TeamWATCH: Visualizing Development Activities Using a 3-D City Metaphor to
Improve Conflict Detection and Team Awareness

A dissertation presented to
the faculty of
the Russ College of Engineering and Technology of Ohio University

In partial fulfillment
of the requirements for the degree
Doctor of Philosophy

En Ye
April 2017

© 2017 En Ye. All Rights Reserved.
This dissertation titled
TeamWATCH: Visualizing Development Activities Using a 3-D City Metaphor to
Improve Conflict Detection and Team Awareness

by

EN YE

has been approved for
the School of Electrical Engineering and Computer Science
and the Russ College of Engineering and Technology by

Chang Liu
Associate Professor of Electrical Engineering and Computer Science

Dennis Irwin
Dean, Russ College of Engineering and Technology
ABSTRACT

YE, EN, Ph.D., April 2017, Computer Science

TeamWATCH: Visualizing Development Activities Using a 3-D City Metaphor to Improve Conflict Detection and Team Awareness

Director of Dissertation: Chang Liu

Awareness of others’ activities has been widely recognized as essential in facilitating coordination in a team among Computer-Supported Cooperative Work communities. Several field studies on software developers in large software companies such as Microsoft showed that coworker and artifact awareness are the most common information needs for software developers; however, they are also the most frequently unsatisfied information needs. As a result, they may duplicate work, or create conflicts without knowing the status of others and the whole project.

To address this problem, we propose a new approach to visualize the developer’s activities using a 3-D city metaphor and implement it in a workspace awareness tool named TeamWATCH (Team based Workspace Awareness Toolkit and Collaboration Hub). TeamWATCH extracts awareness information of artifacts, revisions, and developers from their local workspaces, version control repository, and bug tracking system. It then visualizes both real time and history awareness information together in a 3-D common view shared by the whole team. It also highlights active artifacts that are being changed locally via eye-catching animations and provides the customized personal view for each developer.
The main contributions of this dissertation are 1) a 3-D software visualization scheme that improves workspace awareness and enhances team collaboration; 2) the design and implementation of the workspace awareness tool TeamWATCH using this visualization scheme; and 3) evaluations of the effectiveness of such awareness tools using TeamWATCH as an example in maintaining project awareness and detecting and resolving conflicts via three controlled use experiments. The experiment results showed that the subjects using TeamWATCH performed significantly better in software revision history and project evolution comprehension, and early conflict detection and resolution.
DEDICATION

To my wife, two children, and parents.
ACKNOWLEDGMENTS

Every beginning has an ending, so does the journey of my Ph.D. study. I would like to extend my sincerest thanks to all the people who accompanied me throughout this journey.

Firstly, no words can really represent the appreciation that I have to my advisor Dr. Chang Liu. I definitely could not finish my Ph.D. without him. I will never forget the first day when my wife and I arrived in Columbus airport around midnight from China, it was Dr. Liu who picked us up, and arranged the housing for us. It was him who advised me with patience and kindness, gave me the independence to explore new research ideas, and pushed me to do my best. It was him who understood me when I had to find a job before finishing the dissertation to support my family at my father’s late-stage cancer. And it was also him who has been keeping in touch with me and encouraging me to continue working on my dissertation till I finished it.

Secondly, I would like to say thank you to my committee members Dr. Jundong Liu, Dr. Frank Drew, and Dr. Razvan Bunescu, who provided insightful feedback on my dissertation. I also want to thank Dr. Li Xu and Dr. Xia Jing for their interest in my research and serving as college representatives on my committee. I also want to thank my colleagues in the lab, Reuben Dlamini, Ying Zhong, Qing Zhu, Xin Ye, Hiep Dinh, Lev Neiman, Matthew Sargent, Andrew Goodnite, Sertac Ozercan, Jarrod Reuter, Rayan Alsubail and Sai Dong for their help on my research.

Last but not least, I would like to thank my family for their continuous support. My wife and I came to the USA together and has been through thick and thin since then.
During it, we have two lovely children, who motivate me not to give up in the middle and
be a role model of perseverance for them. Three of them always make me forget my
worries when I arrive home and see them. Also without the unconditional support from
my parents, I could not reach so far. I am sure that my father will be very delightful to
know from the heaven that I eventually obtain my Ph.D. degree.
## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>3</td>
</tr>
<tr>
<td>Dedication</td>
<td>5</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>6</td>
</tr>
<tr>
<td>List of Tables</td>
<td>11</td>
</tr>
<tr>
<td>List of Figures</td>
<td>12</td>
</tr>
<tr>
<td>Chapter 1: Introduction</td>
<td>14</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>14</td>
</tr>
<tr>
<td>1.2 Workspace Awareness</td>
<td>16</td>
</tr>
<tr>
<td>1.3 Proposal</td>
<td>18</td>
</tr>
<tr>
<td>1.4 Contributions</td>
<td>20</td>
</tr>
<tr>
<td>1.5 Dissertation Structure</td>
<td>20</td>
</tr>
<tr>
<td>Chapter 2: Related Work</td>
<td>22</td>
</tr>
<tr>
<td>2.1 Existing Workspace Awareness Tools</td>
<td>22</td>
</tr>
<tr>
<td>2.1.1 Palantir</td>
<td>24</td>
</tr>
<tr>
<td>2.1.2 Jazz</td>
<td>25</td>
</tr>
<tr>
<td>2.1.3 FASTDash</td>
<td>25</td>
</tr>
<tr>
<td>2.1.4 Workspace Activity Viewer</td>
<td>26</td>
</tr>
<tr>
<td>2.1.5 War Room</td>
<td>26</td>
</tr>
<tr>
<td>2.1.6 Syde</td>
<td>27</td>
</tr>
<tr>
<td>2.1.7 CollabVS</td>
<td>28</td>
</tr>
<tr>
<td>2.1.8 Celine</td>
<td>29</td>
</tr>
<tr>
<td>2.1.9 TUKAN</td>
<td>29</td>
</tr>
<tr>
<td>2.1.10 State Treemap</td>
<td>30</td>
</tr>
<tr>
<td>2.1.11 Crystal</td>
<td>31</td>
</tr>
<tr>
<td>2.1.12 Summary</td>
<td>32</td>
</tr>
<tr>
<td>Chapter 3: Proposed Solution</td>
<td>37</td>
</tr>
<tr>
<td>3.1 Hypothesis</td>
<td>37</td>
</tr>
<tr>
<td>3.2 Approach</td>
<td>37</td>
</tr>
<tr>
<td>3.2.1 Information Collection</td>
<td>38</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 1. Comparisons among Workspace Awareness Tools ............................................ 35
Table 2. Mapping between the 3-D Building and the Software Project ........................... 44
Table 3. Null Hypotheses and Alternative Hypotheses of TeamWATCH in History Awareness Information Visualization ................................................................. 59
Table 4. Questions that Subjects were Asked to Answer on an Open-source Project, Notepad++ ................................................................................................................ 66
Table 5. Comparison of Average Correctness and Average Time in the First Experiment ......................................................................................................................... 73
Table 6. Comparison of Average Correctness and Average Time in the Second Experiment ...................................................................................................................... 74
Table 7. Mean Satisfaction of the Two Groups in the First Experiment ......................... 80
Table 8. Mean Satisfaction of the Two Groups in the Second Experiment ...................... 81
Table 9. Null Hypotheses and Alternative Hypotheses of TeamWATCH in Real-time Awareness Information Visualization ................................................................. 92
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Differences among workspace awareness, group awareness, and history awareness</td>
</tr>
<tr>
<td>2</td>
<td>TeamWATCH visualization in Second Life (SL) a) Eclipse b) SL c) smoke representing active artifact d) token representing the developer making local changes e) active task f) highlighted artifact</td>
</tr>
<tr>
<td>3</td>
<td>TeamWATCH visualization in Unity 3D (a) revision statistics; (b) developer statistics; (c) a single cylinder; (d) a stack of cylinders; (e) filters; (f) time slider; (g) blue district; (h) transparent cylinders representing deleted files; (i) return to the main menu</td>
</tr>
<tr>
<td>4</td>
<td>c (a) filtering by both the author name and the file name; (b) filtered visualization result; (c) path, action, author, revision date and time of a file’s specific revision</td>
</tr>
<tr>
<td>5</td>
<td>The TeamWATCH architecture diagram</td>
</tr>
<tr>
<td>6</td>
<td>Visualization of the project Notepad++ in TeamWATCH Unity 3D client</td>
</tr>
<tr>
<td>7</td>
<td>Visualization of the project Azureus in TeamWATCH SL client</td>
</tr>
<tr>
<td>8</td>
<td>Comparison of the average correctness and average time spent for each question</td>
</tr>
<tr>
<td>9</td>
<td>Comparison of overall mean correctness, time, and satisfaction for all groups</td>
</tr>
<tr>
<td>10</td>
<td>The developer 6 used TeamWATCH (shown in the right side) to find out the answer to the CVS question (shown in the left side) in the task 1</td>
</tr>
<tr>
<td>11</td>
<td>The developer 6 received IM from his team mate asking about his status in the task 1</td>
</tr>
<tr>
<td>12</td>
<td>The developer 6 noticed the committed changes from the developer 5 via TeamWATCH</td>
</tr>
<tr>
<td>13</td>
<td>The developer 6 synced with the repository to get the latest changes from the developer 5</td>
</tr>
<tr>
<td>14</td>
<td>The developer 6 checked-in his local changes for the task 1</td>
</tr>
</tbody>
</table>
Figure 15. The developer 6 noticed that the developer 5 was also working on the task 2  
......................................................................................................................................... 102

Figure 16. The developer 6 sent an IM message to confirm it with the developer 5..... 102

Figure 17. The developer 6 got the confirmation from the developer 5 about the status of  
the task 2 .................................................................................................................................. 103

Figure 18. The developer 6 synced with the repository to get the latest changes........... 104

Figure 19. The developer 6 finally committed his changes for chapter 7, i.e. task 2 ..... 105

Figure 20. The number of conflicts detected and resolved early by each group......... 106
CHAPTER 1: INTRODUCTION

1.1 Motivation

As reported in [1], software engineers spend about 70 percent of their time on cooperative activities, thus collaboration is essential for software development. Collaboration is also difficult due to the intangible nature of software, which makes it hard for software developers to create a common view of the team. A shared view of a software system can help developers better understand its complexity during collaboration. It also helps them know more about how their work relates to others’ in the context of the entire system. With the increasing size and complexity of software systems produced in large-scale software development, it is more difficult and even impossible to divide them into several independent modules, which in the end incurs a high degree of parallel development [2]. Therefore, coordination among different team members in a team or across teams working on different modules is necessary. With the progress of globalization and outsourcing, a growing number of software development projects are geographically and (or) temporally distributed, which increases the difficulty of collaboration, since increasing distance between team members usually leads to less effective teams [3][4].

Awareness of others’ activities has been widely recognized as essential in facilitating coordination in a team among Computer-Supported Cooperative Work (CSCW) communities. Awareness can be defined as “an understanding of the activities of others, which provides a context for one’s own activities”[5]. Much attention has been given to the importance of awareness in the coordination of software development due to
the complexity and interdependency of software system [6] [7]. According to a two-month field study of “collocated” software developer teams at Microsoft [8], Ko et al. found that the most common information and the second most frequent information these developers are seeking are coworker awareness, i.e. “what a developer’s coworkers have been doing”, and another awareness information need about artifacts, i.e. “How have resources I depend on changed” was ranked as the third common one. However, they are also regarded as two of the most frequently unsatisfied information needs. That is to say, collocated software developers have difficulty acquiring coworker and artifact awareness information. The importance of awareness about coworkers and artifacts and inadequate tool support to obtain it has been substantiated in other two similar studies on software developers at Microsoft [9], [10]. Maintaining group awareness becomes even more difficult in distributed software development. The distributed team not only cannot take advantage of ad-hoc communication commonly used in the collocated situations, but also have to overcome the impact of working across different time zones, languages, and cultures. As a result, team members may duplicate work, or create conflicts without knowing the status of other’s and the whole team [3], [4], which may, in the end, impact the project schedule, and cause the delay in the project delivery [11].

Such problems can be shown in developers’ traditional usage of version control system [12], [13]. By using version control systems such as CVS [14] and SVN [15], developers work independently at most of the times. They usually become aware of others’ activities only when they perform operations (e.g. check-in, check-out, update) on the central repository. As a result, developers usually can only detect conflicts at these
times. For example, when a developer A wants to check in the modified artifact, he finds that another developer B has checked in the same artifact with different modification, which causes a conflict. At this time, the conflict may be too big, thus resolving it (i.e. merging changes, retesting artifact, etc.) is a significant and time-consuming process. It may be still too late even if A can use some tools such as CVS-Watch [14] to get a notification as soon as B checks in modified artifact. To avoid the efforts in resolving conflicts, developers may rush to check in their changes [16], [17].

1.2 Workspace Awareness

To solve these problems and thus enhance developers’ collaboration, tools need to be created to help developers acquire coworker and artifact awareness information easily, quickly and correctly. As suggested by Dourish and Bellotti, information of past activity and information of current activity were two facets of a single view of awareness information [5]. Awareness information includes both awareness of existing project artifacts (such as when the latest revision of an artifact was committed, who has checked-in an artifact most often, and how many revisions are contained in an artifact) and awareness of ongoing activities by other team members (such as who is online or not, which task they are working on, and which artifacts they are manipulating). Gutwin et al. defined workspace awareness as “the collection of up-to-the-minute knowledge a person uses to capture another's interaction with the workspace” [18]. Gutwin et al. also referred to group awareness as “the understanding of who is working with you, what they are doing, and how your own actions interact with theirs” [19]. In this sense, group
awareness is similar to workspace awareness, albeit with a closer focus on people instead of artifacts.

As shown in Figure 1, in software projects, the most important elements are people, artifacts, and tasks. Artifacts are often stored in source code version control repositories with their history preserved. Tasks are often recorded in issue-tracking databases or other project management documents. General workspace awareness in the context of software projects covers all interactions among all elements, whereas group awareness is centered more closely on people. The focus of this dissertation is workspace awareness, centering on source code and its revision history. Software workspace awareness is the understanding of developers’ up-to-date activities on software artifacts, who is working/has committed what at what time, and why.

Figure 1. Differences among workspace awareness, group awareness, and history awareness.
1.3 Proposal

Workspace awareness tools aim to solve this problem by providing developers with awareness information on which artifacts other developers are working on, what kinds of changes they are making, and whether their changes will affect local workspaces [20]. In the version control system example, conflicts could be detected and thus resolved earlier [21] if workspace awareness tools indicate to developer A that developer B is changing or has changed the same artifact he plans to change, no matter whether B’s change has been committed (i.e. in central repository) or not (i.e. in local workspace). Then A can start a conversation with B to discuss their changes to solve the conflict. He can also switch to working on other artifacts first until B has committed the change.

To help developers effectively maintain heightened awareness and enhance their collaboration, a workspace awareness tool should possess the following desirable properties.

- (Property 1) It should inform developers about the overall structure of the code repository in an “at-a-glance” view.
- (Property 2) It should allow developers to easily identify history and other attributes of individual items of interest.
- (Property 3) It should automatically zoom in to or center/highlight on items relevant to current activities of the developer.
- (Property 4) It should allow developers to interactively search for specific information when needed, and do so more efficiently than traditional version control client software.
• (Property 5) As an awareness tool, most of the time, it should remain in a
developer’s peripheral view, not the focal view.

With these in mind, we propose a new approach to visualize workspace awareness
information in a 3-D city metaphor and implement it in a workspace awareness tool,
called TeamWATCH (Team based Workspace Awareness Toolkit and Collaboration
Hub). TeamWATCH informs developers of workspace awareness information by
monitoring team members’ activities on their local workspaces, version control
repository, and bug tracking system. It then extracts and analyzes the awareness
information of artifacts (such as files and folders), revisions, committing developers, and
events (such as adding, deleting, and modifying artifacts), and finally visualizes it real-
time as a common view shared by the whole team using a 3-D city metaphor.

With TeamWATCH, developers can know not only real-time awareness
information (such as who is online or not, which task (i.e. a bug item or a feature request
in the bug tracking system) they are working on, and which artifacts they are
manipulating), but also history information (such as when the latest revision of an artifact
was committed, who has checked-in an artifact most often, and how many revisions are
contained in an artifact). By providing multiple types of information in one 3-D view,
TeamWATCH makes it easier for developers to obtain an overall understanding of the
whole project. Developers can quickly acquire a general awareness of the size of the
project, how many developers have been working on the project, how many revisions has
this project went through, who contributed what to this project, which file went through
what kind of revisions, etc. After obtaining the overview information, developers can use
filters to focus on the ROI (Region of Interest) for detailed information such as what files have been modified in a specific revision; who and at what time committed a specific revision; who and at what time added, modified, deleted a specific file.

