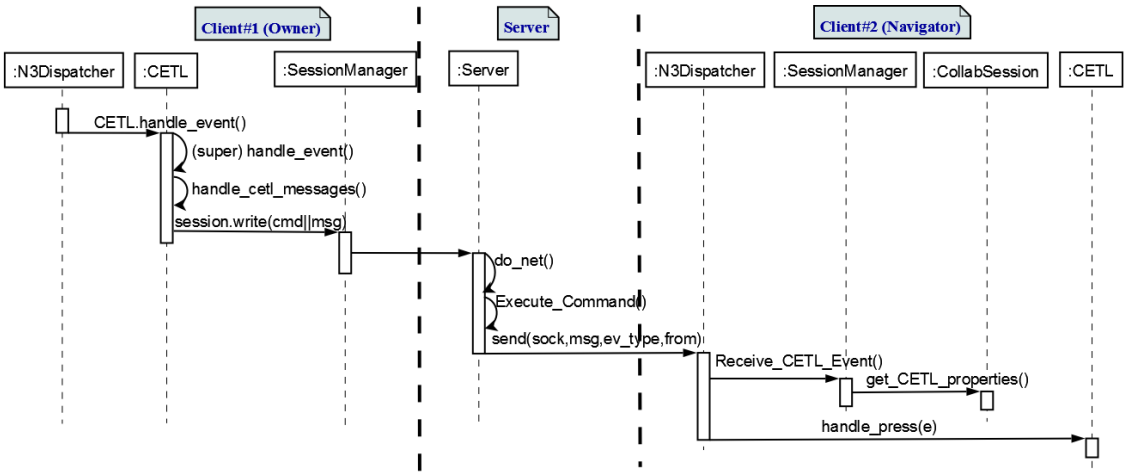

Figure 4.8 GUI Events are Transmitted to the Server and Forwarded to other Clients

4.5 Summary

This chapter proposes a collaborative programming environment called ICI as a part of the CVE virtual environment that enables developers to collaborate synchronously on a variety of development activities. As the foundations of ICI; chapter 4 presented structuring guidelines and architectures for the design of the system's primary components, the collaborative editor, and collaborative debugger.

Chapter 4 also presented the structure for the collaboration scenarios and sessions within the environment. In addition this chapter described the communication, awareness, and presence requirements needed to support the development community collaboration and interactions.

Chapter 5
Asynchronous Social IDE Features

This chapter presents the design and implementation of the asynchronous features of the social development environment named SCI (Social Collaborative IDE) [88, 89] that extends the ICI collaborative IDE and lives within a collaborative virtual environment named CVE.

Section 5.1 presents the hypotheses of this dissertation. The motivation for the proposed system is presented in Section 5.2. In Section 5.3, a brief introduction and definition of social presence are given. The importance of social teams is given in Section 5.4. In Section 5.5 a brief introduction and definition of activity and group awareness are given; also the supported activity and group awareness inside SCI is described. The architectural design of SCI is detailed in Section 5.6. Finally, Section 5.7 gives a detailed discussion about SCI implementation.

5.1 Hypotheses

The research project presented was undertaken to test two hypotheses. The first hypothesis is that integrating social networking features inside a CDE makes the CDE an effective/usable classroom programming environment. Preliminary results to test this hypothesis are gained from the case study presented in chapter 7.

The second hypothesis is that the combination of social network with a virtual environment will provide the collaborative IDE with the social activities, online presence, and awareness information. It increases use, and usefulness of collaboration tools. This dissertation believes that virtual environment represents a collaborative social environment, and adds social bonds between developers; CVE provides a “social presence”- the sense in which developers feel that there are others present in the collaborative development environment. This dissertation includes case studies to test the correctness of those hypotheses. To sum up, the hypotheses reflect the belief that IDEs and social networks complement each other in collaborative software development, because software development communities are social networks.

5.2 Motivation

The synchronous collaborative integrated program development environment called ICI, described in Chapter 4 is well-suited for scheduled interactions between people who already know each other and work together. In order to support communication and collaboration

between less cohesive development groups whose community members are geographically distributed, substantial additional capabilities are needed.

Consider a niche programming community, such as the one for which SCI was developed. Niche developer communities are common. They often consist of a few dozens, hundreds or thousands of developers whose work uses particular software tools such as domain-specific languages, library API's, or open source software projects. Supporting such a community requires more than the ability to e-mail, leave each other messages, or post a comment to the project mailing list. Developers need the ability to find others with common interests or needed expertise, and ask for their assistance easily. The intent of the SCI project is to support asynchronous collaboration, increasing the developers' awareness of each other, and of artifacts, resources, and activities, and encouraging team communication.

5.3 Social Presence

Social presence is an important factor for an online community where high levels of interactions reflect the group's cohesion. Presence is traditionally defined as the sense of "being in", "existing in", or belonging to a group [90, 91]. Several definitions are used for social presence. This section introduces some of the available definitions.

Garrison and Anderson [92] defined presence as *"the ability of participants in a community of inquiry to project themselves socially and emotionally as 'real' people, through the medium of communication being used"*. Social presence is the critical factor in a communication and the ability to work collaboratively is at the heart of social presence theory [91, 92, 93].

Previous definitions focused on physical presence. However, researchers who study social presence have used many definitions. *Social presence* has been defined as the degree to which a person experiences the feeling of being present, the *"sense of being with the other"*, and takes part in the interaction in any community, or the degree to which a person is perceived as real in an online conversation [94, 95]. Annetta and Holmes [91] argue that social presence is strongly attached to individuality: "*If a student in an online community feels they are perceived as an individual then they feel a sense of presence within that community."*

In the context of this work, social presence within social teams and community members is referred to as being aware of the other team members' roles and activities, as well as the resources in the community relevant to a given project. In order to catch people when they are "in" and focused on a given task, team members need to be aware of the other team members' primary activity windows and/or activity history.

5.4 Social Teams

Software developers routinely work in teams. While some projects involve only their immediate team members, many projects involve a broader community of individuals from different institutions [96]. Many developers contribute to several projects in any given week. One of the pervasive challenges facing any software development team is getting the right level of communication to coordinate their work and perform their tasks effectively, and this problem is more difficult for distributed teams.

Large scale software development is a social endeavor. For team members to function effectively, they must maintain a certain level of social awareness. Developers must be aware of their other team members' roles and activities, as well as the resources in the community relevant to a given project. Also, in order to catch people when they are "in" and focused on a given task, developers need to be aware of the other team members' primary activity windows and/or activity history [97].

5.5 User and Group Awareness

Awareness is an important factor for a cohesive software development community where the gathered awareness information supports and reflects the groups' cohesion. *Awareness* has a broad range of meanings. The next section gives a list of common definitions in the literature.

5.5.1 Definitions

A common definition, from Answers.com (http://www.answers.com/), is *"Awareness refers to the ability to perceive, to feel, or to be conscious of events, objects or patterns, which does not necessarily imply understanding."* A more specific definition in the context of collaborative work relates awareness to the working environment: *"awareness is an understanding of the activities of others, which provides a context for your own activity"* [98]. This definition implies a group of people working together. This kind of awareness is often referred to as *group awareness* in the CSCW research community.

According to that definition, a lot of information can be considered awareness information. Gutwin and Greenberg [99] proposed the following list of elements as group awareness relevant information elements. The list in Table 5.1 shows which of those elements is addressed in SCI.

		Addressed in SCI
Presence	Who is participating in the activity? Is anyone in the workspace?	X
Location	Where are they working?	X
Activity Level	How active are they in the workplace?	X
Actions	What are they doing? What are their current activities and tasks?	X
Intentions	What will they do next? Where will they be?	
Changes	What changes are they making, and where?	X
Objects (Artifacts)	What objects are they using?	X
Extents	What can they see? *(view)* How far can they reach? *(reach)*	
Abilities	What can they do?	X
Sphere of Influence	Where can they make changes?	X
Expectations	What do they need me to do next?	

Table 5-1 Supported Group Awareness-Relevant Information Elements

The above list of elements gives a basic idea of what information SCI captures and distributes in the distributed groupware environment. Awareness of *presence* is simply the knowledge that there are others in the workspace and who they are. Awareness of *actions* and *intentions* is the understanding of what another person is doing, either in detail or at a general level. Awareness of *objects (artifacts)* means the knowledge of what object a person is working on. *Location* relates to where the person is working. Awareness of *extents* includes *reach* and *view* awareness; awareness of reach involves understanding the area of the workspace where a person can change things, since sometimes a person's reach can exceed their view. From the above it is clear that the elements addressed in SCI relate to the past and the present because past and present information can be determined from raw perceptual information. SCI does not include any elements relating to the future. Awareness of expectations is not addressed in SCI because information about the future requires prediction.

Gutwin and others [100] stated that *"group awareness is the understanding of who is working with you, what they are doing, and how your own actions interact with theirs"*. Also they stated that the complexity and interdependency of software systems suggests that group awareness is necessary for collaborative software development.

Awareness in the context of this dissertation refers to a specific kind of awareness called *social software awareness* [90] and is defined as the "*Combination of passive and active information about developers' activities and artifacts, proportional to their interconnections.*" Awareness is proportional to the users' interconnections in order to avoid information overload. Several levels of friends are needed in order to give closer friends more access to the awareness information than friends-of-friends, which should have more access than the random strangers or other community members.

To summarize, *presence* and *awareness* complement each other. Where presence is the extent to which information about users, their locations, their activities etc. is available to others, awareness describes the means and extent that others are informed of this available presence information.

5.5.2 Awareness Types

Research literature also demonstrates that there exist different types of awareness. Acknowledging that there are many forms of awareness that exist, Greenberg et al. [99] identify five types of awareness that people in collaborating groups maintain, these types as follows:

- ***Personal awareness:*** information that users maintain about themselves and their roles and activities in the group. This information is either synchronous (eg. current locations and progress within the system) or asynchronous (e.g. where the user has been within the system) [101].
- ***Informal Awareness:*** refers to a general sense of who is around, what they are doing, and what they are going to do. This is the kind of knowledge people have when they work together in a face-to-face environment. Informal awareness provides information on the presence, location, and absence of collaborators [99, 102].
- ***Group-structural awareness:*** This type of awareness provides knowledge about things such as people's and group member's roles, their positions on an issue, their status and role in group processes [102]. Group-structural awareness is essential to support knowledge of other collaborators' expertise based on the roles they assume, this knowledge can prove important in finding appropriate project partners and choosing who to initiate an interaction with on project related activities [103].
- ***Social awareness:*** Information that a developer maintains about others in a social context, including information such as: whether a developer is paying attention, their emotional state, and their level of interest. This awareness information helps minimize interruptions and conflicts when engaging in collaborative processes as described by Schmidt [104].

— ***Workspace awareness:*** This information is about other collaborators' interactions with a shared project workspace and the artifacts it contains [101]. This means awareness of where others are working, what they are doing, and what they are going to do next, and where they're going to do it. Gutwin et al. mentioned that this awareness information is useful for many collaboration activities, such as: "coordinating action, managing coupling, talking about the task, anticipating other's actions and finding opportunities to assist one another" [105].

Liccardi et al. [101] represent these different kinds of awareness with the diagram shown in Figure 5.1, extending the work of Greenberg.

Figure 5.1 Awareness Types

The above awareness types are addressed and supported by SCI.

5.5.3 Importance

For many years software development has presented serious coordination, communication, and collaboration problems, especially when teams are geographically distributed [100, 106, 107]. This leads to cases were developers affect other team members' code. According to researchers in software engineering and CSCW, this is due to the lack of awareness about what is happening in other parts of the project, and the other team members' activities.

Researchers have found a number of problems that still occur in team projects and software development. They found that it is difficult to: determine when developers are making changes to the same piece of the project; communicate with others due to time zones barriers and different work schedules; find developers for closer collaboration or assistance; determine the expert/right developers who have the knowledge about the project different artifacts. As Herbsleb and Grinter [108] state, lack of awareness--"*the inability to share at the same environment and to see what is happening at the other site*"--is one of the major factors in these difficulties. When working with groups who are geographically distributed, awareness of other team members' activities provides

information that is important to build an effective collaboration. This awareness includes: observing who is working with you, and their activities or plans. Developers can use the knowledge of other team members' activities for many other purposes that help to assist the overall cohesion of the project. For example, knowing the specific files and objects that another developer has been working on gives an indication of their expertise within the project; tracking who made changes most recently to a particular file gives an indication of whom to ask before making further changes; and gathering information about who is currently active can help developers to find a possible closer collaboration and real assistance on particular issues [109]. A few systems do track and visualize awareness information (eg. Palantir [49], and TUKAN [110]).

According to Gutwin and others [100], in co-located situations, awareness can be maintained in three ways: first, when developers tell each other about their activities (explicit communication); second, by watching others work developers gather information about their activities and plans (consequential communication); lastly, developers find out about other team members' activities by observing the changes to project artifacts (feedthrough).

5.6 Design

Figure 5.2 shows the integration of the presence information, collaboration tools, and software development facilities in a single environment. The inner oval represents the CVE collaborative virtual environment where users can interact with each other within a 3D virtual world. CVE provides developers with a general view of other users and what they are doing.

In the middle oval, ICI developers use synchronous collaborative software development tools that extend CVE's generic virtual environmental capabilities to communicate, interact, and collaborate in solving their programming problems. The outer oval provides the developers with online presence, activity awareness, and social network features. SCI's asynchronous features help users to select and coordinate their active synchronous collaborations. In general, the asynchronous tools drive the use of the synchronous tools, and the two categories complement each other. CVE, ICI, and SCI are complementary tools that work together to provide a unique development environment.

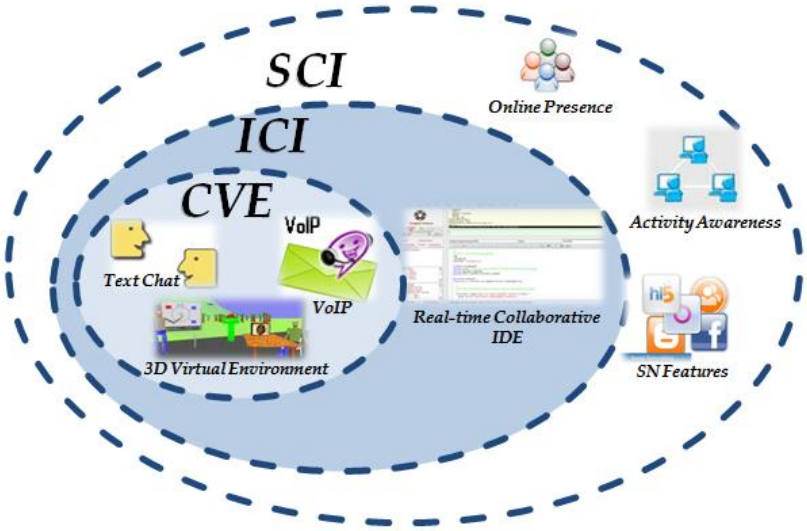

Figure 5.2 The SCI Architecture

5.6.1 SCI's User Interface Components

SCI is a desktop application written as an extension of ICI. ICI and SCI run on a variety operating systems including: Microsoft Windows, Linux, and Mac OS. Figure 5.3 shows the structure of the current SCI environment interface components and the activities related to each component.

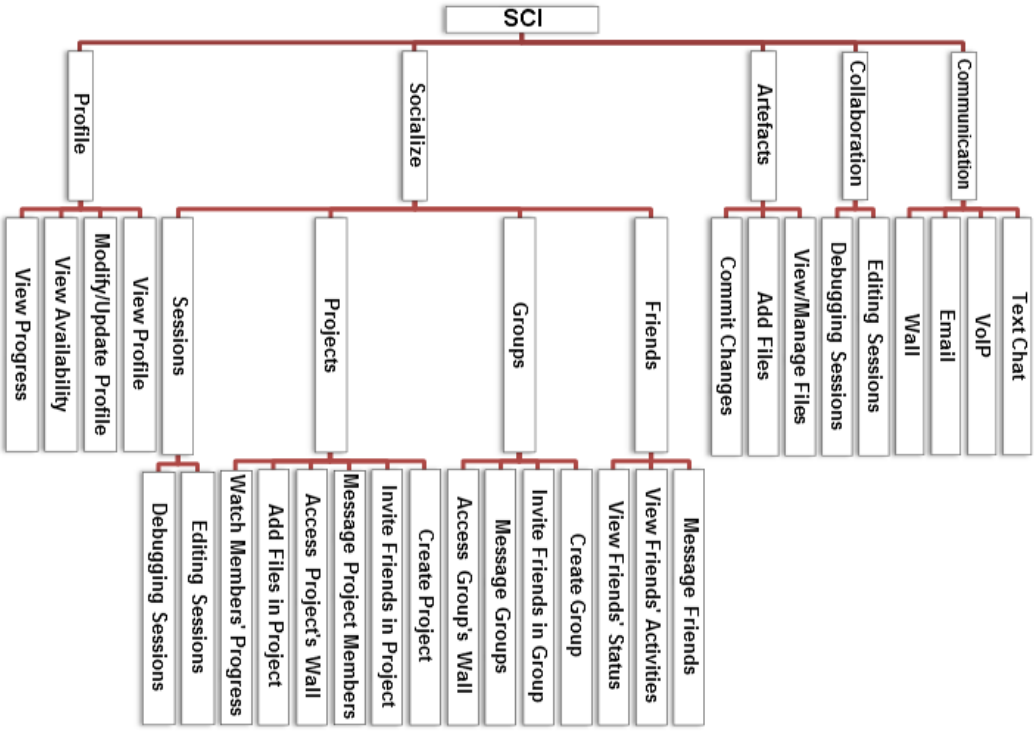

Figure 5.3 Structure of the SCI Components.

The major components of SCI are shown in Figure 5.4, such as:

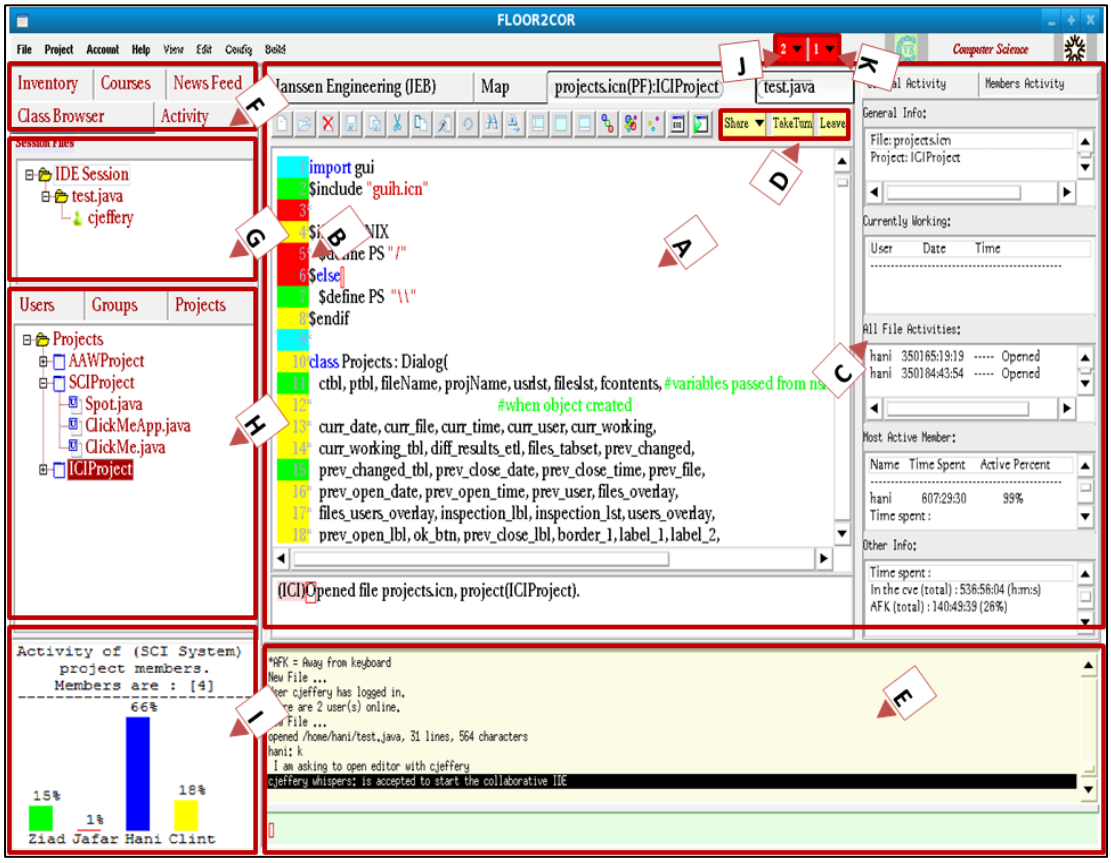

Figure 5.4 A View of the SCI Integrated Development Environment

— Collaboration Spaces (A).

— The ChangeBar (B) uses color coding to depict the user who committed the most recent updates or changes to each line.

— Activities on the current collaboration space (C).

— Text Chatting (E).

— Tabs (F): News Feed Tab, and Activity Tab (that shows both of G and H).

— Session Tree (G).

— Mini Tab Set (H): includes Users', Groups', and Projects' Trees.

Tabs (F), to the left side of Figure 5.4, show social awareness of the users and friends, their status (online, offline, or idle), their location in the CVE virtual environment, active collaborative sessions and members of each session. Information in the tabs allows users to observe the presence of the available teams (groups) and who belongs to each team. Also, they show the

presence of each team member, their activity in the project, and their activity history (see Figure 5.5).

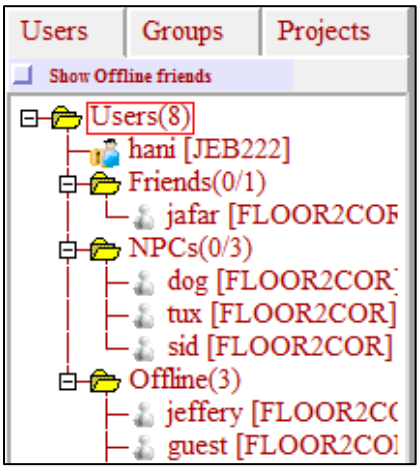

Figure 5.5 Users', Groups', and Projects' Awareness

The social parts (G, H, and I) represent that subset of the awareness information that users get "for free" while concentrating on their project tasks; it includes sessions' tree, users' tree, groups' tree, and projects' tree. Figure 5.4(I) is a bar chart that shows additional information on a project from the user projects list. This detail view cycles semi-randomly through the user's projects, allocating more time to projects with high activity. The chart shows the project members' activity and percentage of the time each spent working in the project. Icons (J) and (K) show passive awareness notifications of pending invitations and requests (Figure 5.4(J)); and emails (Figure 5.4(K)) that users receive from friends and other community members. Following is an explanation of these components:

Collaboration Spaces (Figure 5.4 (A))***:*** The IDE's major collaboration spaces are its text editor and shell areas; they are where developers and team member collaborate in editing their code, and debugging their projects. It uses a color coding to depict the user who committed the most recent updates or changes to each line (see ChangeBar Figure 5.4 (B)), and shows activities on the current collaboration space (Figure 5.4 (C)). Collaboration icons (Figure 5.4 (D)) allow users to share their own collaboration space and start collaboration sessions, take a turn editing the shared space, and leave the collaboration session. These spaces were developed for the ICI collaborative IDE.

Chatting: This is provided by the CVE virtual environment. It allows developers to chat via text (Figure 5.4 (E)) with team members and other developers in real-time.

Email: Developers can send private emails to one or more users, and public emails to the whole group. Public emails also appear as an entry inside the news feed, where developers can comment

on to the feed entry and share their opinion about the topic of the feed, or respond to the email privately.

Special Interest Groups (Figure 5.4 (H)): Developers can join special interest groups (SIGs) where they can find other team members or developers that share the same interest. The system began with four original SIGs: Java Group, C++ Group, Unicon Group, and Software Engineering Group.

Profiles and News Feed: Users are allowed to view others' information within the community circle (teams or groups). Also, they can view information about the community and the project friends circle using the available news feed, and mini-feed that are available in every user profile. Also, SCI provides a wall for each group and project in the system to allow their members to post the group/project news and progress. These wall are only accessible by the group's/project's members. Detailed information is in sections 5.6.1.1 and 5.6.1.2.

5.6.1.1 Profiles

Profiles (Figure 5.6) show the user's friends, groups they are part of, their projects showing the project name, owner, creation date, number of members, number of files, and tree of the project files, and a mini-feed (wall) that will show the users' activities, and recent events.

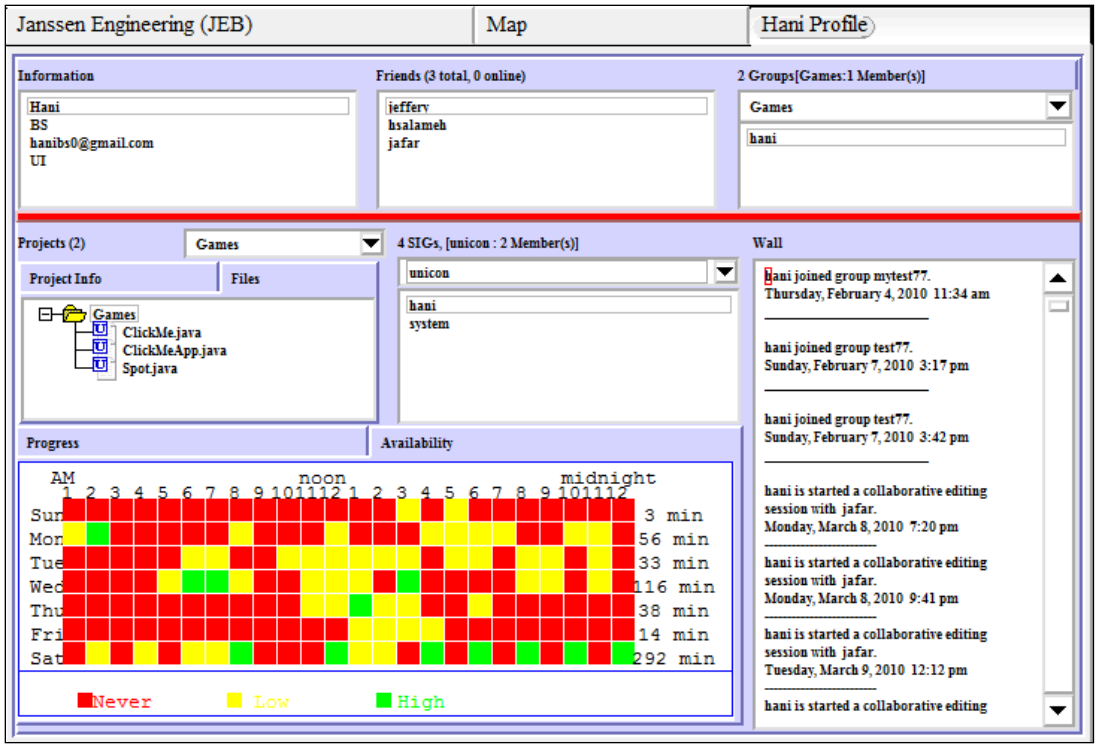

Figure 5.6 User's Profile

Profiles show information related to the user's availability patterns and what hours of each day are they likely to be available for assistance and/or collaboration. They also show the user's progress in all the projects they own or are members of. The profile can also play a central role by letting people at other sites know when the user might be free, and which projects they are familiar with.

Users can set the access to their profiles to either "private", where it can be accessed by the users' friends circle and their team members, or "public" where it can be accessed by all the other developers within the community, unless they are blocked by the owner of the profile. Opening a profile, users can view other members' personal information, friends, groups, projects, talks, progress, availability, and any other content they want to display. The value of displaying users' personal interests alongside the work information is because collaboration is between people, and knowing about the people you are working with, especially those who are distributed all around the globe, can be a key for building effective teams.

5.6.1.2 News Feed and Discussion Threads

The News Feed (Figure 5.4 (F)) highlights information that includes new projects, changes to projects, new groups, members who have joined groups, active sessions (debugging or editing sessions), and other updates. The News Feed also shows conversations taking place between the users and their friends (see Figure 5.7). Each group member can chat with all other members in that group, or view and add posts to the discussion thread in the SCI system.

Figure 5.7 Showing the News Feed from Inside the SCI Development Environment

There are two kinds of feeds: 1) system generated feeds; and 2) user input feeds. The developer can define the priority for the entered feed (low, medium, or high) and set access permissions, such as: private that will be available for his friend's circle to access and reply to, or public that can be accessed by everybody. The system prioritizes the posted feed in each user client proportional to his/her relation to the owner of the post (eg. friends and friends-of-friends (high), project and group partners (medium), and others (low)). Users have the ability to designate which specific user or list of users can access posted information, if they so choose, so that groups can maintain privacy.

The system provides developers with automatically generated postings about team members' activities. These activities include: 1) creating a collaborative editing session; 2) editing a piece of code, compiling a program and/or starting a collaborative session; and 3) other topics related to the project artifacts. Users can view the post contents by right clicking on it and choose the view post option from the generated pop-up menu (see Figure 5.8).

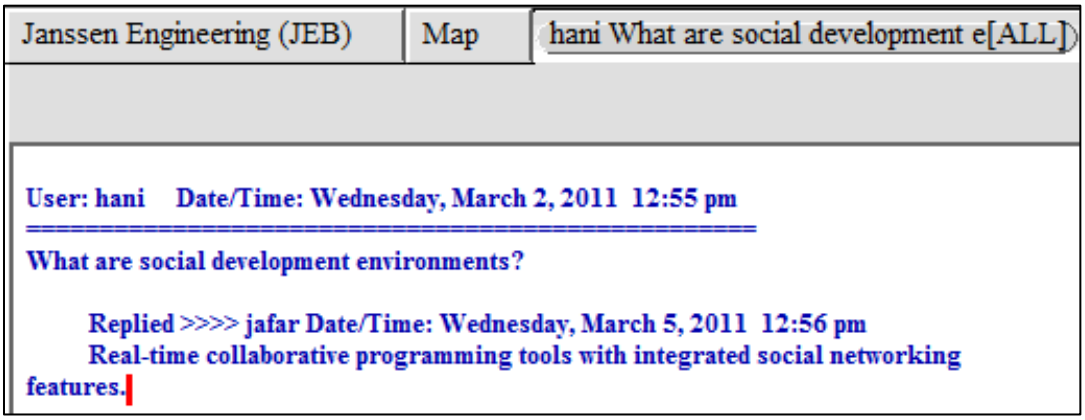

Figure 5.8 Showing the Contents of the Highlighted News Post

A special kind of news feed (*personal feed*) also is available in the user's profile page that shows the updates tailored to that user. The user can delete events from the *personal feed* after they appear so that they are no longer visible to profile visitors. Personal feeds makes it easy for developers to track changes in projects, and team members. Users are able to control what types of information are shared automatically with friends. Users may prevent friends from seeing updates about several types of private activities. The feed shows people who share interests, so they can ask for assistance and find experts easily. Users have the ability to designate which specific user or list of users can access posted information, if they so choose, so that groups can maintain privacy.

As mentioned before SCI provides a wall for each group and project in the system. Figure 5.9 shows the project's wall.

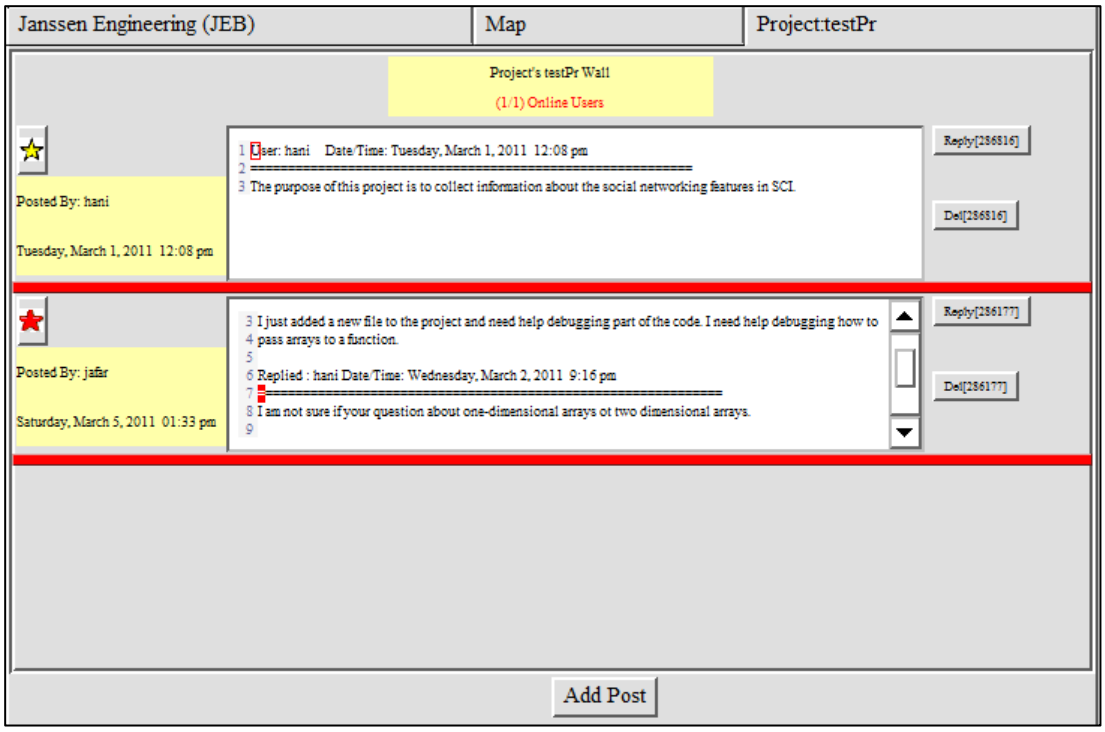

Figure 5.9 A Project's/Group's Wall from Inside the Development Environment

5.6.2 Virtual Environment and Project Presence Features

Developers can create different types of projects with significant permission differences. The owner can set the project to "Group Open", "Group Closed", "Community Open", or "Community Closed", and control access to its files. All the project files are stored in the server. "Group Open" projects allow the owner's friends in the social network to join the project group automatically. "Group Closed" projects require friend status in order to request permission to join the project. "Community Closed" projects require permission but not friend status, to join the project. "Community Open" projects allow any developer to join the project.

Every project is allocated a space inside the CVE virtual environment when the project is created. Project spaces vary depending on their size; a project starts with a simple room sized for its initial membership.

Any user who joins the project causes their avatar to be teleported to the project room, and each time the user opens the project within the CVE his/her avatar is teleported there. Users typically will be members of multiple projects. A teleport menu lists the rooms for each of their

projects and interest groups. The presence of the avatars in a virtual room gives the developers the feeling of the team's presence and encourages interaction between them regardless of the variety of locations and cultures. Users can create rooms for specific groups and purposes. Each room's owner controls its membership and access.

5.6.3 Awareness Requirements

Distributed developers need to maintain general awareness of the entire team, as well as more detailed awareness of people of special interest (people that a developer is working with, or wishes to ask assistance from). Developers in distributed software development projects maintain their awareness primarily through text-based communication tools, such as: mailing lists, email systems, and chat systems, along with voice tools, such as: Voice over Internet Protocol (VoIP).

SCI is implemented so that developers can maintain awareness of the people working on their project, in what parts of the code others are working, what their areas of expertise are, as well as who has joined (or left) the team. Users gain this information from different sources, such as: (1) watching the team news feed; and (2) observing the people who access, edit, or change the group projects' artifacts. This helps developers keep up-to-date with both changes to the projects' artifacts and the activities of other distributed team members. Developers maintain awareness of each other's activities, tasks they are working on, their activity history, and when they are likely to be available.

The system provides the user with a list of the experts in a specific subject. Users who need help reviewing and debugging their code, when trying to share their files, can choose to either share it with an available user or request help from an expert in this piece of code or programming language. Also users can check who is expert in a specific programming language by browsing the special interest groups' members for this language.

Also, mailing lists and news feeds help developers gather information about others at their convenience, for example, to find out whom the experts are in an area. In some cases, developers gather the information by simply initiating a discussion: because the messages go to the entire group, the 'right people' will identify themselves by joining the conversation, answer questions, and comment on the discussion thread. They also can gather information about who are the experts in a specific part of the code, by checking the changes history to this code and the time each team member spent working on this code.

5.6.4 Passive and Active Awareness Features

SCI supports two types of awareness: passive and active. Passive awareness describes information that the system provides automatically in order to make the user aware of the surrounding artifacts of interest. Examples of such information are the notifications of emails and other pending invitations users get for free on their client while focusing on their tasks. Active awareness tools are user-directed and provide additional details about other users and projects.

To promote group awareness, SCI supports a tree structure of the available users that provides awareness of one's team members. Developers can view others' status and profile, and check what project they are working on. Each team member is represented by a node in the tree and users can tell who is online and working in the SCI environment at a glance. Clicking on the user icon reveals further details about developers' activities, such as what files they are currently editing or debugging, their active projects, their interest groups, and so on.

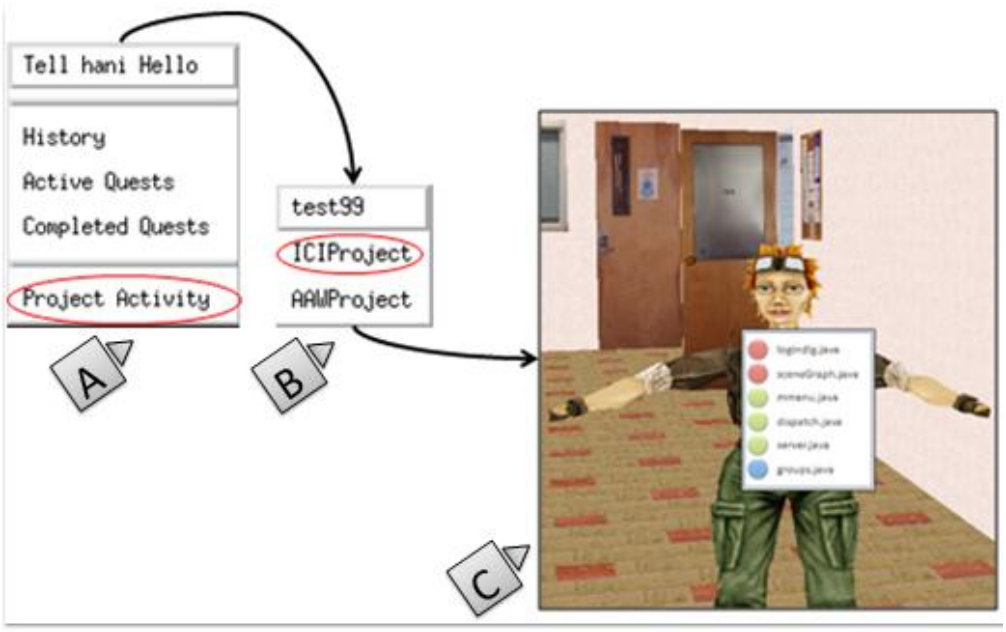

Figure 5.10 Project Artifacts Access

Users in the virtual environment, whose avatars reside in the project/group virtual room, can obtain information about other users' tasks. Right-clicking over their avatar's head causes a popup menu (see Figure 5.10 (A)) to appear with a list of options. Clicking on "Project Activity" causes another popup menu to appear with a list of the projects the user is a member of (see Figure 5.10 (B)). Clicking on any of those projects causes a window to appear with a list of the files he/she accessed in the project decorated with different colors showing the files they are

modifying at the moment (red), files they accessed in the last few days (green), and new files they added to the project (blue) (see Figure 5.10 (C)). Files with the same color decoration are listed under a specific section with headers (modified, accessed, or added) to help developers who have problem with colors easily recognize the differences.

Users gather project artifact awareness information by: (1) hovering over the project files tree (class browser) and the activity tab, where they can see who is editing a specific file, what files have been edited or are being edited at the moment, what kind of sessions have been created, and what are the active sessions (editing, debugging, chatting, and etc). From the users/groups tree tab, users can start a variety of interactions, including text chat, VoIP session, and inviting others for pair programming, debugging, or code reviews.

Developers who use version control systems (e.g. SVN [110], CVS [111]) can observe others' changes by checking the committed changes history in the software repository; however, their local uncommitted changes to the projects' artifacts are not recorded in the shared repository. Unlike the other version control systems Git [77] and Mercurial are distributed peer-to-peer revision control systems with no central repository. SCI does not yet support decentralized revision control systems.

In addition to the ability to observe changes to the files during the collaborative editing sessions by watching others editing, SCI records each user's uncommitted changes in the shared copy of the code, so changed uncommitted pieces of the code appear highlighted with a different color. Recording local uncommitted changes supports awareness in distributed software development and proactively assists users to avoid changes that conflict with others' changes.

Once developers decide to commit the changes, a window will pop up showing who is currently editing the same file, what time they started, who previously changed the file, when they started, and when they finished. They can also compare their own version of the file with the previously saved copies after each commit a developer made. This commit-time information helps users check for any conflicts and decide whether they want to commit their changes or not. Figure 5.11 shows the commit window.

SCI also provides session awareness using a tree structure of the available sessions. By hovering over a session icon (node), a tooltip appears with the session history, showing the owner of the session, who is editing the file at a particular moment, and a list of all the current members and those who made changes to project artifacts and left the session, to help developers gather awareness information about others who may be familiar with a piece of code. Awareness of others editing and/or debugging a specific file helps direct the creation of collaborative IDE editing/debugging sessions. Developers, editing a file or needing some assistance while working

on a piece of code, benefit from being aware of others working on the same code. Also, they can invite them to start a collaborative session.

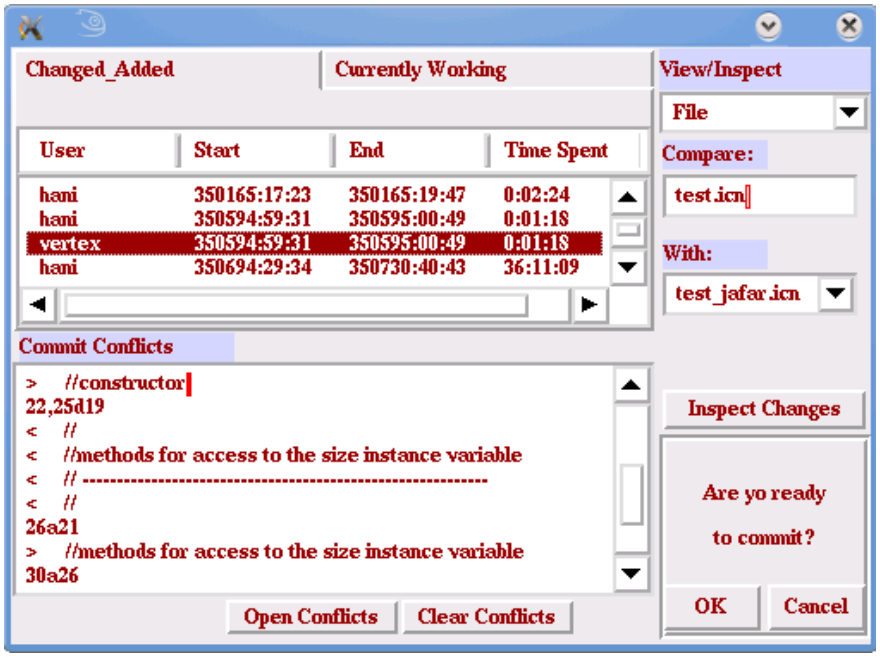

Figure 5.11 Project Files Commit Window. Awareness about the Currently Edited Files

To summarize, SCI provides activity awareness information that covers the following categories: social awareness (the presence of one's collaborators and community members), action awareness (awareness of what collaborators are doing or what they have recently done), and artifacts awareness (information about all the different files or sub-projects that make up the overall project).

5.7 Implementation

This work uses the CVE virtual environment framework/infrastructure to implement and support SCI development environment integration. The software architecture of SCI is depicted in Figure 5.12. The SCI frontend, its editor and shell, is from ICI, while other components and features were implemented from scratch.

5.7.1 Network Protocol

The SCI network protocol messages follow the same design for ICI. Messages are divided into different categories (for detailed information see Appendix B). A BNF grammar that describes the structure for the network messages appears in Appendix D.

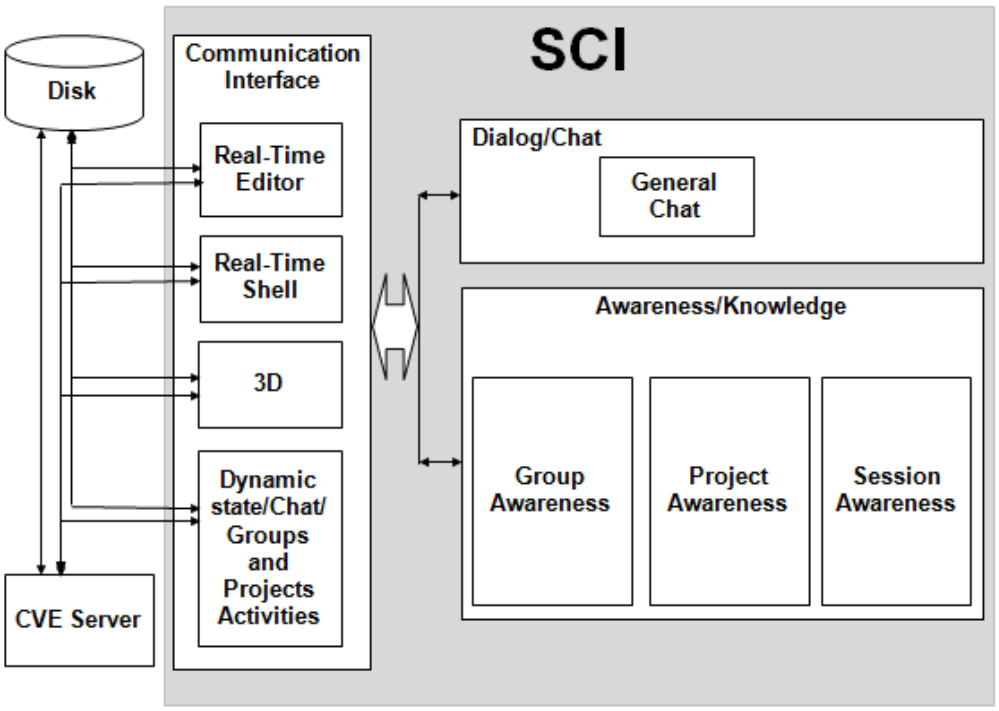

Figure 5.12 SCI Internal Architecture

5.7.2 Source Code

This section includes the SCI source code classes. SCI's source code is organized into five different groups of classes. The first four categories are discussed in Section 4.5.2. Figure 5.13 shows the UML diagram for the classes related to SCI. Following is a brief discussion of the fifth group (labeled *Group # 1*) presented in Figure 5.13 inside the rounded rectangle.

Social Domain Tools includes seven major classes. 1***) ICIGroups*** provides group management functionality, and describes the behavior and properties of the groups in the SCI system; 2) ***NewsFeed*** manages the discussion threads and the system generated feeds; 3) ***Project*** the main software development projects management class provides functions and methods to create projects, invite members, join projects, add files to, access(edit) files from, and commit changes to software development projects; 4) ***Profile*** allows users to create their own personal profiles and view other community members' profiles; 5) ***ForwardDlg*** is responsible of forwarding the collaborative editing/debugging session invitations when a user is busy or not available; 6) ***PendingsDlg*** is the place for the pending invitations organization methods. This class allows the management and organizes the actions a user can take in response to any pending item, such as: accept, ignore, forward, email, and chat; and 7) ***AwarenessMC*** class that

represents the interface between the server and the awareness and social domain tools. It acts as a protocol coordinator that manages delivering the awareness information within the SCI environment.

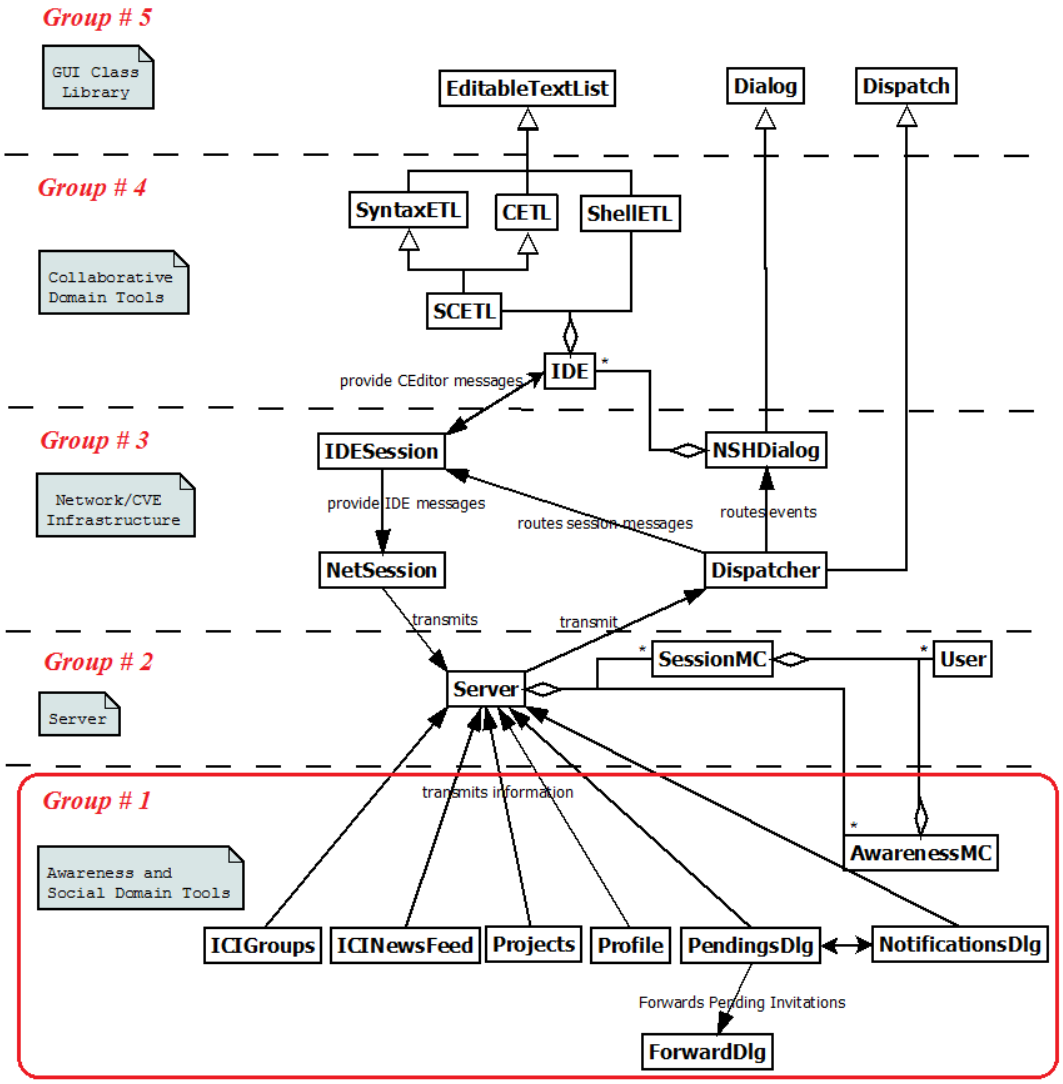

Figure 5.13 SCI's Social Development Environment Class Diagram

5.8 Summary

The design and development of SCI environment were the focus of this chapter. SCI is a social development environment within the CVE system, and also functions as a standalone programming environment.

Chapter 5 presented a social programming environment that supports developers' collaboration, and interaction. It allows them to be aware of the actions of others in real time,

aware of artifacts changes, avoiding coding errors, and potentially improves the collaboration during the software development process. An environment that combines: communication, collaboration, awareness, and social networking and online presence features from inside a single environment.

In addition, this chapter presented the architectures for the design and implementation of the system primary components. It supported features such as: online presence, activity awareness, and social network features. It also described and defined various basic concepts including: awareness, presence, and social presence as desirable properties of social collaborative development environments.

Chapter 6
Notifications Management

This chapter introduces the design and implementation for a framework that manages pending invitations and requests inside the SCI development environment. Section 6.1 introduces the problem and presents the solution. Section 6.2 provides a preliminary taxonomy of notification types in SCI. Section 6.3 introduces the presented framework model. Section 6.4 introduces the supported use cases.

6.1 Introduction

As mentioned earlier, collaborative software development revolves around interactions between developers. SCI is designed to enhance such interaction between the team developers. Development in such an environment is adaptive to constant changes happening to the team, and it requires notifications to help delivering the constantly rapid changes.

These notifications include file and project sharing invitations, and editing and debugging session invitations. Social features add different kinds of interactions such as friend requests, group invitations, project discussions, emails, instant messages, and feed posts. In such an environment, users might get a lot of invitation traffic, which tend to be immediate and interrupt the users suddenly without regard for their current tasks [113, 114], affecting the development process and productivity. This creates a need for a framework that can transparently manage all kinds of invitations.

A notification is an essential feature in a collaborative system that determines the system's capability in supporting different collaborative work activities, and distinguishes collaborative systems from other general multi-user systems where users are normally not notified of the actions performed by others, such as database management systems [115]. Managing notifications is required for three main reasons. First, notifications can lead to interruption among the development community as mentioned in related research [116]. Second, the recipient of the invitation may be unavailable. In this case, invitations can be held until the invited user gets a chance to see them. The third reason is that managing invitations is required from a usability and scalability point of view. For example, an invitation to a collaboration session might intuitively be delivered via a popup dialog, but intrusions do not scale well for busy users. Managing several

opened windows and switching between them can be cumbersome especially if there is a large number of incoming invitations.

6.2 Notifications Taxonomy

Like other related research studies [113, 116], SCI classifies the notification types supported and delivered by the system. This classification is presented using a two dimensional matrix. The first dimension reflects the way the notification is delivered and/or initiated (how). The second dimension reflects who initiates the notification (who). This taxonomy refines the notification taxonomy introduced by McGrenere et al. [116]. Table 6-1 shows the taxonomy matrix of SCI's notifications.

	Who	
How	*Individual*	*Group*
Direct	Occur as a direct result of the user's intentional actions.	Notifications related to the development process among teams' members.
Indirect	Notifications occur as a result of external or unintentional actions.	

Table 6-1 Taxonomy of SCI Notifications (Adapts and Refines Jazz's Notification Taxonomy)

— **Individual-Direct**: notifications happen as a direct result of the user's actions. In general, any notification generated as a result of the user's task can be treated as a direct notification. Other examples are:

- An individual sends an instant message, an email, or starts a VoIP call to an individual or team.
- An individual sends an announcement to a team.

— **Individual-Indirect:** notifications occur as a result of the users' unintentional actions. Following are examples of such actions that generate notifications in the recipient(s) receiving.

- A developer creates a project, a notification to all members is delivered that a new project has been created.
- A developer commits changes to any of the project artefacts, and team members are notified about the code commit event.
- A developer adds a new comment or changes the project wall of a bug report, and other members of the project are notified about the changes.

- A developer leaves a collaborative session without warning his/her partner

— **Group-Direct** and **Group-Indirect:** notifications related to the development process among teams' members.

6.3 Design

The primary emphasis in the design of the proposed system is to minimize the transition effort from individual work to collaborative sessions and interaction with other users, and then back to individual work, so that transition can easily occur dozens of times during the course of a work period. A Facebook-style visible indicator of pending notifications and email are implemented, adding the ability to review the queue at one's convenience to see who has been waiting for what, and for how long, and to provide the developers with passive awareness notifications. Figure 6.1 shows the awareness notification's icons.

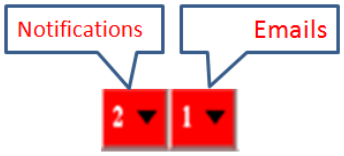

Figure 6.1 Notifications' and Emails' Icons

Once the users get an invitation, they can access the notifications and emails by clicking on the icons in Figure 6.1, and a list of all the available notifications appears. Clicking on any of the list items causes a window to appear showing several actions (see Figure 6.2).

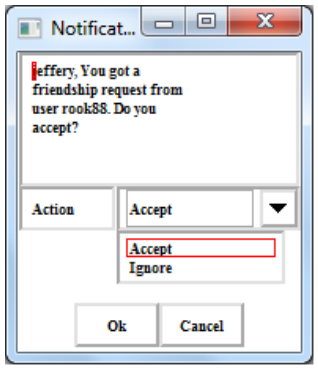

Figure 6.2 A Notification Window.

Users can choose one of several provided actions. They can accept, reject, or discuss the invitation by starting a chat session or write an email to reply to the invitation. In some cases such as collaborative IDE editing and debugging sessions, users may forward the invitation to another user or expert. For example, user "A" invited user "B" for a collaborative IDE session, and user "B" is busy assisting another team member. In this case, user "B" might forward the session to user "C", and the user will make an accept or reject action. All invitations and requests are placed in the pending list until an action is taken, except the initiation and joining of the collaborative IDE sessions. The invitation is removed from the list when either the sender or the receiver of the

```
1   #
2   # Shows how the server prioritize the notifications
3   # between the users uname and sname.
4   #
5   if isFriend (uname, sname) then
6      pinv.ppriority := 5
7   else if isPartner (uname, sname, "sig") |
8           isPartner (uname, sname, "project") then
9      pinv.ppriority := 4
10  else if isFriend_Of_Friend (uname, sname) then
11     pinv.ppriority := 3
12
```

Figure 6.4 Snapshot of the Code that Shows How the Server Prioritize the Notifications

invitation signs out of the system for any reason. The system provides a prioritization for invitations. Invitations from friends' circle and project team members get higher priority than other invitations and requests, meaning that some kinds of invitations are more important (See Figure 6.3).

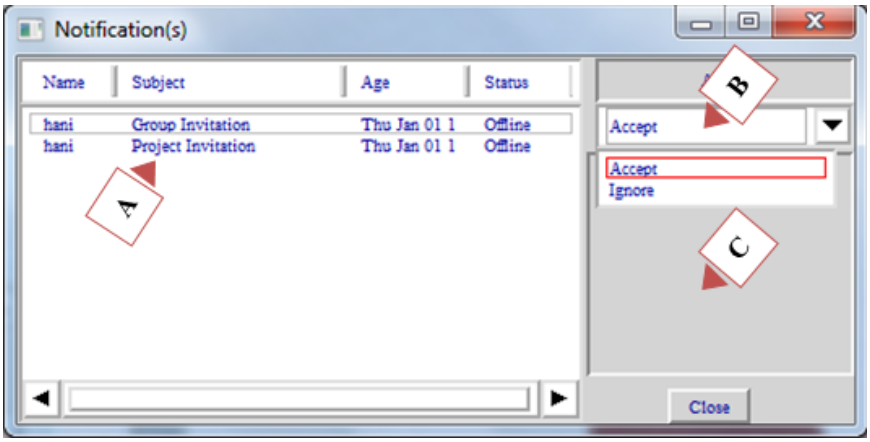

Figure 6.3 Pending Invitations/Requests Management Window

The framework is designed to handle duplicate invitations smoothly. Users can browse the notifications that they got by clicking on the notifications icon in Figure 6.1 and choose *Browse*, and a new window will appear with all the notifications (See Figure 6.4).

Figure 6.4 shows the pending invitations management window. The user's pending invitations are shown in a main window (A). Users can select the action to reply to the pending item from (B), information about the selected pending item appears in (C).

Also, users can browse the old emails by clicking on the emails icon appears in Figure 6.1 and choose *Inbox*, and a new window will appear showing the emails they got (See Figure 6.5).

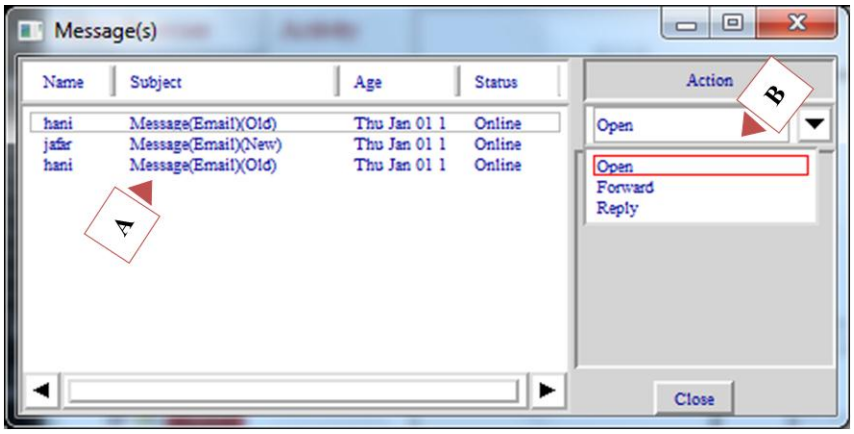

Figure 6.5 Inbox Window

For such a framework there must be a shared workspace that consists of participants and pending actions (entities). For every single invitation or request that occurs, the participant needs to invoke an operation on an object such as menu or button (by means of sending it a message) in the proposed framework. For example, if a participant initiates an editing session, the framework needs to reflect this activity by sending a message add_pending() to the system. Later, when the invited participant accepts/rejects the invitation, the framework sends a message del_pending() to remove the pending entity from the queued operations or entities.