1.4 Contributions

The main contributions of this dissertation are:

1. A 3-D software visualization scheme that improves workspace awareness and enhances team collaboration [22];
2. The design and implementation of the workspace awareness tool TeamWATCH using this visualization scheme [22], [23];
3. The evaluations of the effectiveness of such awareness tools using TeamWATCH as an example in maintaining project awareness and detecting and resolving conflicts via three controlled use experiments. The experiment results showed that the subjects using TeamWATCH performed significantly better in software revision history and project evolution comprehension, and early conflict detection and resolution [23].

1.5 Dissertation Structure

The remainder of the dissertation is organized as follows:

- Chapter 2: Related Work – This chapter provides the literature review of existing workspace awareness tools and the comparisons among them.
- Chapter 3: Proposed Solution - This chapter presents the dissertation hypothesis, the proposed approach, and the comparison between ours and other existing work.
• Chapter 4: Implementation - This chapter describes the implementation of the TeamWATCH prototype and its application on open-source projects’ source code history visualization.

• Chapter 5: Evaluations - This chapter presents the three controlled use experiments and our research findings from the experiment results.

• Chapter 6: Conclusions - This chapter summarizes the research contributions of the dissertation.

• Chapter 7: Future Work - This chapter discusses the future direction of the dissertation.
CHAPTER 2: RELATED WORK

2.1 Existing Workspace Awareness Tools

To create a workspace awareness tool, we first need to know what kinds of awareness information software developers are interested in, and where and how they can gain this awareness information. According to [19], developers in open-source projects tend to maintain both a general awareness of the whole team and more detailed knowledge of people that they plan to work with. First, developers maintain a broad awareness of who are the main people working on their project, and what their areas of expertise are. They obtain these kinds of awareness information from three sources: mailing lists, text chat, and commit logs. Second, when developers plan to work in a particular area, they must gain more detailed knowledge about who are the people with experience in that part of the code. Developers maintain this specific awareness by using a variety of information sources available on the project. These information sources include “maintain” field in the source tree, version control repository logs, issue trackers, help from senior developers, and project document. They also post a query to the mailing list. To summarize, open-source developers maintain group awareness by manually “pulling” information from several information sources. This thesis is also true for other open-source and commercial software developers based on the studies in [7]–[10], [24].

According to Gutwin et al. [19], [25]–[27], Pinelle et al. [28], and Storey et al. [20], information about “who”, “what”, “where”, “when” and “how” is essential for collaborations. LaToz and Myers also reported that “Who, when, how, and why was this code changed or inserted” was one of the most frequent questions developers needed, and
code history was one of the most frequent question categories that developers asked [29]. Furthermore, Dagenais et al. reported that newcomers joining a software project needed to explore in an unfamiliar project landscape and get familiar with a number of landscape features, many of which are related to people, artifacts, and tasks [30]. Therefore, providing software history information efficiently to help maintain software awareness not only benefits developers for their team development, but also helps newcomers become familiar with projects quickly. The software awareness is instrumental in software development because design decisions, requirement changes, development priorities, and other important information are often implicitly embedded in the history of source code evolution. Therefore, maintaining heightened awareness improves team software development.

Many tools have been developed to maintain group awareness. Some of them (e.g., SeeSoft [31], COOP/Orm [32], BSCW [33], Xia [34], and Augur [35]) provide awareness of activities based on information currently available at the repository, thus they can only show changes that have already been committed, without real time information on current activities in developers’ local workspaces. Others (i.e. workspace awareness tools) improve them by adding visualization of up to date information of ongoing changes in developers’ local workspaces (e.g., Palantir [36], JAZZ [37], FASTDash [10], Workspace Activity Viewer [38], and War Room [39]). We are interested in creating a workspace awareness tool in this dissertation, so we will only go over workspace awareness tools below.
2.1.1 Palantir

Sarma et al. have developed a workspace awareness tool named Palantir for distributed software developing teams [36][40]. Palantir provides developers the information about others’ activities that are related to their own workspaces, such as what others do on the artifacts they are changing, by monitoring operations done to the central version control repository and local workspaces. Palantir also measures the “severity” of the changes by dividing the number of lines that has been changed by the total number of lines in an artifact. Palantir can not only detect direct conflict (i.e. “arises due to concurrent changes to the same artifact”), but also a type of indirect conflict (i.e. “arises due to ongoing changes in one artifact affecting concurrent changes in another artifact”), that is, those indirect conflicts caused by changes to class signatures [41]. To avoid information overload, Palantir is integrated with an event notification service called Siena, with which developers can register and receive only the events they are interested. Palantir has been integrated into Eclipse as a plug-in, which provides awareness information to developers by using decorators and text annotations on file icons and file names in the Package Explorer [42]. It also supports other kinds of visualizations as a standalone view including a simple ticker tape, tabular form, explorer form, and pairwise comparison among different developers [43]. The effectiveness of Palantir has been verified through controlled experiment on a mock software project, where subjects are undergraduate and graduate students [44].
2.1.2 Jazz

Hupfer et al. have developed Jazz, a plug-in in Eclipse, to support collaboration among small software developing teams [37], [45]. The collaborative features provided in Jazz include “Jazz Band”, chat, IM, “Team Jam”, screen sharing, and “Concert Awareness”. “Jazz Band” is a view in Eclipse used to show the team members’ icons with status information such as whether they are online or not, and which file they are working on currently. Software engineers can touch the user icon to chat with or IM other team members. They can also initiate a chat session under selected code snippets, and chat logs can be saved and linked to those snippets. “Team Jam” is a discussion board used to support asynchronous discussion among the team, and store all the team messages. Developers can also use screen sharing to discuss the project by sharing current code snippets with other team members. “Concert Awareness” provides awareness information about team members’ activities by putting decorators on file icons in the Package Explorer in Eclipse to indicate a file’s change status (e.g. locally saved), with tooltips to show who has made that change.

2.1.3 FASTDash

Biehl et al. have developed a workspace awareness tool named FASTDash for small software developing teams (3-8 programmers) [10]. FASTDash extracts awareness information such as who is working on which file from version control repository and local workspaces. Built on Visual Studio and existing version control systems such as Team Foundation Server, it can detect whether a developer is viewing or editing or debugging a file, and even the specific class or method name if available. It displays the
whole file structure of project code repository as a Squarified Treemap [46], with a
rectangle representing a file. Developers’ file operation activities are shown in the
corresponding rectangles using visual properties such as color, border, highlight, and
icons. Developers can also directly add an annotation to the related rectangle, which can
be easily checked by others. A field study on a software developing team in Microsoft
has shown the effectiveness of FASTDash.

2.1.4 Workspace Activity Viewer

Ripley et al. have developed a 3-D visualization tool named Workspace Activity
Viewer to visualize workspace activities and their evolution on a project-wide basis [38].
Built on Palantir, which focuses on individual developers’ awareness needs, Workspace
Activity Viewer provides developers and managers an overview of all the workspaces’
current activities that can be grouped by developer or artifact. Workspace Activity
Viewer also traces the project evolution history by recording all the activities in the
workspaces. The tool has been tested on five open source projects by visualizing their
simulated workspace activities since the real data is not available.

2.1.5 War Room

O’Reilly et al. have developed a workspace awareness tool named War Room,
based on the metaphor of “War Room Command Console” [39]. War Room extracts
information from version control repository and local workspaces and presents
information in eight linked consoles, which are viewed through a large public display
space. War Room visualizes the whole structure of the code repository using a
hierarchically stacked layout. It also represents the complexity of the software system by
showing a thumbprint of each artifact’s code structure (different code constructs are shown in different colors) in the stacked layout. Also, only the details of active artifacts are shown in the stacked layout, while inactive artifacts represented by a small pixel. Developers’ activities on artifacts are represented using flags, and conflicts represented by coloring artifacts. A five-week case study of the tool has been performed in DivX Networks Inc., a digital media technology company, and the helpfulness of the tool in showing the team progress and detecting conflict has been identified.

2.1.6 Syde

Lile et al. have developed Syde, a collaborative tool (including different Eclipse plug-ins) to provide workspace awareness information to the developers via three lightweight 2-D visualizations [47]. The Word-Cloud view (implemented as a separate view in Eclipse) shows the names of the classes in the project as a cloud such as the view of tagged words for the photos in the website Flickr¹, in which the color of the word represents the last developer who has changed the corresponding file, the order represents the recency of the change, and the size represents the number of the changes. The Buckets view (implemented as a separate view in Eclipse) visualizes each file as a bucket with each change to that file as a square, in which the color represents the developer who has made the corresponding change, the height represents the number of the changes, and the changes is ordered by the time with the latest change shown on top of the bucket. The Decoration View (implemented as the Package explorer decorations in Eclipse) uses an overlay icon to mark the files that are currently being changed locally, an arrow to represent the last person making the change, and a textual annotation to show the

¹ http://www.flickr.com
timestamp of last change. The case study of Syde has been performed on two multi-developer projects (two developers per project) in a programming course.

2.1.7 CollabVS

Hegde and Dewan [48] have developed an extension to Visual Studio named CollabVS to provide various collaboration functionalities for software developers. They applied a collaboration-centered approach, in which each collaborative feature is implemented as a new view in Visual Studio, making it optional for the developers to use. CollabVS support the following collaborative functionalities: (a) the general presence status such as online and the real-time presence up to the method level (e.g. who is modifying which method in which class) (b) various communication means including IM, Voice/Video, and Screen Sharing (c) concurrent access awareness and potential direct and indirect conflict notifications (which artifacts are being modified by who that will be in conflict with the local changes) (d) the local code change sharing among workspaces and the diff between local change and remote change (e) awareness event subscription mechanism (i.e. the developer can choose to watch for any changes to an artifact, and will be notified when the changes happen). The authors have carried out a user study on the CollabVS, involving sixteen software developers divided into pairs working on a programming project together using CollabVS, and the after-experiment survey results showed that the subjects used more than half of the collaborative features and were happy with the tool.
2.1.8 Celine

Estublier and Garcia [49] have proposed to integrate different cooperative process models into the awareness systems to reduce the awareness info overload and implemented them in a system named Celine. The cooperative process models divide the developers’ workspaces into several groups based on the relativity of the tasks they work on, and only awareness info from the other workspaces in the same group will be shown to the developers. Each group has a reference workspace (aka the central repository), and the star topology is usually used to map the relationships between other workspaces and the reference one. Under the star topology, the artifact in the local workspace can have the one or two of the following four states: unchanged (i.e. in sync with the reference workspace), modified (i.e. locally modified), obsolete (i.e. the reference workspace has the newer version), changed (i.e. modified in the remote workspace). Based on the states of the artifacts, developers can know the status of other developers and detect direct conflicts. For example, an artifact with modified and obsolete may prompt the developer to sync to the latest change to avoid the conflict later. Celine has been daily used at STMicroelectronics.

2.1.9 TUKAN

Schummer and Haake [50] have proposed to support collaborative software development, especially for Extreme Programming via different Modes of Collaboration (MoCs). There are eight MoCs with increasing degree of collaboration: offline mode (developers work individually and can only be contacted by email), process level mode (developers can know the current task each other is working on), change level mode
(developers expose all their source code change activities to others), change aware mode (developers can be notified others’ source code change activities), presence level mode (developers expose all their source code changes and browsing activities to others), presence awareness mode (developers can be notified others’ source code change and browsing activities), communication mode (developers can plan and discuss the work together using the planning tool and communicator tools such as IM), and tightly-coupled collaboration tools (developers can share their workspace among each other). Schummer and Haake implemented the above modes in a tool named TUKAN based on the metaphor of software space. In TUKAN, a software space consists of the artifacts the developer is currently working on and their related artifacts such as those being depended on. It is represented as a graph with its nodes corresponding to the software artifacts and the weighted edges between nodes corresponding to their semantic relationships. To visualize presence awareness and change awareness in the software space, different colors of user figures are added to the nodes and the edges showing the degree of close relationships, and different types of weather icons are also added to the nodes and the edges to show the degree of conflicts. TUKAN has been evaluated by two groups for a week: one group is from the author’s institute, and the other is the outside company. Both groups thought TUKAN improved their collaboration, although they didn’t like the tool’s speed.

2.1.10 State Treemap

Molli et al. [51] have proposed to build a workspace awareness widget named State Treemap for a virtual team supporting platform, the Motu environment. State
Treemap displays all the artifacts in the workspace as rectangles in 2-D treemap [52]. The color of the rectangles represents the state of the artifact in the local workspace. An artifact can have one of the following exclusive states: Local Up to Date, Local Modified, Remotely Modified, Need Update (i.e. there is a newer version in the repository), Potential Conflict (i.e. the artifact is being modified locally and remotely), Will Conflict (the artifact is modified locally, and there is a newer version in the repository). State Treemap has been integrated into Motu, which also provides presence awareness via a tool similar to ICQ, and the communication functionalities such as IM and Audio/Video conferencing. Motu has been used by a virtual team of architects and is available for download in SourceForge².

2.1.11 Crystal

Brun et al. [53]–[55] have analyzed the patterns of conflicts between developers’ changes from nine open-source projects totaling 3.4 million lines of code, proposed to use a speculative analysis technique to detect potential direct and indirect conflicts (or textual and higher order conflicts as mentioned in paper), and implemented it in a tool named Crystal. The speculative analysis technique tends to simulate the developers’ typical actions (e.g. publishing their changes to the central or master repository, building and running the test) after committing their changes locally in a distributed version control repository, and report any potential conflicts found during the simulation with the recommended resolutions. The simulation results including the developers’ local repository state in comparison with others’ and guidance on conflict resolution (if there

² http://motu.sourceforge.com
are any) are shown in an unobstructed way and it is up to the developers to decide whether and when to resolve the conflicts.

2.1.12 Summary

From our analysis on workspace awareness tools above, we have been able to make several observations:

- All the tools only extract awareness information from version control repository (and local workspaces). According to the observation in [19] that developers obtain awareness information from several sources, awareness information gained only from a single source is incomplete, which may thus be incorrect, or misleading. Meanwhile, if developers want to know the awareness information from other sources, they have to “pull” information by themselves, which incurs additional efforts.

- All the tools except Workspace Activity Viewer only present developers’ real-time information regarding ongoing changes to artifacts (and may also the latest check-in info related to these artifacts). Although Workspace Activity Viewer records the historical awareness information, it can only show developers the raw data, i.e. a snapshot of all ongoing changes to artifacts at a particular time. Actually, developers are also interested in historical information, especially the statistical results from this information. For example, in the beginning of this section, we have discussed that developers are interested in gaining more detailed knowledge about who are the people with experience in that part of the code. If the statistical information about who has changed each artifact most often can be
provided, developers can ask for help from this person when they have questions on this artifact, since this person usually tends to be more familiar with it.

- Among all the tools, some tools such as Palantir and Jazz display awareness information in a filtered view customized for individual developers: only show information regarding activities related to artifacts that are either included in a developer’s local workspace or artifacts in which he has specifically registered interest. The others such as Workspace Activity Viewer, FASTDash and War Room provide an overview of all the ongoing activities on project’s code repository. FASTDash and War Room even create a layout representing the file structure of project repository and show change information on it. Both of these visualizations are equally important. The filtered view can be helpful in solving the information overload problem for developers especially in a large-scale software project, while the overview layout can show the global state of the entire system so that developers can understand how their work relates to others in the context of the entire system.

- Among all the tools, Palantir, Workspace Activity Viewer, CollabVS, and Celine provide filter mechanism to handle developers’ cognitive load, under which they can only see the changes they are interested. And probably due to the privacy concern, only CollabVS and Celine can enable developers to see the others’ locally-changing or locally-changed code in their workspaces. This function may be helpful in helping developers solve the conflicts quickly by comparing (or
diffing) their own locally change code with the others’ when two or more developers are changing the same artifact.

- Among all the tools, Jazz, CollabVS, and StateTreemap provide communication functionality. FASTDash’s annotation function is actually asynchronous communication among the team. Jazz supports contextual communication, where developers can chat under the context of a certain artifact, and chat logs can be linked to the related code. Without this functionality integrated into the awareness tool, developers usually use other standalone communication tools to chat with others when they find conflicts or other problems on artifacts. This not only incurs the cost of context switch between awareness tools and communication tools, but also makes the valuable chat logs easier to be neglected or even lost.

- Among all the tools, Palantir, CollabVS, TUKAN and Crystal also support detecting indirect conflict through dependency analysis, while the others can only detect direct conflicts, i.e. the same file locally changed by more than one developer.