In this model, an event will be created each time an activity occurs (pending invitation or request is sent). The system stores information about each activity in a log file, such as sender, receiver, time-stamp, type of activity to keep tracking of the activities and ease their management.

6.4 Use Case Model of Pending Management Framework

The proposed framework needs to initiate the actions (Events) and deliver them to remote users. Thus it needs Initiation functionality and distribution functionality. Events occur as a result to the users' actions form the input of the framework and called *PendingEvents*. The information

(pending entities) delivered to the users form the output of the system and called *PendingNotification* (See Figure 6.6).

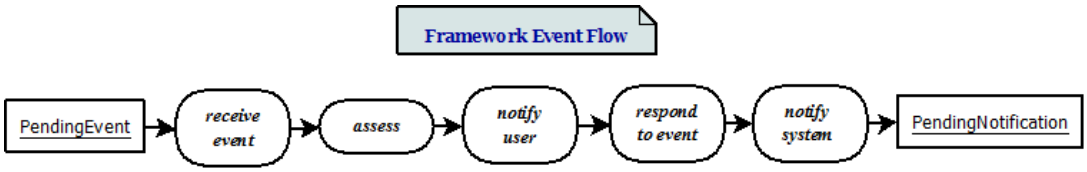

Figure 6.6 Events Managing Process [97]

The inputs from the users are not simply forwarded to the other users, and the pending notifications need assessment before it will be sent. For each *PendingEvent*, the system needs to assess the event nature and the target. Only if the *PendingEvent* appears to be new, not a duplicate, the framework will send the event. Once the user accepts the invitation/request, the framework will send the *PendingNotification*.

The pending framework can be described in terms of two key use cases: the initiation functionality and distribution functionality. Each pending activity requires two main actor types: the sender and the receiver (See Figure 6.7).

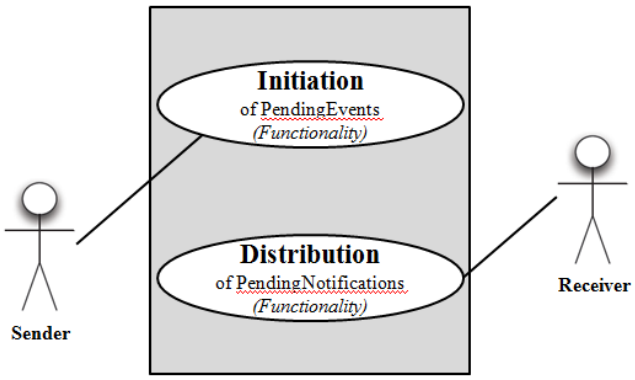

Figure 6.7 Notifications Management Framework Main Actor Types (Sender and Receiver)

6.5 Pending Scenarios

This section introduces all the supported pending invitation scenarios, and illustrates the activity using UML activity diagrams. As mentioned earlier, interactions among developers include collaborative session (editing, compiling, and debugging) invitations, friend requests, group invitations, and project discussions.

SCI supports general collaborative software development tasks. However, it was built to serve the specific needs of computer science and software engineering education, particularly distance

education. Also, it serves niche programming communities and their developers. There are several requirements scenarios in computer science teaching environments (See Section 4.2).

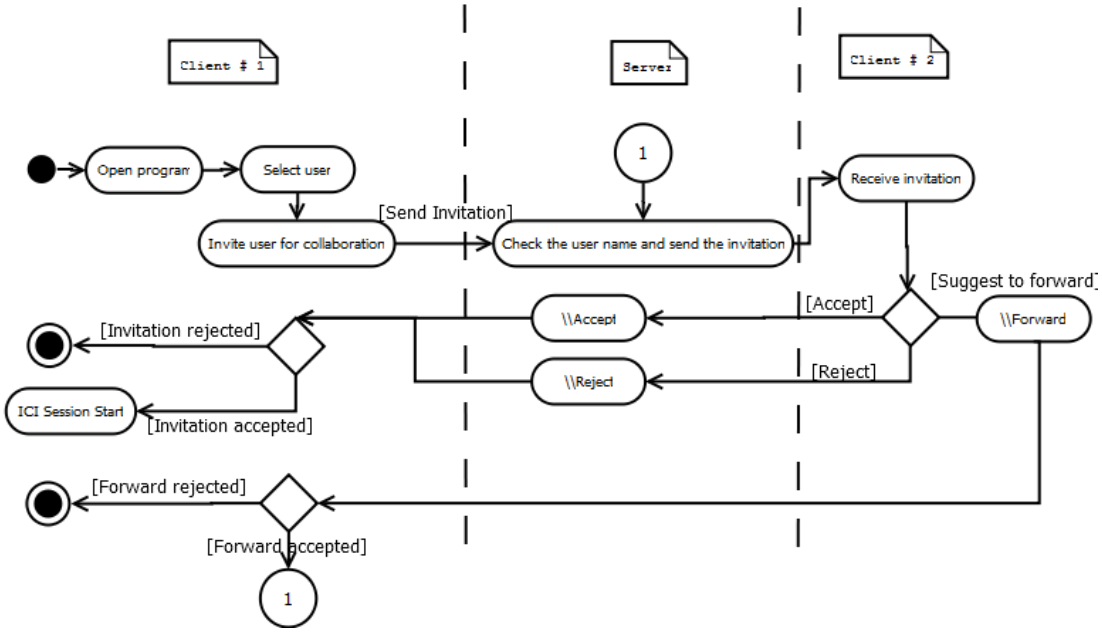

Figure 6.8 IDE's Session Invitation Activity

Figure 6.8 shows the IDE collaborative session invitation activity.

Another kind of interaction that is related to the collaboration session is the take turn action where a collaborator who is in a watch mode during the collaborative editing session requests permission to switch to the edit mode.

During a collaborative debugging session, developers can take turns controlling the debugger, while other developers watch and discuss the debugging commands and messages. Figure 6.9 shows the collaborative editing session take turn request activity.

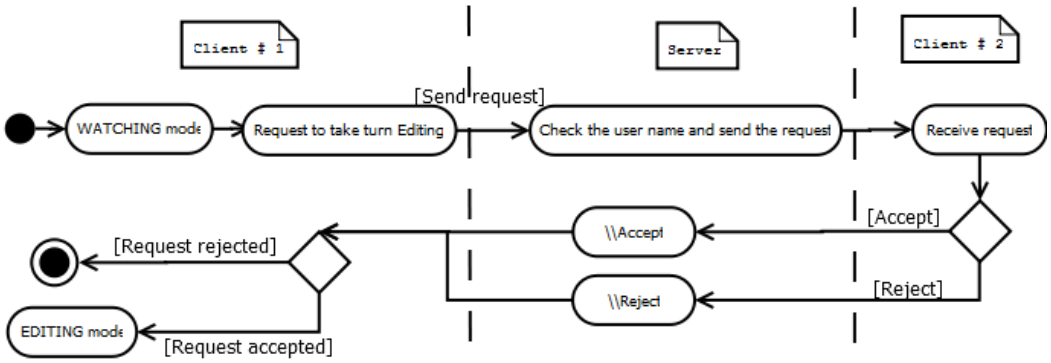

Figure 6.9 IDE's Session Take Turn Request Activity

Another common type of interaction that takes place between developers is adding new friends. In order to build a social community circle, developers request other peers to become their friends. The invited developer has to reply to the friendship request sent by the others, and if the reply is positive and he/she accepted the request, both the developers become friends (see Figure 6.10).

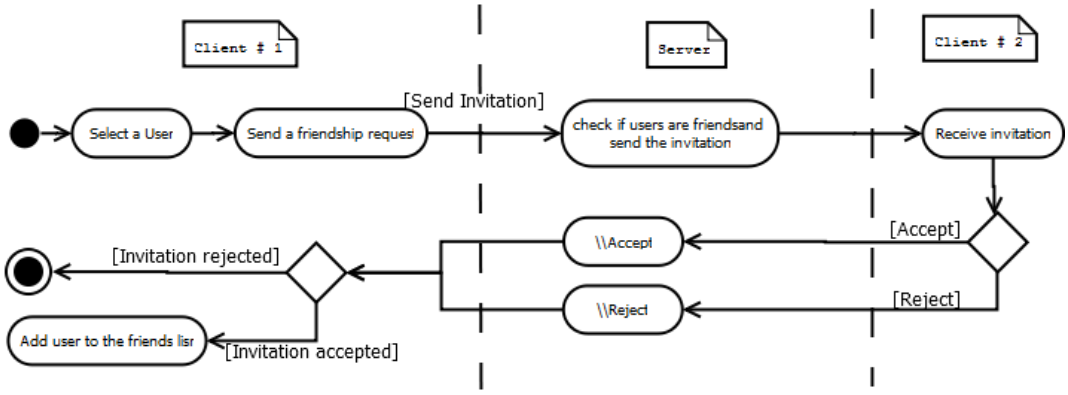

Figure 6.10 Friendships' Request Activity

Another two important types of interaction that takes place between developers in a distributed environment are project and group join invitations (See Figure 6.11). Those interactions are common practices between developers who work together to achieve a goal, and finish a specific task. Developers send others invitations to join a project or group. Also, sometimes they request to be members of a specific project or group.

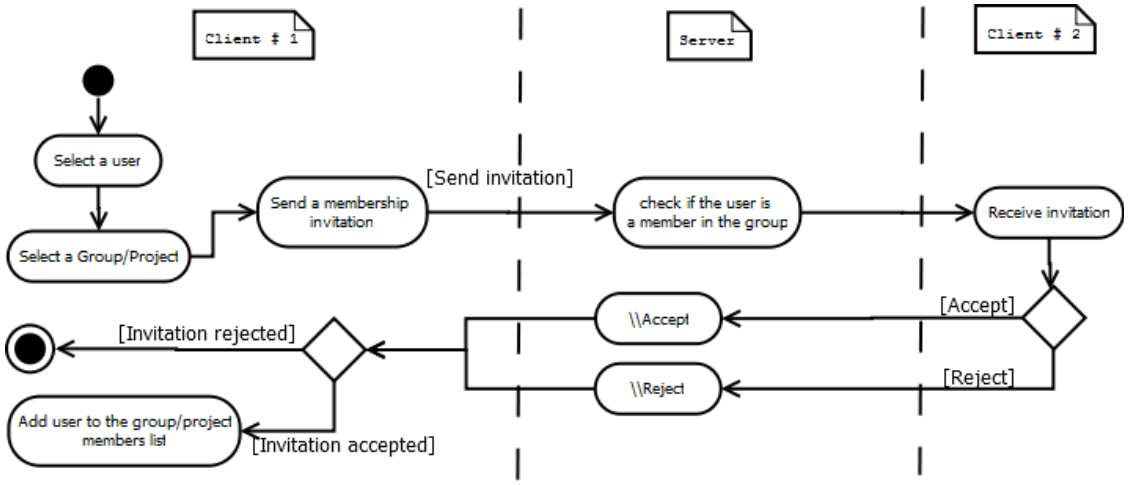

Figure 6.11 Projects' and Groups' Membership Invitations Activity

6.6 Summary

Collaborative software development revolves around interactions between developers. The design and development of notifications framework was the focus of this chapter. Chapter 6 presents a general notifications management framework that controls and manages the emails and any other kind of notifications a distributed developer using SCI would receive.

This chapter also presents a classification of the notification types supported and delivered by the system. It classifies the notifications to reflect both of (1) the way the notification is delivered and/or initiated, and (2) who initiates the notification.

This framework is designed to enhance the interaction between the team developers. The framework was applied to the design of a notification component for the SCI system, and tested during the evaluation process of the system. Also it was evaluated to be a usable and useful feature.

Chapter 7
Using SCI in Introductory Computer Science Classrooms: a Case Study

Teaching computer programming has always been a challenging task. Students find it difficult to learn programming languages and write programs. The difficulties of teaching and learning programming contribute to the decline in the number of students taking computer courses. When it comes to teaching areas of technology and communications such as computer programming, efficient collaboration tools are required for two important reasons: firstly, improving computer-based communication techniques is crucial in this era where students depend on computers to perform most of their tasks; and secondly, those students will be the future software engineers that are needed to meet the huge demands for software development. Teaching computer science courses usually starts with teaching programming prior to teaching advanced topics, and programming is generally viewed as a difficult part of such courses [35].

Students taking computer science courses may attend class and work in different places away from each other, and from their instructor(s); to make it easier for them to seek assistance solving their assignments, remote collaboration tools are needed. Thus the computer science courses are a primary venue for collaborative and social IDE's such as SCI.

One natural application of SCI in these contexts is distributed pair programming. Pair programming is a style of programming in which two programmers develop software side by side at one computer, collaborating on the same design, project code, or test [117]. Distributed Pair Programming (DPP) also known as virtual pair programming, is a style of pair programming where the two programmers are in different locations [118], working via a collaborative real-time editor, shared desktop, or a collaborative development environment. Previous research has shown the benefit of using pair programming in collaborative learning for introductory computer science classes [27]. Also pair programming has shown a notable improvement in the students' performance. "The research on teaching and learning over the past 50 years suggests that the early use of collaborative learning leads to higher interest, higher retention, and higher academic performance in students. Early use of these techniques can also increase the sense of belonging for students and can lead to the early development of collaborative skills to prepare students for team experiences in subsequent courses and future careers" [119].

The research goal for this study is to make it easier for students to experience virtual team work, get some experience with the distributed software development process and the expected challenges, and test the effect of using the SCI system in supporting distributed software development. This study aims to provide insight into three main research questions. These are:

1. How efficient is SCI in the classroom? How feasible and usable is the SCI system?
2. Are the supported means of communication and collaboration adequate?
3. Is the supported awareness information adequate?

In the context of this study, adequate can be measured by: 1) observing whether the collaboration and communication tools integrated within the SCI environment were valuable (whether they helped the participants finish their work and increased their productivity); and 2) surveying the users and asking if the integrated features are enough, or if they prefer to have other tools integrated within the environment.

The first question is answered by the participants' observations (*feedback*). The second and third questions are answered by observing: a) how often the group members communicated while in collaboration; b) what communication tools they used the most, c) if that affected their progress on the task and helped them finish the task, d) how fast a user responded to a notification or an invitation, and e) if the supported passive awareness features were helpful.

This chapter presents the format of the study, background of participants and study setup. This is followed by a discussion of the design of the questionnaire administrated at the end of the study. Finally, the results generated from the study are presented.

7.1 Study

Software Engineering and related subjects aim to teach students how to do collaborative work and enhance their programming expertise. The following study was inspired by these mentioned goals and is implemented during the "CS120: Computer Science I" class at the University of Idaho, in Summer 2011.

As mentioned earlier, this study investigates the efficacy of using SCI in the classroom, the required means of communication, and the needed level of awareness and presence information for SCI to be useful.

7.2 Methodology

The study consists of three separate tasks. The first task focused around finishing a simple programming exercise in the classroom. The task required using both the collaborative editing and debugging features supported by the system.

The second task is a simple collaborative debugging activity where students were given a C++ program with errors and missing methods and were required to debug and fix the code. Participants involved in the second task were not aware of the other group member at the start of the session. They worked individually to finish the assignment.

The third task focused around performing activities that were related to both the 3D environment and the social networking features, and finishing simple programming task. Similar to the second task, participants were not aware of the other team member at the start of the session. Participants worked with their teammate(s) to finish the assignment.

Every participant that was involved in any of the mentioned three tasks belonged to a group of two members; participants were randomly allocated to these groups. Each task session was scheduled to take between 45 and 60 minutes. Before the participants started the session, a printed document with instructions about the task was distributed and the instructor made sure that everybody was aware of their part of the task (see Appendices E1-E3); a short training session (30 minutes) was held to make the participants familiar with the features available inside SCI and the objective of the study in order to make their job easier.

In addition to studying the user interface, the study evaluated the effect (net gain or loss in productivity) of integrating and using the SN features such as the awareness notification on collaboration practices such as editing, and debugging. Our goal was not to test whether the SCI environment is as efficient as the co-located one, but to test if SCI provides an environment that could "work well enough" to allow the participants produce functioning software in a reasonable time. Figure 7.1 shows the process flow chart.

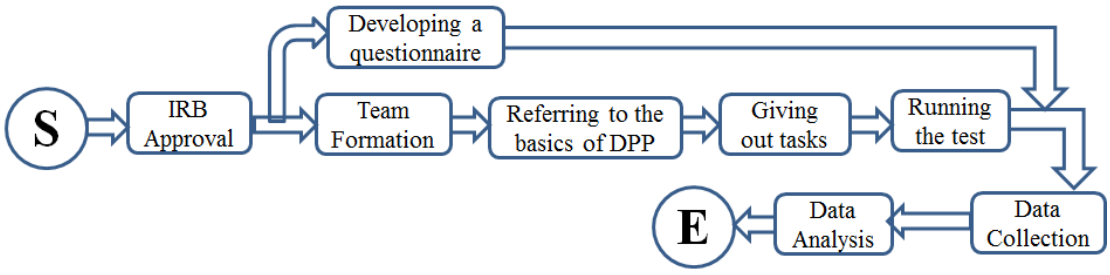

Figure 7.1 Case Study's Process Flow Chart

7.2.1 Background of Participants

The study targeted individuals with beginner-level expertise in the *C++* programming language who were willing to use the SCI development environment to finish the assigned tasks. The real names of the participants were anonymous and no personal information was revealed or required. Since the study focus does not depend on the nature of the groups, participants were randomly allocated to groups of two by the system administrator and the class instructor. A letter-name is assigned to each of the formed groups (eg. groupA, groupB …).

7.2.2 Task

During the study, teams carried out tasks. Each team of students worked on the same task. Students used *C++* as the programming language.

Task One:

Before teams started working on the assigned task, a master (driver) for the session was assigned. Although the other participant was asked to finish some parts of the task, he/she acted primarily as a navigator or watcher. Participants divided the work between them (assigning specific method(s) for each to finish) or just collaborated in finishing the missing parts.

The students that participated in the first task (collaborative editing and debugging task) followed the following steps:

1. Participants were allowed to communicate with each other only using on-line tools such as: email, and instant messaging (Google Talk, MSN messenger). They were asked to exchange their contact information.
2. They were asked to agree on a name for their group.
3. Participants were asked to join the C++ special interest group.
4. Then, they were asked to start a collaborative session. The driver opened an empty document and saved it as a *[groupName].cpp* file.
5. After that, the driver started working on his/her part of the task. The other member acted as a navigator, assisted via chat and watched changes to the shared code. Also, the watcher was allowed to request to take turn and help with the assigned task.
6. When the driver finished his/her part of the work, the watcher asked for permission to take turn and added their part of the code within the shared code and gave the floor control back to the driver of the session, and moved to the watch mode and vice versa.

7. Once each of the group members finished his/her part of the task, participants discussed the design of their program. The driver had control of the shared code and only the driver could edit/change the code. Watchers continuously and actively examined the work of the driver, watched for errors and thought of other alternatives.

Task Two:

Students that participated in the second task performed a simple debugging task in the classroom, consisting of the following steps:

1. First, the instructor created a number of software development projects within the CVE (a project for each team). Then, the instructor added a file required for the task to each project (a C++ file with some bugs and errors).
2. Participants were required to join their assigned project.
3. They were required to open the file *"taskprog.cpp",* and fix the bugs.
4. Participants were required to send friendship requests to other users available in the environment. They were required to establish friend status with their assigned partners.
5. They were allowed to chat with other participants and request help, and ask how to solve a specific bug (all from inside the CVE). Also, they were allowed to ask for assistance from the other project member using email.
6. In case the participants had a problem figuring out how to complete a missing piece of the code or to find and fix a bug in the code, they were allowed to post questions to the project's wall, and request assistance from other project member.
7. After finishing the task, students were asked to compile, run, and make sure that their own versions of the code were error free.
8. Then, they were asked to commit their changes to the server, and to add their own version of the file to the project files.

Task Three:

Participants were required to perform activities that were related to both the 3D environment and the social networking features (students had the choice to finish this part after the class time). Before starting the session, the system administrator created the project "***My3D***", assigned the participants to teams, and assigned a location (room) for each team. Teams and locations were assigned randomly. Participants involved in this task performed the following activities (interactions):

Interactions from inside the 3D world

1. Switched to the 3D world tab.
2. Checked the other team member's locations from the users' tree.
3. Asked to visit the closest rooms to their location, and interacted with avatars (sent greetings, and viewed activities).
4. Asked to teleport to the general meeting room (the virtual room **JEB321**), see who is around and introduce themselves.

— **For example:** to teleport to JEB321 room, you type **teleport JEB321** in the chat box.

5. Asked to interact with the nearest avatar in **JEB321** room (interactions such as sending greetings, and viewing activities).
6. Asked to check who is their assigned partner, introduce themselves, and then teleport to the group's room.
7. They were asked to perform a list of activities with their partners after moving to the group's room. These were asked to:

— Send a greeting to their partner(s).
— View their activity "History".
— Check their "Project Activity".
— Exchange roles (one of them is master/driver and the second is watcher/helper)
— Join the project "**My3D**".
— Discuss the solution for the assigned program.
— The driver of the session opened a C++ file and started the program.
— After finishing his/her part of the code, the driver added the file to project "**My3D**".
— The helper opened the file and added his part of the code, and compiled the program to make sure it is working.

Interacting with the SCI Social network features

8. Viewed their partner's profile (or any of the CVE users).
9. Checked the availability table and progress chart available at their profiles, and checked to figure out at what time probably his/her partner is available for help?
10. Viewed their partner's availability, by checking the "User's Usage Report".
11. Sent an email(s) to their teams' partners, and asked questions about their experience with the SCI environment. Also, they were asked to send email to the system administrator to suggest changes to the system and leave comments.

Depending on the nature of the third task, and the questionnaire statements, a descriptive analysis of the third task results appears in Section 8.3.

As with any other social network, the SCI users form the social network's nodes, and relationships and activities among these users form the edges of the network/graph. In SCI, there are four kinds of active edges: friendship (friends) edge, group partners, project partners, and expert edges. After finishing the assigned tasks, the actions the participants performed show the activity on the graph edges.

7.2.3 Questionnaire Design and Measures

After completing each task, students were asked to fill out a survey to answer questions about their experience while using the SCI system, the difficulties they faced, and things they liked and things they didn't like about the system. The questionnaire is designed to obtain preliminary feedback on the SCI system and its implementation. It contains both open and closed questions and is designed to measure the effect of the SCI system in the classroom. This was given to participants at the end of the study (See Appendices F1-F3).

Participants were required to respond to closed statements on a Likert scale where they needed to specify their level of agreement ranging from the response "Strongly Agree" = 5 to the response "Strongly Disagree" = 1. Other questions are also asked so participants can suggest features they would prefer/like to integrate within the collaboration space.

7.3 Goals

There is one main observation goal of the case study; the goal is to observe how participants perform their programming task in the sense of interaction design and communication support. This research obtains data from this case study and uses this data to improve the communication tools inside the CVE virtual environment and the SCI development environment. Collecting data is achieved in the following directions:

- Observing interactions among the students/players within the group.
- Observing what tools and features are needed to improve the user's awareness of changes to the project, and awareness of others.
- Analysing information design specifications (how easy and useful a certain interface features are and how they should be presented in a new electronic version)

7.4 Discussion and Findings (Descriptive Analysis)

As mentioned above in Section 7.2.2, the study consisted of three separate tasks. Table 7-1 introduces a brief statistics to the participants' nature, tools they used, and tools they would like to see integrated in the system.

Task #	Participants (Total)	Participants responded to the survey (Total)	Participants suggested/liked tools integration inside the CVE (%)					
				Chat	**Email**	**Wall**	**Newsfeed**	**SN Features**
1	8	7	Like	86%	29%	58%	N/A	N/A
			Used	N/A				
2	7	7	Like	100%	100%	100%	86%	100%
			Used	86%	57%	57%	N/A	N/A
3	7	4	Like	N/A				
			Used	N/A				

Table 7-1 Summarized data collected from the survey distributed during the study

Following is a discussion of the data collected during each task.

7.4.1 Task One

The first task focused around performing a simple collaborative programming task in a classroom where participants can work from inside the CVE and communicate using external communication tools. In this session (task), the population consisted of 8 students, enrolled in the University of Idaho CS120 course, Computer Science I, in the Summer of 2011. The study assesses the effectiveness of SCI system on the participants' experience. This task evaluates the student engagement to the CVE environment without the benefit of SCI, and using a student survey.

The surveys were given to students to complete and returned anonymously. Figures 7.2 – 7.5 show a summary of results of this survey for the first task. The bar graph in Figure 7.2 corresponds to the answers of the following statements:

Statement1 (Q1) Rate your knowledge/expertise of C++ programming.

Statement2 (Q2) I have experience working with collaborative tools.

Statement3 (Q3) I have had experience working with pair programming teams before.

Statement4 (Q4) I usually work physically close to my teammate(s).

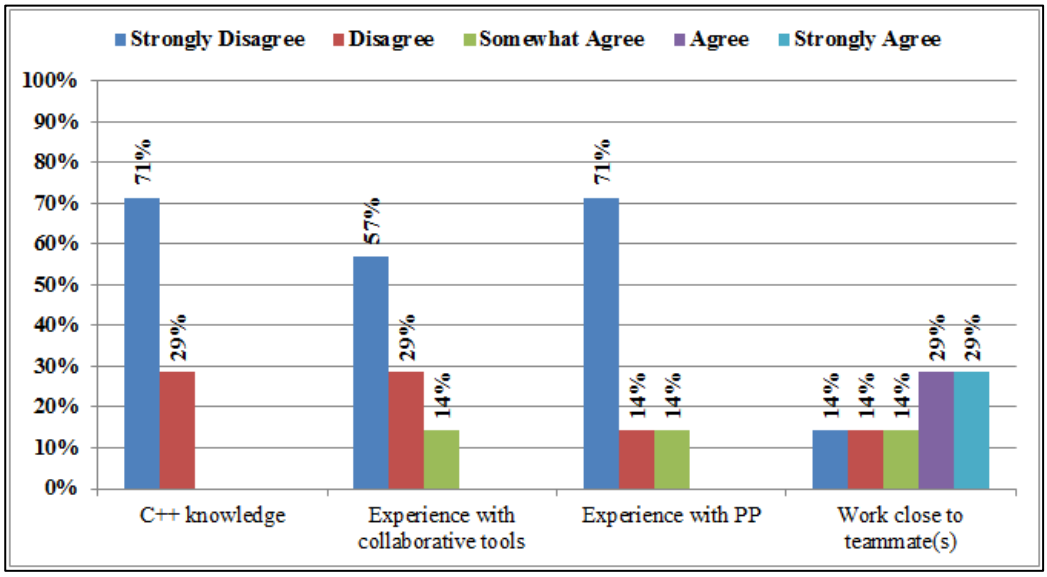

Figure 7.2 Responses to Participant Related Questions.

It was observed from the data shown in Figure 7.2 that 86% of the participants didn't have experience working with collaborative tools. Also, a significant majority of the participants (85%) had no experience working with pair programming teams before the session. It was also observed that 72% of the participants usually work physically close to their teammate(s).

The bar graph in Figure 7.3 corresponds to the answers of the following statements:

Statement5 (Q5) It was easy to communicate (start an online chat/conversation, and/or send email) with my team partner.

Statement6 (Q6) It was easy to watch my partner make changes to the code while conversing using the external chat/email system.

Statement7 (Q7) I prefer having the following tools integrated inside the CVE environment:

(Q7-1) Chat (text/audio)

(Q7-2) Email

(Q7-3) Wall/Forum

Also, it was observed from the responses that only 43% of the participants found it easy to communicate with their teammates during the collaboration session, and 42% of them found it easy to watch their partners' changes to the shared code while conversing. Participants' responses showed that students used chat to communicate with their teammates more than email and walls. Participants showed the need to have these tools integrated inside the SCI environment; 86% of the responses showed the need for integrating a chat tool, 72% of the participants showed interest

in having an email tool, and 72% showed interest in having a wall/forum integrated inside the environment.

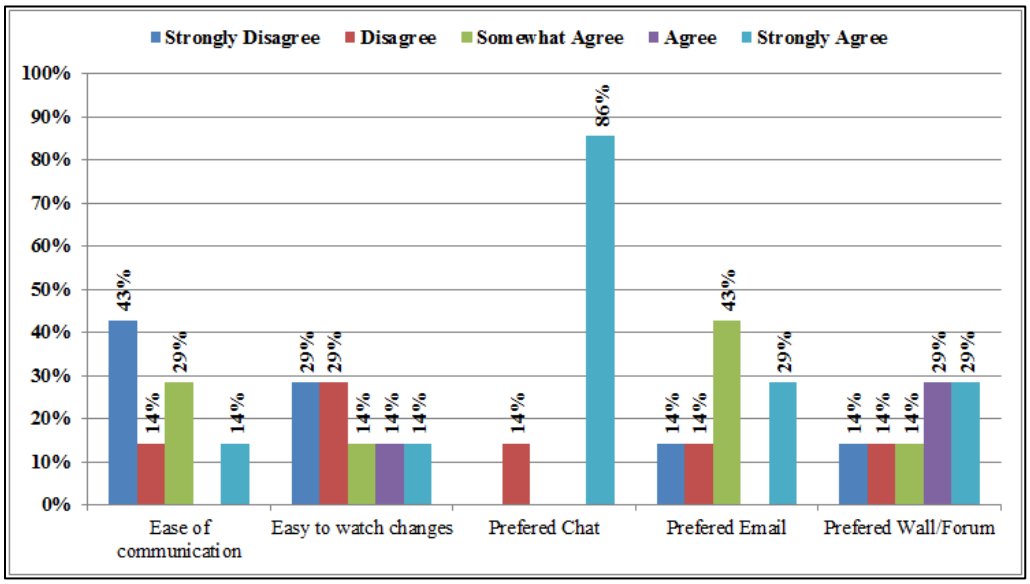

Figure 7.3 Responses to Communication Related Questions.

The bar graph in Figure 7.4 corresponds to the answers of the following statements:

Statement8 (Q8) It was easy to start and end the collaborative editing session.

Statement9 (Q9) It was useful to watch my partner editing/debugging the code.

Statement10 (Q10) It was easy to get assistance within CVE compared to face-to-face meetings.

Statement11 (Q11) I was more comfortable with my solution knowing that someone was watching the code and helped debugging it.

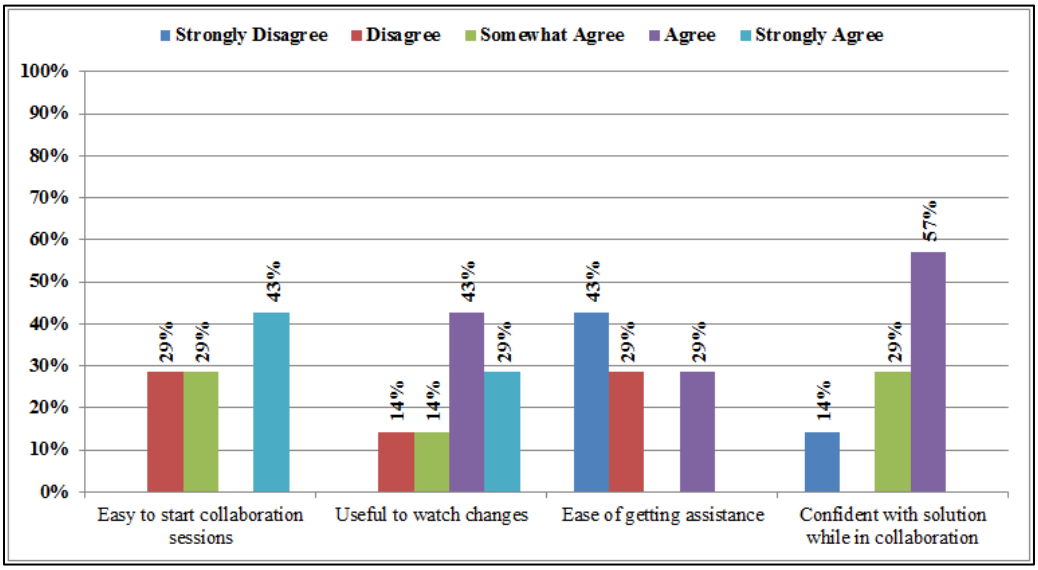

Figure 7.4 Responses to Task Related Questions.

From the data appears in Figure 7.4 it was observed that 29% of the participants found it difficult to start a collaborative session in SCI. 86% of the responses showed that the participants found it useful to watch their partners editing and/or debugging the shared code. 71% of the participants claimed that it was not easy to get assistance while collaborating within CVE compared to face-to-face meetings. Also it was observed that 86% of the participants felt more confident of their solution knowing that somebody was watching the code and helping to debug it.

The bar graph in Figure 7.5 corresponds to the answers of the following statements:
Statement12 (Q12) Overall, it was easy for me to use CVE to collaborate with my partner and finish my task.
Statement13 (Q13) Overall, the CVE IDE window was distracting.
Statement14 (Q14) Overall, I liked finishing my work with a partner and within CVE more than working alone.

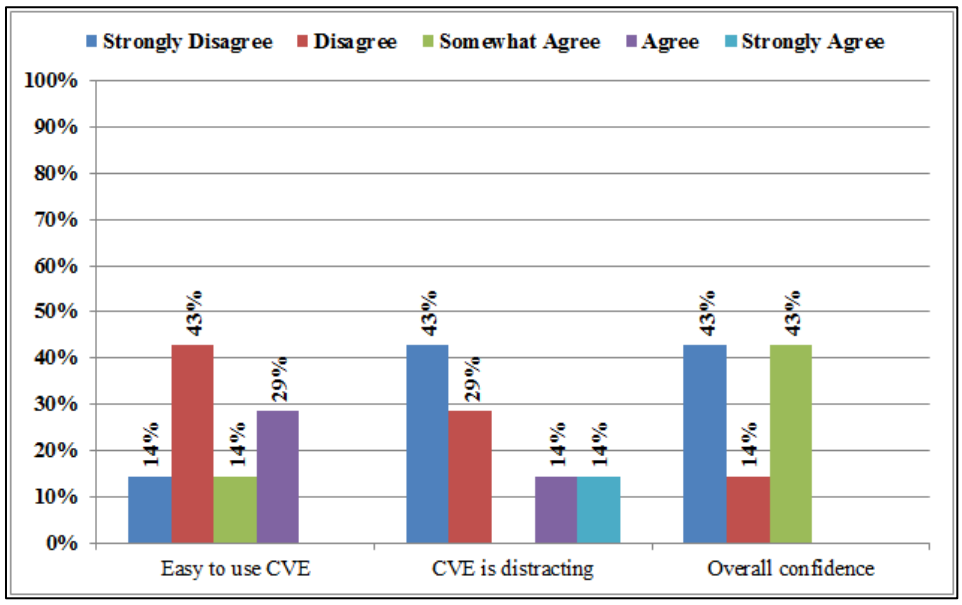

Figure 7.5 Responses to General Questions

The responses showed that only 57% of the participants found it easy to collaborate and finish the assigned task from inside the CVE, and that was related to two important factors: 1) the server instability, and 2) the difficulty to communicate with their partners using tools from outside the CVE. For the same reasons, 57% of the participants expressed dissatisfaction, and disliked working with a partner from inside the CVE. Even though some of the responses reported

usability issues, 72% of the respondents strongly disagreed or disagreed that the CVE IDE window was distracting.

Student's general comments appear in Table 7.1. The majority of their comments were beneficial. Overall, the students agreed and noticed the need for tools integration inside the CVE environment, and suggested some usability changes to the way notifications and awareness information were delivered.

Other Comments
"The system often crash and fail connecting."
"No built in chat features made communicating difficult. Using a third party chat program was slow and hard to into the development."
"Notifications for starting a collaborative session and request for "drivers" should be more visible."
"Server instability made starting a session and maintaining difficult."
"Include a compiler with the CVE install package."

Table 7-2 Summary of other comments from participants took part in the first task.

Summary of findings

The responses showed that using tools from outside the environment made it difficult for the participant to interact and collaborate. Also, it prevented them from requesting assistance while finishing the assigned task. On the other hand, they felt confident of their solutions knowing that someone is helping by watching and debugging the shared code. The participants expressed the need for tools integration inside the SCI environment, and found it useful to watch their teammates' changes to the shared code even though it was not easy for them to communicate with them. The server instability and the lack of tools integration made the student dissatisfied with the system effectiveness and usability.

7.4.2 Task Two

The second task focused around performing a simple debugging task in classroom where participants collaborate to finish the task, communicate, and use all the supported features from inside the CVE virtual environment. Participants worked on this assignment individually, but they

were allowed to ask for assistance from their instructor and/or classmates. In this session (task), the population consisted of 7 students, enrolled in the University of Idaho CS120 course, Computer Science I, in the Summer of 2011. The study assessed the effectiveness of SCI system on the participants' experience. This task evaluated the students' involvement in the CVE environment with the benefit of SCI, and using a student survey.

The surveys were given to students to complete and returned anonymously. Figures 7.6 – 7.10 show the summary results of the second task survey. The bar graph in Figure 7.6 corresponds to the answers of the following statements:

Statement1 (Q1) During the session, I used the following communication tools:

(Q1-1) Text chat (Q1-2) Email

(Q1-3) Wall (Q1-4) Newsfeeds

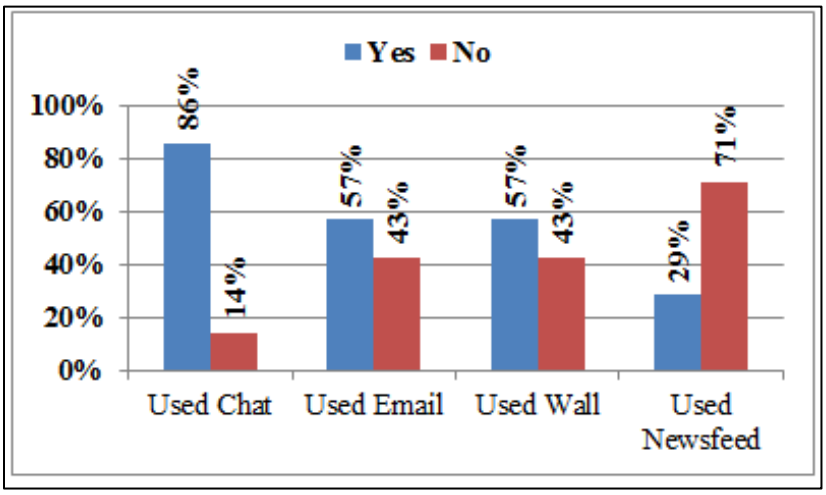

Figure 7.6 Responses to Communication Related Questions. Showing What Communication Tools Participants Used More (Task #2).

It was observed from the above data that, during the session, 86% of the participants used text chat to communicate with their teammates, 57% used email, 57% used the wall tool, and 29% used the newsfeed. From the above it appears that the main communication tool (mostly used) is the text chat.

The bar graph in Figure 7.7 corresponds to the answers of the following statements:

Statement2 (Q2) It was easy to communicate (start an online chat/conversation, and/or send email) with my team partner.

Statement3 (Q3) It was useful to have the following tools/features integrated inside the CVE system, and make the task completion easier.

(Q3-1) Text chat

(Q3-2) Email

(Q3-3) Wall

(Q3-4) Newsfeed

(Q3-5) Awareness/Presence features

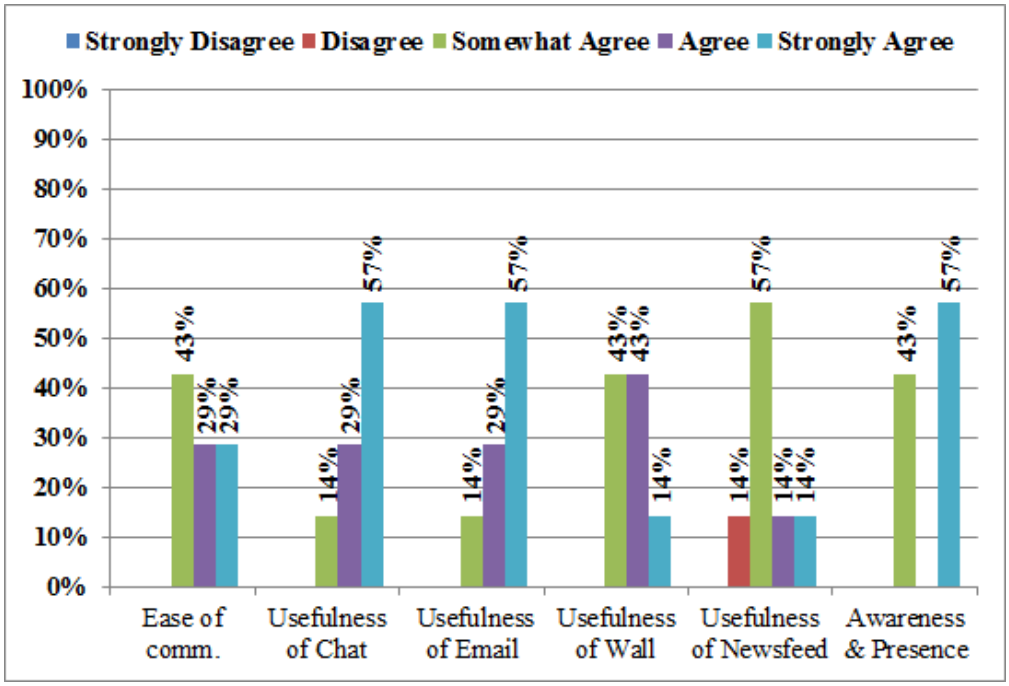

Figure 7.7 Responses to Communication Related Questions (Task #2)

It was also observed, from the responses, that all the participants found it easy to communicate with their teammates from inside the CVE environment. The responses showed that the participants found it useful to have access to the communication tools from inside the CVE. The responses showed that all the participants found it useful to have access to the text chat tool, email system, and wall/forum tool from inside the CVE environment. Also, 86% of the participants found it useful to have access to the newsfeed. All the participants found the supported awareness information useful even though some of them had no experience working with collaborative tools.

The bar graph in Figure 7.8 corresponds to the answers of the following statements:
Statement4 (Q4) It was easy to access the project file(s) and fix the bugs.
Statement5 (Q5) It was useful to be aware of who is currently working on the same file by observing the awareness information.

Statement6 (Q6) It was useful to see my team mate(s) presence and activity in the project (even when I did not have any direct benefit).

Statement7 (Q7) It was easy to ask for assistance within CVE while finishing the assigned task.

Statement8 (Q8) It was easy to get assistance within CVE while finishing the assigned task.

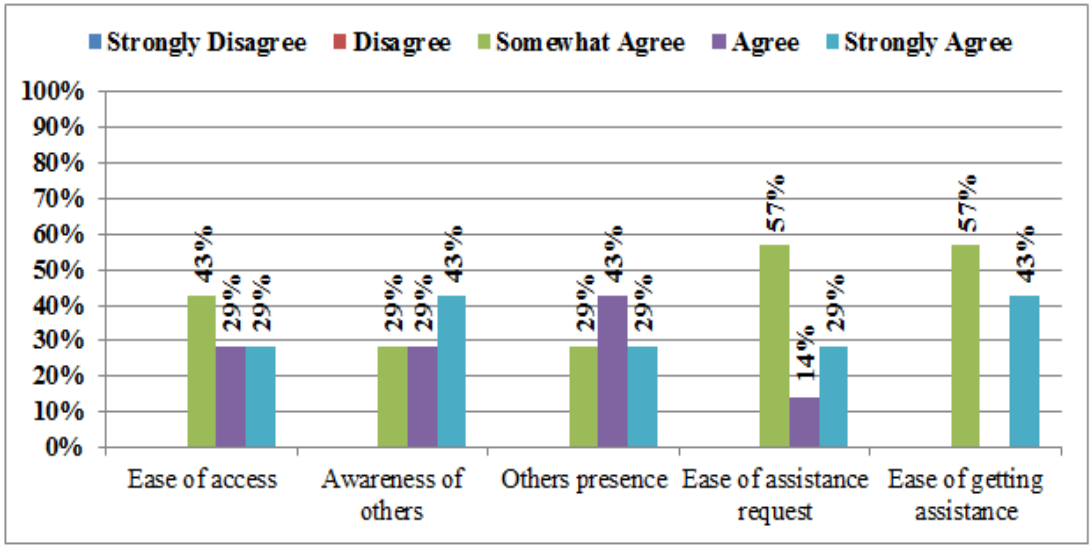

Figure 7.8 Responses to Task Related Questions (Task #2)

From the data that appears in Figure 7.8, it was observed that none of the participants found it difficult to access the project files and fix the bugs. The responses showed that the participants found the awareness information useful, and liked to be aware of their teammates' activities and presence. Also, it was observed from the data that all the participants found it easy to ask and get assistance while finishing the assigned task. Overall, the participants found it useful to be aware of their teammates' activities even when they did not have any direct benefit.

The bar graph in Figure 7.9 corresponds to the answers of the following statements:

Statement9 (Q9) Overall, it was easy for me to communicate with my teammate(s) and finish my task.

Statement10 (Q10) Overall, the CVE IDE window was distracting.

Statement11 (Q11) Overall, the supported awareness information was enough and useful.

The responses showed that only 14% of the participants found it difficult to communicate with their teammates and finish the assigned task from inside the CVE. 28% of the responses suggested that the CVE IDE window is distracting. The responses showed that 86% of the participants found the supported awareness features to be useful, and enough.

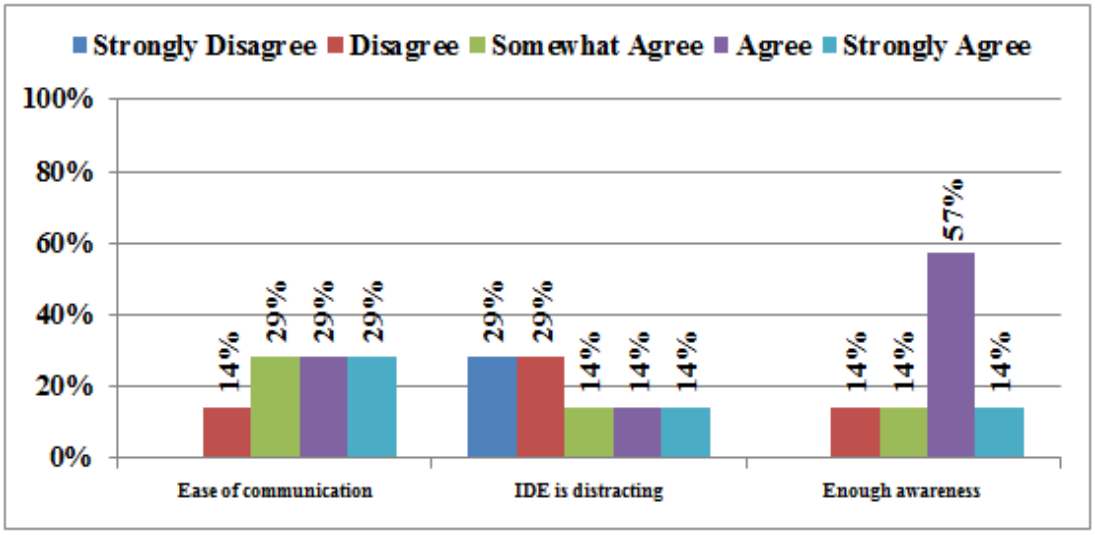

Figure 7.9 Responses to General Related Questions (Task #2)

The bar graph in Figure 7.10 corresponds to the answers to the following statements:

Statement12 (Q12) It was easy to perform and/or view the following activities:

(Q12-1) Friends request

(Q12-2) Assistance request

(Q12-3) Walls post(s)

(Q12-4) Users status

(Q12-5) Projects/Groups request

Statement13 (Q13) It was easy to notice the notifications, emails, requests, and invitation that I received from other participants.

Statement14 (Q14) Joining the software project helped me contact and request help from programming language experts.

Statement15 (Q15) I find the notification icons important features in the environment.

Statement16 (Q16) I find it useful to see my teammate's presence (away/busy/offline/online).

It is observed from the data in Figure 7.10 that none of the participants found it difficult to interact with the others; they found it easy to perform the social networking activities with their teammates. The activities included: sending requests and invitations, using communication tools (eg. Text chat, email, wall, and newsfeed), and watching their teammates' status and availability. From the responses it was appeared that 29% of the participants found it difficult to notice the supported notifications of the emails, requests, and invitations, while 86% of the participants found the notifications icons important features in the environment. All the participants showed interest in watching their teammates' presence and activity awareness, and found it very useful.

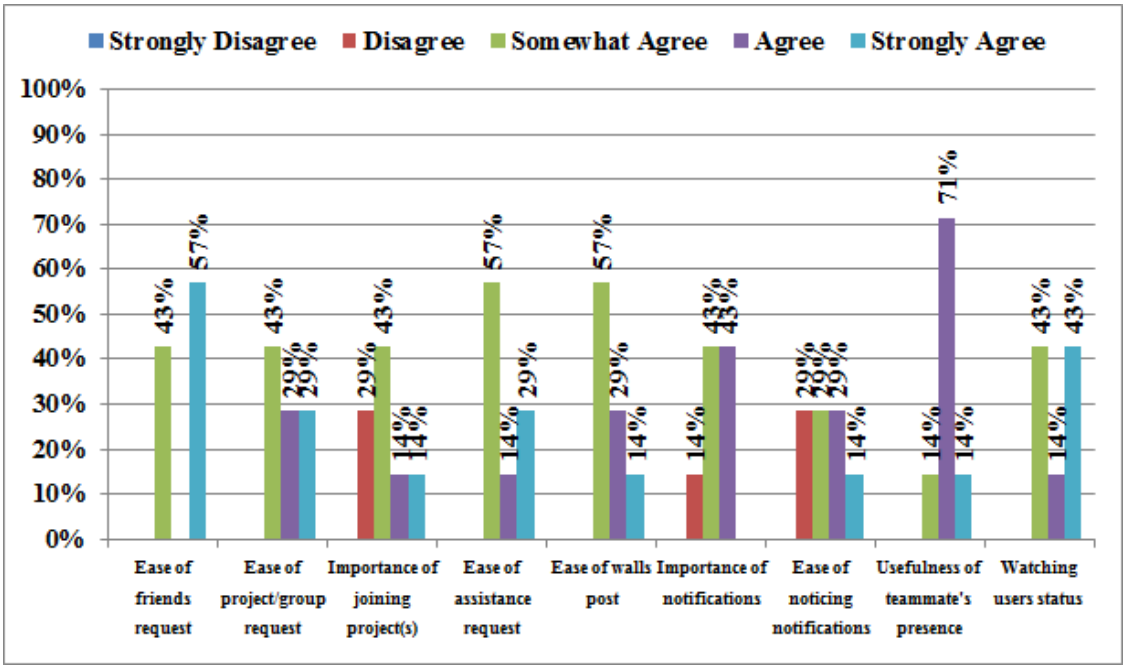

Figure 7.10 Responses to SN Related Questions (Task #2)

The bar graph in Figure 7.11 corresponds to the answers of the following statements:

Statement17 (Q17) ***Ease of deployment***. It was easy to configure the environment to initiate and/or take part in a collaborative session.

Statement18 (Q18) ***Fidelity***. The supported awareness information is correct and up-to-date.

Statement19 (Q19) ***Collaboration Awareness***. The system provided me with enough cues about my teammate's activity and presence.

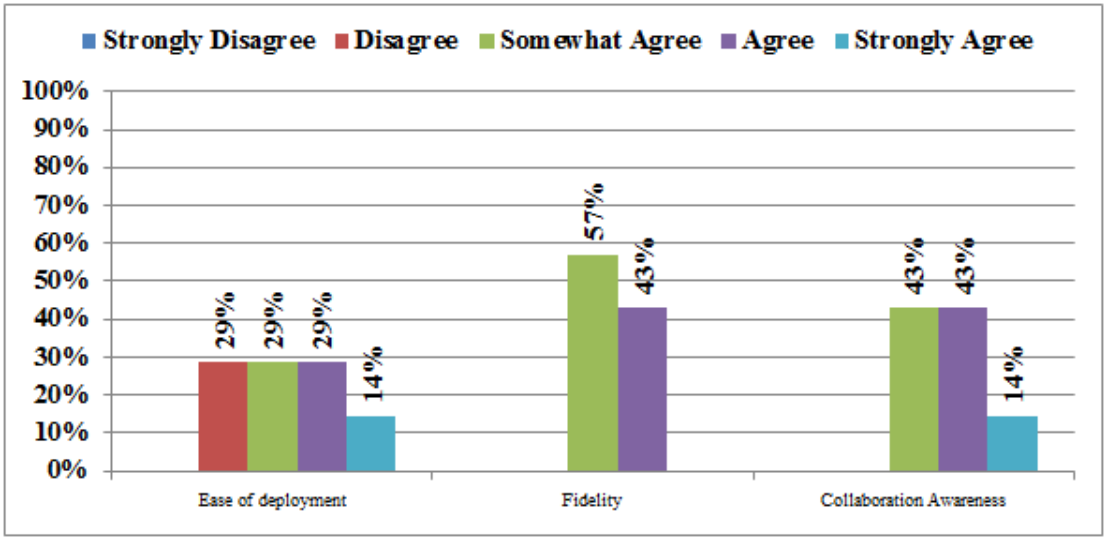

Figure 7.11 Responses to Usability Related Questions (Task #2)

The data that appears in Figure 7.11 showed that 29% of the participants found it difficult to initiate and/or take part in a collaborative session. The responses showed that all the participants found the awareness, presence, and social networking features support were up-to-date, and enough to give them cues about their teammates' activities and progress in the project. The data above tested the usability of the supported awareness and social network features depending on three factors: *deployment*, *fidelity*, and *collaboration awareness*. The responses gave an indication to the ease and usability of the environment with the integrated/supported features.

Student's general comments appear in Table 7.2. The majority of their comments were positive. Overall, the students agreed and noticed the benefits of integrating the communication tools, awareness, presence, and social networking features inside the CVE environment, and suggested some usability changes to the environment.

Other Comments
"It would be useful to have a column with time of item entry to the night of each of the server message lists at the bottom of the page to see which message is more recent and therefore more likely in effect. This would be especially good to know whether the users' agent has crashed or has been restored in the system."
"User notification of emails and friend requests has to be actively searched out. If this flashed or changed colors this would be handy."
"To enter text after selecting an action, the user must select the cursor position."

Table 7-3 Summary of other comments from participants took part in the second task.

Summary of findings

The participants recognized the importance of integrating the awareness information and social networking features inside the CVE environment, and almost all the participants agreed that the new features made collaboration and communication with others easier, and helped them collaborate with their teammates to finish their assigned tasks.

7.5 Quantitative Findings

A study of the effect of integrating the social networking feature inside the SCI development environment was implemented. Six items were selected from the questionnaire responses collected from the first two tasks.

A summary of students' responses appears in Table 7.4. The study compared participants' responses from session A (participants used SCI without tools integration) and session B (participants used SCI with tools integration). This study was performed on six items.

	Task #1 (A) (Without features integration)	Task #2 (B) (With features integration)
Student's Responses (%)	43.00	100.00
	76.67	97.20
	86.00	100.00
	29.00	100.00
	43.00	86.00
	28.00	28.00

Table 7-4 A summary of the t-test data and statistics

Figure 7.13 shows a comparison of the participants' responses.

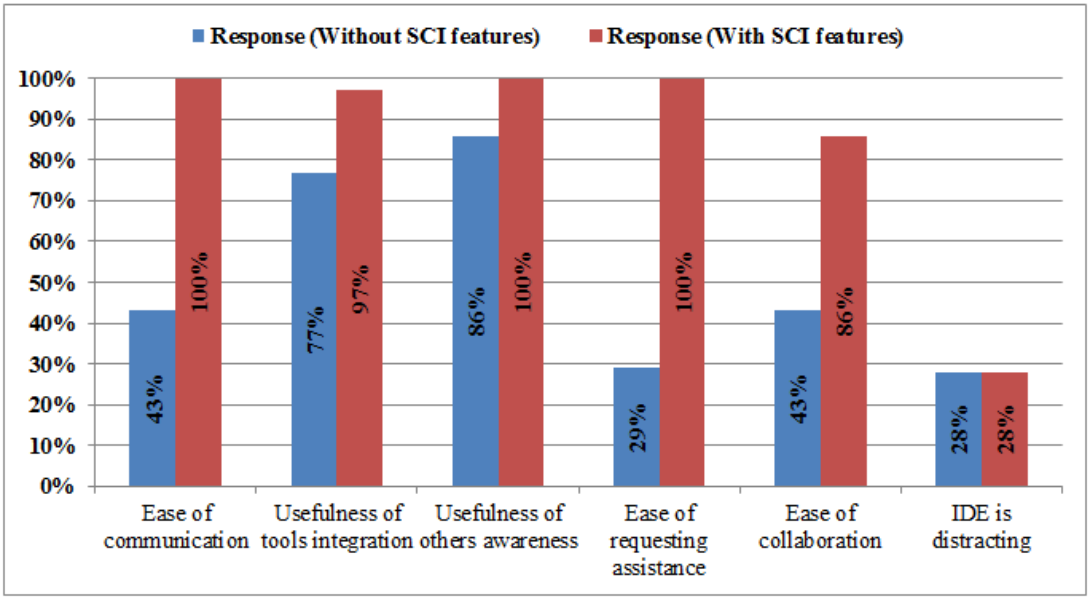

Figure 7.12 The T-Test Data Comparison

Fisher's Exact Test

$$\frac{((\boldsymbol{a}+\boldsymbol{b})!\,(\boldsymbol{c}+\boldsymbol{d})!\,(\boldsymbol{a}+\boldsymbol{c})!\,(\boldsymbol{b}+\boldsymbol{d})!)}{\boldsymbol{a}!*\boldsymbol{b}!*\boldsymbol{c}!*\boldsymbol{d}!*\boldsymbol{N}!}$$

This section shows a summary of statistics made using fisher's exact test. During this test a set of statements are taken from the survey, and the responses of the two tasks groups for those statements were compared. Following is a summary of the selected statements and the calculated statistics. The test used a predetermined alpha level of significance ($\alpha = 0.05$).

The suggested null and alternative hypotheses are as the following:

H_0: *there is no effect in integrating the social networking features inside the CDE.*

H_1: *there is an effect in integrating the social networking features inside the CDE.*

Statement2 It was easy to communicate (start an online chat/conversation, and/or send email) with my team partner.

Observed Data			
	Without	With SN	Total
Disagree	a = 4	b = 1	5
Agree	c = 3	d = 6	9
Total	7	7	14

Expected Data		
	Without	With SN
Disagree	2.33	2.67
Agree	4.67	5.33

```
> x = matrix(c(4,1,3,7),
+            nrow = 2,
+            ncol = 2)
> fisher.test(x)
```

Fisher's Exact Test:

P-value = 0.1189

Alternative hypothesis: true odds ratio is not equal to 1. 95% confidence interval lies between 0.5027782 and 521.2731861.

Odds ratio = 7.876343

Since p < a (4), the null hypothesis was rejected. There is evidence that the features integration affects the environment usability, and increases the participants' involvement.

Statement3 It was useful to have the following tools/features integrated inside the CVE system, and make the task completion easier.

Observed Data			
	Without SN	With SN	Total
Disagree	a = 2	b = 1	3
Agree	c = 5	d = 11	16
Total	7	12	19

Expected Data		
	Without SN	With SN
Disagree	1.11	1.89
Agree	5.89	10.1

```
> x = matrix(c(2,1,5,11),
+            nrow = 2,
+            ncol = 2)
> fisher.test(x)
```

Fisher's Exact Test:

P-value = 0.5232

Alternative hypothesis: true odds ratio is not equal to 1. 95% confidence interval lies between 0.1725787 and 281.9943236.