Comparisons among workspace awareness tools mentioned above are shown in Table 1.
### Table 1

*Comparisons among Workspace Awareness Tools*

<table>
<thead>
<tr>
<th>Tool</th>
<th>Type</th>
<th>Awareness Info Source</th>
<th>Awareness Info Visualization</th>
<th>Awareness Info Filter</th>
<th>Integrated Communication Functionality</th>
<th>Conflict Detection</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Palantir</td>
<td>Eclipse plug-in</td>
<td>Version control repository and local workspaces</td>
<td>2-D views customized for individual developers integrated into Eclipse</td>
<td>Allow developers to select the awareness info in which they are interested, and only notify developers that info</td>
<td>N/A</td>
<td>Both direct and indirect conflict</td>
<td>Controlled experiment on mock software projects in a programming course</td>
</tr>
<tr>
<td>Jazz</td>
<td>Eclipse plug-in</td>
<td>Version control repository and local workspaces and presence</td>
<td>2-D views customized for individual developers integrated into Eclipse</td>
<td>N/A</td>
<td>Synchronous communications such as IM and Screen Sharing, asynchronous communications such as discussion board, and contextual communication based on related source code</td>
<td>Direct conflict</td>
<td>Turned into a software product in IBM</td>
</tr>
<tr>
<td>FASTDash</td>
<td>Standalone tool</td>
<td>Version control repository and local workspaces and presence</td>
<td>2-D common view built on the file structure of project repository shared by the whole team</td>
<td>N/A</td>
<td>Asynchronous communications such as annotations to the visualized file</td>
<td>Direct conflict</td>
<td>Field study on a software developing team at Microsoft</td>
</tr>
<tr>
<td>Workspace Activity Viewer</td>
<td>Standalone tool</td>
<td>Version control repository and local workspaces</td>
<td>3-D common view shared by the whole team</td>
<td>Filter by developer, and filter by artifact</td>
<td>N/A</td>
<td>Direct conflict</td>
<td>Visualization on simulated workspace activities on five open source projects</td>
</tr>
<tr>
<td>War Room</td>
<td>Standalone tool</td>
<td>Version control repository and local workspaces</td>
<td>2-D common view built on the file structure of project repository shared by the whole team</td>
<td>N/A</td>
<td>N/A</td>
<td>Direct conflict</td>
<td>Case study in a real software developing company</td>
</tr>
<tr>
<td>Tool</td>
<td>Description</td>
<td>Version control repository and local workspaces</td>
<td>2-D views customized for individual developers integrated into tool</td>
<td>Notification strategy</td>
<td>Conflict Resolution</td>
<td>Case Study</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------</td>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------</td>
<td>-----------------------</td>
<td>---------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>Syde</td>
<td>Eclipse plug-in</td>
<td>Version control repository and local workspaces</td>
<td>2-D views customized for individual developers integrated into Eclipse</td>
<td>N/A</td>
<td>N/A</td>
<td>Direct conflict</td>
<td>Case study on two multi-developer projects in a programming course</td>
</tr>
<tr>
<td>CollabVS</td>
<td>The extension to Visual Studio</td>
<td>Version control repository and local workspaces and presence</td>
<td>2-D views customized for individual developers integrated into Visual Studio</td>
<td>Allow developers to select the awareness info they are interested in, and only notify developers that info</td>
<td>Synchronous communications such as IM, audio/video, and screen sharing</td>
<td>Both direct and indirect conflict</td>
<td>User study on software engineers at Microsoft</td>
</tr>
<tr>
<td>Celine</td>
<td>Standalone tool</td>
<td>Version control repository and local workspaces</td>
<td>2-D views customized for individual developers</td>
<td>Apply different strategies to provide developer only relevant info</td>
<td>N/A</td>
<td>Direct conflict</td>
<td>Daily used by engineers at STMicroelectronics</td>
</tr>
<tr>
<td>TUKAN</td>
<td>A plug-in for the Smalltalk system</td>
<td>Version control repository and local workspaces and presence</td>
<td>2-D views customized for individual developers integrated into Smalltalk</td>
<td>N/A</td>
<td>Synchronous communications such as IM and screen sharing, and asynchronous communications such as Email</td>
<td>Both direct and indirect conflict</td>
<td>Case studies on the authors’ research group and a software company for one week</td>
</tr>
<tr>
<td>State Treemap</td>
<td>Integrated into a platform named “MOTU”, which is an open-source project</td>
<td>Version control repository and local workspaces and presence</td>
<td>2-D views customized for individual developers</td>
<td>N/A</td>
<td>Synchronous communications such as IM and audio/video</td>
<td>Direct conflict</td>
<td>Used by a virtual team of architects</td>
</tr>
<tr>
<td>Crystal</td>
<td>A standalone tool</td>
<td>Version control repository and local workspaces</td>
<td>2-D views customized for individual developers</td>
<td>N/A</td>
<td>N/A</td>
<td>Both direct and indirect conflict</td>
<td>Case studies on the nine open source projects</td>
</tr>
</tbody>
</table>
CHAPTER 3: PROPOSED SOLUTION

3.1 Hypothesis

The goal of our work is to help software developers maintain group awareness, enhance their collaboration, which in the end improves their efficiency. Here is the hypothesis to capture the goal.

Enabling software developers to get access to the awareness information they need in an easier and faster way, which in turn helps them make better coordination effort including detecting and resolving the emerging conflicts earlier and reducing the number of merge conflicts.

The hypothesis can be realized through the following research objectives

- Inform the awareness information needed
- Enable easier and faster access to the awareness information
- Provide earlier conflict detection and resolution
- Support better group awareness

3.2 Approach

To achieve our research goal, we propose to create a workspace awareness tool that can first extract and integrate awareness information from several sources, and then visualize relevant information to developers in an appropriate 3-D form. It will have the following features:

- It will extract awareness information from a variety of sources, which at least include version control repository (and local workspaces) and issue tracking
system. They are the only two awareness sources that are based on actual manipulations of the project artifacts.

- It will provide developers both real-time awareness information and historical awareness information, and highlight the information in which the developers are more interested.
- It will visualize awareness information in 3-D from in both ways: the filtered view for individual developers, and the overview layout for the whole team.

To create such a tool, we need to consider mainly from three aspects:

3.2.1 Information Collection

What kinds of awareness information we plan to extract from version control system and issue tracking system? We plan to extract the following awareness information from them based on the review of existing work in last chapter:

1. Presence awareness information
   a. The status of a developer (busy, away, ...)
   b. Which task a developer is currently working on?

2. Real-time resource awareness information such as
   a. For each developer, which artifacts he is changing or has changed in his local workspace compared with the latest version in version control repository. The “changing” operation can represent any of the following actions: add, delete, rename, move, modify, update (i.e. check-out), and commit (i.e. check-in).
(b) For each artifact, which developers are changing or have changed it in their local workspaces compared with the latest version in version control repository.

(3) History resource awareness information such as

(a) Who has checked in each artifact most recently (or lastly), and who has checked in each artifact most often?

(b) Who contributed most to the project, and who is the most active recently?

(c) Which artifact has gone through the most number of revisions?

(d) In which time period, developers have worked on the project most actively, and what are the changes in this period?

The resource awareness information will also definitely include the details of each artifact changed by each developer: version number if the artifact is checked in, the time of the change, the person in charge of the change, and the size of the change (e.g. the number of changed lines).

3.2.2 Information Extraction

How to extract and integrate awareness information from version control system and issue tracking system? To extract awareness information, daemons will be created to monitor operations on version control repository and issue tracking system, and store the extracted awareness information in the database. Since these two sources are mainly related to each other through artifacts, we can combine information from them together by artifacts. Integrating data about commits in the version control system and change requests in the issue tracking system can provide developers a historical view of which
artifacts are related to the bug, who is responsible for it, which version of the artifact resolve it, and when.

3.2.3 Information Presentation

How to present relevant awareness information to developers in an appropriate 3-D form? Shneiderman suggests that: “A useful starting point for designing advanced graphical user interfaces is the Visual Information-Seeking Mantra: overview first, zoom and filter, then details on demand.” [56]. We try to follow this mantra when designing the visualization of TeamWATCH.

1. We first create a common overview of awareness information based on the file structure of project repository.

2. Then we use animations to highlight active artifacts (i.e. all the files that have local changes from developers’ workspace and these changes are not yet committed to the central repository). We also create filters by which developers can locate the artifacts in which they are interested quickly. Meanwhile, developers can use zoom function provided by the 3-D platform we leverage to center/highlight the particular artifacts.

3. Finally, the detailed awareness information can be obtained through a pop-up dialog when developers click the artifact visualization.

These three steps fulfilled the five desired properties for a workspace awareness tool in Section 1.3. And we will go through the detailed visualization design in the following paragraphs.
TeamWATCH visualizes presence information by showing developers’ status through the buddy list and showing the task id and summary (Figure 2e) above the head of developers’ avatar. It uses a 3-D city metaphor to visualize both real-time and history artifact awareness information. It uses city buildings to represent files, and city district to represent folders, with the layout of the city representing the overall file structure of the software project. Because human has remarkable perceptual ability to detect changes in size, color, shape, movement, and texture for visual information [56], some of these attributes are used to represent properties of the files and the revisions. Authorship information was mapped onto the color of the cylinders. Different colors represent different developers. Transparency was used to represent whether files were deleted or not. In addition, animations are used to highlight the active artifact information, since developers are more interested in it compared with history information [10]. Animations are created when any developer makes local changes to an artifact, and disappear if all its local changes have been committed or rolled back.
Figure 2. TeamWATCH visualization in Second Life (SL) a) Eclipse b) SL c) smoke representing active artifact d) token representing the developer making local changes e) active task f) highlighted artifact.

The layout of a project was mapped onto the (X, Y) coordinates so that developers know project’s structure through city’s layout. The area of the virtual city represents the size of the project, and the area of each city district represents the size of each folder. The location of a cylinder represents the location of a file in the project. Information on revisions was mapped onto the vertical Z coordinate. TeamWATCH represents a file as a stack of differently colored cylinders. The height of the stacked cylinders represents the number of the revisions the file has. A single cylinder represents a revision with its color mapping to the author of that revision. If successive revisions have the same author, the corresponding cylinders will be combined into one cylinder.
whose height is the sum of the individual heights of all the revisions. Cylinder stack is sorted by date, such that the most recent revision is always on top. In this way, it is easy to guess how many revisions a file has by the number of cylinders, who is the author of the latest version by the color of the top cylinder, and who has committed most revisions in total by which color of the cylinder is highest.

TeamWATCH uses different animations to visualize active (or real-time) artifact information. When a developer changes a file locally, the building will start to emit smoke (Figure 2c) in the color corresponding to that developer, which rises to the sky. Through this animation, developers can easily tell the popularity of a file, and then decide whether or not to make changes on it, since changes may incur conflicts to the other developer who has changed it. If more than one developer is changing the same file, the whole building will be shaking, warning the team about the possible conflict.

The visualization mapping between the 3-D building and the software artifact is shown in Table 2. Some elements of this mapping were also used in CodeCity built by Wettel and Lanza [57], [58], and sv3D (source viewer 3-D) built by Marcus et al. [59]. CodeCity and sv3D are not project awareness tools monitoring developers’ ongoing activities in workspaces or historical information from repositories. They are visualization tools that aim at providing 3-D representations for software systems to help program comprehension. CodeCity visualizes classes (buildings represent classes and districts represent packages); sv3D visualizes lines of source code (each file is represented by a container in which each cylinder represents a line of code).
Table 2

**Mapping between the 3-D Building and the Software Project**

<table>
<thead>
<tr>
<th>Attributes of the building</th>
<th>Attributes of the software artifact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinates on a horizontal surface (X, Y)</td>
<td>Layout of a project (artifacts including files and folders)</td>
</tr>
<tr>
<td>Height (Z)</td>
<td>Number of revisions of a file ordered by the revision time and the latest one is always on top</td>
</tr>
<tr>
<td>Floor</td>
<td>Revision</td>
</tr>
<tr>
<td>Shape (Cylinder vs rectangular block)</td>
<td>Files versus folders</td>
</tr>
<tr>
<td>Color</td>
<td>The author of the revision</td>
</tr>
<tr>
<td>Transparency</td>
<td>Status (deleted or not)</td>
</tr>
<tr>
<td>Smoke emitting from the top of the building</td>
<td>Active artifact that is being changed locally by the developer corresponding to the smoke color</td>
</tr>
<tr>
<td>The building with lights on (glowing)</td>
<td>Highlighted artifact</td>
</tr>
</tbody>
</table>

With the current visualization, it may be difficult for developers to get the awareness information on an artifact if the code base is large. The filters are created to help developers quickly locate the artifacts in which they are interested. The filters allow the developers to select to which project it will be applied. When the developers type any keyword in the filter dynamically, only the artifacts whose name matches the keyword can be shown in the visualization, and other artifacts will become invisible. They can quickly switch back to the normal state by removing keywords in the filters. The filters are only applied to local developer’s view, i.e., other developers’ view is unchanged by the filter operation. Therefore, the filters can be used to create a customized personal view for each developer without affecting other developers’ view.

To get the detailed information of an artifact, developers can click on the corresponding visualization. TeamWATCH displays textual information when necessary to allow developers to examine details. Information such as path, author, action, revision date and time of a specific revision of a file is displayed on the corner of the TeamWATCH GUI when users point mouse cursor to a cylinder representing that
revision. They can also highlight the artifact in which they are interested by making the corresponding building glow (Figure 2f).

As shown in Figure 3, TeamWATCH visualizes a software project as a 3-D city. The city districts (blue blocks) represent folders (Figure 3g). A stack of cylinders in a column (Figure 3d) represents a file. Each cylinder (Figure 3c) represents a revision of this file. Transparent objects (Figure 3h) represent deleted files that no longer exist in the latest revision. The GUI also shows revision statistics (Figure 3a) and developer statistics (Figure 3b) informing users how many revisions the project has gone through, and how many developers contributed to this project with how many revisions. By providing both the 3-D layout visualization and historical statistics, TeamWATCH helps users obtain project’s overview information efficiently. Developers can establish an overall understanding of a project by briefly browsing its visualization. They can quickly establish an awareness of what the layout of the project looks like, how many files and folders there are, who contributed what, how many revisions the project has gone through, which files have been deleted and whether the project structure has been rearranged, etc.
To improve the efficiency of understanding a specific portion of the project, TeamWATCH provides three filters and a time slider. By using the filters (Figure 4a), users can look at only certain files. Using both the file filter and the author filter together, users get information of how often and when a developer modified certain files (Figure 4b). More information about a revision of a file (Figure 4c) is presented when the mouse cursor hovers over the corresponding cylinder. By using a combination of filters, TeamWATCH users can quickly locate the ROI for a particular portion of the project that is the focus of their current collaborative work. For example, “who has contributed to this artifact” can be answered by using the artifact filter, and “what artifacts were added, modified or deleted by this author” can be answered by using the author filter. By using
both artifact filter and author filter, developers can quickly find out when an author added, modified, or deleted an artifact.

Figure 4. TeamWATCH filters (a) filtering by both the author name and the file name; (b) filtered visualization result; (c) path, action, author, revision date and time of a file’s specific revision.

3.3 Comparisons with Existing Workspace Awareness Tools

Compared with existing workspace awareness tools such as Palantir [36], JAZZ [37], and FASTDash [10], TeamWATCH is different in the following ways. Firstly, TeamWATCH not only extracts awareness information from version control repository
and local workspaces, but also from bug tracking system, while all the tools only extract awareness information from version control repository and local workspaces. Secondly, TeamWATCH visualizes both real-time awareness information and history awareness information together in a 3-D city metaphor, which provides the developers a quick and easy way to get the information they need. Thirdly, TeamWATCH highlights active artifacts that are being changed locally via eye-catching animations and also combines the common view for the whole team with the customized personal view for each developer together by using a filter. Thus, it can help developers acquire the awareness information they want on the artifacts in which they are interested quickly. Last but not least, TeamWATCH (Second Life version, aka SecondWATCH) enables developers to interact with each other through avatars with various communication means under the context of 3-D awareness information visualization in SL. This makes them feel they are much more “there”, which in the end enhances the collaboration among team members.
CHAPTER 4: IMPLEMENTATION

4.1 Implementation Background

4.1.1 Second Life

Second Life (SL) [60] is a 3-D online Virtual World (VW) with about one million active users as of 2013 (i.e. the last time we checked before switching to using Unity 3D to develop the visualization client). VW, also known as Synthetic World [61], is a computer-simulated persistent environment similar to the real world. It is usually created for its users (also known as residents) to inhabit and interact through avatars. An avatar is the user’s representation in the VW.

SL provides various communication methods including text chat, Instant Message (IM), group message, and voice chat. Chat is usually used to talk with other residents locally in a public fashion, and all the residents nearby within a certain distance can hear it. IM and group message are used for private conversation among two or more people. Other people cannot hear it even they are nearby. SL supports voice chat through Voice over Internet Protocol (VoIP). SL also provides various gestures and animations to enrich communication.

SL provides powerful creation tools including a 3-D modeling tool, and a scripting language called Linden Script Language (LSL). Anything you imagine can be created by them. Meanwhile, you own the intellectual property of your creations. Most of the contents in SL are actually created by its users using these tools.

SL has made its client software open source, and plan to open-source its server software in the future. SL also supports in-world (i.e. inside SL) and out-world (i.e.
outside SL) interaction mechanism by using XML-RPC and HTTPRequest. By using these open and extensible features, residents have developed various tools bridging SL and real life. For instance, SLoodle [62] is a virtual environment learning system by integrating SL and the Moodle learning management system.