Odds ratio = 4.03901

Since p < a (2), the null hypothesis was rejected. There is evidence that the features integration affects the environment usability, and increases the participants' involvement.

Statement5 It was useful to be aware of who is currently working on the same file by observing the awareness information.

Observed Data			
	Without SN	With SN	Total
Disagree	a = 1	b = 1	2
Agree	c = 6	d = 6	12
Total	7	7	14

Expected Data		
	Without SN	With SN
Disagree	1	1
Agree	6	6

```
> x = matrix(c(1,1,6,6),
+           nrow = 2,
+           ncol = 2)
> fisher.test(x)
```

Fisher's Exact Test:

P-value = 1

Alternative hypothesis: true odds ratio is not equal to 1. 95% confidence interval lies between 0.01094678 and 91.35109674.

Odds ratio = 1

Since p <= a (1), the null hypothesis was rejected. There is evidence that the features integration affects the environment usability, and increases the participants' involvement.

Statement7 It was easy to ask for assistance within CVE while finishing the assigned task.

Observed Data			
	Without SN	With SN	Total
Disagree	a = 5	b = 1	6
Agree	c = 2	d = 6	8
Total	7	7	14

Expected Data		
	Without SN	With SN
Disagree	3	3
Agree	4	4

```
> x = matrix(c(5,1,2,6),
+            nrow = 2,
+            ncol = 2)
> fisher.test(x)
```

Fisher's Exact Test:

P-value = 0.1026

Alternative hypothesis: true odds ratio is not equal to 1. 95% confidence interval lies between 0.7203391 and 817.9820107.

Odds ratio = 11.71745

Since p <= a (5), the null hypothesis was rejected. There is evidence that the features integration affects the environment usability, and increases the participants' involvement.

Statement9 Overall, it was easy for me to communicate with my teammate(s) and finish my task.

Observed Data			
	Without SN	With SN	Total
Disagree	a = 4	a = 1	5
Agree	a = 3	a = 6	9
Total	7	7	14

Expected Data		
	Without SN	With SN
Disagree	2.5	2.5
Agree	4.5	4.5

```
> x = matrix(c(4,1,3,6),
+             nrow = 2,
+             ncol = 2)
> fisher.test(x)
```

Fisher's Exact Test:

P-value = 0.2657

Alternative hypothesis: true odds ratio is not equal to 1. 95% confidence interval lies between 0.4199818 and 456.2874326.

Odds ratio = 6.788517

Since p <= a (4), the null hypothesis was rejected. There is evidence that the features integration affects the environment usability, and increases the participants' involvement.

Statement10 Overall, the CVE IDE window was distracting.

Observed Data			
	Without SN	With SN	Total
Disagree	a = 5	b = 4	9
Agree	c = 2	d = 3	5
Total	7	7	14

Expected Data = (rTotal * cTotal) / Total		
	Without SN	With SN
Disagree	4.5	4.5
Agree	2.5	2.5

```
> x = matrix(c(5,4,2,3),
+             nrow = 2,
+             ncol = 2)
> fisher.test(x)
```

Fisher's Exact Test:

P-value = 1

Alternative hypothesis: true odds ratio is not equal to 1. 95% confidence interval lies between 0.1295684 and 32.1159362.

Odds ratio = 1.79176

Since p <= a (5), the null hypothesis was rejected. There is evidence that the features integration affects the environment usability, and increases the participants' involvement.

7.6 Summary

The findings of this study support the social development environment usability and usefulness in general. Comparing the responses from the first two tasks showed that integrating the

communication and social networking features inside SCI makes it an effective and usable classroom programming environment.

Responses from the first task where the participants used external tools showed that only 43% of the participants found it easy to communicate, while 100% of the respondents who used the SCI tools expressed that it was easy to communicate with their teammates and request assistance. Also, they found it useful to watch and be aware of their teammates' activities, presence, and progress in their development projects; 100% of the participants expressed satisfaction of the supported awareness information, and found it enough to give them cues about their teammates' activities.

However, the student's responses and the quantitative findings suggested the benefit and usefulness of integrating the communication tools, awareness information, presence, and social networking features inside the collaborative development environment. The findings suggested that there is evidence that the features integration increases involvement and affects the environment usability.

Chapter 8
Evaluation

This chapter assesses the SCI framework and supporting features. This evaluation is made in order to investigate the benefits of the SCI system features and their role in increasing awareness among distributed developers.

In Section 8.1, the evaluation for SCI is introduced. Section 8.2 presents details about the log files that are used by the SCI system to collect data. Section 8.3 provides a detailed evaluation of the dissertation's hypotheses.

8.1 Introduction

As mentioned earlier in section 5.3, the research presented in this dissertation tests two hypotheses. The first hypothesis is that integrating social networking features inside a CDE makes the CDE an effective/usable classroom programming environment. The second hypothesis is that the combination of social network and IDE with a virtual environment will provide the collaborative IDE with the needed online presence and awareness information. This dissertation includes a case study and evaluation to test the correctness of those hypotheses.

For the SCI framework, group studies were valuable to explore the effectiveness of integrating the proposed features on users' collaborative effort while accomplishing particular categories of tasks. The SCI framework used a logging facility to record data about users, tools, events such as artifact changes and access to specific features that result from user activity on the project artifacts. Section 8.2 presents some of the log files that were used by the SCI system to collect data.

Data for the study was collected in Summer 2011 on the Computer Science I (CS120) course. The class consisted of seven mostly freshman computer science majors. The study was designed as a one-factor randomized paired comparison. The study consisted of three different tasks, details about the tasks appears in Section 7.2.2. A summary of the total number of participants involved in each task, and the total number of participants responded to the surveys handed to them after each task.

8.2 Log files

The SCI system stores data in different log files. SCI creates log files with information about users', groups', projects', and artifacts' activities. This information was reviewed to monitor the users' activity, study the system features, and test the proposed hypotheses.

For this dissertation, the SCI system was instrumented to produce event logs including the following classes of events:

Event	Description
Access	A user accessed a project file.
Add	A user requested a friendship, or added a group, project, file, or feed.
Debug	A user has started or participated in a debugging session.
Delete	A user deleted a group, a project, or a feed.
Edit	A user has edited one of the project's files or the shared document.
Join	A user joined a group or a project.
View	A user viewed a profile or a feed post.

Information about these events is recorded in different log files. These log files are stored in different places in the server. They all store information in line-oriented human readable format. All fields are space separated fields. This section presents some of these files.

— ***dat/notification_response.log:*** this log provides an indication on how quickly the users responded to the awareness notifications. This helped figure out to what extent the awareness notifications were helpful. The following is the format for this log data:

Format				
Type	**From(User)**	**To(User)**	**Sent (Time)**	**Responded (Time)**
All the above five fields are space separated.				
Summary				
Type	Notification type is one of (Project/Join, Group/Join, Group/Message, Session/Edit, Session/TakeTurn, Friend/Add)			
From	User who sent the notification.			
To	User the notification is sent to.			
Sent	Time the notification is sent (seconds). This is calculated by converting the time &clock into a number of seconds past midnight.			
Responded	The time difference between sending the notification and responding to it (seconds).			

Table 8.1 shows information about all kinds of notifications generated by the participants during the case study. The information includes: type, times repeated, total time to respond, and the average response time.

Notification	# Repeated	Total (min)	Avg. Response Time (min)
Session/Edit	18	115.5	6.4
Group/Join	2	39.3	19.7
Friend/Add	9	116.9	13.0
Avg.			**9.37**
Median			**2.5**

Table 8-1 Notifications Generated by the Study Participants

From the data obtained in this log file it appears that the average response time for the notification was 9.37 minutes, and the median is 2.15 minutes. 2.5 minutes for a response time is above the expected response time. These values give an indication that the presented facebook-style notifications need to be larger and more noticeable, and that SCI requires certain training/familiarization.

— ***dat/partner_stats.log:*** this log file presents projects' and groups' partners. It also presents the number of times interactions of each type have been made. The data collected in this file was compared with the total number of interactions that have been made from inside the SCI environment, and what percentages of those interactions have been made from groups' and projects' partners. This gave an indication of to what extent partners collaborated and communicated in the SCI environment. This log data is of the format:

Format			
Name	**Type**	**Event_Type**	**Count**
All the above four fields are space separated.			
Summary			
Name	Name of interaction is one of (Friend Accept, Profile View, or Session Edit/Debug)		
Type	Type of partnership is either (G: Group) or (P: Project).		
Event_Type	Event type is one of (Edit, Debug, View, or Update)		
Count	# of times this interaction is made		

Table 8.2 shows information about the interactions occurred between the participants during the case study. The information includes: partnership relation (F, G, P), number of interactions occurred between each, and the percentage of interactions occurred between each partnership relation (partners' interactions compared to the total number of interactions).

Relation	Partners' Interactions (#)	Partners' Interactions (#) / (Total Interactions)
P: Project	101	0.30
G:Group	75	0.22
F:Friend	14	0.04

Table 8-2 Interactions Occurred between Partners

Results appear Table 8.2 shows that 56% of the total amount of interactions was interactions that occurred between projects' partners, groups' partners, and friends. On the other hand, 44% of the interactions occurred between non-partner class members, or participants who did not create a friendship relation with certain class members that they are in fact friendly with. This gives an indication to the noticeable social activities that occurred inside the environment, and helped to form the social bonds between the participants.

— ***dat/stats.log:*** This log file collected data about all events, and the number of times an event occurred. The following is the format for this log data:

Format	
Event_Type	**Count**
Summary	
Event_Type	Event type is one of (Project/Add, Group/Add, Session/Edit, …)
Count	# of times this interaction is made

Table 8.3 presents a list of all the activities that occurred between the group study participants. It shows the times each activity occurred between partners, and the total times it occurred.

Activity	Times Occurred by Partners	Total Times Occurred	Avg.
File/Edit	51	65	0.78
Profile/View	12	18	0.66
Friend/Add	8	14	0.57

Table 8-3 List of the Activities Occurred between the Participants

Observing the above table, it appears that the partners' interactions were focused around three activities. The first activity, *File/Edit*, shows that the participants spent time editing and collaborating with others to finish and debug project files. The second activity, *Profile/View*, shows that the participants were interested in watching who is around, and what their activities are. The third activity, Friend/Add, gives an indication that they were trying to socialize and form social bonds with other while finishing their assigned tasks.

— ***dat/users/[userID]/stats.log:*** This log file collected data about the events made by the user (userID), and the number of times an event was made. The following is the format for this log data:

Format	
Event_Type	**Count**
Summary	
Event_Type	Event type is one of (Project/Add, Group/Add, Session/Edit, …)
Count	# of times this interaction is made

Event	Times Occurred
Group/Add	1
Feed/View	2
Profile/View	9
Friend/Add	10
Project/Add	10
Session/Edit	11
Group/Join	11
Session/Debug	28
Feed/Add	29
Project/Access	38
File/Open	38

Table 8-4 Events Generated by the Participants

The data that appears in Table 8.4 shows information about the events generated by the group study participants. It shows that the participants tried to socialize with each other. They created both friendship and/or partnership relations with each other. They also tried to watch their teammate's availability and progress by watching their profiles. They also tried to collaborate by: 1) joining projects and groups; 2) initiating editing and debugging sessions; and 3) posting comments to the wall or replying to others posts.

— ***dat/communication_stats.log:*** This log file collected data about the communications made by the user (userID), and the number of times each communication tool had been used. The following is the format for this log data:

Format	
cType	**Count**
Summary	
cType	Communication type is one of (Chat, Email, Feed, and Wall)
Count	# of times this communication tool has been used by the users.

The data in Table 8.5 shows communication tools, and the total number of times each tool was used during the case study. This supports the data generated from the group study responses regarding the communication tools. It is showing that users mostly used and liked communicating with others using text chat (63% of the total communications that occurred were chat messages).

Communication Type	Times Used
Email	4
Wall	6
Feed (mostly system generated)	77
Chat	145
System Chat	490

Table 8-5 Used Communication Tools

— ***dat/sessions/acvtivities.log:*** This log file recorded the activities for all the sessions created from inside the SCI environment. This log file was used to store records of the shared file, partner(s), start time, time spent, and the session output when running the program.

Format						
Fname	**sUser**	**Start(Time)**	**End(Time)**	**Action**	**cOutput**	**Spent(Time)**
All the above five fields are separated using a special string " --- ".						
Summary						
Fname	Shared file name.					
sUser	Name of user who created or joined the session.					
Start(Time)	Time the session started (seconds). This is calculated by converting the time &clock into a number of seconds past midnight.					
End(Time)	Time the session ended (seconds). This is calculated by converting the time &clock into a number of seconds past midnight.					
Action	Owner : Created, Partner: Joined					
Spent(Time)	The time difference between starting and ending the session (seconds).					

Table 8.6 shows a list of the collaborative sessions occurred between the participants. This table shows the name of the session, and the time partners spent working on their assigned task.

Session Name	Time Spent (min)
Untitled0	41
PIMath.cpp	18
dell.cpp	72
teamfemale.cpp	98
csgeeks.cpp	20

Table 8-6 Collaborative Sessions and Time Spent by Each Group

— ***dat/projects/[projName]/objName_spent_time.log:*** This log file recorded the total time each member spent working in a project artefact. The *objName* can be either *file_name* or *project_name*. This log file records consist of two fields: *UserID* and *Time_Spent*.

Project	User	Time Spent (min)
projectA	A	11
	B	24
projectB	C	0
projectC	D	74
	E	42
projectE	F	0

Table 8-7 Shows the Total Time Each Participant Spent Working on the Project Artefacts

Table 8.6 shows a list of software development projects and the time each participant spent working on the projects artefacts.

— ***dat/users/[userID]/interactions_in3D.log:*** This log file recorded the total number of times a user interacted with the 3D virtual world. These interactions include: changing locations, viewing activities, viewing changes history, teleporting to different rooms, and greeting other users. This log file records consist of two fields: *Interaction_Type* and *Count*.

The data from this log file showed that the participants changed their location inside the 3D and that they were immersed in the environment. They tried to teleport from room to room, and interacted with others avatars by: sending greetings, checking their history, and view their activities. Table 8.7 shows a list of the activities occurred between the participants.

Interaction	Times Occurred
Greeting	12
Door Open/Close	108
Teleport	64
Change Location	909
View Activity	3
History	7

Table 8-8 3D Interactions Occurred between the Group Study Participants.

— ***dat/users/[userID]/locations.log:*** This log file recorded information about changing locations and moving between the environment rooms (navigation). It recorded the total number of times a user moved between locations, and the number of times each location was visited. This log file records consist of two fields: *Location* and *Count*.

User	*Times Asked to Change Location*	*# of Times Changed Location*	*Difference*
A	7	86	79
B		91	84
C		93	86
D		95	88
E		107	100
F		112	105
G		114	107
H		170	163

Table 8-9 Total Number of Navigations Made by Each Participant

During the study each participants was asked to visit the closest location to his/her room (~5 locations) and to teleport to two locations (~7 locations the number of locations was asked to visit each). The data in Table 8.9 shows the number of navigations occurred by each participant. The data from this log file showed that the participants navigated the environment and changed different locations inside the 3D, and that gives indication that they were immersed and enjoyed the 3D world experience. Table 8.9 shows anonymous names for participants, and how many times they changed locations.

— ***dat/users/[userID]/availability.log:*** This log file records the login and logout entries for each user in the system (a log file for each user). This log file is used from inside SCI to show the availability patterns for each user, and what time are they likely to be available. The data extracted from this file is shown in a 2D window that resides in the user's profile (see Figure 5.6).

— ***dat/availability.log:*** This log file recorded the availability table entries for all the system users (a single log file for all the users). This log file was used from inside SCI to show the availability patterns for all the users, and how many logins happened each day of the week, and what hours show more logins.

— ***dat/users/[userID]/ide_3d_activity.log*** *and* ***dat/users/[userID]/ide_3d_usag.log:*** The *ide_3d_activity.log* log file recorded the times a user entered, and left a specific mode when switching between the system tabs. The available modes are: 3D, Editor, Profile, and Wall. *ide_3d_usag.log* analyses the records from the first log file, and generates a user's usage report at the end of each week. The following is the format for those data log files:

Format (*ide_3d_activity.log*)		
Day	**Time**	**In/Out Mode**
All the above three fields are space separated.		
Summary		
Day	The week day	
Time	Time in the following format *hh:mm:ss*.	
In/Out Mode	Mode is one of the following four mode: 3D, Editor, Profile, Wall	
Examples		
Saturday 17:48:26 In 3D Saturday 17:55:31 Out 3D Saturday 17:55:31 In Editor Saturday 18:01:34 Out Editor		

Format (*ide_3d_usage.log*)			
Date: Shows the date the records added.			
Day	**IDE (min)**	**3D (min)**	**AFK (min)**
The above four fields are formatted in a table and separated using spaces.			
Summary			
Day	The week day		
IDE (min)	Time a user spent working in the IDE.		
3D (min)	Time a user spent working in the 3D world.		
SN (min)	Time a user spent using the Social Network features.		
AFK (min)	Time a user spent in the environment and away from the keyboard.		
Example			

Saturday, May 28, 2011 12:00 pm

Day	IDE (min)	3D (min)	SN (min)	AFK (min)
Sunday	93	6	0	7.6
Monday	27	144	0	7.23
Tuesday	907	1262	0	17.91
Wednesday	441	270	0	12.75
Thursday	0	0	0	0.0
Friday	20	111	0	3.35
Saturday	0	0	0	0.0

The logging facility within SCI provides raw information enabling valuable insight as to how software projects develop over time. In addition to the fine-grained modification records (modifications made to the software development project artifacts and shared files) supported by SCI and similar to the history logs of CVS and Subversion, SCI provides the history of changes with information about: changed files, who made the changes, the impact of each individual change, and timestamp.

8.3 Methodology

8.3.1 Evaluating Awareness and Presence Integration

"Integrating social networking features inside a CDE makes the CDE an effective/usable classroom programming environment."

In evaluating the first hypothesis, we conducted experiments over a period of one semester (approximately two months during the Summer 2011 semester). Participants were students from the "CS120: Computer Science" class. Our goals were to study the effect of integrating the social and awareness features SCI's usability and usefulness, and to validate the claimed hypothesis. The study consisted of three separate tasks (see Section 7.2.2).

This dissertation used both qualitative and quantitative methods. Qualitative analysis deals with non-quantified data that typically was obtained through questionnaires where users are required to fill a questionnaire after finishing each task (see Section 7.4). Quantitative analysis studies used two statistics analyses: *T-Test* and *Fisher's Exact Test* (see Section 7.5).

8.3.1.1 Summary of findings

The presented experiments (see Section 7.2.2) aimed to test the usability and usefulness of integrating social networking feature inside SCI as a classroom programming environment. In

spite of a noticeable effect of the SN integration, the instability of the CVE server software has put a negative effect on the results collected during the group study.

Our hypothesis that the social networking features add value, usability, and usefulness to the environment was confirmed by both the qualitative and quantitative analyses presented in chapter 7. The group study and the quantitative tests showed evidence that the features integration affects the environment usability.

8.3.2 Evaluating 3D Virtual Environment Integration

"The combination of social networking with a virtual environment will provide the collaborative IDE with the online presence and awareness information that increases use and usefulness of collaboration tools."

Team members need to evaluate each other, determine who has what knowledge, and who they can ask for help on specific technical topics. In an environment such as SCI, team members need to feel that they belong to the team, and need to feel the other team members' presence in order to achieve their goals [120].

The second hypothesis suggests that if improving distributed teams' collaborative-effort requires more social contact, contact in a 3D virtual environment might provide it. This discussion is supported both by literature [121, 122, 123] as well as observation of multiplayer online games like World of Warcraft and virtual worlds like Second Life. In such environments, users who never meet face-to-face are able to meet, form groups and make plans to perform complex tasks within the virtual world. This creates a belief that distributed development could be improved using such worlds, either in the communication or the development environment.

Validating the second hypothesis was achieved by testing the effectiveness of the virtual world and the avatar presence in increasing the sense of belonging to the group, and to what degree developers feel socially present, and perceive each other as real in virtual interactions.

In this section the presence and feeling of belonging was tested using both subjective and behavioral measures. Depending on the subjective nature of the presence [124], it was logical to measure presence relying on the SCI system users' self-reported sense of belonging. Measuring the subjective feeling of presence in such an environment is done by conducting surveys and questionnaires on the system users.

To help mitigate the subjective nature of the questionnaire, this dissertation used another approach which is a behavioral measure. The belongingness was tested by measuring the users'

responses and interactions that were produced automatically while using the system, and related to being present in the environment.

The major classes of interaction in collaborative virtual environments are navigation, selection, manipulation, system commands and symbolic input [125]. In order to validate the suggested hypothesis, some responses (interactions) were examined as a possible measure of presence; the interactions used are mainly *navigation*, *selection*, *system commands*, and *symbolic input* such as : changing location and moving between the environment rooms (*navigation*), using the teleport from inside the 3D to move between different projects' virtual rooms (*navigation*), opening/closing a room door (*selection*), clicking an avatar or sending a greeting (*selection and symbolic input*), viewing a summary of projects' users and accessing others' history from inside the 3D (*system commands*). The premise is that the more users repeat the process, the more they were demonstrating a sense of belonging in the 3D environment, and the more real the environment was to them.

Usability:

Experience from many different projects has shown that different people encounter different usability problems. Therefore, it is possible to improve the effectiveness of the tools significantly by involving multiple participants.

Evaluating usability is an important factor to the success of the software development, and to ignore evaluation is to risk failure. Evaluating the 3D environment usability can include metrics. Bowman et al. [125] categorizes metrics as:

- Task performance metrics such as speed and accuracy.
- System performance metrics. A major example is the latency issues that affect the user's experience while collaborating.
- User preference metrics.

For the purpose of this study, user preference metrics were used. These metrics refer to subjective awareness and immersion of the system by the user, such as the ease of use, ease of learning, and support functionality that are highly valued by the participants/users [126]. Such issues that are correlated with subjects' perception were commonly evaluated with questionnaires that indicate the user's experiences of presence and user's comfort issues as an example.

8.3.3 Experiment

As mentioned earlier in Section 7.2.2, the group study consisted of three separate tasks. The third study focused around performing social activities related to the 3D world and SN working. Section 8.3.4 presents a discussion of the data collected during the third task.

8.3.4 Results

After performing the experiments, the collected data were subjected to qualitative and quantitative analyses. Qualitative analysis describes the data that has been obtained through questionnaires and surveys. Quantitative analysis describes and compares numeric data collected by the system log files. This section describes both of the qualitative and quantitative results.

Qualitative Analysis

After finishing the task, the participants were given a survey to complete and return anonymously. The bar graph in Figure 8.1 corresponds to the answers of 3D related statements that were asked in the survey.

Statement1 (Q1) I have had experience in using 3D virtual environments before the session.
Statement2 (Q2) The 3D world brings a real world value to the collaboration.
Statement3 (Q3) The use of 3D virtual environment and avatars creates a sense of presence of others.
Statement4 (Q4) Having visual representation of others inside the 3D world is sufficient to create a high sense of presence.
Statement5 (Q5) Interaction and collaboration is needed to create a high sense of presence.
Statement6 (Q6) The sense of my teammate's presence increased with the interaction and collaboration inside the 3D world.
Statement7 (Q7) In a virtual environment, you can see other people's avatar moving around, and you can see what they are doing. Do you think that helps you to interact and be more social than if it would be chat-only or web-based environment?

It was observed from the data shown in Figure 8.1 that only 50% of the participants had experience in using a 3D virtual environment before the session. All the participants agreed that the 3D world brings a real world value to the collaboration. Also, they expressed that the use of 3D worlds and avatars made them feel a sense of others' presence in the environment. 75% of the responses indicate that having the avatars visual representation within the 3D world was enough to make them feel the presence of their teammates. The responses showed that the interaction and collaboration within the 3D world were important factors to create the feeling of belongingness

between the teammates, and that their sense of presence increase by the interaction and collaboration. They also expressed the usability of the 3D interactions and that it helps the interaction and socializing more than face-to-face interactions.

The bar graph in Figure 8.2 corresponds to the answers of the following statements:

Statement8 (Q8) The CVE environment improved my team productivity, and made the task

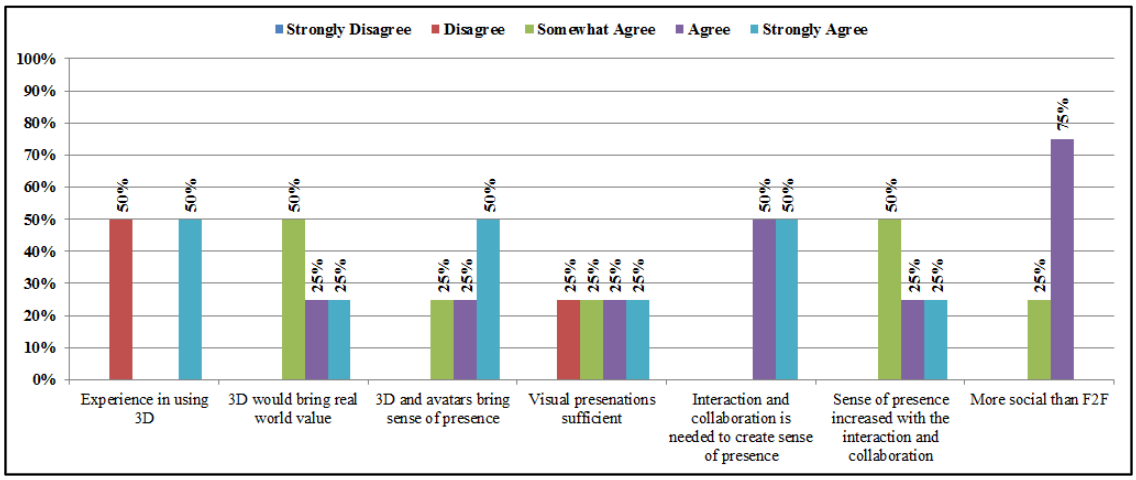

Figure 8.1 3D Related Statement's Responses

enjoyable.

Statement9 (Q9) I find CVE easy to learn.

Statement10 (Q10) I find CVE to be a useful environment.

Statement11(Q11) Rate your satisfaction with the collaboration outcome.

Statement12(Q12) Rate your satisfaction with the collaboration process.

Statement13(Q13) Other than the instability of the CVE server software,

(Q13-3) The supported communication and collaboration tools are enough to provide a productive environment.

(Q13-4) The supported awareness and presence information is enough and adequate.

(Q13-5) Rate your willingness/motivation to use the SCI environment for collaboration tasks again.

The data collected from the above statements responses showed the usefulness and usability of the SCI development environment. All participants agreed that the CVE is useful and easy to learn. 75% of the respondents expressed satisfaction of both the collaboration process and outcome, and claimed that the environment helped to increase their team productivity and made the task more enjoyable. 75% of the participants found the integrated features (communication

and collaboration tools, awareness, presence, and social network features) were enough and adequate.

The bar graph in Figure 8.3 corresponds to the answers of the following statements:

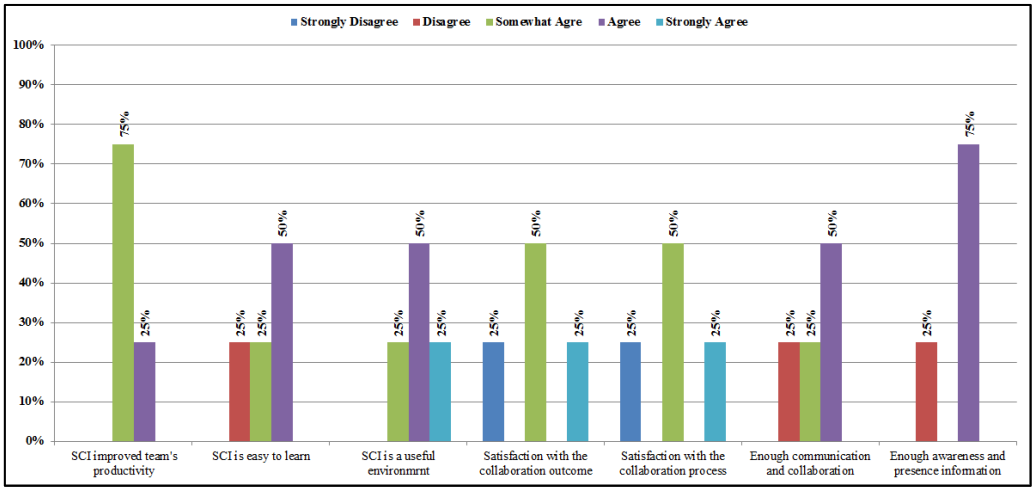

Figure 8.2 General Related Statement's Responses

(Q13-1) I find that CVE is a suitable classroom programming environment?

(Q13-2) I am willing / motivated to use the SCI environment for collaboration tasks again?

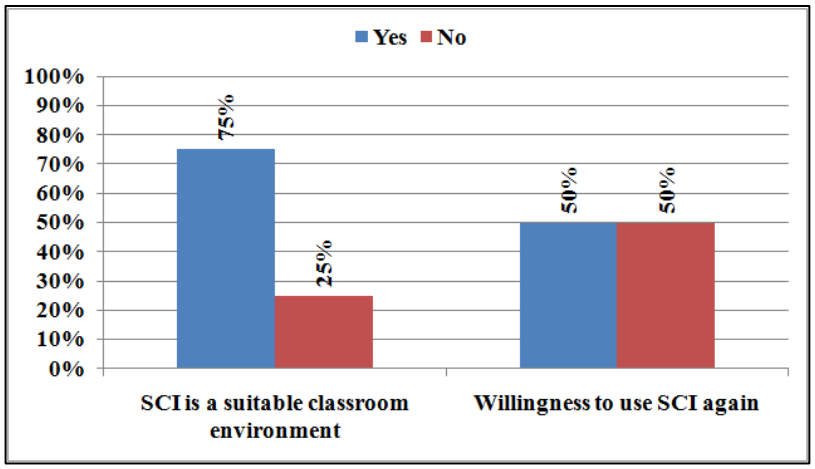

Figure 8.3 Willingness/Motivation Related Statement's Responses

The responses showed that 75% of the participants showed satisfaction and agreed that SCI is a suitable classroom environment. 50% of the participants found SCI enjoyable, and expressed willingness to use it again for collaboration tasks. On the other hand, 50% expressed unwillingness to use it again, and that is very probably related to server instability. Also they

found difficulty in using text chat for participants who do not type fast, and suggested to integrate a voice chat system.