SL-like VWs are well known as immersive and collaborative persistent environments. They support avatar-based communications, a wide spectrum of creative and collaborative activities, and 3-D representation of information. They can also be an immersive environment for software developers to collaborate with each other through avatars. By integrating developers’ avatars and the visualization of awareness information in SL, even members of geographically distributed teams can interact with each other by flying around, pointing to, and manipulating elements of the visualization, which keeps the whole team on the same page. We sense that enriched visualization and collaboration functionalities provided by SL-like environments may help software developers maintain group awareness better and enhance their collaboration.

4.1.2 Unity 3D

Unity 3D is a game engine that allows a developer to create a game once and then publish it on multiple platforms. Originally a Mac-only development tool, Unity is now available for Windows since its version 2.5 release. Developers have the option to publish a Windows application, Mac application, a web player (which requires a user to install the Unity web player), or an iPhone or Wii application if the developer purchases the proper license. Scripting for these games can be done in JavaScript, C Sharp, Boo, or a mixture of the three languages. Creating games with Unity not only involves scripting,
but it also includes connecting various scripts and objects visually in the Unity IDE. For example, to import 3d assets from supported programs a developer can drag the 3d object file directly into the editor. If a script has a variable targeting another game object, the target game object can be dragged on to the variable in order to assign itself as the target. While these drag and drop features are available, using scripts to assign variable values is still possible. Unity games can be created as a single player or multiplayer games. Perhaps the most notable multiplayer game is the MMORPG Fusion Fall, which was created for Cartoon Network.

One of the features of Unity that are used in our visualization is what Unity calls a Prefab. A Prefab is a game object that can be used repeatedly in a scene. Any change to the Prefab model will be updated in any instance of the Prefab currently in use. Suppose that a Prefab of a cube with a script to make it rotate in place was being used in a game. If a developer notices that the cubes are spinning too quickly, the developer only needs to change the speed in the Prefab model. No matter how many copies of the Prefab are in use, all of them will be updated with the slower speed. Suppose Prefabs were used for all of the objects included in a visualization, then if different shapes or behaviors are decided upon at a later date for the visualization, all that must be updated is the shape or behavior of the old Prefab.

4.2 TeamWATCH

TeamWATCH is implemented in a client-server architecture, as shown in Figure 5. Arrows represent information flow. It mainly consists of three components:

TeamWATCH server, TeamWATCH visualization client, and TeamWATCH plug-in.
The server side is implemented as a Java Web service. The Monitor (or Extractor) on the TeamWATCH server extracts the project’s real-time awareness information from the developers’ local workspaces, and the project’s history awareness information from the version control repository and issue tracking system, via TeamWATCH plug-in (currently only available in Eclipse). The TeamWATCH plug-in is built on top of Eclipse CVS plug-in to get notifications for developers’ operations on the local workspace and central repositories, and Eclipse Mylyn plug-in for developers’ current task in the issue tracking system. Then it sends the awareness information to the extractor. The extractor also supports directly extracting project’s historical log information from a repository through SCM (Software Configuration Management) clients such as SVN command in Subversion. For instance, it sends Subversion commands such as “svn log –v >> some_document.txt” to the Subversion client, and obtains a log file of the project. In either way, the awareness information extracted from the client contains information such as author, revision date and time, and files that are being or have been added, modified, or deleted for every revision of the project. The input to the Analyzer is the awareness log information extracted by the Extractor. The Analyzer formats the raw data sent from the extractor, and also calculate the statistical results such as how many check-ins each developer has contributed to the project, how many revisions each file has gone through, etc. before sending the info to the tree-mapping component. The tree-mapping component maps project’s structure information onto 3-D coordinates based on the mapping strategy as described in chapter 3, using the Quantum Treemap algorithm [63]. The tree-mapped output is then serialized into a string through the Serializer component.
The final output of the TeamWATCH Web service is a serialized string to be sent to the TeamWATCH client side for visualization. To avoid repeated analysis, both the original awareness information and the final generated serialized string information are stored. When the project source version tree is updated, only newly committed information needs to be calculated and mapped onto an existing 3-D layout. For example, if the serialized 3-D layout result of a project with 735 revisions is stored in the database, when someone committed a new revision 736, information of revision 1 to 735 can be retrieved from the database; only information of revision 736 needs to be processed.

Figure 5. The TeamWATCH architecture diagram.

The TeamWATCH client was implemented as a standard-alone application using either Second Life or Unity 3D. The client side of TeamWATCH is mainly the
visualization of the project’s real-time and history changes. It obtains information from the Serializer on the server side and presents the final visualization for users. The Unity 3D client can run on multiple platforms including Windows PCs, Mac computers, and iOS devices, with which developers can set up a 2nd display to display awareness information.

The TeamWATCH client in Second Life (aka SecondWATCH) was implemented based on the modified open-sourced SL client viewer. It took advantage of SL’s avatar based on VW to simulate the developer’s workspace, leveraged its 3-D object building feature to create 3-D objects representing software artifacts, and utilized its various communication functionalities (text chat, Instant Message (IM), group message, and voice chat) to provide interaction between team members. Later we found that the SL client is heavy-weighted, having a dependency on SL server, which was not stable and had the performance and latency issue, especially when visualizing the large size of software projects. This affected the usability of the TeamWATCH tool when we did the initial evaluation of the tool. Therefore, we implemented another client in Unity 3D, and the evaluation of the TeamWATCH tool discussed in this dissertation is based on the Unity 3D client.

4.3 Application

An introduction video for TeamWATCH is available on YouTube\(^3\).

TeamWATCH visualization can be loaded on a second personal display for an individual developer or a large display in the shared workspace for the whole team. TeamWATCH was used as our workspace awareness tool when we were developing it, i.e., used to show

\(^3\) http://youtu.be/xPDilTwfySU
awareness information on its own developing process (as shown in Figure 2). It has also been successfully used to visualize the history awareness information of real-world open-source projects including Notepad++, jEdit, Firebird, Hugin, OpenNMS, FreeMind, GUJ, Azureus, etc. Descriptions and screenshots of these visualizations, as well as TeamWATCH software and user guide, are available on the VITAL Lab website. A screenshot of awareness information visualization on Notepad++ project using TeamWATCH Unity 3D client is shown in Figure 6, while visualization of GUJ project using SL client is shown in Figure 7.

Figure 6. Visualization of the project Notepad++ in TeamWATCH Unity 3D client.

4 http://notepad-plus-plus.org
5 http://www.jedit.org
6 http://www.firebirdsql.org
7 http://hugin.sourceforge.net
8 http://www.opennms.org
9 http://freemind.sourceforge.net
10 http://sourceforge.net/projects/guj
11 https://sourceforge.net/projects/azureus/
12 http://vital.cs.ohiou.edu
Figure 7. Visualization of the project Azureus in TeamWATCH SL client.
CHAPTER 5: EVALUATIONS

To determine the evaluation strategies, let us first revisit our research objectives, i.e.

- Inform the awareness information needed
- Enable easier and faster access to the awareness information
- Provide earlier conflict detection and resolution
- Support better group awareness

Based on the research objectives, we plan to separate the evaluation of TeamWATCH into two parts:

(1) We will first evaluate whether and how the awareness tools such as TeamWATCH can help inform the awareness information needed, and enable easier and faster access to the awareness information. To achieve this goal, we will focus this part of the evaluation on TeamWATCH history awareness information visualization, since project revision history information is one of the most frequent question categories that the developers need and ask most.

(2) We will then evaluate whether and how the awareness tools such as TeamWATCH can help provide earlier conflict detection and resolution and support better group awareness. To achieve this goal, we will focus this part of the evaluation on TeamWATCH real-time awareness information visualization since the real-time awareness information is the source to enable early conflict detection and resolution. By combining it with the history awareness information we evaluate in the first part together, we can then evaluate how the overall
awareness information provided by TeamWATCH can support better group awareness.

5.1 Experimental Design of History Awareness

To answer the question of whether and how TeamWATCH help facilitate the understanding of project revision history as compared with traditional source repository clients (e.g. SVN clients without 3D visualization), two controlled experiments were conducted. The experiments were designed to evaluate the efficiency of searching and understanding software history information between subjects using TeamWATCH and subjects using the command-line SVN client and TortoiseSVN (an SVN client with GUI). In each experiment, subjects from both groups were asked to answer the same set of questions, which were based on historical information from the source code version control repository of an open-source project. The first experiment uses a between-subject design. To further eliminate bias introduced by varied subject background and experience, the second experiment uses a within-subject design. In both experiments, subjects’ subjective opinions about the tools and their objective performance data were captured and evaluated.

5.1.1 Research Questions and Hypotheses

The experiment was intended to address the following five research questions.

(1) Does the use of TeamWATCH help increase the correctness of searching software history information from version control repository, compared with traditional SVN clients such as the command-line SVN client and TortoiseSVN?

---

13 Since the study involved human subjects, it was approved by Institutional Review Board (IRB) at Ohio University before the experiments.
(2) Does the use of TeamWATCH help reduce the time to search software history information from version control repository, compared with traditional SVN clients?

(3) Whether users using TeamWATCH are more satisfied with their tool in searching software history information from version control repository, compared with users using traditional SVN clients?

(4) What kind of software history information does TeamWATCH perform significantly better in searching of, compared with traditional SVN clients, and why?

(5) What kind of software history information do traditional SVN clients perform significantly better than TeamWATCH in searching of, and why?

The null hypotheses and alternative hypotheses that correspond to the first three research questions are described in Table 3.

Table 3

*Null Hypotheses and Alternative Hypotheses of TeamWATCH in History Awareness Information Visualization*

<table>
<thead>
<tr>
<th>Null hypothesis</th>
<th>Alternative hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>H10: Using TeamWATCH does not improve the accuracy of searching software history information.</td>
<td>H1a: Using TeamWATCH improves the accuracy of searching software history information.</td>
</tr>
<tr>
<td>H20: Using TeamWATCH does not reduce the time used in searching software history information.</td>
<td>H2a: Using TeamWATCH reduces the time used in searching software history information.</td>
</tr>
<tr>
<td>H30: Users do not consider TeamWATCH helpful in software history information exploration subjectively.</td>
<td>H3a: Users consider TeamWATCH helpful in software history information exploration subjectively.</td>
</tr>
</tbody>
</table>
The first three research questions, which were answered through quantitative analyses, help draw a relatively objective conclusion on the effectiveness of history awareness provided by TeamWATCH in improving developer efficiency of software history information exploration. The fourth and fifth questions were answered through qualitative analyses, which further revealed both advantages that TeamWATCH enjoyed over traditional SVN clients, and shortcomings of TeamWATCH.

5.1.2 Independent and Dependent Variables

In both controlled experiments, the tool used by subjects in searching software history information is the only independent variable, because the intention is to test the null hypotheses and to answer the research questions by comparing the effectiveness and performance of TeamWATCH over a baseline in software history information exploration. Therefore, TeamWATCH and the chosen baseline are the two levels of the independent variable.

5.1.2.1 Choosing a Baseline

To evaluate the general assumption that software history awareness by tool support can improve developer efficiency, this study compares the performance of subjects with history awareness tools such as TeamWATCH and the performance of subjects without any form of awareness tools in doing the same tasks. Therefore, the baseline chosen should be neither any form of awareness tools nor any materials with answers filled in intentionally for the experiment. To represent users searching software history information under their normal working environments but without a history
awareness tool, traditional SVN clients such as SVN command and TortoiseSVN were chosen as the baselines of the experiment.

In the first controlled experiment, control subjects were allowed to use any mainstream Subversion clients. However, all subjects preferred using the command-line SVN client. After the introduction and exercises, they all demonstrated proficiency in combining SVN commands with shell commands such as “svn log -r 711:735 -v -q | grep filename -c” to search, filter historical information from the repository. Therefore, the command line Subversion client was used in the experiment as the baseline.

In the second controlled experiment, a popular Subversion client with GUI, TortoiseSVN \(^\text{14}\), was chosen as the baseline, because it was an easy-to-use Subversion client supporting both SVN operations and 2-D graphical views, and it is an up-to-date SVN client widely used by practitioners. It had been downloaded 2,052,557 times during the period from January 1st to April 1st in 2013 when the experiment was carried out. TortoiseSVN version 1.7.12 released on April 4th, 2013 supporting Subversion 1.7.9 was used in the experiment.

5.1.2.2 Dependent Variable

The dependent variables of the experiment are: (1) the correctness of answers for each question for which information searched from the version control repository of the object system, (2) the time used in answering each question, and (3) subjects’ satisfactions with tools they used in answering each question.

Also recorded is general feedback from subjects toward their feelings in the experiment after they answered all the experimental questions. Such feedback serves as

\(^{14}\) http://tortoisesvn.net/
important complementary material for qualitative analysis of why subjects are satisfied or
unsatisfied with TeamWATCH, the command-line SVN client, or TortoiseSVN.

5.1.3 Contextual Project

An open-source project named Notepad++, a source code editor and Microsoft
Windows Notepad replacement on Sourceforge, was chosen as the object system for this
experiment because of the following reasons: (1) It was a real-world project actively used
by users (It was downloaded 101,005 times from January 1st, 2013 to April 16th, 2013 when the experiment was carried out); (2) Its log files and historical information were
accessible from the Sourceforge Subversion repository; (3) Its development had gone
through over five years and accumulated enough historical information (1035 revisions);
(4) Its size was suitable for running smoothly on the computers used in the study that did
not contain any dedicated graphics cards; (With over 934 files in more than 96 folders, it
was big enough for the visualization to be interesting, but not too large to degrade
graphical performance during the experiment.) (5) It was developed by a small software
team of four members. (This way, the subjects were not forced to learn about too many
developers in the short period of time during the experiment.) Therefore, the Notepad++
project was a suitable project for this experiment on the effectiveness of Team-WATCH
on improving the efficiency of software history exploration for small software teams.

5.1.4 Subjects

In the two controlled experiments, Computer Science undergraduate and graduate
students aged from 19 to 35 were recruited to serve as the subjects. The experiments were

15 “Notepad++ Download Statistics”. Available: http://sourceforge.net/projects/notepad-
plus/files/stats/timeline
voluntary for the subjects, and they did not receive any compensation. Twenty subjects were recruited for the first experiment. Thirteen subjects from a class dual-listed for both undergraduate and graduate CS students were assigned to the treatment group, which used TeamWATCH. Their average experience in version control systems such as CVS, SVN, and GIT were 5.38 months. Seven other subjects were individually recruited and placed in the control group. The control subjects had an average of 13-month experience on version control systems, more than twice as compared to the treatment group. In the treatment group, 7 out of 13 participants had more than one-year experience in 3-D games. Background info on 3-D gaming for the control subjects was not collected because it was irrelevant; the control subjects did not use 3-D tools at all. In the control group, all 7 subjects showed proficiencies in SVN commands. When asked to choose an SVN client, they all chose the command-line SVN client.

In the second controlled experiment, nineteen CS undergraduate and graduate students were recruited. Their average experience with version control systems such as CVS, SVN, and GIT was 4.05 months. Their average experience on 3-D games or 3-D applications was 96.63 months. Twelve of the nineteen subjects were chosen randomly and assigned to group A, and the other seven to group B. The difference in the sizes of the two groups was the result of the random selection, not intentional. To reduce subject dependency on one tool, subjects in both groups were asked to use TeamWATCH and TortoiseSVN in turns to answer two sets of questions in two rounds. In the first round, subjects were asked to work on the Question Set A, with Group A using TeamWATCH and Group B using TortoiseSVN. In the second round, subjects were asked to work on
Question Set B, with Group A using TortoiseSVN and group B using TeamWATCH. The tool swap between the two rounds ensured that no tool could gain an advantage because the subjects may be more familiar with the problem after the first round.

The two cohorts of subjects in the two experiments were from different classes and were prepared differently for the experiments. The designs of the two experiments were also different. Therefore, the results were not comparable between the two experiments.

5.1.5 Tasks

The design of the experimental tasks aims at evaluating the effectiveness of TeamWATCH as compared to traditional SVN clients on helping developers keep aware of software history information. The experimental questions are designed to represent awareness information of software history required by developers in their team activities. Gutwin [18], [19], [25]–[27], [64] considered the following as important awareness information: (1) who did what where and when, (2) how an operation happened, and (3) how an artifact turned into its current state. Pinelle [28] also coded the mechanics of collaboration in which the basic awareness information was described as who, where and what. These principles were followed in our experiment and evaluation question design.

To test our hypotheses, answer the research questions, and evaluate the TeamWATCH tool, sixteen questions in five categories were designed for the subjects to answer, as shown in Table 4. “Who” represents information about developers; “What” represents what happened to the artifacts and what happened in previous revisions; “Where” represents locations (folders) that developers worked on; “When” represents
revision time; and “How” represents how artifacts were developed. This set of questions covers both overall and detailed information of developers, artifacts, revisions, and events. Their answers cover information in the who, what, where, when, and how categories that are frequently asked about in team developments.