8.3.4.1 Quantitative Results

Comparing the numeric data obtained from the log files collected by the system during the tasks, it appeared that the participants were immersed in the 3D world, and they spent time wandering inside the 3D environment. The data in Table 8.6 shows a list of the interactions the participants made while in the 3D world. Table 8.8 shows a summary of these interactions.

	Navigation	Selection	System Commands	Symbolic Input
Times Occurred	973	108	10	12
%	88%	10%	1%	1%

Table 8-10 3D Interactions Statistics

Also it appeared that 65% of the total interactions were interaction occurred between projects' partners, groups' partners, and friends, and 63% of the total communications that occurred between the participants were initiated by text chat.

8.3.4.2 Summary of findings

The presented experiment (the third task) aimed to get results to the question of whether 3D virtual environments can add real value to the SCI environment and online collaboration. In spite of a noticeable effect of the 3D integration, the instability of the CVE server software had a negative impact on the results of the 3D CVE participants' responses, satisfaction, motivation or willingness to use the CVE.

Our hypothesis that the 3D world integration, being virtually embodied in a configurable 3D collaborative environment, provides the collaborative IDE with online presence, and increases the usability and usefulness of the collaborative tools was confirmed by the groups study survey responses, and the data collected from the system log files.

Part V

Conclusions and Future Work

Chapter 9
Conclusions and Future Work

This chapter summarizes the main results of this research, discusses their significance, and offers suggestions for future work.

9.1 Conclusions

Social support for software development is an important emerging field of research. Conventional single-user tools are not able to provide the needed environment for smooth collaboration between distributed developers due to the size and complexity of today's development projects. CDE tools that support and provide project artifacts' updates in real time have the potential to raise the level of communication, and coordination between distributed developers.

Most current CDEs have inherent limitations, including little support for awareness and online presence, missing social networking features, and weak support for source code repositories' features. The multitude of tools increases the friction that results from switching among different tools.

This dissertation presented two primary contributions. The first contribution is ICI (Idaho Collaborative IDE), which allows developers to share, in real-time, the process of editing, and supports real-time collaborative compiling, linking, running, and debugging sessions, and provides an environment where developers can communicate easily; all from within the same tool. ICI combines a synchronous collaborative program editor and a real-time collaborative debugger within a 3D multi-user virtual environment.

The second primary contribution is a social collaborative development environment that addresses the mentioned problems above, and eliminates several current limitations. SCI combines both synchronous and asynchronous features. It is composed of a presence and activity awareness information component, an integrated development environment, and collaboration tools that all reside within a single environment, which serves as a virtual collaborative development environment. The merger of these tools increases their benefit to the development community.

SCI supports development in mainstream languages, such as C, C++, and Java. Depending on the obtained research results, augmenting related and specific awareness information, online

presence, and social network features within a single environment makes SCI a usable and useful development environment for developers and teams working in a distributed environment.

The combination of both the synchronous and asynchronous features makes the tools more usable. In general, the asynchronous features support the usability of the synchronous tools, and both categories complement each other. The combination improves the traditionally used collaborative tools with features and techniques, which may include:

— Online presence that introduces the developers with a sense in which they feel that other developers are socially present in the development environment.
— Activity awareness information that supports the developers with a way to understand who is working with them, what they are doing, and how their own actions interact with theirs.
— Social network features that introduce developers with a better awareness of the surrounded development community and their projects' development activity.

SCI is a social real-time collaborative IDE that helps software development teams and makes communication, and collaboration effective and more productive. Future work directions aim to solve the limitations presented in Section 9.2, and make SCI as complete and efficient as possible and attractive to the software development community. News about the project are available at http://cve.sf.net/socialide.

9.2 Limitations

Like any other distributed groupware system, SCI's implementation poses technical challenges that need to be addressed, especially as SCI becomes more feature and functionality rich. This section touches on a handful of these challenges, some of which can be mitigated using known methods, while others are open issues.

9.2.1 Concurrent Editing

Collaborative editing applications have been designed to support groups of users and allow them to edit their documents simultaneously. Using such an application, many challenges may arise, ranging from the technical challenges of *maintaining consistency* to the *social challenges of supporting group activities*.

Due to network latency challenges, simultaneous editing by multiple users can be either slow or else potentially inconsistent edits must be reconciled. The challenge is to figure out exactly how to make the changes appear in the document, which were created in remote versions of the

document that never existed locally, and make sure that no conflict occurs with the client's local document edits. SCI dodges the communication lag issue by having just one version of the program and allows the clients to edit the program one at a time.

Concurrency is an important requirement for real-time collaborative editing systems, i.e. all participants should be able to concurrently edit any piece of the shared code [127]. Approaches such as turn-taking [128] and locking [129] are used by the software development community.

For the scenarios envisioned in SCI's requirements, collaborative editing does not include concurrent editing of the same file by multiple users. Concurrency control was added to the editor widget by implementing a way to lock the developers who do not have permission, and to guarantee that users access the shared workspace one at a time. To address the issue of maintaining consistency, in its current state, SCI is forcing the main client window to stay the same size and don't allow users to resize the window.

A future work would be to integrate the operational transformation approach for maintaining consistency of the copies of the shared document in real-time collaborative editing systems [130]. Various operational transformation algorithms have been proposed, such as dOPT [131], adOPTed [132], and SOCT3 and SOCT4 [133].

9.2.2 Lack of social communication and coordination

The interactions between distributed team members are associated with challenges and disadvantages due to the lack of the rich social communication and coordination that is only possible when team members are co-located.

Being aware of these challenges is important for evaluating the effectiveness of a groupware system. Following are some of the problems that are facing distributed teams [134]:

1. Distributed teams lose awareness of social interactions and other members' activities. They are limited in the spontaneous interactions that may occur with other team members. Also, finding collaborators is a very important direction that SCI has not done enough to address. Trusting people is more important in such environments than trusting the artifacts. Integrating trust with a matching expertise needs more research, and there is promise in exploring this area.
2. In distributed environments, communication relies on lower-bandwidth tools and applications such as: phone and email. Also, they are constrained by transaction delays in relative to face-to-face interactions and encounters. In general, there is less interaction between developers in distributed environments because they mostly do not meet with each other in person.

Sometimes developers have problems getting timely advice and potential help from other developers.

3. In order for the distributed teams to manage their project artifacts, they face numerous challenges, including version control and user access management.
4. SCI doesn't support any links or communications with other tools, social networks, or software repositories. Section 9.4 shows some possible areas of interoperability.

9.2.3 Case Study Limitations

The small sample size is a primary limitation of the study introduced in this dissertation that limited the generalizability of the results. With a larger sample size, the results would have been different and more generalized. Further study with a large sample in different population (eg. students who are taking advanced computer and software engineering tasks) would be required.

9.3 Directions for Future Work

This research has brought to light several areas that require more exploration.

9.3.1 Feature Additions and Improvements

Aside from addressing the limitations discussed in Section 9.2, several opportunities for improving the SCI research project exist.

9.3.1.1 User Interface

In the area of user interface there are several issues that need to be addressed in the future. First, the user interface could be redesigned by integrating advanced visualization techniques. Visualizations can show the users and their activities and changes to the artifacts as objects rendered in the 3D worlds, similar to those visualizations applied in the tower project [135]. The 3D world change can be extended to as artifacts are created, modified and/or deleted. For example, the avatars' colors might change as their progress in the project increase.

These visualizations represent a mapping of the avatars' activities to a 3D rendering where the users will be represented as avatars and the activities are physical metaphors (e.g. collaboration sessions, groups, and projects will be represented as a gathering of avatars; also places vary in colors depending on the gathering type).

9.3.1.2 Turn-Based Control (Floor Control)

As mentioned in section 9.2.1, concurrency control was added to SCI's editor widget by implementing a way to lock the developers who do not have permission, and to guarantee that users access the shared workspace one at a time. Although successful as a real-time software development collaboration tool, SCI's turn-based explicit control is not appropriate for all collaboration scenarios. Future work would include a "traffic-light" control mode in which editing permission switches between users automatically. For example, a user in edit mode has a green light until another user attempts to edit, after which edit mode user's light switches to yellow, and control transfers after the edit mode user is idle for an appropriate time. It would also be useful to mark user changes by different colors.

9.3.1.3 Security and Privacy

In a system such as SCI, security refers to the preservation of confidentiality of the information shared or delivered in a groupware application, such as the users' identities. Security is of great importance to users of groupware and multiplayer online games, who are concerned about theft of sensitive data and important projects' artifacts.

Privacy is an important issue and the SCI implementation did not ignore privacy concerns. SCI did not implement enough security and privacy features. A future work would include careful experimentation to find out how much privacy people will voluntarily give up in order to get the benefits of increased awareness.

9.3.1.4 Collaborative Session Management

The synchronous communication between the users of SCI system is organized around the basic concept of collaborative sessions. A collaborative session involves two or more participants (teachers and/or learners) that synchronously interact by means of a set of collaborative applications. To carry out their task, the students regularly create and join collaborative sessions to work together.

Real-time collaborations can be either scheduled or opportunistic. In either case, potential collaborators need to be able to create, lookup, and join collaborative sessions. Currently, each real-time collaborative system uses a different session management mechanism. A standard mechanism could ease the use and integration of groupware systems supported by any vendor.

A future improvement direction will focus on implementing a session management subsystem. This subsystem supports the creation of collaborative sessions that: helps ease users' communication and collaboration. It will extend SCI to provide better support for collaborative sessions that span multiple files in a project.

9.3.1.5 Interruption Management

The current notifications management implementation worked well enough for the system users and case study participants, and they responded to the notifications in a short time. A future improvement is by including an option so users specify if they can be interrupted by notification and collaboration invitation, or by allowing the system to predict/detect when a user is interruptible.

The improvement will include: 1) considering the importance of relevance and relevance of the interruption content; 2) allowing the users to specify, filter, or limit the types of notification they get. Another improvement will focus on implementing the slow-growth approach [116] to help minimizing the response time to the notifications.

9.3.1.6 Collaborative UML drawing tool

CASE tools and whiteboards are commonly used in software development design activities. Many CASE tools are designed to support online users, and allow them to share and collaborate on their UML diagrams with others. Whiteboards enable efficient collaboration and communication in drawing and designing in collaborative environments because of their ease of use and availability. However, since their design doesn't support powerful computations, they are still problematic and not enough solution for the collaborative development design and modeling [136].

This provides insight to the need for a better design and modeling tools. This area of improvement will provide the SCI development environment with a collaborative UML diagram editor. This tool will allow users to browse around within their UML-based software projects; it provides a focus of attention and a sense of location. It will provide real-time awareness of others online; not just who is on-line but what they are doing, and where.

9.3.2 Future Evaluations

The evaluations of the SCI tool presented in this dissertation show the effectiveness of integrating awareness and social network features in increasing developers' productivity and code quality, and save the time required to switch between different tools. They provide evidence that integrating CDEs within a 3D virtual environment provides the CDE with online presence and social awareness information.

An important further step for the progression of SCI is to investigate how developers interact with each other and SCI features given more complex and open-ended sets of development tasks.

Longitudinal Studies (Repeated observations of the same features over long periods of time)

Software engineering tasks typically take days or weeks to complete. An investigation into the fine-grained actions of participants during collaborative tasks over short periods of development has been conducted, but a more thorough examination gives better results. Given the SCI framework's event logging capabilities, a longitudinal study of collaborative development behavior will be of considerable value. Aspects to consider include bug counts, design aspects and frequency of compilation and debugging attempts.

Comparisons to Other Tools

Another interesting evaluation would be the comparison of existing user evaluation results with other broadly comparable tools such as Jazz [27] and CollabVS [48]. It would be interesting to compare the costs and benefits of applying these tools in practice in the distributed development environments.

9.4 Interoperability

In its current state, SCI doesn't have its own compiler or debugger. In order to support those functionalities, SCI uses compilers and debuggers from outside the system such as: gcc, g++, and javac compilers, and gdb, jdb, and udb debuggers.

SCI supports only internal chat and email facility, meaning that users can just interact from inside the system. Future work directions would include inter-operating with external and extensively used chat (e.g. MSN, Yahoo Messenger, Google Talk, Skype ... etc.) and email (e.g. Gmail, Yahoo, and Hotmail … etc.) systems.

Another area is to interoperate with version control systems such as CVS and SVN to allow users manage files in both sides, compare versions, check history, and avoid coding conflicts. The collaboration server can be layered on top of an ordinary revision control server that remembers all changes made to the project files. The SCI environment will exploit awareness of other team members; its text editor graphically depicts (using distinct background colors) the text lines with (a) pending updates to files that others have committed, (b) lines having uncommitted changes in other developers' copies of the files, and (c) lines that were recently viewed by others. This level of detail can minimize conflicts from concurrent revision of the same code, and help team members know when consultation or a meeting is in order.

Part VI

Appendices

Appendix A:
Groupware Taxonomy

This section gives a list of functional categories that represent logical taxonomies for groupware tools.

Overview

Researchers developed several framework classifications to help them understand the functionalities of these tools. Each classification has a different focus: some provide a detailed taxonomy to compare tools in a particular area, some classify tools based on: (1) their functionality, (2) predictability of the actions that they support, (3) the collaboration approach that they take, and (4) the user effort required to collaborate effectively. This section summarizes some of those frameworks.

A.1 Time/Space Matrix

Groupware tools have been classified in many ways, but most of them are based on an original matrix created by Johansen in 1988 as illustrated in Figure A-1 [137, 138, 139]. According to Ward [10], groupware technologies are typically categorized along two primary dimensions:

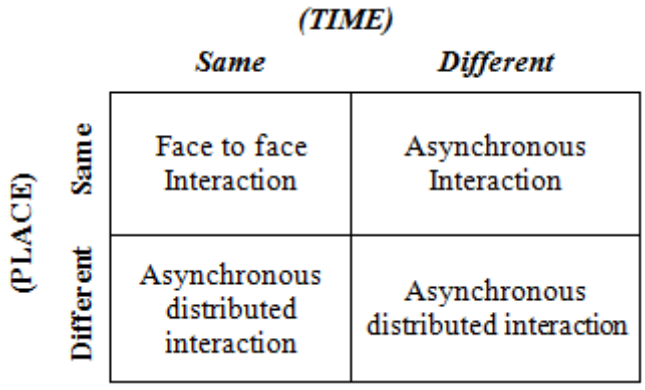

Figure A- 1 Time/Space Taxonomy (adapted from [138])

Whether users of the groupware are working together at the same time ("real-time" or "synchronous" groupware) or different times ("asynchronous" groupware), and whether users are working together in the same place ("co-located" or "face-to-face") or in different places (distributed or "distance").

In 1989, Johansen [139] refined DeSanctis and Gallupe's [137] space and time classification framework to create a 3x3 matrix (see Figure A-2) [8, 140, 141]. He improved DeSanctis and Gallupe's framework by distinguishing the tools based on the predictability of the actions that they support. Johansen's framework classifies tools based on the temporality of activities, location of the teams, and the predictability of the actions [4].

		TIME		
		Same	**Different but predictable**	**Different but unpredictable**
PLACE	**Same**	Meeting facilitation	Work shift	Team rooms
	Different but predictable	Teleconferencing, Videoconferencing, Desktop conferencing	Electronic mail	Collaborative writing
	Different but predictable	Interactive multicast seminar	Computer boards	Work flow

Figure A- 2 A variant of DeSanctis and Gallupe's Time/Space Taxonomy Framework [139]

Nutt classifies workflow systems based on the characteristics of the underlying workflow model. He created a three dimensional domain space that models a work procedure as illustrated in Figure A-3 (adapted from [4]). The three dimensions in the model space are:

- X axis − the amount of conformance that is required by the organization for which the process is a model.
- Y axis − the level of detail of the description of the process.
- Z axis − the operational nature of the model.

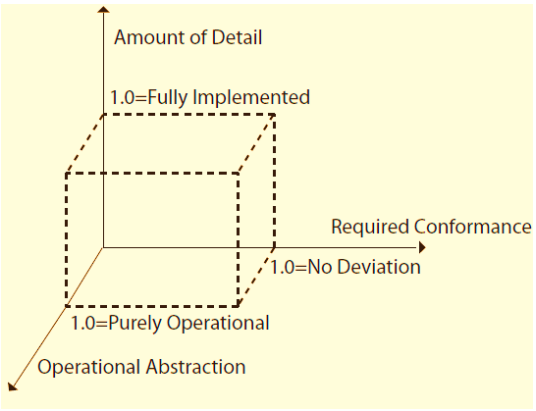

Figure A- 3 The Model Domain Space [4]

Andriessen, in his book "Working with Groupware: Understanding and Evaluating Collaboration Technology" (2003) extends Johansen's standard taxonomy further by distinguishing the groupware application from the other information and communication technology applications by having functions that serve five possible groups of processes: Communication, Cooperation, Coordination, Information Sharing, and Social Interactions [134].

A.3 Pyramid Taxonomy from the User's Effort Perspective

All of the above frameworks look at coordination tools from a functionality point of view. Sarma [4] introduces a classification framework that revolves around different classes of user effort required to collaborate effectively. This framework is based on two principal characteristics of collaborative tools: (1) the level of coordination support provided to users, and (2) the focus of a tool on one of the three essential elements of collaboration: communication, artifact management, and task management. It combines these two characteristics to form a pyramid, as illustrated in Figure A-4 where five levels of coordination support are organized vertically and called "layers" and three different foci of tools are organized horizontally, and called "strands".

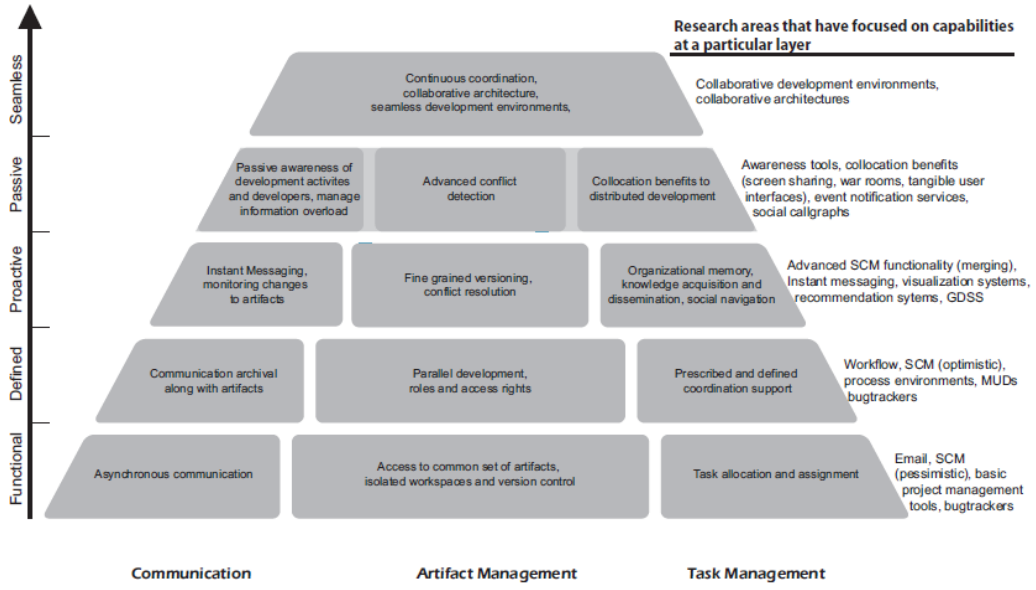

Figure A- 4 Sarma's Classification framework [4]

Penichet and others [142], try to present a more flexible taxonomy for the CSCW systems. Their proposed solution is to classify the tools and systems by showing the relation between the system with the time-space features and with the typical CSCW characteristics: information sharing, communication and coordination. A specific system can have several characteristics. Figure A-5 shows the possibilities regarding CSCW characteristics.

Type	CSCW Characteristics		
	Information Sharing No = 0, Yes = 1	Communication No = 0, YES = 1	Coordination No = 0, Yes = 1
X	0	0	0
A	0	0	1
B	0	1	0
C	0	1	1
D	1	0	0
E	1	0	1
F	1	1	0
G	1	1	1

Figure A- 5 Possibilities regarding CSCW Characteristics [142]

This dissertation, along with Sarma [4], agrees that most of the previous frameworks look at coordination tools from a functionality point of view, and none classify tools based on the user's effort required to collaborate effectively. Also, it observes that none of the previous studies classify tools based on online presence. For this reason we extend the taxonomy introduced by Penichet et al. [142] to show the relation between the system with the time-space features and with the four CSCW characteristics instead of three: information sharing, communication, coordination, and social and online presence. Someone could argue that awareness and interaction cannot be separated from the other characteristics. But the fact is that, in software engineering, awareness is generally linked to issues related to coordination or managing dependencies between activities. Awareness is seen as a way to inform team members of the work of others that is interdependent with their current tasks, therefore it enables better coordination of teams, but is not itself a coordination [143].

Appendix B:
Network Protocol Messages

This appendix provides information about the network protocol messages that both of ICI and SCI use. The protocol uses a string switching technique to send and receive messages in the form of string. The appendix explains the format of the rules that specify how data is packaged into messages sent and received. The next section describes the ICI message categories, followed by the SCI message categories.

B.1 ICI messages

ICI messages are divided into three categories. The following subsections contain a discussion of each category and the messages that belong to each one of them.

B.1.1 General ICI messages

This messages category consists of seven messages. Once a user chooses to open a collaborative IDE session and invite another user, the *CETLOpen* message will be fired and a request for collaboration will be initiated. It takes a string a string argument consisting of the user_name, filename, and encoded file contents.

Users can accept or reject to join the IDE session. If the invited user decides to start collaboration, then *CETLAccept* will be called to inform the server of the other user acceptance to the invitation. This message has the parameters (recipient, index_counter, file_name, and slave). It opens a collaborative IDE tab on the accepting client, and changes the background color of the editor and shell widgets on all clients to light yellow as an indication for the collaboration session. Choosing to reject or ignore the collaboration, the message *RejectIDE* is called to inform the server after the user rejects an invitation to join a collaboration session.

In addition to the editing sessions, ICI users can initiate compilation and debugging sessions. Compile sessions start with the *CETLCompile* message, which convey the compilation and linking messages, notably the error messages, through the server to the other collaboration users.

As mentioned earlier in section 4.4.2, ICI's collaborative editor supports two modes: watch and edit. During a collaborative session, only the lock owner may edit; the lock control gets assigned once the editing session starts with the call for *CETLLock* message. The rest of the participants are in watch mode, but they still can ask for permission to take a turn at the controls. Once a user asks to take a turn, ICI sends *CETLLockTransfer* message to indicate that the user requests a turn

at program editing. If the owner closes the session, then the state will change to unlocked, and the background color of the editor and shell widgets will change to white.

B.1.2 Collaborative editor messages

The collaborative editing messages category includes four main *CETL* messages. Those *CETL* messages get generated by *syntaxCETL* class in response to GUI events coming from the collaborative editor. Those messages are *CETLevent*, *CETLmouse*, *CETLkey*, and *CETLscrol*. Once the events got fired and the messaged called, the server calls the *handle_CETL_Event()* method to forward the event to the participating clients.

B.1.3 Collaborative shell messages

The collaborative editing messages category includes four main *SHL* messages. The class ShellETL generates those *SHL* messages in response for both GUI user input events and output received from the external process (the compiler, debugger, or program being executed). The *SHL* messages are *SHLevent*, *SHLmouse*, *SHLkey*, *SHLscrol*. As soon as the server gets the events through the SHL commands, the server forwards the event to all the collaborative shells at the other side by calling the *handle_CETL_Event()* method.

B.2 SCI messages

SCI messages cover four different message categories. This section gives brief information about each category and the messages that belong to each one of them.

B.2.1 Group messages

Groups help developers connect and share resources and discussion with colleagues around a common interest. They can connect with colleagues who are interested in a specialty or a specific topic area, working on a project or attending a class together. Following is a list of the group management network commands:

SCI allows its users to make their own special interest groups (SIGs), by simply passing the chosen *sigName* and their names (*sigOwner*) as string arguments to the server, and through the *makesig* network message.

Only SIG owners can remove them. When users chose to remove their SIGs, SCI initiates *groupRemove* message to inform the server that the SIG owner is asking to remove the group. It

also informs its members that this group is no longer available. This string attached to this message takes two arguments *sigName* and *sigOwner*. And

SCI users invite others to join their own SIGs, join other members SIGs, and leave SIGs. When a user sends an invitation to another to join the group, the group *groupInvite* messages will be called attached with a string consists of *invitedUser*, *sigName*, and *groupOwner* to inform the *invitedUser*. To join a group the client sends a *groupJoin* meesage to the *groupOwner* requesting to join a group. Users (both of the *invitedUser* and *groupOwner*) can either accept or reject the invitation; if accepted then the *sigOwner* client side sends a *groupAdduser* message to inform the server to add the *invitedUser* to the SIG members list.

SIG members may leave the group at any time. Once decided to leave the group, a *groupLeave* will be sent indicating that the user is leaving the group *sigName*, then the server calls *userRemove* message to remove the user from the members list and updates the group tree.

B.2.2 News Feed messages

The news feed provide users with information of the recently and frequently updated contents and project artifacts. This messages category allows users to post a feed (*addFeed*), remove their own feeds (*removeFeed*), view the contents of the group, friend, project, or community feeds (*viewFeed*), and respond to a feed by explanations or answers to questions (*replytoFeed*).

The news feed messages get generated by the *ICINewsFeed* class in response to the GUI events coming from the newsfeed popup menu events *newsfeed_popup_menu()*.

B.2.3 Project messages

The Projects class generate those messages in response to the GUI events coming from the projects popup menu events *project_popup_menu()*. Those messages allow accessing the project artifacts: open, close, commit changes, and check differences between file versions. The project messages are *projfileOpen, projfileClose*, *commitChanges, readytoCommit, checkdiff*.

B.2.4 User messages

Those messages allow users to manage contacts activities, such as: adding friends (*addFriend*), accepting friends' requests (*addFriendAccept*), remove users from the friend list (*userRemove*), and view contacts' personal profiles (*userProfile*).

Appendix C:
SCI's Real-Time Collaborative Editing

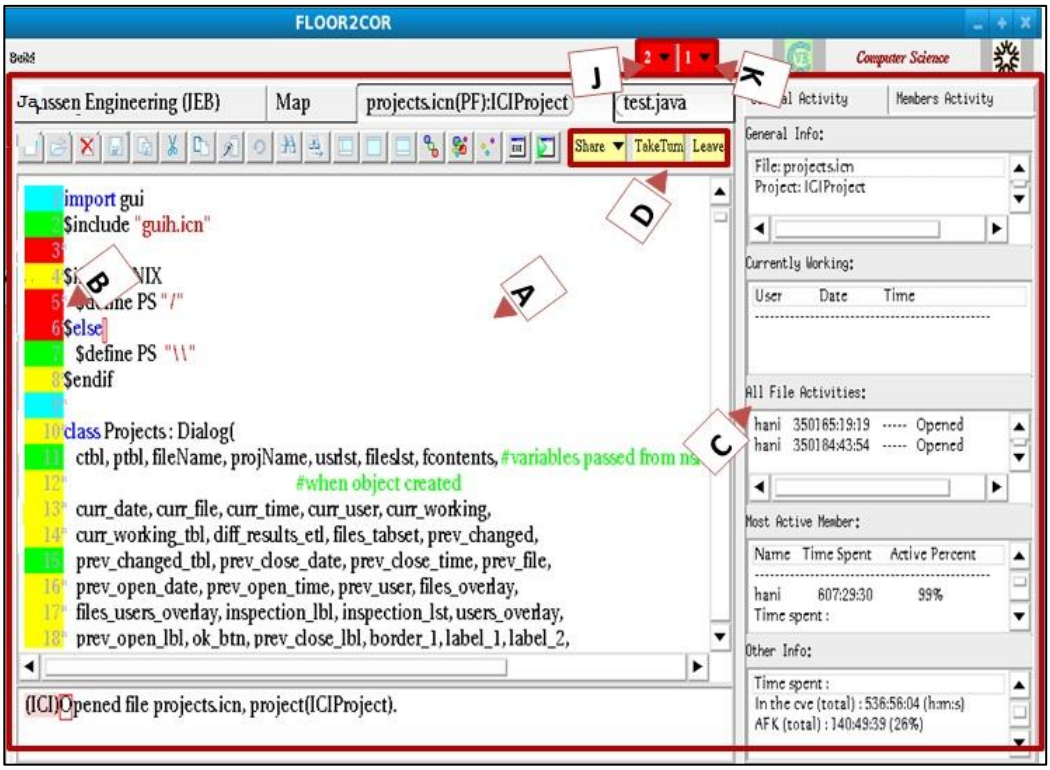

Figure C- 1 Main SCI IDE View

This appendix details how you can develop a program code and share your code in CVE with others from inside the Collaboration Area.

In CVE it is possible to share documents such as computer code that you have written either in Java, C/C++ or Unicon by clicking on the Share button shown in Figure C-1. As mentioned in the previous chapter, CVE's Integrated Development Environment (IDE) lets you write, edit, run, compile and debug code inside the virtual environment.

The collaboration area can also be used to share your files in a collaborative session. This makes the IDE especially suited for education purposes, because students can debug their code with the help of other students or their instructor by sharing their code with them.

It is possible to share your code with others by you initiating a collaborative session with another user (other users can be either available user's friends or users who are experts in the shared file programming language). This area lets you organize your projects, connect with other users and share your code by inviting another user to share your screen. Sharing code will lead to a greater sharing of knowledge

This appendix describes the following items:

— Sharing your code with others through the Collaborative IDE

— Inviting a Slave user to collaborate

— Collaborative Session Options

— Taking a Turn on a Session

— Closing a Collaborative Session

The IDE gives users the ability to work on a project in the virtual world individually, as well as share these files with other users for troubleshooting. This lets users work on projects together and read each other's code by using the collaborative function of the IDE (see Figure C-2).

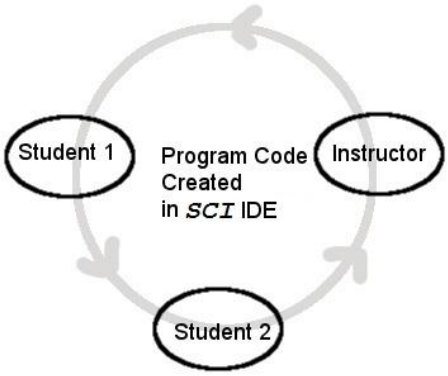

Figure C- 2 Collaboration Cycle.