Separately, Sillito et al. [65] described 44 types of code-based questions programmers asked in software development, they also observed programmer behaviors around these questions. They categorized these 44 types of questions into 4 categories. They found that questions belonging to the "finding a focus point" or the "expanding a focus point" categories were better supported by tools, and questions in the "understanding a subgraph" or the "connecting groups of subgraphs" categories were only partially supported by tools. Our study tries to find out the impact of history awareness tools such as TeamWATCH in answering both questions of a focus point of the project such as "who at what time revised a file", and questions of the overall project such as "what happened to the whole project in a major revision". Nine of the sixteen experiment questions, shown in Table 4, belong to the “finding or expanding a focus point” categories that need detailed information related to an entity. The other seven questions belong to the “understanding a sub-graph” category that needs overall information of the project.

Alwis et al. [66] assigned 36 types of programmer's conceptual queries into 5 categories, which were inter-class, intra-class, inheritance, declarations, and evolution. Our work focuses on the evolution category to answer questions about files, developers,
and revisions. All sixteen questions in Table 4 fall under the evolution category as defined by Alwis et al.

Table 4

<table>
<thead>
<tr>
<th>Specific Questions</th>
<th>Category</th>
<th>Result (Which tool was better)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1. List all authors who have contributed to the Notepad++ project, then identify who made the most contributions.</td>
<td>Understanding a subgraph</td>
<td>Who</td>
</tr>
<tr>
<td>Q2. List all authors who have contributed to file Notepad_plus.cpp, then identify who last revised this file.</td>
<td>Expanding focus points</td>
<td>TeamWATCH</td>
</tr>
<tr>
<td>Q3. Who created the /trunk/PowerEditor/src/tools/NppShell folder, and in which revision and at what time?</td>
<td>Expanding focus points</td>
<td>TeamWATCH</td>
</tr>
<tr>
<td>Q4. Roughly how many deleted files are there in the repository? 0, 5, 10, 20, 50, 100, 700, 1500?</td>
<td>Understanding a subgraph</td>
<td>TeamWATCH</td>
</tr>
<tr>
<td>Q5. What happened in revision 460?</td>
<td>Finding a focus point</td>
<td>TeamWATCH</td>
</tr>
<tr>
<td>Q6. How many files were modified in the latest revision, and when did the last revision happen?</td>
<td>Expanding focus points</td>
<td>TeamWATCH performed worse than SVN command but better than TortoiseSVN</td>
</tr>
<tr>
<td>Q7. Is it true that &quot;harrybsharry&quot; and &quot;donho&quot; used to work together closely by modifying the same files, but not anymore?</td>
<td>Understanding a subgraph</td>
<td>TeamWATCH</td>
</tr>
<tr>
<td>Q8. Which folders was &quot;harrybsharry&quot; last working on?</td>
<td>Expanding focus points</td>
<td>TeamWATCH</td>
</tr>
<tr>
<td>Q9. When was file WordStyleDlg.cpp last modified?</td>
<td>Expanding focus points</td>
<td>TortoiseSVN</td>
</tr>
<tr>
<td>Q10. When did &quot;yniq&quot; last commit a revision?</td>
<td>Expanding focus points</td>
<td>No significant difference</td>
</tr>
<tr>
<td>Q11. Which file went through the most revisions?</td>
<td>Understanding a subgraph</td>
<td>TeamWATCH</td>
</tr>
<tr>
<td>Q12. Who at which revision modified the file /trunk/PowerEditor/src/Notepad_plus.cpp? List the author's name and revision number.</td>
<td>Expanding focus points</td>
<td>No significant difference</td>
</tr>
<tr>
<td>Q13. List one subfolder that all files inside went through only one revision.</td>
<td>Understanding a subgraph</td>
<td>TeamWATCH</td>
</tr>
</tbody>
</table>

---

16 Question 12 was used only in the first experiment.

17 Question 13, 14, 15 and 16 were used only in the second experiment.
Table 4: continued

| Q14. List three files that were edited by both “donho” and “yniq”. | Understanding a subgraph | TeamWATCH |
| Q15. List all authors who have committed works in folder /trunk/PowerEditor/src/tools/NppShell/src(bitmap. | Expanding focus points | No significant difference |
| Q16. List one subfolder that most files inside were frequently edited by more than one developer. | Understanding a subgraph | TeamWATCH |

The rightmost result column of Table 4 is based on the results of the experiment, which are discussed later in this section.

5.1.6 Procedures

Two separate controlled experiments were conducted. The first experiment evaluates the efficacy of TeamWATCH in answering project history related questions, compared with SVN command. The second experiment focuses on the comparison of TeamWATCH and TortoiseSVN.

The first experiment was conducted with each group separately. Before the start of this experiment, the treatment group was presented with a 10-minute introduction plus a 5-minute exercise (totally 15 minutes of training) of Team-WATCH. During the training period, participants were allowed to ask questions about problems they encountered.

During the experiment, subjects from both groups were asked to answer the same set of questions (Q1 to Q12) as described in Table 4. The treatment group was asked to use TeamWATCH to search information for answers, and the control group was asked to use the SVN command to do the same. Subjects in both groups were provided with answer sheets, on which they were asked to write down their answers for each question, and they were also asked to record the start time and end time of answering each
question. Subjects were asked to answer each question in three minutes. If a subject feels
the tool not suitable for answering a question, or if a subject cannot find an answer for a
question in three minutes, the subject was asked to write down the reason why an answer
cannot be found before skipping to the next question. However, when analyzing the
experimental result, the accurate time was used to calculate the mean value, even though
some subjects spent and record more than three minutes. Both the accuracy of their
answers and the time spent for each question were analyzed later.

After answering each question, all participants were then asked to fill a 5-point
Likert scale survey based on their satisfaction of how helpful they felt using
TeamWATCH or the SVN command to answer each question. Participants’ satisfactions
were used to evaluate the consistency between their subjective feelings and their
objective experimental results.

At the end of the experiment, subjects from the treatment group were asked to fill
another 5-point Likert scale survey to provide their feelings of using TeamWATCH in
this experiment that whether they would like to continue to use TeamWATCH, whether
they would like to introduce this tool to their friends, whether they felt TeamWATCH is
more helpful than traditional Subversion clients such as the SVN command or
TortoiseSVN in searching software history information from repositories, etc. Control
subjects were not asked to answer these questions because they were not exposed to
TeamWATCH. Subjects from both groups were encouraged to provide additional
feedback on their experience in this experiment, which is included in the discussion later
in this section. It took them about an hour for the whole experiment.
The second experiment was conducted slightly differently. To reduce subject dependency on one tool, subjects were divided into two groups and the experiment into two phases. Subjects swapped the tool they used between the two phases. Experiment questions used are shown in Table 4. Q1 to Q11 are the same as in the first experiment. Q13 to Q16 are added to further evaluate how TeamWATCH performs on searching holistic information of project revision history as compared to TortoiseSVN. These questions were divided into two question sets. Question Set A contains Q1, Q3, Q4, Q5, Q7, Q10, Q11, and Q13. Question Set B contains Q2, Q6, Q8, Q9, Q14, Q15, and Q16. Questions in A are independent with questions in B. As discussed in subsection 5.1.4 Subjects, subjects were randomly divided into two groups A and B. In the first phase of the experiment, subjects in Group A were asked to answer Question Set A using TeamWATCH, while subjects in Group B were asked to use TortoiseSVN. In the second phase of the experiment, Group A subjects were asked to answer Question Set B using TortoiseSVN, while Group B subjects were asked to use TeamWATCH. After completing both Questions Sets A and B, all subjects were asked to fill a survey providing feedback on the use of TeamWATCH. Correctness, time spent on searching answers, and satisfaction of using the assigned tool for answering each question were recorded for quantitative analysis. Survey results and general feedback from subjects were recorded for qualitative analysis. It took them about an hour for the whole experiment.
5.2 Experiment Results of History Awareness

The experiment results on efficiency in tasks related to software source code history and satisfaction for the tools are compared in the following sub-sections. General feedback from the subjects is also discussed. The overall results are presented in the end of this section.

5.2.1 Analysis of Correctness and Time

To better understand the difference between TeamWATCH and traditional Subversion clients, comparisons of the mean correctness of answers per person to each question and mean time per person used to answer each question were performed. Figure 8 shows mean correctness scores of answers across subjects to each question. The correctness score of answer for each question was determined by how much correct information is provided by subjects. Partial credits are allowed. For example, for Q1 “List all authors who have contributed to the Notepad+ project, then identify who made the most contributions”, participants were expected to provide five names of developers. They would get a correctness score of 100% if they provided all five names correctly; they would get 80% if they missed one; 60% if they missed two. Figure 8, besides, shows the average time needed per person to answer each question.
In both the first and the second experiments, subjects using TeamWATCH did well in Q1 “List all authors who have contributed to the Notepad+ project, then identify who made the most contributions.”, Q4 “Roughly how many deleted files are there in the repository? 0, 5, 10, 20, 50, 100, 700, 1500?”, and Q11 “Which file went through the most revisions?” For both Q1 and Q4, subjects using TeamWATCH achieved significantly higher efficiency than subjects using SVN command and achieved

---

18 Question 12 evaluates only TeamWATCH and SVN commands. Questions 13 to 16 evaluate only TeamWATCH and TortoiseSVN.
significantly higher correctness than subjects using TortoiseSVN. For Q11, no subjects using SVN command, and only one subject using TortoiseSVN provided a correct answer. Some of them gave up within a short period of time realizing they could not find the answer through SVN commands. Others tried some time before they gave up. This shows that TeamWATCH is an enabling tool when holistic understanding of project history is needed and can help programmers in situations where detail-oriented traditional Subversion clients simply cannot help.

Both the mean correctness and the mean time were taken into account when evaluating the efficiencies of experimental groups in answering each question. A non-parametric statistical test was applied in this comparison since the probability distributions of results from both the treatment group and the control group were unknown, as were common in this type of studies. The Mann-Whitney U test was applied to assess significance levels of the two groups. Results are shown in Table 5 and Table 6, in which n is the number of participants in an experiment group answered the question; U is the Mann-Whitney U test statistic, which is then used to determine P, which in turn indicates whether the result was statistically significant. In order to understand which group achieved a better correctness and which group used less time, the higher mean correctness scores, as well as the lower mean time, were bolded. Results showing statistically significant differences were highlighted. The dark gray color indicates that the treatment group did better than the control group in answering a question; the white color indicates the reverse situation.
| Question | Correctness (%) |  | Time (seconds) |  |
|----------|-----------------|------------------|-----------------|
|          | Correctness (%) | Time (seconds)   |  |
|          | TeamWATCH | SVN cmd | U | P | TeamWATCH | SVN cmd | U | P |
|          | Mean | Mean | Mean | Mean | Mean | Mean | Mean | Mean |
| Q1       | 13 | 98.46 | 7 | 97.14 | 42.5 | 0.648 | 13 | 46.64 | 7 | 176.00 | 3.0 | 0.00 |
| Q2       | 13 | 81.54 | 7 | 77.14 | 43.5 | 0.863 | 13 | 73.10 | 7 | 234.40 | 3.0 | 0.00 |
| Q3       | 13 | 76.92 | 7 | 85.71 | 39.0 | 0.497 | 13 | 69.20 | 7 | 89.60 | 20.5 | 0.58 |
| Q4       | 13 | 80.77 | 7 | 92.86 | 34.5 | 0.273 | 13 | 77.60 | 7 | 161.00 | 8.0 | 0.03 |
| Q5       | 13 | 61.54 | 7 | 92.86 | 23.0 | 0.045 | 13 | 96.30 | 7 | 49.20 | 13.0 | 0.14 |
| Q6       | 13 | 57.69 | 7 | 100 | 14.0 | 0.005 | 13 | 60.00 | 7 | 73.50 | 10.5 | 0.17 |
| Q7       | 13 | 84.62 | 7 | 28.57 | 20.0 | 0.015 | 13 | 37.55 | 7 | 180 | 0 | 0 |
| Q8       | 13 | 92.31 | 7 | 100 | 42.0 | 0.463 | 13 | 52.30 | 7 | 208.00 | 2.0 | 0.00 |
| Q9       | 13 | 76.92 | 7 | 100 | 35.0 | 0.179 | 13 | 43.80 | 7 | 81.60 | 9.0 | 0.05 |
| Q10      | 13 | 61.54 | 7 | 100 | 28.0 | 0.065 | 13 | 44.00 | 7 | 56.20 | 15.0 | 0.31 |
| Q11      | 13 | 84.62 | 7 | 0 | 7.0 | <0.001 | 13 | 36.08 | 7 | 180 | 0 | 0 |
| Q12      | 13 | 50.77 | 7 | 75.71 | 27.5 | 0.131 | 13 | 85.70 | 7 | 231.00 | 8.0 | 0.09 |

The treatment group using TeamWATCH did significantly better in the dark gray rows.
The light gray rows indicate no significant difference.
The control group using SVN commands did significantly better in the white rows.
Table 6

Comparison of Average Correctness and Average Time in the Second Experiment

<table>
<thead>
<tr>
<th>Question</th>
<th>TeamWATCH Correctness (%)</th>
<th>TortoiseSVN Correctness (%)</th>
<th>U</th>
<th>P</th>
<th>TeamWATCH Time (seconds)</th>
<th>TortoiseSVN Time (seconds)</th>
<th>U</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n Mean</td>
<td>n Mean</td>
<td></td>
<td></td>
<td>n Mean</td>
<td>n Mean</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q1</td>
<td>12 100</td>
<td>7 78.57</td>
<td>24</td>
<td>0.016</td>
<td>12 45.25</td>
<td>7 89.43</td>
<td>24.5</td>
<td>0.138</td>
</tr>
<tr>
<td>Q2</td>
<td>7 88.57</td>
<td>12 81.67</td>
<td>39</td>
<td>0.783</td>
<td>7 79.43</td>
<td>12 93.42</td>
<td>31</td>
<td>0.352</td>
</tr>
<tr>
<td>Q3</td>
<td>12 100</td>
<td>7 82.86</td>
<td>30</td>
<td>0.057</td>
<td>12 73.67</td>
<td>7 134.14</td>
<td>18</td>
<td>0.042</td>
</tr>
<tr>
<td>Q4</td>
<td>12 87.5</td>
<td>7 0</td>
<td>0</td>
<td>0</td>
<td>12 60.42</td>
<td>7 83.43</td>
<td>24.5</td>
<td>0.138</td>
</tr>
<tr>
<td>Q5</td>
<td>12 62.5</td>
<td>7 78.57</td>
<td>31</td>
<td>0.305</td>
<td>12 92.42</td>
<td>7 39.43</td>
<td>15</td>
<td>0.022</td>
</tr>
<tr>
<td>Q6</td>
<td>7 85.71</td>
<td>12 33.33</td>
<td>20</td>
<td>0.032</td>
<td>7 69.43</td>
<td>12 50.92</td>
<td>27.5</td>
<td>0.219</td>
</tr>
<tr>
<td>Q7</td>
<td>12 91.67</td>
<td>7 57.14</td>
<td>31</td>
<td>0.224</td>
<td>12 76.67</td>
<td>7 95.57</td>
<td>32.5</td>
<td>0.421</td>
</tr>
<tr>
<td>Q8</td>
<td>7 85.71</td>
<td>12 83.33</td>
<td>41</td>
<td>0.894</td>
<td>7 115.86</td>
<td>12 68.08</td>
<td>22</td>
<td>0.089</td>
</tr>
<tr>
<td>Q9</td>
<td>7 57.14</td>
<td>12 100</td>
<td>24</td>
<td>0.016</td>
<td>7 47.14</td>
<td>12 47.83</td>
<td>41</td>
<td>0.932</td>
</tr>
<tr>
<td>Q10</td>
<td>12 83.33</td>
<td>7 100</td>
<td>35</td>
<td>0.266</td>
<td>12 73.83</td>
<td>7 45.43</td>
<td>23</td>
<td>0.108</td>
</tr>
<tr>
<td>Q11</td>
<td>12 91.67</td>
<td>7 14.29</td>
<td>9.5</td>
<td>0.001</td>
<td>12 84.92</td>
<td>7 153.29</td>
<td>15</td>
<td>0.021</td>
</tr>
<tr>
<td>Q12</td>
<td>12 70.83</td>
<td>7 14.29</td>
<td>17</td>
<td>0.017</td>
<td>12 57.17</td>
<td>7 123.29</td>
<td>12</td>
<td>0.011</td>
</tr>
<tr>
<td>Q13</td>
<td>7 100</td>
<td>12 33.33</td>
<td>14</td>
<td>0.006</td>
<td>7 84.29</td>
<td>12 150</td>
<td>16</td>
<td>0.027</td>
</tr>
<tr>
<td>Q14</td>
<td>7 92.86</td>
<td>12 100</td>
<td>36</td>
<td>0.190</td>
<td>7 40.71</td>
<td>12 59.75</td>
<td>30.5</td>
<td>0.331</td>
</tr>
<tr>
<td>Q15</td>
<td>7 100</td>
<td>12 41.67</td>
<td>17.5</td>
<td>0.013</td>
<td>7 31.14</td>
<td>12 134.42</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The treatment group using TeamWATCH did significantly better in the dark gray rows.
The light gray rows indicate no significant difference.
The control group using TurtoiseSVN did significantly better in the white rows.