C.1. Sharing your code with others through the Collaborative IDE

In the collaborative IDE a student can ask the instructor to see his/her computer code, so the instructor can help the student in debugging the code. In the collaborative IDE there are two roles that can be played by users in collaborating with each other: Master and Slave (See Figure C-3).

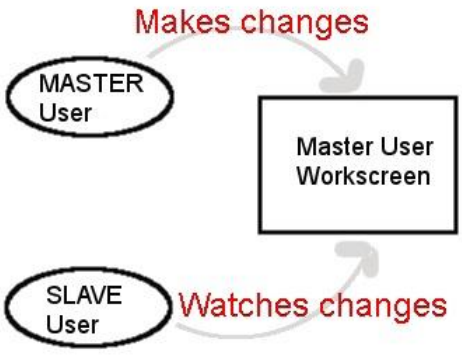

Figure C- 3 Collaborative IDE Main Roles

The Master user is the person currently working on a program and shares access to their screen, whereas the Slave is witness to what the Master user is doing on their screen. However, a Slave user can ask for permission from a Master user by Taking a Turn to work on a file if they want to try their hand at it. This then switches the roles around between Master and Slave users.

NOTE:

Master user: The Master user invites another person in sharing their work-screen, and is the person who controls all the events in the shared editor screen.

Slave user: The Slave user accepts the invitation from the Master user, but can only see what actions the Master user performed in the editor screen. The Slave user can ask for permission to Take a Turn and become a Master user who can edit the file.

Anybody can be a ***Master*** or ***Slave***; it just depends on which person initiates a collaborative session. For instance, if a student wants to debug his code with the help of an instructor, the student (as ***Master*** user) will need to invite the instructor (***Slave*** user), so that the instructor can see the student's interactions on screen. The instructor (the Slave user) can then give feedback through chat or Take a Turn to help the student debug their code.

In order to use the ***Collaborative IDE***, a Master user will always need to take the following steps (See Figure C-4):
Step 1: Open or create a file to collaborate on
Step 2: Invite a Slave user to collaborate

Above is an overview of the process, followed by the individual steps to opening a file and inviting a user to collaborate.

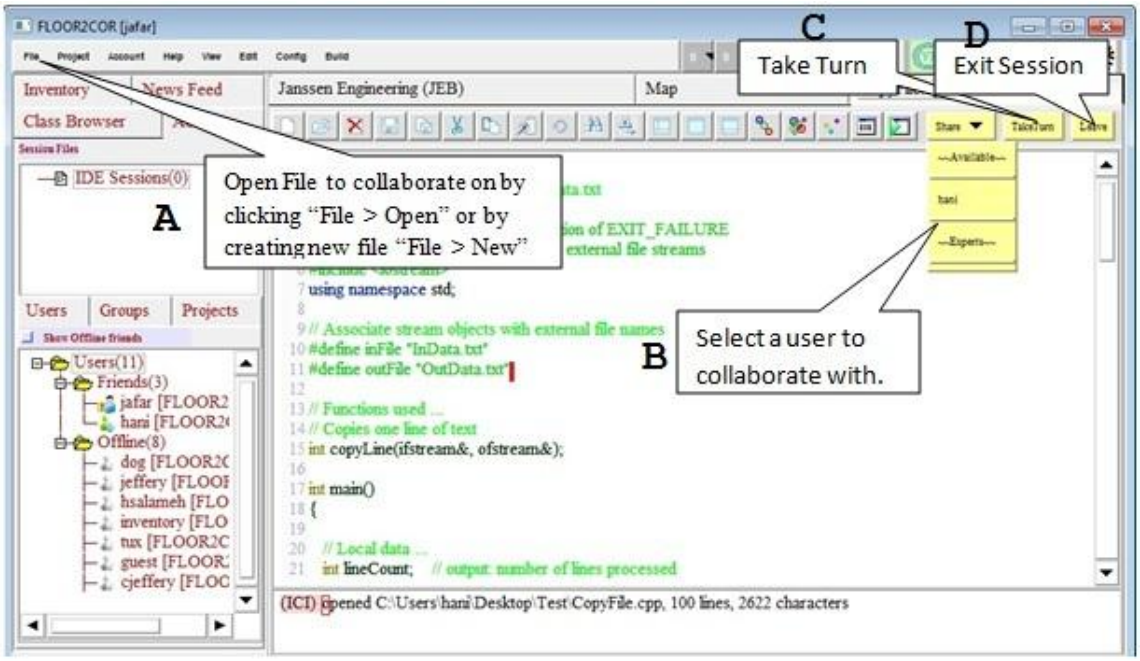

Figure C- 4 Session's Main Actions (Share, Take Turn, and Leave)

NOTE:

In order to collaborate on a file, a Master user will always need to take these two steps:

— Open an existing or new file to work on

— Invite a Slave user to a collaborative session

C.2. Inviting another User to Collaborate

NOTE: In order to invite another user, a file owner will need to first have opened a file (as described above) prior to initiating collaboration with a guest user.

Inviting a guest user in CVE is easy:

— With a file opened, click the "Share" button (Figure C-4 (B)).

— From the generated drop down menu, choose one of the users to do a collaborative IDE session with.

A notification message (see Figure C-5) will appear in the selected Slave user's client window saying that User [Master User Name] is asking to open collaborative editor session. This signals to the Slave user that the Master user sent him/her a collaborative editing session invitation.

Figure C- 5 Invitation's Notification

The slave user can accept, reject, or forward the invitation by choosing the invitation item appears in Figure C-5, and a notification dialog will appear (See Figure C-6).

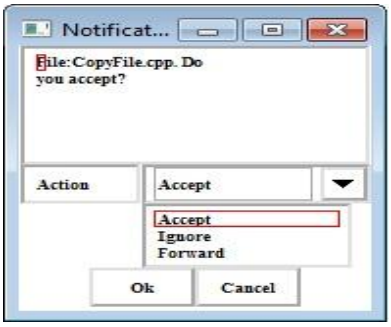

Figure C- 6 Notification's Response Window

— Modify the shared file in the editor screen.

The invited user will see the inviting user's name in the ***Session Tree*** (See Figure C-7) with a ***(owner)** next to the name to indicate that this is the owner of the session and the one who currently have the permission to edit the file (*).

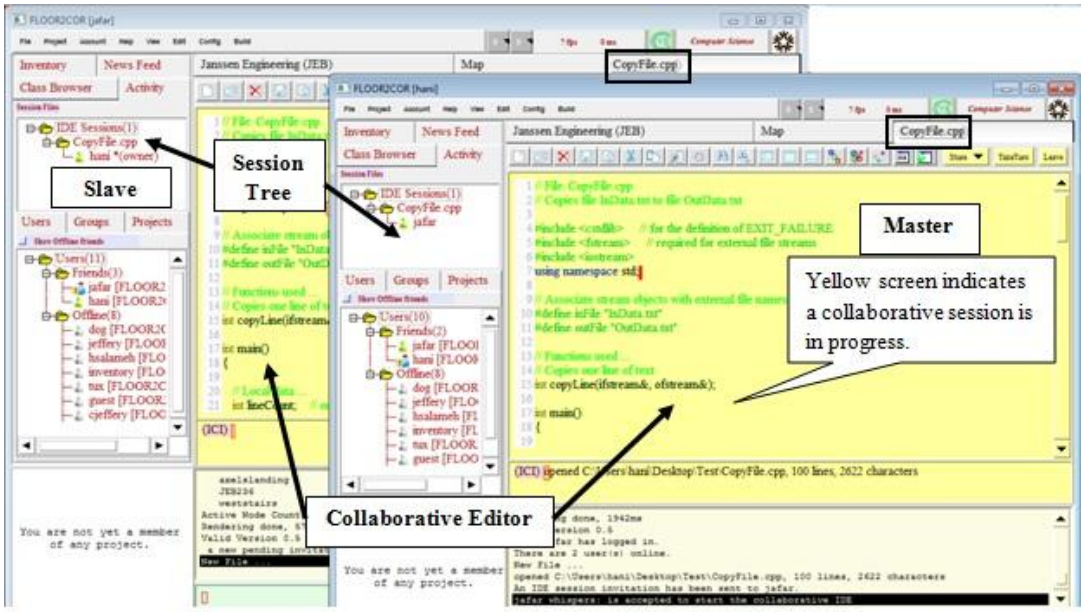

Figure C- 7 SCI's Collaboration Session

C.3. Collaborative Session Options

As mention earlier and shown in Figure C-6, when invited by a Master user, the Slave user can Accept, Reject, or Forward the Master user's request for a collaborative editing session.

When a collaborative session is rejected

If the Slave user rejects the invitation, the new editor tab will not open on the Slave user's screen, and the Master user's request for a collaborative session is rejected. A message from the user invited will appear in the Master user's screen indicating that the session invitation is rejected.

When a collaborative session is accepted:

If the slave user accepts the invitation, a new editor tab will open in the slave user. A chat message will appear in the Master user's screen indicating that "*[User Name]* is accepted to start the collaborative IDE", which indicates to them that their request for a collaborative session is accepted by the Slave user.
As displayed in Figure C-7, a collaborative session changes by the background color to light yellow in both the Master's and Slave's side (which indicates that the tab is currently used for a collaborative session).

NOTE:

The Slave user has no ability to type or make any changes in the editor, since the Master user can only make changes. Hence, during the collaborative session, the tab for the Master user is unlocked, whereas the Slave user's tab is locked.
The Slave user can give feedback to the Master user by using voice chat or text chat. Moreover, the Slave user can ask to *Take a Turn*, which will allow the Slave user to become the owner of the lock and edit the file while the Master becomes their Slave user.
When a Collaborative Session has started, anything the Master user does in his/her editor will appear in the Slave user's editor screen:

- Typing from the keyboard
- Selecting text using mouse or keyboard
- Using the scrollbar
- Mouse Click, PgUp, PgDn keys.
- Selecting all (Ctrl-A), Copy (Ctrl-C), Cut (Ctrl-X), Paste (Ctrl-V), Undo (Ctrl-Z) and Redo (Ctrl-Y)

During the collaborative session, the Slave user can chat with the Master user and give feedback to the Master user about the collaborative session. A Slave user can also ask to ***Take a Turn*** to edit the file themselves.

C.4. Taking a Turn on a Collaborative Session

In order to Take a Turn and work on a program, a guest user must ask for permission for this from the Master user. This ensures that the Master user is finished with their work and is ready to share their program with the Slave user.

In order for a Slave user to ***Take Turn***, the following will need to be done:

— The Slave user will need to click on the Take Turn button (See Figure C-4(C)), and notification will be sent to the owner of the lock (Master) (See Figure C-8).

Figure C- 8 Take Turn Notification

— The Master can either accept or ignore the notification when asked if the Slave user can Take a Turn and start editing the Master user's file.

If a Master user is ready, they can accept to switch roles with the Slave user.

If a Master user is not ready to give the Slave user a turn at editing the file, they can choose to ignore the invitation.

C.5. Closing (Leaving) a Collaborative Session

At any time the user can leave the collaborative session, by clicking on the Leave button (see Figure C-4(D)).

The Shared editor area will return from the collaborative light yellow color to a regular white background to indicate that the collaborative session has ended. Closing (Leaving) a session will automatically save the last version of the file on the owner and guest user's side.

Appendix D:
SCI's BNF Grammar

This appendix provides a BNF grammar that describes the structure for SCI network messages.

```
<Network_Protocol> ::= <Command> <Message> crlf
<Command>          ::= <GeneralICI_Commands> |
                       <CollabEditor_Commands> |
                       <CollabShell_Commands> |
                       <Group_Commands> |
                       <User_Commands> |
                       <NewsFeed_Commands> |
                       <Project_Commands> |
                       <Pendings_Commands>
<Message> ::= <params> <extra> SPACE
<params>  ::= <sender> <recipient> <project_name> <project_owner>
              <sessionID> <file_contets> <comiledebugg_output>
              <grpup_name> <group_owner>
<GeneralICI_Commands> ::= CETLAccept | CETLCompile | CETLLock |
                          CETLLockTransfer | CETLOpen | RejectIDE
<CollabEditor_Commands> ::= CETLevent | CETLkey | CETLmose |
                            CETLscrol
<CollabShell_Commands> ::= SHLevent | SHLkey | SHLmose | SHLscrol
<Group_Commands> ::= makesig | groupAdduser | groupInvite |
                     groupJoin | groupRemove | groupLeave
<User_Commands> ::= addFriend | addFriendAccept | userRemove |
                    userProfile
<NewsFeed_Commands> ::= addFeed |  removeFeed | replytoFeed |
                        viewFeed
<Project_Commands> ::= projfileOpen | projfileClose | checkdiff |
                       readytoCommit | commitChanges
<Pendings_Commands> ::= ICIPendings | DELPending |
                        FWDPendingSuggest
<crlf>  ::= <carriage_return> <line_feed>
```

Appendix E-1:
Instructions Sheet (I)

Goal: Test the effectiveness of SCI development environment in software engineering classrooms, and study the effectiveness of the supported communication and social networking features.

Task: Perform a simple collaborative programming task in the classroom. On your own or with the assistance of the instructor, choose a partner and form a team of two, and once you are in your team, decide with your partner who will be the driver and who the watcher, and then partners can trade each other's user name so they can communicate with each other from inside the SCI system.

Procedure: The work is distributed between two participants so they can collaborate in finishing the assigned task. One of the participants, the *driver*, controls the shared editor and starts the collaborative debugger. The driver is responsible for opening the source code. The second participant, the *watcher*, examines the driver's work, offering advice, and suggesting corrections. Driver and watcher can trade places every few minutes or as often as desired in order to complete the task.

Communication: Participants communicate with each other using on-line tools such as: 1) Email (Vandal's email), and 2) Instant Messaging (Google Talk, MSN messenger ….). If you and your partner have a common instant messaging tool, please use it, and otherwise please communicate using emails.

Time: The programming session is scheduled to take 50 minutes.

1. Participants will be required to join the C++ special interest group.
   Right-clicking on a group node, a user can join a special interesst group.
2. The session driver starts the collaborative session, and invites his/her partner to join the collaboration.

To start a collaborative IDE session:

A. Open new file from the file menu.
B. Save the file as **[groupName].cpp**.
C. The owner (driver) of the session can invite other users to a collaborative session by clicking on the **Share** menu button (top right corner of the shared file tab), and then choose one of the users (available and expert users) to join the collaborative IDE session.
D. Once the driver sends the invitation, a pending item will appear on the watcher client.

E. The watcher accepts/rejects the invitation as follows:

A notification message will appear in the selected client window (*watcher*) saying that "*You got an IDE Session from [Master User Name]"*. This signals to the *watcher* that the *master* user has sent him/her a collaborative editing session invitation.

Figure E- 1 *Pending notifications (showing IDE session invitation).*

F. The watcher user can accept, reject, or forward the invitation by choosing the invitation item that appears in Figure E-1.

G. Accepting the invitation changes both the "text editor" and "shell message box" background color to light yellow in both sides, indicating the start of the collaborative session. Also, it will lock the tab of all the collaborated users except the owner of the session. Only the owner of the session can edit the file/program.

H. Then, any changes the owner does to the file/program in his side will appear in the watcher side. The watcher can see the owner actions such as:

- Typing characters from the keyboard.
- Selecting text using mouse or keyboard.
- Using the scrollbar.
- Mouse click; also PageUp and PageDown keys.
- Shortcuts such as: Select all (Ctrl-A), Copy (Ctrl-C), Cut (Ctrl-X), Paste (Ctrl-V), Undo (Ctrl-Z), Redo (Ctrl-Y), and Save (Ctrl-S).

3. Participants are required to collaborate and complete the missing parts of the code. They can switch roles finishing the code.

To switch roles (IDE: Take Turn) (Participants switch roles at regular intervals.)

In order for a watcher user to ***Take Turn***, the following must occur:

A. The watcher user clicks on the **Take Turn** button, causing a notification to be sent to the owner of the lock (see Figure E-2).

B. The driver can either accept or ignore the notification when asked if the watcher user may Take a Turn and start editing the file.

Figure E- 2 Pending notifications (showing Take Turn invitation).

If a driver user is ready, they can accept to switch roles with the watcher user.

If a driver user is not ready to give the watcher user a turn at editing the file, they can choose to ignore the invitation.

The driver can accept the request by pressing the notifications menu. A notification dialog will appear similar to the one shown in Figure E-2, and then the owner may choose the accept choice from the list.

4. When finished, participants are required to compile and run their programs.

Appendix E-2:
Instructions Sheet (II)

Goal: Test how easily students can get assistance and collaborate while working on their assignments.

Task: Perform a simple debugging task in the classroom, and using the CVE environment. Participants will work on this assignment individually, but they still can ask for assistance from their instructor and/or classmates.

Procedure: The participants will open a project file and try to fix bugs in this file.

Communication: Participants communicate with each other from inside the CVE environment.

Time: The debugging session is scheduled to take 50 minutes.

Note: Each time a user sends an invitation, a new notification will be added to the recipient pending notifications.

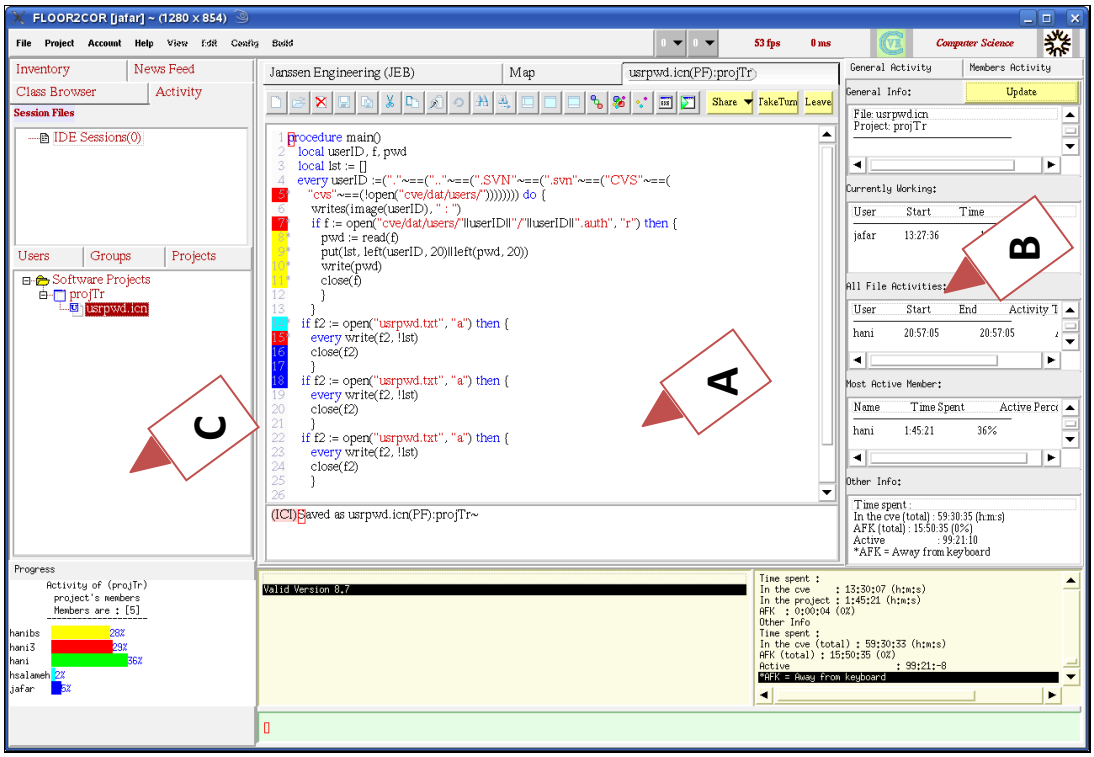

Figure E- 3 The main SCI window. Showing the opened file area (A), the activity tab (B), and the project(s) functionality tab (C).

Before the session starting, the instructor (CVE admin) created a project called *"projectA"*, and added a file called "taskprog.cpp" to the project (a C++ file with some bugs and errors).

1. Join the development project called *"projectA"*.

   Right-clicking on the project node, select "Project", and then click "Join" to join the ptoject.

   This project contains a single source file called taskprog.cpp.

2. Open the file called "taskprog.cpp", and start fixing the bugs.

   Right-clicking on the assigned file node, and then click "Open".

3. To the right side of the opened file area, you can see awareness of the opened file activity, who is working on it, and other project related information.

4. Participants are required to send friendship requests to other users available in the environment. They are required to establish friend status with their assigned partners.

   Right-click on an online/offline node, and then click "Add as Friend".

5. They can chat with other participants and request help, and ask how to solve a specific bug.

   \tell userid message

6. Participants are welcomed to ask for assistance from their group members using the email.

   Right-click on an online/offline node, and then click "Email".

7. Also, they need to post either questions or comments to the group wall.

   Right-click on the project's node, and then click "Wall …".

8. After finishing the task, students are required to compile, run, and make sure that their own versions of the code are error free.

9. Finally, they need to commit their changes to the server.

   Right-click on the opened file node, and then click "Commit".

Appendix E-3:
Instructions Sheet (III)

Goal: Test the effect of integrating the social networking features and 3D activities on the development environment.

Task: Perform social activities using both the supported social networking features and the 3D world environment.

Procedure: The participants should follow some steps to perform the assigned activities and socialize with others available in the environment.

Communication: Participants communicate with each other from inside the CVE environment.

Time: The session is scheduled to take 50-60 minutes.

Notes:

(1) Each time a user sends an invitation, a new notification will be added to the recipient pending notifications.

(2) In order to send chat message(s):

- Private : **\tell** userid **message**
- Public : **\say** message

Before starting the session, the system administrator created the project "***My3D***". Participants were randomly assigned to teams, and a location (room) was assigned for each team. Participants involved in this task should perform the following activities (interactions):

Interactions from inside the 3D world

5. Switch to the 3D world tab.
6. Check the other team member's locations from the users' tree.
7. Visit the closest rooms to your location, and interact with avatars (send greetings, and view activities).
8. Teleport to the general meeting room (the virtual room JEB321), see who is around and introduce yourself.

— **For example:** to teleport to **csac** room, you type **\teleport csac** in the chat box.

9. Interact with the nearest avatar in **csac** room (interactions such as sending greetings, and viewing activities).
10. Your team's name is _____.
11. Check who is your assigned partner, introduce yourself, and then teleport to the team's room.
12. After moving to the team's room, with your partner perform the following list of activities:

- Send a greeting to your partner(s).
- View your partner's activity "History".
- Check your partner's "Project Activity".
- Exchange roles with your partner (one is master/driver and the second is watcher/helper)
- Join the project **"My3D"**.
- See the assigned programming task.
- Discuss the solution with your partner.
- Discuss with your partner what part of the code each should finish.
- The driver of the session must open a C++ file and start the program.
- Save the file with your team name.
- After finishing his/her part of the code, the driver adds the file to project **"My3D"**.
- The helper opens the file, adds his/her part of the code, and compiles the program to make sure it is working.

Interacting with the SCI Social network features

13. View your partner's profile (or any of the CVE users).
14. Check the availability table and progress chart available at your partner's profile, and check to figure out at what time your partner is available for help?
15. View your partner's availability by checking the "User's Usage Report".
16. Send an email(s) to your partner, and ask questions about his/her experience with the SCI environment. Also, send an email to the system's administrator to suggest changes to the system and leave comments.

Appendix F-1:
SCI's Collaborative Programming and Usability Survey (I)

Participant related questions						
1.	Rate your knowledge/expertise of C++ programming.					
	Beginner					Expert
	1	2	3	4	5	
2.	I have experience working with collaborative tools.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
3.	I have had experience working with pair programming teams before.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
4.	I usually work physically close to my teammate(s).					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
Communication related questions						
5.	It was easy to communicate (start an online chat/conversation, and/or send email)					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
6.	It was easy to watch my partner make changes to the code while conversing using					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
7.	I prefer having the following tools integrated inside the CVE system:					
		Strongly Disagree				Strongly Agree
	Chat (text/audio)	1	2	3	4	5
	Email	1	2	3	4	5
	Wall/Forum	1	2	3	4	5
Task (Code sharing) related questions						
8.	It was easy to start and end the collaborative editing session.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
9.	It was useful to watch my partner editing/debugging the code.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
10.	It was easy to get assistance within CVE compared to face-to-face meetings.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
11.	I was more comfortable with my solution knowing that someone was watching the					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	

General questions					
12.	Overall, it was easy for me to use CVE to collaborate with my partner and finish				
	Strongly Disagree				Strongly Agree
	1	2	3	4	5
13.	Overall, the CVE IDE window was ***distracting***.				
	Strongly Disagree				Strongly Agree
	1	2	3	4	5
14.	Overall, I liked finishing my work with a partner and within CVE more than				
	Strongly Disagree				Strongly Agree
	1	2	3	4	5

Appendix F-2:
SCI's Collaborative Programming and Usability Survey (II)

Communication related questions						
1.	During the session, I used the following communication tools:					
	Text chat	Yes	No			
	Email	Yes	No			
	Wall	Yes	No			
	Newsfeeds	Yes	No			
2.	It was easy to communicate (start an online chat/conversation, and/or send email) with my team partner.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
3.	It was useful to have the following tools/features integrated inside the CVE system, and make the task completion easier.					
		Strongly Disagree				Strongly Agree
	Text chat	1	2	3	4	5
	Email	1	2	3	4	5
	Wall	1	2	3	4	5
	Newsfeed	1	2	3	4	5
	Awareness/Presence features	1	2	3	4	5
Task related questions						
4.	It was easy to access the project file(s) and make fix the bugs.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
5.	It was useful to be aware of who is currently working on the same file by observing the awareness information.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
6.	It was useful to see my team mate(s) presence and activity in the project (even when I did not have any direct benefit).					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
7.	It was easy to ask for assistance within CVE while finishing the assigned task.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
8.	It was easy to get assistance within CVE while finishing the assigned task.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
General questions						

9.	Overall, it was easy for me to communicate with my teammate(s) and finish my task. Strongly Disagree 1 2 3 4 5 Strongly Agree
10.	Overall, the CVE IDE window was ***distracting***. Strongly Disagree 1 2 3 4 5 Strongly Agree
11.	Overall, the supported awareness information were enough and useful. Strongly Disagree 1 2 3 4 5 Strongly Agree
Social Networking related questions	
12.	It was easy to perform an/or view the following activities: Strongly Disagree / Strongly Agree Friends request 1 2 3 4 5 Assistance request 1 2 3 4 5 Walls post(s) 1 2 3 4 5 Users status 1 2 3 4 5 Projects/Groups request 1 2 3 4 5
13.	It was easy to notice the notifications, emails, requests, and invitation that I received from other participants. Strongly Disagree 1 2 3 4 5 Strongly Agree
14.	Joining the software project helped me contact and request help from programming language experts. Strongly Disagree 1 2 3 4 5 Strongly Agree
15.	I find the notification icons important features in the environment. Strongly Disagree 1 2 3 4 5 Strongly Agree
16.	I find it useful to see my teammate's status/presence (away/busy/offline/online). Strongly Disagree 1 2 3 4 5 Strongly Agree
Usability related questions	
17.	**Ease of deployment**. It was easy to configure the environment to initiate and/or take part in a collaborative session. Strongly Disagree 1 2 3 4 5 Strongly Agree
18.	**Fidelity.** The supported awareness information is correct and up-to-date. Strongly Disagree 1 2 3 4 5 Strongly Agree
19.	**Collaboration Awareness.** The system provided me with enough cues about my teammate's activity and presence. Strongly Disagree 1 2 3 4 5 Strongly Agree

Appendix F-3:
SCI's Collaborative Programming and Usability Survey (III)

3D related questions	
1.	I have had experience in using 3D virtual environments before the session. Strongly Disagree 1 2 3 4 5 Strongly Agree
2.	The 3D world brings a real world value to the collaboration. Strongly Disagree 1 2 3 4 5 Strongly Agree
3.	The use of 3D virtual environment and avatars creates a sense of presence of others. Strongly Disagree 1 2 3 4 5 Strongly Agree
4.	Having visual representation of others inside the 3D world is sufficient to create a high sense of presence. Strongly Disagree 1 2 3 4 5 Strongly Agree
5.	Interaction and collaboration is needed to create a high sense of presence. Strongly Disagree 1 2 3 4 5 Strongly Agree
6.	The sense of my teammate's presence increased with the interaction and collaboration inside the 3D world. Strongly Disagree 1 2 3 4 5 Strongly Agree
7.	In a virtual environment, you can see other people's avatar moving around, and you can see what they are doing. Do you think that helps you to interact and be more social than if it would be chat-only or web-based environment? Strongly Unsatisfied 1 2 3 4 5 Strongly Satisfied
SN related questions	
8.	I have had experience in using social network(s) before the session. Strongly Disagree 1 2 3 4 5 Strongly Agree
9.	Having the walls, chat, and other social network tools integrated within the SCI environment, helped me finish my task, and established a successful social Strongly Disagree 1 2 3 4 5 Strongly
10.	It was easy to perform and/or view the following activities: Strongly Disagree / Strongly Agree Projects and Group Join 1 2 3 4 5 Walls Post(s) 1 2 3 4 5 Users Status 1 2 3 4 5

	View Profile	1	2	3	4	5
	Check Usage Report	1	2	3	4	5
	Check Availability	1	2	3	4	5
	Assistance Request	1	2	3	4	5
11.	The progress bar chart provides useful awareness information of my project's activities.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
12.	Using the supported awareness information and social network features to socialize made communication easier and the environment more suitable for the task.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
General related questions						
13.	The CVE environment improved my team productivity, and made the task enjoyable.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
14.	I find CVE easy to learn.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
15.	I find CVE to be a useful environment.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
16.	Rate your satisfaction with the collaboration outcome.					
	Strongly Unsatisfied					Strongly Satisfied
	1	2	3	4	5	
17.	Rate your satisfaction with the collaboration process.					
	Strongly Unsatisfied					Strongly Satisfied
	1	2	3	4	5	
18.	Other than the server instability,					
	I find that CVE is a suitable classroom programming environment?					
	Yes				No	
	I am willing / motivated to use the SCI environment for collaboration tasks again?					
	Yes				No	
	Why?					
	The supported communication and collaboration tools are enough to provide a productive environment.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
	The supported awareness and presence information is enough and adequate.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	
19.	Rate your willingness/motivation to use the SCI environment for collaboration tasks again.					
	Strongly Disagree					Strongly Agree
	1	2	3	4	5	

References

[1] Hazzan, O., Dubinsky, Y.: Social Perspective of Software Development Methods: The Case of the Prisoner Dilemma and Extreme Programming. XP 2005: 74-81.