5.2.1.1 Correctness – Research Question 1

As indicated by the results of the first experiment in Table 5, subjects using TeamWATCH achieved significantly higher correctness for Q7 and Q11, but had significantly lower correctness for Q5 and Q6, compared with SVN subjects. In the second experiment, the result as shown in Table 6, TeamWATCH subjects performed significantly better than TortoiseSVN subjects for Q1, Q4, Q6, Q11, Q13, Q14, and Q16. Even though TeamWATCH subjects achieved better correctness than TortoiseSVN subjects, it did not outperform SVN subjects. Therefore, we could not reject H10, which indicates that there is no evidence in this experimental result showing the advantage of
TeamWATCH over SVN commands on the accuracy of searching software history information.

5.2.1.2 Time – Research Question 2

Subjects using TeamWATCH spent significantly less time than subjects using SVN commands for 6 out of 12 questions in the first experiment shown in Table 5. In the second experiment, as shown in Table 6, TeamWATCH subjects used less time than TortoiseSVN subjects for 11 out of 15 questions (significantly for 5 including Q11 and Q13, with which most subjects using TortoiseSVN failed to find an answer in three minutes; non-significantly for 6). It is, therefore, reasonable to reject H20 in favor of H2a, which means that TeamWATCH reduces the time used in searching software history information, compared with the traditional SVN clients.

5.2.1.3 Task Analysis – Research Question 4 and 5

As shown in Table 5 and Table 6, there are significant differences (p < 0.05) between subjects using TeamWATCH and subjects using SVN commands or TortoiseSVN to search overall information about developers (Q1, a “Who” question), overall information of what happened in past revisions and in current revision (Q4, a “What” question), and holistic understanding of the past revisions of a file (Q11, a “How” question). To answer these three questions, subjects using TeamWATCH achieved significantly either higher correctness or higher efficiency than subjects using SVN commands or TortoiseSVN, in both the first and the second experiment. TeamWATCH performed significantly better in searching project overall information to answer these three questions.
In addition, subjects using TeamWATCH in the second experiment achieved significantly higher correctness and efficiency than subjects using TortoiseSVN in obtaining a holistic understanding of how the project was organized (Q13), how different developers cooperated (Q14), and how developers contributed to the same part of the project (Q16). The result of these questions in the second experiment is consistent with the result of the first experiment, which showed that the 3-D visualization in TeamWATCH helps obtain a holistic understanding of the project history more efficiently than traditional SVN clients. This was evidence of the intuitiveness of the TeamWATCH 3-D visualization and indicated that TeamWATCH’s advantage was most obvious when a holistic understanding of the project history was relevant and instrumental.

For Q4 and Q8 in the first experiment, the treatment group demonstrated significantly higher efficiency, but the control group achieved non-significantly higher correctness scores. Both questions required subjects in the treatment group to use TeamWATCH to locate a specific 3-D object among many. It was possible that some subjects located the object, but as they looked up to record the information from the top left corner, their cursors drifted and information of a nearby object was recorded instead. This identified an area to improve for this particular TeamWATCH implementation. But it was not necessarily a weakness for the general 3-D visualization methodology underlying TeamWATCH. In fact, the significantly higher efficiency of the treatment group showed that this could instead be another area of strength.
For Q5 “What happened in revision 460?” and Q6 “How many files were modified in the latest revision? When did the last revision happen”, the control group using SVN commands spent non-significantly less or similar time and achieved significantly higher mean correctness scores. This showed that SVN commands were advantageous to programmers to obtain well-defined, detail-oriented specific information. For Q5, subjects using TortoiseSVN did significantly better as well, likely because that Subversion explicitly logged detailed information of changes in a revision, and that the log information can be easily accessed by SVN commands and TortoiseSVN. In contrast, subjects using TeamWATCH only knew that many files were deleted (as indicated by transparent cylinders) and many others were created in a specific revision. However, they would not know that these two sets of files were related unless they check out detailed information of these objects.

For Q9, Q10, Q12, and Q15, subjects using TeamWATCH obtained non-significantly lower correctness scores (significantly only for Q9 in the 2nd experiment) but spent non-significantly less time (expect for Q10 in the 2nd experiment). To answer Q3, Q9, and Q10, subjects using TeamWATCH had to navigate and locate the ROI for detailed information in the 3-D scene; to answer Q6 and Q12, subjects using TeamWATCH had to count the number of many specific objects. Some subjects may have missed some objects when navigating the 3-D scene, which led to lower mean correctness scores. This may be related to some subjects’ lack of experience with 3-D navigation. Six out of thirteen subjects in the treatment group in the first experiment had less than one year or no experience in 3-D gaming. These six subjects had lower mean
correctness score than the other seven subjects more experienced in 3-D for Q6 (mean score of the 6 less experienced subjects $\mu_1 = 41.67$ versus mean score of the other 7 subjects $\mu_2 = 71.43$), Q9 ($\mu_1 = 66.67$ vs. $\mu_2 = 85.71$), Q10 ($\mu_1 = 50.00$ vs. $\mu_2 = 71.43$), Q3 ($\mu_1 = 66.67$ vs. $\mu_2 = 85.71$), and Q12 ($\mu_1 = 50.00$ vs. $\mu_2 = 51.43$). In contrast, subjects in the second experiment had more experience in 3-D gaming (an average of 96.63 months) and hence achieved higher correctness as shown in Table 6. Subjects with more experience in 3-D gaming were more familiar with navigating the 3-D scene and therefore achieved higher correctness scores.

For all seven questions in the “Understanding a subgraph” category, Q1, Q4, Q7, Q11, Q13, Q14, and Q16 in Table 4, TeamWATCH subjects performed better than SVN commands and TortoiseSVN. This result shows that TeamWATCH is more suitable for answering questions that required a higher-level understanding of the project repository.

Furthermore, the sixteen questions in Table 4 are all related to project evolution, which was reported by Alwis et al. as not being well supported by current tools [66]. Subjects using TeamWATCH did better than subjects using traditional SVN clients in answering ten out of these sixteen questions, achieved either higher correctness or higher efficiency. This result provides evidence for the advantages of TeamWATCH in helping answer conceptual queries related to project evolution.

To summarize, the experiment result showed that (1) TeamWATCH was much better in answering questions that required holistic understanding of software project history, in some cases enabling subjects to answer questions that they could not answer using SVN commands (Q7, Q11) and TortoiseSVN (Q4, Q11, Q13); (2) SVN commands
and TortoiseSVN were more advantageous in obtaining well-defined, detail-oriented specific information (Q5 and Q6 when using SVN commands, Q5 and Q9 when using TortoiseSVN); and (3) TeamWATCH enabled subjects to achieve higher time efficiency (significantly and non-significantly for 12 out of 16 questions designed to cover the full spectrum of awareness issues of "who, what, where, when, and how") in exploring both overall and detailed software history information.

5.2.2 Analysis of the Results on Satisfaction

In the first experiment, subjects in the treatment group were generally satisfied with using TeamWATCH to complete the tasks in the experiment. Even though some of them were not experienced with 3-D games, they felt comfortable using TeamWATCH to search historical information for answers after a 15-minute introduction and training of the tool. To evaluate the difference between satisfactions of both groups, all subjects were asked to respond to a 5-point Likert scale survey (1 = not helpful at all; 5 = the tool did everything I wanted) that whether they felt TeamWATCH or SVN command was helpful in answering each question. Mean satisfaction scores across individuals for each question are shown in Table 7, where n is the number of participants in each group and U is the Mann-Whitney U test statistic.
### Table 7

**Mean Satisfaction of the Two Groups in the First Experiment**

<table>
<thead>
<tr>
<th>Question</th>
<th>TeamWATCH</th>
<th>SVN Commands</th>
<th>U</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>n=13</td>
<td>Mean 4.55</td>
<td>10</td>
<td>0.002</td>
</tr>
<tr>
<td>2</td>
<td>n=13</td>
<td>Mean 4.45</td>
<td>15</td>
<td>0.002</td>
</tr>
<tr>
<td>3</td>
<td>n=13</td>
<td>Mean 3.17</td>
<td>19</td>
<td>0.207</td>
</tr>
<tr>
<td>4</td>
<td>n=13</td>
<td>Mean 3.55</td>
<td>17</td>
<td>0.241</td>
</tr>
<tr>
<td>5</td>
<td>n=13</td>
<td>Mean 4.18</td>
<td>27</td>
<td>0.949</td>
</tr>
<tr>
<td>6</td>
<td>n=13</td>
<td>Mean 3.55</td>
<td>23</td>
<td>0.635</td>
</tr>
<tr>
<td>7</td>
<td>n=13</td>
<td>Mean 3.83</td>
<td>3</td>
<td>0.004</td>
</tr>
<tr>
<td>8</td>
<td>n=13</td>
<td>Mean 3.58</td>
<td>24</td>
<td>0.543</td>
</tr>
<tr>
<td>9</td>
<td>n=13</td>
<td>Mean 4.33</td>
<td>18</td>
<td>0.183</td>
</tr>
<tr>
<td>10</td>
<td>n=13</td>
<td>Mean 3.91</td>
<td>4</td>
<td>0.008</td>
</tr>
<tr>
<td>11</td>
<td>n=13</td>
<td>Mean 4.00</td>
<td>2</td>
<td>0.003</td>
</tr>
<tr>
<td>12</td>
<td>n=13</td>
<td>Mean 1.48</td>
<td>12</td>
<td>0.047</td>
</tr>
</tbody>
</table>

Subjects using TeamWATCH were significantly more satisfied in the dark gray rows.

The light gray rows indicate no significant difference.

Subjects using SVN commands were significantly more satisfied in the white rows.

Subject satisfactions in the second experiment, as shown in Table 8, coincide with the result of the first experiment that subjects were more satisfied with using TeamWATCH than using TortoiseSVN for 10 out of 15 questions (significantly for 7 and non-significantly for 3).
Table 8

Mean Satisfaction of the Two Groups in the Second Experiment

| Question | TeamWATCH | | | TortoiseSVN | | |
|----------|-----------|-----------------|-----------------|-----------------|-----------------|
|          | n | Mean | Standard Deviation | n | Mean | Standard Deviation | U | P  |
| 1        | 12 | 5.0 | 0 | 7 | 4.29 | 1.11 | 24 | 0.017 |
| 2        | 7 | 4.86 | 0.38 | 12 | 3.83 | 0.94 | 14.5 | 0.012 |
| 3        | 12 | 4.42 | 0.67 | 7 | 3.29 | 0.95 | 14.5 | 0.012 |
| 4        | 12 | 3.0 | 0.95 | 7 | 3.14 | 1.46 | 38.5 | 0.759 |
| 5        | 12 | 3.75 | 0.97 | 7 | 4.71 | 0.76 | 18 | 0.03 |
| 6        | 7 | 4.14 | 0.9 | 7 | 4.42 | 0.9 | 33.5 | 0.429 |
| 7        | 12 | 3.75 | 0.62 | 12 | 2.71 | 1.5 | 23 | 0.089 |
| 8        | 7 | 3.29 | 1.6 | 12 | 3.83 | 1.4 | 33.5 | 0.452 |
| 9        | 7 | 4.86 | 0.38 | 12 | 4.67 | 0.89 | 30.5 | 0.842 |
| 10       | 12 | 2.79 | 1.16 | 7 | 4.86 | 0.38 | 5 | 0.001 |
| 11       | 12 | 3.83 | 1.19 | 7 | 2.0 | 1.41 | 12.5 | 0.01 |
| 13       | 12 | 3.92 | 1.44 | 7 | 2.14 | 1.07 | 11 | 0.007 |
| 14       | 7 | 4.29 | 0.76 | 12 | 1.83 | 1.11 | 5 | 0.001 |
| 15       | 7 | 4.71 | 0.49 | 12 | 4.5 | 0.9 | 39 | 0.756 |
| 16       | 7 | 4.0 | 0.82 | 12 | 1.79 | 1.27 | 8 | 0.003 |

Subjects using TeamWATCH were significantly more satisfied in the dark gray rows.
The light gray rows indicate no significant difference.
Subjects using TortoiseSVN were significantly more satisfied in the white rows.

5.2.2.1 Satisfaction – Research Question 3

Results show that the treatment group in the first experiment was more satisfied for 10 out of 12 questions (significantly for Q1, Q2, Q7, Q11, Q12, and non-significantly for Q4, Q5, Q6, Q8, Q9). The treatment group was less satisfied only for 2 out of 12 questions (significantly for Q10 and non-significantly for Q3). In the second experiment, subjects were significantly more satisfied with TeamWATCH for 7 out of 15 questions, and significantly less satisfied for only 2 questions. The total mean satisfaction across individuals using TeamWATCH was 3.92, whereas the total mean satisfaction of subjects using SVN commands was 3.0, and subjects using TortoiseSVN was 3.47. This shows that subject satisfactions with TeamWATCH were higher than subjects with SVN.
commands and TortoiseSVN. Therefore, we consider it sufficient to reject H30 in favor of H3a, which means that users using TeamWATCH are more satisfied with their tool than users using the traditional Subversion client in searching software history information they require.

5.2.2.2 Consistency with Correctness and Time

Furthermore, in the first experiment, the lower mean satisfactions of the control group for Q11 and Q7 (mean = 1.0, standard deviation = 0.0 for Q11; mean = 1.2, standard deviation = 0.45 for Q7) are consistent with the experimental results that none in the control group was able to provide an answer for Q11 and only 1 out of 7 participants found the correct answer for Q7 using Subversion. In comparison, the mean satisfactions of the treatment group were 3.91 for Q11 and 3.83 for Q7.

When being asked what happened in revision 460 (Q5) and how many files were modified in the latest revision (Q6), the treatment group was more satisfied even though their correctness scores were significantly lower. The reason may be the same as discussed in the previous sub-section. Some subjects in the treatment group had less experience in 3-D navigation, and may not be aware that they had missed several 3-D objects (files) by the way they set up their camera positions and angles when viewing the 3-D scene in which there were many objects.

Results also showed that the treatment group was significantly less satisfied than the control group (mean of 2.45 versus 4.60, p < 0.05) for Q10 “What is the last time "yniq" committed a revision”. The reason was because subjects using TeamWATCH needed to filter out objects committed by yniq first, then find out which has the latest
revision number, while the control subjects just need one command “svn log –q | grep yniq”.

It is true in the second experiment that subjects were significantly less satisfied with TeamWATCH for Q5 and Q10, while they did significantly worse than subjects using TortoiseSVN for answering these two questions. Overall, subjects’ satisfaction results are consistent with their efficiency results. Subjects achieved better time efficiencies and were more satisfied with TeamWATCH than using SVN commands or TortoiseSVN, in most cases.

5.2.3 Qualitative Analysis of General Feedback

Subjects from the treatment group in the first experiment and all subjects in the second experiment were asked to respond to general feedback questions on TeamWATCH usability on a 5-point Likert scale at the end of this experiment. Four subjects from the treatment group (selected based on their availability) and all control subjects were interviewed on their use of TeamWATCH and Subversion, respectively.

Sixteen out of thirty-two (50%) subjects in both the first and second experiment agreed or strongly agreed with the statement that they would like to use TeamWATCH in their future projects involving version control. Only 5 out of 32 (15.6%) disagreed or strongly disagreed. Furthermore, 2 interviewees stated that they would like to use TeamWATCH if this tool is compatible with their future projects and is accessible. One subject commented that the 3-D city view made searching software history information like playing an interesting game with fun.
For the statement that “I would recommend TeamWATCH to other developers”, 16 subjects (50%) agreed or strongly agreed. Only 3 (9.4%) disagreed or strongly disagreed.

One subject in the treatment group commented that he felt TeamWATCH would be helpful for those who had little experience with the command-line Subversion client or traditional Subversion GUI clients such as TortoiseSVN. The learning curve was much smoother. Separately, one interviewee from the control group said that he felt more experience with Subversion would be instrumental in completing the tasks. Both of these suggest that TeamWATCH could be particularly helpful to programmers who were not experienced in traditional Subversion clients.

Nineteen out of thirty-two (59.4%) subjects in both the experiments agreed or strongly agree with statement “Team-WATCH was more helpful than Subversion command line and traditional GUI clients”. But 9 subjects were neutral with this statement. One subject in the treatment group of the first experiment happened to be among those who were interviewed. He said that he circled neutral because he knew little about the SVN commands and TortoiseSVN for comparison.

Twenty-eight out of thirteen-two (87.5%) subjects agreed or strongly agreed with statement “the visualization is intuitive and easy to understand”.

One subject in the first experiment said that he felt it was hard to adjust the camera when navigating the visualization. He suggested a smoother rotation of the camera when zooming in and zooming out.
One control subject wrote down a piece of feedback next to the answer for Q1 “List all authors who have contributed to the Notepad+ project, and list who made the most contributions” in the answer sheet that he had to list the whole log and count the number manually. This explains why the treatment group achieved significantly higher efficiency in answering Q1. This is also consistent with our assumption that statistical information and overview information in a 3-D view can help improve the efficiency of exploring software history information.

Overall, the subjective feedback was consistent with the objective experimental results and offered support to the conclusion that TeamWATCH helped improve efficiency in software history information exploration.

5.2.4 Overall Results

Figure 9 shows the overall mean correctness score, the overall mean time, and the overall mean satisfaction score across questions and subjects in both the treatment group and the control group. Results show that subjects using TeamWATCH achieved an overall mean correctness of 81.05% with an overall mean time of 65.91 seconds. In contrast, subjects using SVN commands spent more than twice as much time (144 seconds) but only achieved a slighter lower mean correctness score (79.2%). Subjects using TortoiseSVN obtained a 60% correctness score by spent an average time of 91.23 seconds. This shows that subjects using TeamWATCH spent 27% less time and achieved higher correctness. In terms of satisfaction, on a 5-point Likert scale (1 = not helpful and 5 = very helpful), the overall mean satisfaction of subjects using TeamWATCH was 3.92, whereas the overall mean satisfaction across individuals using TortoiseSVN was 3.47 and
individuals using SVN commands was 3. The higher mean satisfaction of subjects using TeamWATCH is consistent with its lower mean time spent. This suggests that subjects using TeamWATCH achieved higher efficiency, and was more satisfied with TeamWATCH than subjects using SVN commands and TortoiseSVN.

![Figure 9. Comparison of overall mean correctness, time, and satisfaction for all groups.](image)

The experimental result confirmed the effectiveness of TeamWATCH in helping answer questions related to software source code historical information. Subjects in the treatment group, in general, demonstrated competence in searching for information using the 3-D user interface, even though all of them were new to TeamWATCH, and some were even unfamiliar with 3-D environments before the 15-minutes training. One participant with more than one-year Subversion experience commented that a 3-D user interface was more understandable than traditional SVN clients and thus improved his confidence using this tool. When being asked, “Is it true that ‘harrybsharry’ and ‘donho’ used to work together closely by modifying the same files, but not anymore” (Q7),
subjects in the treatment group provided correct answers quickly by finding whether two colors representing two authors overlapped on the same locations. In contrast, most subjects using SVN commands and TortoiseSVN gave up on this question, and one explained that it would be too time-consuming to compare all works of one author’s with all of another’s. Through intuitive perception of colors, shapes, and spatial relationship in a 3-D visualization, developers can easily become aware of a project’s overall historical information, which may be time-consuming for users of conventional SVN clients to obtain from the repository. This experimental result clearly demonstrates that TeamWATCH possesses Properties 1, 2, and 4 in Section 1.3.

5.3 Threats to Validity of History Awareness Experiment

5.3.1 Internal Validity

5.3.1.1 Subjects

The number of subjects in the experiment is low. In addition, the difference between the number of subjects in the treatment group (13 subjects in the first experiment) and the number of subjects in the control group (7 subjects in the first experiment) is also a potential threat. In the second experiment, Group A had 12 subjects, but Group B had only 7 subjects. The results of the comparison between two groups would be more accurate if we could recruit more subjects.

Subjects in both experiments were CS graduate and undergraduate students, and they had low experience in Subversion, compared to experienced professional software engineers. However, before the experiments, subjects were presented with an introduction of Subversion and were asked to use SVN commands and TurtoiseSVN to
answer exercise questions similar to the experimental questions. They were comfortable with using SVN commands and TurtoiseSVN to answer the experimental questions after the brief training.

Separately, in the first experiment, subjects in the treatment group were less experienced in Subversion than those in the control group. This is likely not a threat because it is conceivable that the difference in efficiency could be even greater if similar groups were used for both the control group and the experiment group.

5.3.1.2 Baseline

The baseline in this study was SVN commands in the first experiment and TortoiseSVN in the second experiment. These two traditional Subversion clients are the most popular tools currently available for searching software revision history information. When being asked to choose whatever SVN clients they felt comfortable with, controlled subjects in the first experiment chose SVN commands, and they searched the project repository using various SVN commands and shell commands proficiently.

5.3.1.3 Tasks

To mitigate the threat that the experimental question design may be biased to the advantage of TeamWATCH, the most common software history information required frequently by developers were selected. In the designed tasks, some questions requiring detailed information (such as how many and what files were modified in the last revision) were more biased to the advantage of traditional version control clients.
5.3.1.4 Data Difference

The treatment group searched information for answers using TeamWATCH, while the control group used SVN commands or TortoiseSVN. The information required to answer every question was from the same software repository. Therefore, there was no bias for data between two groups.

5.3.2 External Validity

5.3.2.1 Subjects

Subjects in both the control group and the experiment group were Computer Science students. Their experience in software development was different from real-world professional software developers, who were the ultimate target audience of this prototype tool.

5.3.2.2 Baseline

In this study, the direct comparison was among TeamWATCH, SVN commands, and TortoiseSVN. The immediate result from the experiment did not directly cover other Subversion clients with a graphical representation of certain repository information. However, TortoiseSVN is an easy-to-use client for both SVN operations and 2-D graphical views integrated. It is a popular SVN client widely used by many developers. It represents one of the most efficient SVN clients currently used by developers. The direct comparison study between TeamWATCH and TortoiseSVN is appropriate to obtain the conclusion that TeamWATCH allows developers to be more efficient in search for source code historical information than conventional SVN clients.
5.3.2.3 Object System

The experimental project (Notepad++) chosen in this study was a medium size project with more than 934 files and 1035 revisions developed by four developers. While further evaluation of TeamWATCH’s effectiveness on larger sized projects (e.g. those with more than 1000 files and more than 10000 revisions) developed by a larger team is desirable, current results from this experiment are informative and worthwhile to share with the Software Engineering community.

5.3.2.4 Tasks

The sixteen questions used in the two controlled experiments did not cover all types of questions developers may encounter in collaborative work. Nevertheless, these questions were designed based on the “who, what, when, where, how” criterion and represented the most frequently encountered questions regarding software source code history.

5.3.2.5 TeamWATCH

The experiment designers were aware that TeamWATCH did not represent all history awareness tools. Therefore, even though the experiment would be a fair evaluation of the TeamWATCH tool, to generalize the specific outcome from TeamWATCH for software history awareness, in general, could be a threat to the validity of the general result. However, better tool designs will likely only produce even better results than what has been shown with TeamWATCH.
5.4 Experimental Design of Real-time Awareness

To answer the question of whether and how TeamWATCH helps maintain group awareness, improve development efficiency, and thus enhance team collaboration, a controlled experiment was conducted\(^{19}\). The experiment was designed to evaluate the efficiency of detecting and resolving the conflicts earlier between subjects using TeamWATCH and subjects without using it. In the experiment, subjects from both groups were randomly divided into teams of two, each of them working together to finish a few tasks from a text editing project hosted in CVS repository. Subjects’ subjective opinions about the tool and their objective performance data were captured and evaluated.

5.4.1 Research Questions and Hypotheses

The experiment was intended to address the following two research questions:

1. Does the use of TeamWATCH help developers with it detect and resolve the potential conflicts earlier, thus reduce the number of the merge conflicts, compared with the developers without using TeamWATCH?

2. Does the use of TeamWATCH help developers with it maintain group awareness better, compared with the developers without using TeamWATCH?

The null hypothesis and alternative hypothesis corresponding to the two research questions are listed in the following Table 9.

\(^{19}\) Since the study involved human subjects, it was approved by IRB at Ohio University before the experiment.
Table 2

Null Hypotheses and Alternative Hypotheses of TeamWATCH in Real-time Awareness Information Visualization

<table>
<thead>
<tr>
<th>Null hypothesis</th>
<th>Alternative hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>H40: Using TeamWATCH does not help developers detect and resolve the potential conflicts earlier</td>
<td>H4a: Using TeamWATCH helps developers detect and resolve the potential conflicts earlier</td>
</tr>
<tr>
<td>H50: Using TeamWATCH does not help developers maintain group awareness better.</td>
<td>H5a: Using TeamWATCH helps developers maintain group awareness better.</td>
</tr>
</tbody>
</table>

The first research questions would be answered through quantitative analyses, while the second questions would be answered through qualitative analyses.

5.4.2 Independent and Dependent Variables

In the controlled experiments, the tool used by subjects in finishing the tasks is the only independent variable, because the intention is to test the null hypotheses and to answer the research questions by comparing the effectiveness and performance of TeamWATCH over a baseline in software project development. Therefore, TeamWATCH and the chosen baseline are the two levels of the independent variables.

5.4.2.1 Choosing a Baseline

To evaluate the general assumption that workspace awareness by tool support can improve developer efficiency, this study compares the performance of subjects with workspace awareness tools such as TeamWATCH and the performance of subjects without any form of awareness tools in doing the same tasks. Therefore, the baseline chosen should be none of any form of awareness tools for the experiment. To represent users developing software under their normal working environments but without a workspace awareness tool, Eclipse IDE was chosen as the baselines of the experiment.
5.4.2.2 Dependent Variable

The dependent variables of the experiment are: (1) the number of the potential conflicts detected and resolved to avoid the merge conflicts and (2) subjects’ feedback on whether and how the tools help them maintain group awareness. Also recorded is general feedback from subjects toward their feelings in the experiment after they finished all the experimental tasks. Such feedback serves as important complementary material for qualitative analysis of why subjects are satisfied or unsatisfied with TeamWATCH.

5.4.3 Contextual Project

In the selection of the project, we started with an open-source Java project and came up with a few coding tasks based on it. However, during the trial of the experiment, we found that the subjects mostly only focused on finishing their own coding tasks without having the time to check their team member’s status. This is because that the recruited subjects are undergraduate or graduate students who don’t have much Java programming skills. Although the coding tasks are straightforward, it still took them some time to figure it out. Also, the subjects’ programming skills vary, which would affect the validity of the experiment results. Therefore, we decided to switch to a text editing project, in which the subjects will serve as the editors working on a few text editing tasks. Compared with the coding tasks, the students’ text editing skills are well enough for the experiment, and should not vary a lot to create the bias. Another reason we started from text edit project in the experiment is to evaluate early conflict detection
and resolution, in which case, the edits to the text and the edits to the code are very similar activity that triggers the similar conflicts.

The text editing project is based on a book named "EFF's Guide to the Internet" (formerly “The Big Dummy's Guide to the Internet”), which is written by the staffs from the Electronic Frontier Foundation in 1994. It was chosen as the project for this experiment because of the following reasons: (1) The topic of the book, i.e. the Internet is well known for the subjects, who are computer science students, thus should not create any bias to the subjects on the familiarity of the content. (2) It went through a couple of editions, thus providing the opportunities for the experiment designer to not only create the number of revisions on the repository based on the realistic book revisions but also come up with the related text editor tasks.

5.4.4 Subjects

In the controlled experiment, Computer Science undergraduate and graduate students aged from 19 to 35 were recruited to serve as the subjects. The experiment was voluntary for the subjects, and they did not receive any compensation. Twenty-four subjects were recruited from a class dual-listed for both undergraduate and graduate CS students. Originally we had planned to divide the 24 participants equally into two groups, however, two machines prepared to be used by the treatment group had the issue of running TeamWATCH before the experiment, so we had to move two students to the control group. Among them, ten were randomly picked and assigned to the treatment group, which used TeamWATCH. Their average experience in version control systems such as CVS, SVN, and GIT was around six months (based on the survey results from 8
Fourteen are placed in the control group. The control subjects also had an average of 6-month experience on version control systems (based on the survey results from 12 subjects), similar to the treatment group. In the treatment group, 6 out of 8 participants who finished the pre-experiment survey had less than one-year experience in 3-D games. Background info on 3-D gaming for the control subjects was not collected because it was irrelevant; the control subjects did not use 3-D tools at all.

5.4.5 Tasks

The design of the experimental tasks aims at evaluating the effectiveness of TeamWATCH as compared to traditional IDEs on helping developers keep aware of software awareness information (both history and real-time) and detect potential conflicts. The experimental tasks are designed to simulate the awareness information required by developers that we already discussed in the previous experiments, and potential conflicts they may hit in their team activities, especially the most frequent conflicts that developers hit during daily development [67].

To test our hypotheses, answer the research questions, and evaluate the TeamWATCH tool, five tasks were designed for the subjects to work on, as shown in Appendix A. Each task consists of one subtask asking about the history information of the project, and one or two subtasks that requires making changes on the files in the repositories. Four of these five tasks require two subjects in a team to work on the same file, thus creating the potential conflicts.
5.4.6 Procedures

The experiment was conducted with each group in a different room in the same time. The subjects were randomly assigned to either treatment group or control group. Then the subjects from both groups were randomly assigned to a team of two. All participants were asked to answer some survey questions related to their familiarity with source control system, 3-D games, team development, etc. They also need to set up the experiment by checking out the text edit project from the CVS repository in Eclipse, installing and running a video capture software to record their experiment process. The treatment group also needed to set up TeamWATCH to visualize the repository of the text-editing project. During this period, participants were allowed to ask questions about problems they encountered.

During the experiment, they are required to work on the same set of five tasks (but the two subjects in each team were assigned five different tasks) as described in Appendix A. The treatment group was asked to use Eclipse plus TeamWATCH to finish the task, and the control group was asked to use the Eclipse to do the same. Subjects in both groups were provided with answer sheets, on which they were asked to write down their answers for each subtask asking about the history information of the project, and they were also asked to check in the changes for the text editing subtasks. Subjects were asked to check in at least once per task.

After finishing the five tasks, all participants were then asked to fill out a survey about the overall experiment experience. Then subjects from the treatment group were asked to fill another survey to provide their feelings of using TeamWATCH in this
experiment such that whether they felt TeamWATCH is more helpful in maintaining
group awareness, etc. Control subjects were not asked to answer these questions because
they were not exposed to TeamWATCH. Subjects from both groups were encouraged to
provide additional feedback on their experience in this experiment, which is included in
the discussion later in this section. It took them about an hour for the whole experiment.

5.5 Experiment Results of Real-time Awareness

Overall, we’ve collected the following experiment data

(1) The video recording of each subject’s experiment process via the video recording
software running in the experiment computer. In the experiment group, we
successfully recorded nine (out of ten) subjects’ experimental process, totally
about 366 minutes; in the control group, we collected thirteen (out of fourteen)
subjects’ video recording, totally around 311 minutes, although the recordings of
some subjects in the control group are very short (around eight minutes), not
capturing the whole process. One subject from each group ran into the issue of
video recording, thus could not capture their experiment process.

(2) Each subject’s check-in changes for the five tasks in the CVS repository.

(3) The chat logs between the subjects in the same team during the experiment.

(4) Each subject’s answers to the survey questions before and after the experiment.

Before discussing the experiment results, we would start with a detailed example
of how the subjects in the experimental group performed in the experiment. We take the
developer 6 in the team 3 as an example to show how he performed in the first two tasks.
The developer 6 started the experiment with the first CVS question in the first task, i.e. who last revised the Chapter 1, and he got the answer by checking the top cylinder of visualization of chapter 1 changes in TeamWATCH, as shown in Figure 10 below.

![Figure 10.](image)

Then developer 6 started working on the text edit subtask in the task 1, i.e. In Chapter 1, search the first occurrence of the texts “8-1-N” and “7-1-E”, add the texts “(which stands for "8 bits, 1 stop bit, no parity" -- yikes!)” right after “8-1-N”, and add the texts “(7 bits, 1 stop bit, even parity)” right after “7-1-E”. In the middle of it, he received an IM from his teammate, i.e. the developer 5, asking his status on the task 1, as shown in Figure 11 below.
Figure 11. The developer 6 received IM from his team mate asking about his status in the task 1.

The developer 6 was still working on the task 1, then he noticed that a new cylinder appearing on top of the visualization of the chapter 1, i.e. the newly committed changes from the developer 5 for task 1, as shown in Figure 12 below.

Figure 12. The developer 6 noticed the committed changes from the developer 5 via TeamWATCH.
The developer 6 then synced with the repository to get the latest changes on Chapter 1 from the developer 5, as shown in Figure 13 below.

_Figure 13._ The developer 6 synced with the repository to get the latest changes from the developer 5.

After that, the developer 6 finished his task 1 and checked in the changes, as shown in Figure 14 below.
Figure 14. The developer 6 checked-in his local changes for the task 1.

The developer 6 started working on task 2 (i.e. the chapter 7) and noticed that the developer 5 was also working on the task 2 from the visualization, as shown in Figure 15 below.
Figure 15. The developer 6 noticed that the developer 5 was also working on the task 2.

The developer 6 sent an IM message to the developer 5 to discuss the status of task 2, as shown in Figure 16 below.

Figure 16. The developer 6 sent an IM message to confirm it with the developer 5.
The developer 6 got the confirmation from the developer 5 that he had checked in his changes for the chapter 7, i.e. task 2, as shown in Figure 17 below.

Figure 17. The developer 6 got the confirmation from the developer 5 about the status of the task 2.