[2] Gutwin, C., Schneider, K., Paquette, D., and Penner, R.: Supporting Group Awareness in Distributed Software Development. EHCI/DS-VIS 2004: 383-397.

[3] Cook, C. Towards Computer-Supported Collaborative Software Engineering, PhD Dissertation, University of Canterbury, Christchurch, New Zealand, 2007. Available at http://www.cosc.canterbury.ac.nz/research/reports/PhdTheses/2007/phd_0701.pdf. Accessed June 10, 2011.

[4] Sarma, A. A Survey of Collaborative Tools in Software Development. ISR Technical Report UCI-ISR-05-3, Donald Bren School of Information and Computer Science, University of California, Irvine, USA, 2005.

[5] Booch, G., and Brown, A. Collaborative Development Environments. January 11, 2007. Available at http://drdobbs.com/architecture-and-design/196900222. Accessed June 10, 2011.

[6] Booch, G. and Brown, A. W. Collaborative Development Environments. Advances in Computers, 59:2–29, 2003.

[7] Introduction to Doug Engelbart's Revolution. Available at http://www.co-intelligence.org/CollectiveIntellTakesOff.html. Accessed June 10, 2011.

[8] Grudin, J. Computer-Supported Cooperative Work: History and Focus, Computer, vol. 27, no. 5, pp. 19-26, May 1994, doi:10.1109/2.291294.

[9] Rama, J., and Bishop, J. A Survey and Comparison of CSCW Groupware Applications, Proceedings of the 2006 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries SAICSIT '06, ACM.

[10] Ward, S. Making Sense of Computer Supported Cooperative Work: A Taxonomy of Terminology, PhD Dissertation, Central Queensland University, Australia, 2007.

[11] Greenberg, S. Computer Supported Cooperative Work and Groupware: An Introduction to the Special Edition. International Journal of Man Machine Studies, 34(2), pp. 133-143, February, 1991.

[12] Dover, W. A Survey of Groupware. Available at http://www.cc.gatech.edu/classes/cs6751_97_fall/projects/spin/groupware/index.html. Accessed June 10, 2011.

[13] Baecker, R. M., Grudin, J., Buxton, W. A. S., and Greenberg, S. Readings in Human-Computer Interaction: Towards the Year 2000, 1995, Morgan Kaufmann Publishers, Inc.

[14] Graham, T. C. GroupScape: Integrating Synchronous Groupware and the World Wide Web. INTERACT 1997: 547-554

[15] Wang, W. Powermeeting on Common Ground: Web Based Synchronous Groupware with Rich User Experience, Proceedings of the Hypertext 2008 Workshop on Collaboration and Collective Intelligence, June 19-21, 2008, Pittsburgh, PA, USA [doi>10.1145/1379157.1379166].

[16] Bates, J. The State of the Art in Distributed and Dependable Computing, a CaberNet Sponsored Report, October 1998. Available at http://research.cs.ncl.ac.uk/cabernet/www.laas.research.ec.org/cabernet/sota/report/sota.pdf. Last Accessed June 10, 2011.

[17] Day, M. What Synchronous Groupware Needs: Notification Services. Position Paper for the 6th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VI), Chatham, MA, 1997.

[18] Graham, T. C. N., Phillips, W. G. and Wolfe, C. Quality Analysis of Distribution Architectures for Synchronous Groupware, Proc. CollaborateCom, pp. 1-9, 2006.

[19] Zafer, A. NetEdit: A Collaborative Editor, Master of Science, University of Virginia, USA, 2001, 82 pages.

[20] Begole, J., Smith, R. B., Struble, C. A., and Clifford, A. S. Resource Sharing for Replicated Synchronous Groupware, IEEE/ACM Transactions on Networking, vol. 9 pp. 833-843, Dec. 2001.

[21] Reiss, S. P. The FIELD Programming Environment: A Friendly Integrated Environment for Learning and Development. Kluwer Academic Publishers, Norwell, MA, USA, 1995.

[22] Eclipse Platform Technical Overview. Object Technology International Incorporated, February 2003(updated for 2.1; originally published July 2001). Available at http://www.eclipse.org/whitepapers/eclipse-overview.pdf. Accessed June 10, 2011.

[23] Eclipse Communication Framework. Available at http://www.eclipse.org/ecf/ documentation.php. Accessed June 10, 2011.

[24] Lewis, S. Eclipse Communication Framework. Eclipse Foundation, April 2005. Available at http://www.eclipse.org/ecf/. Accessed June 10, 2011.

[25] Visual Studio Developer Center. Microsoft Corporation, March 2007. Available at msdn2.microsoft.com/en-gb/vstudio. Accessed June 10, 2011.

[26] Hupfer, S., Cheng, L.-T., Ross, S., and Patterson, J. Introducing Collaboration into an Application Development Environment, CSCW'04, Nov. 2004, Chicago, Illinois, vol. 6, Issue 3, 2004 ACM, pp. 21-24.

[27] Cubranic, D., Storey, M.-A. Collaboration Support for Novice Team Programming. In Proceedings of ACM GROUP'05, pages 136–139, November 2005.

[28] Dourish, P., and Bellotti, V. Awareness and Coordination in Shared Workspaces, CSCW November 92 proceedings of ACM.

[29] Ellis, C. A., Gibbs, S. J., and Rein, G. L. Groupware: Some Issues and Experiences, CACM 34(1), 1991, 38-58. Reprinted in Baecker, R.,M. Readings in Groupware and Computer-supported Cooperative Work: Facilitating Human-Human Collaboration, Morgan Kaufmann, 1993.

[30] Baecker, R. M., Nastos, D., Posner, I. R., and Mawby, K. L. The User-centred Iterative Design of Collaborative Writing Software, Proceedings of InterCHI'93, ACM, 1993, 399-405, 541.

[31] Mitchell, A. Communication and Shared Understanding in Collaborative Writing, MS Thesis, University of Toronto, Department of Computer Science, 1996. Available at http://www6.ufrgs.br/limc/escritacoletiva/pdf/com_and_shared_understand.pdf. Accessed June 10, 2011.

[32] Begole, J. Flexible Collaboration Transparency: Supporting Worker Independence in Replicated Application – Sharing Systems, PhD Dissertation, Virginia Polytechnic Institute and State University, Department of Computer Science, 1998.

[33] ITeach, A Collaborative Editing Environment, at Institue of Advanced Studies in Humanities.

[34] Mullick, S. MUSE: A Collaborative editor. Masters Project. University of Kentucky. Available at http://www.cs.engr.uky.edu/~raphael/studentWork/muse.html. Accessed June 10, 2011.

[35] Network Text Editor, an Application Developed at Networked Multimedia Research Group, University College London. Available at http://www-mice.cs.ucl.ac.uk/multimedia/software/nte/. Accessed June 10, 2011.

[36] Schanda, J. WLMH-5: CAMP – A Pair Programming Support Environment. Master of Computing, 77 pages. Available at http://www.dcs.shef.ac.uk/intranet/teaching/projects/archive/ug2005/pdf/u1js2.pdf. Accessed June 10, 2011.

[37] Langton, J., Hickey, T., and Alterman, R. Integrating tools and resources: a case study in building educational groupware for collaborative programming. Journal of Computing Sciences in Colleges, 2004. 19(5): p. 140 – 153.

[38] Cheng, L-T., De Souza, R. B. C., Hupfer, S., Patterson, J., and Ross, S. Building Collaboration into IDEs. ACM Queue 1(9): 40-50, 2003.

[39] Storey, M.-A., Michaud, J., Mindel, M., et al. Improving the Usability of Eclipse for Novice Programmers. OOPSLA Workshop: Eclipse Technology Exchange. pp. 35-39, Anaheim CA, October 2003. Available at http://gild.cs.uvic.ca/docs/publications/oopsla.pdf. Accessed June 10, 2011.

[40] CodeBeamer: Available at http://www.intland.com and http://www.codebeamer.com. Accessed June 10, 2011.

[41] Ho, C., Raha, S., Gehringer, E., and Williams, L. Sangam: A Distributed Pair Programming Plug-in for Eclipse, Eclipse Technology Exchange (Workshop) at the Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) 2004.

[42] Ripley, R., Sarma, A., and Van der Hoek, A. Workspace Awareness in Application Development, Proceedings of Eclipse Technology eXchange Workshop, Vancouver, Canada, 2004, pp. 17–21.

[43] Hattori, L., and Lanza, M. An Environment for Synchronous Software Development. In Proceedings of the 31st International Conference on Software Engineering (ICSE 2009), Vancouver, Canada, May 16-24, 2009.

[44] Frost, R. Jazz and the Eclipse Way of Collaboration, IEEE Software, vol. 24, no. 6, pp. 114-117, Nov./Dec. 2007, doi:10.1109/MS.2007.170.

[45] Cheng, L.-T., Hupfer, S., Ross, S., and Patterson, J. Jazzing up Eclipse with Collaborative Tools, Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange, p.45-49, October 27-27, 2003, Anaheim, California [doi>10.1145/965660.965670].

[46] JBuilder: Available at http://www.codegear.com/products/jbuilder. Accessed June 10, 2011.

[47] Product Review, JBuilder 8. Reviewed by David Neuendorf and Richard Wiener. Available at http://www.jot.fm/products/review4.pdf. Accessed June 10, 2011.

[48] Collaborative Development Environment using Visual Studio. Available at http://research.microsoft.com/en-us/projects/collabvs/default.aspx. Accessed June 10, 2011.

[49] SourceForge. www.sourceforge.net.

[50] Sarma, A., Noroozi, Z., and Van der Hoek, A. Palantír: Raising Awareness among Configuration Management Workspaces, Proceedings of the Twenty-fifth International Conference on Software Engineering, Portland, Oregon, USA, 2003, pp. 444–454.

[51] Boyd, D., Ellison, N. Social Network Sites: Definition, History, and Scholarship. Journal of Computer-Mediated Communication 13 (1), 2007. Available at http://jcmc.indiana.edu/vol13/issue1/boyd.ellison.html. Accessed June 10, 2011.

[52] Liccardi, I., Ounnas, A., Pau, R., Massey, E., Kinnumen, P., Lewthwaite, S., Midy, M. A., and Sakar, C. The Role of social Networks in Students' Learning Experiences. *ACM SIGCSE Bulletin (December Issue)*, 2007. pp. 224-237.

[53] Treese, W. (2008). Social Network? Which One? NetWorker, 12 (1). Retrieved on April 2, 2009, from ACM Digital Library. Digital Library of Technology at Monterrey.

[54] Mitchell-Wong, J., Kowalczyk, R., Quoc Vo, B. Social Network Profile and Policy, Policies for Distributed Systems and Networks, IEEE International Workshop on, pp. 207-210, 2008 IEEE Workshop on Policies for Distributed Systems and Networks, 2008.

[55] Marczak, S., Kwan, I., Damian, D. Social Networks in the Study of Collaboration in Global Software Teams. Poster in IEEE Int'l Conf. on Global Software Engineering (ICGSE 2007), Munich, Germany, 2007.

[56] Willard, T. Social Networking and Governance for Sustainable Development. International Institute for Sustainable Development, March 2009.

[57] Howard, B. 2008. Analyzing online social networks. Commun. ACM 51, 11 (Nov. 2008), 14-16. DOI= http://doi.acm.org/10.1145/1400214.1400220.

[58] Social network downtime in 2008, February 2008. Available at http://royal. pingdom.com/2008/02/26/social-network-downtime-in-2008/. Accessed June 10, 2011.

[59] Google Launches OpenSocial to Spread Social Applications across the Web. Google. 2007-11-01. Available at http://www.google.com/intl/en/press/pressrel/ opensocial.html. Accessed June 10, 2011.

[60] MySpace and Google Join Forces to Launch Open Platform for Social Application Development. Google. 2007-11-01. Available at http://www.google.com/intl/en/ press/pressrel/myspace_opensocial.html. Accessed June 10, 2011.

[61] Developer/Documentation/The OpenSocial API. Introduction to OpenSocial. Available at http://en.netlog.com/go/developer/documentation/article= opensocialapi . Accessed June 10, 2011.

[62] Helft, Miguel; Brad Stone (2007-11-02). MySpace Joins Google Alliance to Counter Facebook. New York Times. The New York Times Company. http://www.nytimes.com/ 2007/11/02/technology/02google.html. Accessed June 10, 2011.

[63] 7 Things You Should Know About Twitter. Available at http://net.educause.edu/ ir/library/pdf/ELI7027.pdf. Accessed June 10, 2011.

[64] Java, A., Song, X., Finin, T., Tseng, B. Why We Twitter: Understanding Microblogging Usage and Communities, Proceedings of the 9th WebKDD and 1st

SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis, p.56-65, August 12-12, 2007, San Jose, California.

[65] Google Launches OpenSocial to Spread Social Applications across the Web. Google. Available at http://www.google.com/intl/en/press/pressrel/opensocial.html. Accessed June 10, 2011.

[66] Communities of Practice. Available at http://www.infed.org/biblio/communities_of_practice.htm. Accessed June 10, 2011.

[67] Social Networking Site for Programmers. Available at http://www.acmsolver.org/index.php?option=com_content&view=article&id=67:social-networking-site-for-programmers&catid=35:web-20&Itemid=27. Accessed June 10, 2011.

[68] IBM's Social Network for Software Developers. Available at http://www.mytechboxonline.com/mtosn/sn-ibmdev-04.html. Accessed June 10, 2011.

[69] SPTechBlog. The SharePoint Technology Blog. Bridging the SharePoint-IBM Divide. Available at http://www.sptechblog.com/2009/05/bridging-sharepoint-lotus-divide.html. Accessed June 10, 2011.

[70] Bringing Social Networking to Jazz. Available at http://sharepointlotus.net/content/bringing-social-networking-jazz. Accessed June 10, 2011.

[71] My developerWorks. Available at https://www.ibm.com/developerworks/mydeveloperworks. Accessed June 10, 2011.

[72] Paul, R. SourceForge Wants to be Collaboration Powerhouse, Buys Ohloh. May 29, 2009. Available at http://arstechnica.com/open-source/news/2009/05/sourceforge-acquires-foss-code-metric-web-site-ohloh.ars. Accessed June 10, 2011.

[73] GitHub Social Coding. Available at http://github.com/.

[74] GitHub. Available at http://www.crunchbase.com/company/github.

[75] GitHub: Making Code More Social. Available at http://radar.oreilly.com.cn/blog/2009/brady/github-making-code-more-social. Accessed June 10, 2011.

[76] Catone, J. GitHub: A Social Network for Programmers. April 11, 2008. Available at http://www.readwriteweb.com/archives/github_a_social_network_for_programmers.php. Accessed June 10, 2011.

[77] Insider Guide to GitHub. Available at http://www.pragprog.com/screencasts/v-scgithub/insider-guide-to-github. Accessed June 10, 2011.

[78] Anderson, G., Anderson, P., Fast, T., and Webster, C. Assemble the Social Web with Zembly. Prentice Hall PTR (1st Ed.). December 2008. Available at http://www.asgteach.com/books/zemblybook.html. Accessed June 10, 2011.

[79] Bani-Salameh, H., Jeffery, C., Al-Sharif, Z., Idoush, I.: Integrating Collaborative Program Development and Debugging within a Virtual Environment. 14th International Workshop, CRIWG 2008, Omaha, Nebraska, September 2008 Proceedings.

[80] Bouras, C., and Tsiatsos, T. Educational Virtual Environments: Design Rationale and Architecture. Multimedia Tools and Applications 29: 153–173. Springer Netherlands, June 2006.

[81] Dictionary.com. Available at http://dictionary.reference.com/browse/virtual environment.

[82] Collaborative Virtual Environments. Available at http://www.crg.cs.nott.ac.uk/ events/CVE98/. Accessed June 10, 2011.

[83] Ellis, S. R., 1994. What are Virtual Environments. IEEE Computer Graphics and Applications, 14(1), pp.17-22.

[84] Kubo, M.M., Tori, R., and Kirner, C.: Interaction in Collaborative Educational Virtual Environments. CyberPsychology & Behavior, v.5, n.5, p. 399-408, October 2002.

[85] Al-Sharif, Z. Debugging With UDB (July 16, 2008), Unicon Technical Report #10. http://www.unicon.org/utr/utr10.html. Accessed June 10, 2011.

[86] Jeffery, C., Mohamed, S., Parlett, R., and Pereda, R. Unicon Book: Programming with Unicon. (1999-2003). Available at http://unicon.org/book/ub.pdf. Accessed June 10, 2011.

[87] Jeffery, C., and Jeffery, S. An IVIB Primer. (February 21, 2006), Unicon Technical Report #6b. http://www.cs.nmsu.edu/~jeffery/unicon/utr/utr6b.pdf. Accessed June 10, 2011.

[88] Bani-Salameh, H., Jeffery, C., Al-Gharaibeh, J.: SCI: Towards a Social Collaborative Integrated Development Environment. Workshop on Social Computing in Education (WSCE09) that held in association with the 2009 International Conference on Social Computing (SocialCom), Vancouver, Canada, August 2009.

[89] Bani-Salameh, H., Jeffery, C., and Al-Gharaibeh, J.: A Social Collaborative Virtual Environment for Software Development. In proceedings of 2010 International Symposium on Collaborative Technologies and Systems (CTS 2010), Chicago, Illinois, May 2010.

[90] Rogers, P., and Lea, M. Social Presence in Distributed Group Environments: the Role of Social Identity (2005). Psychology: Journal Articles (Peer-Reviewed). Paper 11. http://digitalcommons.bolton.ac.uk/psych_journalspr/11. Accessed June 10, 2011.

[91] Annetta, L.A., & Holmes, S. Creating Presence and Community in a Synchronous Virtual Learning Environment Using Avatars (2006). International Journal of Instructional Technology and Distance Learning 3 (8), http://www.itdl.org/Journal/ Aug_06/index.htm. Accessed June 10, 2011.

[92] Garrison, D. R., Anderson, T., and Archer, W. Critical Inquiry in a Text-Based Environment: Computer Conferencing in Higher Education. The Internet and Higher Education 2 (2-3): pp. 87-105.

[93] Annetta, L.A., Folta, E., and Klesath, M. V-Learning: Distance Education in the 21st Century through 3D Virtual Learning Environments, Springer, 2010. Google Book.

[94] Gunawardena, C., and Zittle, F. Social Presence as a Predictor of Satisfaction within a Computer-mediated Conferencing Environment. The American Journal of Distance Education.

[95] Gunawardena, C. Social Presence and Implications for Designing Online Learning Communities. Paper Presented in the Fourth International Conference on Education Technology, July 2005. Available at http://www.edu.cn/include/new_jiaoyuxxh/ xiazai/gunawardena.ppt. Accessed June 10, 2011.

[96] Ehrlich, K., Valetto, G., and Helander, M. Seeing Inside: Using Social Network Analysis to Understand Patterns of Collaboration and Coordination in Global Software Teams, pp.297-298, International Conference on Global Software Engineering (ICGSE 2007), 2007.

[97] Prasolova-Førland, E., and Divitini, M. Collaborative Virtual Environments for Supporting Learning Communities: an Experience of Use, Proceedings of the 2003 International ACM SIGGROUP Conference on Supporting Group Work, Sanibel Island, Florida, USA, 2003, pp. 58 – 67.

[98] Kobylinski, R. (2005), Building Group Awareness in Distributed Software Development Projects, PhD Dissertation, Technische Universität München, 2005.

[99] Gutwin, C., and Greenberg, S. Workspace Awareness for Groupware, Proceedings of the Conference on Human Factors in Computing Systems (Vancouver, Canada, April 13–18, 1996), ACM Press, New York, NY (1996), pp. 208–209.

[100] Gutwin, C., Penner, R., and Schneider, K. Group Awareness in Distributed Software Development, Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work, Chicago, Illinois, USA, 2004, pp. 72 – 81.

[101] Liccardi, H. C., White, D. S., and Southampton, H. S. O. CAWS: Visualizing Awareness to Improve the Effectiveness of Co-authoring Activities, Special Issue of Collaborative Computing in IEEE Distributed Systems Online, 2008.

[102] Gross, T., Stary, C., and Totter, A. User-Centered Awareness in Computer-Supported Cooperative Work-Systems: Structured Embedding of Findings from Social Sciences. International Journal of Human-Computer Interaction - 18(3):323-360, 2005.

[103] Gutwin, C. Workspace Awareness in Real-Time Distributed Groupware. PhD Thesis, Department of Computer Science, University of Calgary, 1997.

[104] Schmidt, K. The problem with 'Awareness':Introductory Remarks on 'Awareness in CSCW'. Computer Supported Cooperative Work, an International Journal - 11(3-4):285-298, 2002.

[105] Gutwin, C., and Greenberg, S. A Descriptive Framework of Workspace Awareness for Real-Time Groupware, Computer Supported Cooperative Work (CSCW), vol. 11, pp. 411-446, 2002.

[106] Bouillon, P., Krinke, J., and Lukosch, S. Software Engineering Projects in Distant Teaching, 18th Conference on Software Engineering Education & Training (CSEET'05), 2005, pp.147-154.

[107] Grinter, R. E., Herbsleb, J. D., and Perry, D. E. The Geography of Coordination: Dealing with Distance in R&D Work, Proceedings of the 1999 International ACM SIGGROUP Conference on Supporting Group Work, Phoenix, Arizona, USA, 1999, pp. 306-315.

[108] Herbsleb, J. D., and Grinter, R. E. Architectures, Coordination, and Distance: Conway's Law and Beyond, IEEE Software, vol. 16, no. 5, Sep./Oct, 1999, pp. 63-70.

[109] Schneider, K., Gutwin, C., Penner, R., and Paquette, D. Mining a Software Developer's Local Interaction History, Proceedings of the International Workshop on Mining Software Repositories (MSR), Saint Louis, Missouri, USA, 2005.

[110] Schümmer, T. Lost and Found in Software Space, Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Vol. 9, 2001.

[111] SVN. Subversion. Available online at http://subversion.tigris.org.

[112] CVS. Concurrent Version System. Available online at http://www.nongnu.org/cvs.

[113] Wilson, T., Miller, R. Reducing the Cost of Interruption Using Gradual Awareness Notifications. MIT Computer.

[114] Bailey, B., Konstan, J., and Carlis, J. Measuring the Effects of Interruptions on Task Performance in the User Interface. In Proceedings of the IEEE Conf. on Systems, Man, and Cybernetics 2000, 757–762.

[115] Shen, H., and Sun, C. Flexible Notification for Collaborative Systems. In Proceedings of the ACM Conference on Computer-Supported Cooperative Work, pages 77–86, November 2002.

[116] McGrenere, J., Li, J., Lo, J., Litani, E. Designing Effective Notifications for Collaborative Development Environments. The Smart Internet 2010: 65-87.

[117] Cockburn, A., and Williams, L. The Costs and Benefits of Pair Programming, Cagliari, Sardinia, Italy, June 2000. [Online]. Available: http://citeseerx.ist.psu.edu/ viewdoc/summary?doi=10.1.1.26.9064.

[118] Flor, N. Globally Distributed Software Development and Pair Programming. Commun. ACM 49, 10 (October 2006), 57-58. DOI=10.1145/1164394.1164421 http://doi.acm.org/ 10.1145/1164394.1164421.

[119] McKinney, D., Denton, L. Developing Collaborative Skills Early in the CS Curriculum in a Laboratory Environment, Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education, Houston, Texas, USA, Pages 138-142., March, 2006.

[120] Bartholomew, R. Evaluating a Networked Virtual Environment for Globally Distributed Avionics Software Development, Global Software Engineering, International Conference on, pp. 227-231, 2008 IEEE International Conference on Global Software Engineering, 2008.

[121] Nardi, B., Harris, H. Strangers and Friends: Collaborative Play in World of Warcraft, Proceedings Conference on Computer Supported Cooperative Work 2006, ACM, 2006, pp. 149-158

[122] Bainbridge, W. S. The Scientific Research Potential of Virtual Worlds, AAAA Science, 27 July 2007, pp. 472-476

[123] Bowman, D. A., McMahan, R. P. Virtual Reality: How Much Immersion Is Enough?. IEEE Computer, July 2007, pp. 36-43.

[124] Sas, C., (2005), Sense of Presence. In Ghaoui, C. (ed.) Encyclopedia of Human Computer Interaction, Idea Group, 511-517.

[125] Bowman, D. A., Kruijff, E., LaViola Jr., J. J., and Poupyrev, I. 3D User interfaces: Theory and Practice. Addison Wesley, USA, 2005.

[126] Jeffrey Rubin. Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests. Wiley, 1994.

[127] Ignat, C.-L., and Norrie, M. C. Tree-Based Model Algorithm for Maintaining Consistency in Real-Time Collaborative Editing Systems, Workshop on Collaborative Editing, CSCW 2002, New Orleans, Louisiana, USA, Nov. 2002.

[128] Greenberg, S. Personalizable Groupware: Accommodating Individual Roles and Group Differences. In Proceedings of the European Conference on Computer Supported Cooperative Work, Amsterdam, Sept. 1991, pp.17-32.

[129] Greenberg, S., Marwood, D. Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface. In Proceedings of the ACM Conference on Computer Supported Cooperative Work, North Carolina, October 1994, pp. 207-218.

[130] Ignat, C.-L., and Norrie, M. C. Customizable Collaborative Editor Relying on treeOPT Algorithm, in: Proceedings of ACM, ECSCW, 2003

[131] Ellis, C.A., and Gibbs, S.J. (1989): Concurrency Control in Groupware Systems, Proceedings of the ACM SIGMOD Conference on Management of Data, May 1989, pp. 399-407.

[132] Ressel, M., Nitsche-Ruhland, D. and Gunzenbauser, R. (1996): An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group

Editors', Proc. of ACM Conference on Computer Supported Cooperative Work, November 1996, pp. 288-297.

[133] Vidot, N., Cart, M., Ferrié, J., and Suleiman M. (2000): Copies Convergence in a Distributed Real-Time Collaborative Environment, Proceedings of the ACM Conference on Computer Supported Cooperative Work, Philadelphia, USA, December 2000, pp.171-180.

[134] Pole, W. Virtual Workgroups: Hidden Challenges to Supporting Distributed Teams. A whitepaper 1996-2005.

[135] Prinz, W., and Pankoke-Babatz, U. Tower — Theatre of Work Enabling Relationships. ECRIM News, (42):46–47, 2000.

[136] Damm, C. H., Hansen, K. M., Thomsen, M., and Tyrsted, M. The Knight Project. Collaborative UML Modelling Using Gestures on an Electronic Whiteboard. ECOOP, Poster Session, 1999.

[137] DeSanctis, G., and Gallupe, R.B., A Foundation for the Study of Group Decision Support Systems, Management Science, 33(2), 1987, 589-609.

[138] Johansen, R. Groupware: Computer Support for Business Teams. The Free Press. 1988.

[139] Ellis, C., Gibbs, S., and Rein, G. Groupware: Some Issues and Experiences. Commun. ACM 34, 1 (January 1991), 39-58.

[140] Bafoutsou, G., and Mentzsa, G. Review and Functional Classification of Collaborative Systems, International Journal of Information Management, 2002, Vol. 22 pp.281-305.

[141] Pumareja, D.T., and Sikkel, K. Getting Used with Groupware - A First Class Experience. AI & Society, 20 (2), 2006. pp. 189-201. ISSN 0951-5666

[142] Penichet, V. M. R., Marin, I., Gallud, J. A., Lozano, M. D., and Tesoriero, R. A Classification Method for CSCW Systems, Electronic Notes in Theoretical Computer Science (ENTCS), 168, p.237-247, February, 2007.

[143] Damian, D., Izquierdo, L., Singer, J., and Kwan, I. Awareness in the Wild: Why Communication Breakdowns Occur. In Proceedings of the international Conference on Global Software Engineering. Washington, DC. August 2007.

Curriculum Vita

Hani Ahmad Bani-Salameh
Department of Computer Science
University of Idaho
Moscow, ID 83844
hsalameh@vandals.uidaho.edu

Education

University of Idaho, Moscow, ID
Pursuing Ph.D. Degree in Computer Science
Earned August 2011

New Mexico State University, Las Cruces, NM
Master of Science in Computer Science
Earned December 2006

Irbid National University, Irbid, Jordan
B.Sc. of Science in Computer Science,
Earned July 1999

Publications

A listing of all papers related to the work presented in this dissertation is given here.

Conference Papers

CW'11

Al-Gharaibeh, J., Jeffery, C., Bani-Salameh, H.: ***Building a Collaborative Virtual Environment: A Programming Language Codesign Approach***. In Proceedings of the 2011 International Conference on Cyberworlds (CW 2011), Banff, Canada, October 2011.

ICICS'11

Bani-Salameh, H., Jeffery, C., Idoush, I.: ***Introducing Social Development Environments***. In: The 2nd International Conference on Information and Communication Systems (ICICS 2011), Amman, Jordan, May 2011.

CTS'10

Bani-Salameh, H., Jeffery, C., Al-Gharaibeh, J.: ***A Social Collaborative Virtual Environment for Software Development.*** In Proceedings of the 2010 International Symposium on Collaborative Technologies and Systems (CTS 2010), Chicago, Illinois, May 2010.

WSCE'09

Bani-Salameh, H., Jeffery, C., Al-Gharaibeh, J.: ***SCI: Towards a Social Collaborative Integrated Development Environment***. Workshop on Social Computing in Education (WSCE09) that held in association with the 2009 International Conference on Social Computing (SocialCom), Vancouver, Canada, August 2009.

CRIWG'08

Bani-Salameh, H., Jeffery, C., Al-Sharif, Z., Idoush, I.: ***Integrating Collaborative Program Development and Debugging within a Virtual Environment***. In Proceedings of the 14th International Workshop, CRIWG 2008, Omaha, Nebraska, September 2008.

Book Chapters

Hani Bani-Salameh and Clinton Jeffery (Ed.). (2010). Chapter 02: **Teaching and Learning in a Social Software Development Tool**, Irwin King, *Social Media Tools and Platforms in Learning Environments: Present and Future* (pp. 17-37), Springer 2010.