Then the developer 6 synced with the repository to get the latest changes for chapter 7 from the developer 5, as shown in Figure 18 below.
The developer 6 synced with the repository to get the latest changes. The developer 6 finally committed his changes for chapter 7, i.e. task 2, as shown in Figure 19 below.
The developer 6 finally committed his changes for chapter 7, i.e. task 2.

The experiment results on efficiency in tasks related to the potential merge conflicts and feedback for maintaining the group awareness, in general, are compared in the following sub-sections.

5.5.1 Analysis of Conflict Early Detection and Resolution

The primary objective of this experiment was to evaluate whether TeamWATCH can help developers detect and resolve the potential conflicts early to avoid the merge conflicts. Therefore, in the experiment result, we only checked whether there were merge conflicts, but did not differentiate whether the merge conflict was resolved before checking in the changes. Among the experiment data that we collected, this analysis was mainly based on the video recording of the experiment and the check-ins in the repository.

There are totally four conflicts in the ten tasks (five per team member) for each team. The number of conflicts detected and resolved early (i.e. those not turning into
merge conflicts) by both the experimental group and the control group is shown in Figure 20. A non-parametric statistical test was applied in this comparison since the probability distributions of results from both groups were unknown, as were common in this type of studies. The Mann-Whitney U test was applied to assess significance levels of the two groups. $U$ is the Mann-Whitney U test statistic, which is then used to determine $P$, which in turn indicates whether the result was statistically significant. The result is significant at $p \leq 0.05$ using Mann–Whitney U test (The U-value is 4.5. The critical value of U at $p \leq 0.05$ is 5). Therefore, we could reject $H_0$ in favor of $H_1$, i.e. using TeamWATCH does help developers detect and resolve the potential conflicts earlier.

![NUM OF CONFLICTS DETECTED AND RESOLVED EARLY BY EACH TEAM](image)

*Figure 20. The number of conflicts detected and resolved early by each group.*
In the experimental group, we observed that two teams did not pay attention to the smoke emitted from the files in TeamWATCH visualization when they were working on in the first task, thus the team member who checked in the changes late hit the conflict during the check-in time. Then they realized the importance of the hints provided by the TeamWATCH and leveraged it to coordinate together to avoid the merge conflicts in the following four tasks.

The main strategy adopted by most of the team in the experimental group is: one team member checks in the local changes for a file first, after that the other one syncs to the repository to get the changes for that file, and then starts working on the task related to that file. Sometimes, they might even switch the order of the tasks that they were working on to avoid the conflicts. And below is the excerpt of the chat logs from one team in the experimental group

**Developer 1**: just modified chapter 1, sync now

**Developer 2**: changed chapter 1 I think it worked check commit!!!!!!!!!!!!!!!!!!!!!!!!!!!

In the control group, we observed that most teams only detected the conflicts during the checking-in time and had to resolve the merge conflicts. The only exception is one team who coordinated how to avoid the conflicts from the beginning of the experiment by exchanging with each other the tasks assigned to them and suggesting that one team member started from the last task and worked backward while the other one worked forward. And below is the excerpt of the chat logs from this team.

**Developer 1**: I'm going to work on the Chapter 10 project first, which one are you working on now?
Developer 2: just chapter one i guess. we shouldnt get any problem with that one

To make the experiment simulate the real software developing environment, and also make it fair to both groups, we encouraged all the teams in both groups to use Gmail Chat or other IM tools that they’re familiar with to communicate with each other about the project status and collaborate together to finish the tasks. This exception team figured out there were potential conflicts at the beginning, which, in a sense, proved that the TeamWATCH tool would be very helpful for them since it could help detect and avoid the potential conflicts as soon as it occurs.

5.5.2 Analysis of Maintaining Group Awareness in General

The ability to detect and resolve the conflicts earlier is one way to justify that the TeamWATCH tool can help maintain group awareness better. In this section, we will look for the other proof mainly from the post-experiment survey data. The survey data was collected via the online survey website surveymonkey.com, and some participants did not fill out (or submit) their survey before leaving the experiment.

For the survey question “Were you aware of the status of your teammate (e.g. what was he/she working on at any particular moment)?”, eight out of nine (i.e. 88.9%) responders from the experimental group thought they were aware of their team member’s status, while only six out of the eleven (54.5%) responders from the control group were aware of their team member’s status.

The post-experiment survey specifically for the experimental group (since the questions are all related to TeamWATCH) asked the survey question “Did TeamWATCH help you maintain group awareness (i.e. knowing the status of the project artifacts and the
status of your team members) better? If yes, please give an example.

all the seven responders from the experimental group gave the positive answers, and most of them also gave an example, some of which are shown below

- The smoke gave me an idea there is someone working on the same file I'm working on it. But I did not have a conflict since I committed first.
- yes! helped me to find out what my team member was working on
- It helped when someone was actually committing,
- yes checking if a teammate had edited a file I was editing
- yes, I could see when I was making revisions as well as my partner.

It also asked the subjects when they used TeamWATCH during the experiment, most of them said they used it very often

- I used it as often as possible to see who changed the file and when they did so.
- pretty often, checked the visualization of what was happening
- the entire time

When being asked “which features of TeamWATCH were most useful to them”, the answers from the subjects fell into the follow two categories

- The visualization of who is editing/modifying which file
- The filter or search functionality

When being asked “Any suggested improvement on TeamWATCH?”, the answers from the subjects mainly about the 3-D UI interaction improvement.

- I would really love it if I could use it throw only the mouse :)}
I couldn't figure out how to switch from rotate to pan mode, and then I couldn't figure out how to get it back to the mode I wanted.

Based on the qualitative data collected from the post-survey, we could see that TeamWACH was proved to be helpful in maintaining group awareness, but we don’t think it is significantly enough to reject $H_5_0$ in favor of $H_5_a$.

5.6 Threats to Validity of Real-time Awareness Experiment

5.6.1 Internal Validity

5.6.1.1 Subjects

The number of subjects in the experiment is low. The results of the comparison between two groups would be more accurate if we could recruit more subjects.

Subjects were CS graduate and undergraduate students, and they had low experience in version control systems such as CVS and integrated development environments such as Eclipse, compared to experienced professional software engineers. However, before the experiments, subjects were given a CVS and Eclipse assignment to get familiar with them. Meanwhile, the simple guideline on Eclipse (and CVS plug-in) operations needed in the experiment is also included in the experiment sheet.

Subjects were randomly assigned to either experimental group or control group. The pre-survey results showed that both groups share the same (i.e. around 6 months) version control system experience.

5.6.1.2 Contextual Project

The text edit project instead of software project was chosen to mitigate the risk that the experience of the subjects determines the experiment results. Since the subjects
are all computer science students, the book on computer science topic (i.e. Internet) was chosen to set the same level of topic familiarity for the subjects.

5.6.1.3 Tasks

To mitigate the threat that the experimental task design may be biased to the advantage of TeamWATCH, the most common conflicts that developers can encounter in the daily work were selected, and all the tasked can be finished either via Eclipse CVS plug-in or TeamWATCH without much difference. Furthermore, we did not record, not even compare, the time required to answer these questions.

5.6.2 External Validity

5.6.2.1 Subjects

Subjects in both groups were Computer Science students. Their experience in software development was different from real-world professional software developers, who are the ultimate target audience of this prototype tool.

5.6.2.2 Contextual Project

The experimental project chosen in this study was a small size text edit project with 19 files and around 200 revisions developed by four editors (including two subjects). This could not really simulate the real software developing project, however, as explained in the previous section, it was introduced to let the subject focus on the evaluation of the tool instead of spending most of the time figuring out how to finish the coding tasks. While further evaluation of TeamWATCH’s effectiveness on larger sized software projects developed by a larger team is desirable, current results from this experiment are informative and worthwhile to share with the Software Engineering community.
5.6.2.3 Tasks

The five tasks (including coding subtasks and CVS history information questions) used in the experiments did not cover all types of tasks/questions developers may encounter in collaborative work. Nevertheless, these code tasks were designed to cover different types of coding changes that might introduce the potential conflict, and the questions were designed based on the “who, what, when, where, how” criterion and represented the most frequently encountered questions regarding software source code history.

5.6.2.4 TeamWATCH

The experiment designers were aware that TeamWATCH did not represent all the awareness tools. Therefore, even though the experiment would be a fair evaluation of the TeamWATCH tool, to generalize the specific outcome from TeamWATCH for software awareness, in general, could be a threat to the validity of the general result. However, better tool designs will likely only produce even better results than what has been shown with TeamWATCH.

5.7 Summary

The last three sections have discussed our evaluation of TeamWATCH on its effectiveness in enabling users to detect potential conflicts and collaborate to resolve conflicts earlier to avoid the merge conflicts during check-in. Statistically significant results showed that TeamWATCH, as a workspace awareness tool, helped users detect and resolve a larger number of conflicts earlier as compared to users without any workspace awareness support. This clearly demonstrated the effectiveness of
TeamWATCH in reducing the effects of conflicts and thus improving developer’s efficiency in the software development. It also provided qualitative evidence of the effectiveness of TeamWATCH in maintaining group awareness.
CHAPTER 6: CONCLUSIONS

Although coworker and artifact awareness information are essential for collaboration among software developers in a team, there is inadequate tool support to help them acquire it. This dissertation tries to address this problem by proposing a new approach to visualize developer’s activities that can be extracted as coworker and artifact awareness information in a 3D city metaphor, which builds a common view for the team. We first collect the awareness information in which the developers are mostly interested during their daily activities based on the existing research results, then categorize them mainly as history and real-time awareness information, and map them to different properties of the City metaphor. We apply the advanced graphical user interface design mantra to our visualization by creating the common overview of awareness information based on the file structure of project repository, then using eye-catching animations to highlight active artifacts that have real-time ongoing changes, and also providing filters for the developers to create their customized view to only view the information interesting mostly to them.

We then demonstrate the technical feasibility of our approach by building our research prototype, TeamWATCH. It is created as a workspace awareness tool to support visualizing both history and real-time awareness info in a shared common view via the 3-D city metaphor. TeamWATCH has been implemented in a client-server architecture, consisting of Eclipse plug-in, the visualization client in different platforms (i.e. Second Life and Unity3D), and the server. It has been successfully applied to visualize the history awareness information of many real-worlds open-source projects.
Finally, we evaluated the effectiveness of TeamWATCH in informing the awareness information needed, enabling easier and faster access to the awareness information, providing earlier conflict detection and resolution, and supporting better group awareness via three controlled user experiments. The experiment results provided quantitative evidence of the benefits of TeamWATCH in earlier conflict detection and resolution, and group awareness enhancement.

In summary, the main contributions of this dissertation are:

(1) A 3-D software visualization scheme that improves workspace awareness and enhances team collaboration;

(2) The design and implementation of the workspace awareness tool TeamWATCH using this visualization scheme;

(3) The evaluations of the effectiveness of such awareness tools using TeamWATCH as an example in maintaining project awareness and detecting and resolving conflicts via three controlled use experiments. The experiment results showed that the subjects using TeamWATCH performed significantly better in software revision history and project evolution comprehension, and early conflict detection and resolution.
CHAPTER 7: FUTURE WORK

The main improvement suggestion from the experiment subjects on TeamWATCH is that it is vulnerable to disorientation, which is a typical issue in 3D visualizations. With the advance of 3D visualization techniques, the issue should be alleviated, and we will discuss how we plan to improve TeamWATCH UI interaction later in this chapter. Another limitation existing in TeamWATCH is that it can only be used to visualize the awareness information from one branch, however, many branches might be created during the whole development process. This is can be mitigated by only visualizing the branch that the developers are currently working on and usually the developers working on the same components make the changes on the same branch as well, or visualizing the master branch where all the changes will finally get merged into during the integration. It is up to the developers to decide which branch they want to visualize.

Currently, there is not much adoption of workspace awareness tools in the industry. Usually, the change from a research tool to industry practice takes times, and software developers tend to hesitate in accepting new tools since switching the tools requires some training or learning curve for them to reach the same familiarity as they have for the old ones. Anyway, we need to continue improving our tool to make it fit developers’ needs while reducing the learning curve.

In the future, we plan to add the support of indirect conflict detection, integrate with more version control repositories such as Git, improve the UI interaction of
TeamWATCH by leveraging Nature User Interface tools such as Kinect and developing a mobile version of TeamWATCH, and evaluate it in the real software development team.

To better support workspace awareness, several aspects of TeamWATCH can be improved to provide a more accurate and convenient way to search software awareness information through an easy-to-navigate, intuitive visualization. First, we plan to add the support of indirect conflict detection and visualization to the TeamWATCH. The indirect conflict detection can be built based on the dependency analysis of different artifacts and visualized using a special smoke animation.

Second, integrating TeamWATCH with more version control repositories such as Git will make it more available for use to collect more data for further evaluation. And integration of more on-demand statistical information displays to show files’ and events’ statistics on the user interface will save users’ time and make it unnecessary to count objects when an accurate number is needed.

Third, more intuitive interaction with TeamWATCH will help reduce the amount of mental power needed to navigate in the 3-D scene. We are working to add Natural User Interface tools such as Kinect to TeamWATCH so that developers can focus on their tasks at hand when interacting with TeamWATCH to maintain awareness. Meanwhile, developers spend increasingly more time on getting awareness information via their mobile phone [68] and mobile awareness visualization tools start emerging [69], [70], which provides a good future direction for us on how to improve the TeamWATCH UI interaction in the mobile world.
Last, but not the least, it would be desirable to perform a longer-term study over complete software development cycles on the real software development team to measure TeamWATCH’s effectiveness for the overall project, instead of individual tasks performed by the students as studied in this dissertation. We also plan to carry out evaluations comparing TeamWATCH against existing awareness tools on the effectiveness of enhancing group awareness.
REFERENCES


2007.


pp. 202–211.


## APPENDIX A: EXPERIMENT TASKS

<table>
<thead>
<tr>
<th>Tasks for Subject A in each team of two</th>
<th>Tasks for Subject B in each team of two</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. CVS question: who last revised the Chapter 1?</td>
<td>1. CVS question: who last revised the Chapter 1?</td>
</tr>
<tr>
<td>2. In Chapter 1, switch the order of last two paragraph in section 1.1, and also move the following texts “There are the basics. Now on to the Net!” to a new paragraph.</td>
<td>2. In the Chapter 1, search the first occurrence of the texts “8-1-N” and “7-1-E”, and add the texts “(which stands for &quot;8 bits, 1 stop bit, no parity&quot; -- yikes!)” right after “8-1-N”, and add the texts “(7 bits, 1 stop bit, even parity)” right after “7-1-E”.</td>
</tr>
</tbody>
</table>
| 1. In Chapter 7, search and correct the following number errors  
(1) Modify the text “50,000-byte file” to “500,000-byte file”  
(2) Modify the text “24000-baud modem” to “2400-baud modem”  
(3) Modify the text “20,000 books” to “10,000 books”  
(4) Switch the years in the following texts  
In 1994, the project uploaded an average of four books a month to its FTP sites; in 1993, they hope to double the pace.  
2. CVS question: When did the editor “en” last revise the Chapter 7?  
3. In Chapter 7, insert the following texts to the end of the section 7.10 FYI  
The comp.sys.ibm.pc.digest and comp.sys.mac.digest newsgroups provide information about new MS-DOS and Macintosh programs as well as answers to questions from users of those computers. | 1. In Chapter 7, insert the following texts to the end of the section 7.10 FYI  
In the comp.virus newsgroup on Usenet, look for postings that list FTP sites carrying anti-viral software for Amiga, MS-DOS, Macintosh, Atari and other computers.  
2. CVS question: When did the editor “en” last revise the Chapter 7?  
3. In Chapter 7, Reverse the first name and last name in the following texts “Emtage Alan, Heelan Bill and Deutsch Peter” |
| 1. CVS question: when the last revision of Chapter 8 was committed?  
2. At the end of Section 8.1, the order of the following two commands “ms-dos or macintosh” and “ms-dos and macintosh” are wrong. Correct it by switching them. | 1. CVS question: how many editors contributed to Chapter 5?  
2. In the Chapter 5, insert the following two Command-BitnetList-UsenetList triples where two <<ADD COMMAND>>, <<ADD BITNETLIST>>, <<ADD USENETLIST>> strings are found.  
(1) sub NEW-LIST Your Name, listserv@ndsuvm1.bitnet, bit.listserv.new-list sub INFONETS Your Name, info-nets-request@think.com, bit.listserv.info-nets |
| 1. CVS question: how many revisions have Chapter 2 went through so far?  
2. In the Chapter 2 section 2.1, search and replace the texts “<<ADD DOMAIN SUFFIX EXAMPLES>>” with the below domain suffix examples and then order them alphabetically  
.com for businesses  
.org for non-profit organizations  
.gov and .mil for government and military agencies  
.net for companies or organizations that run large networks. | 1. In the Chapter 2 section 2.1, insert the following paragraph right after <<ADD DOMAIN SUFFIX EXAMPLES>>  
Sites in the rest of the world tend to use a two-letter code that represents their country. Most make sense, such as .ca for Canadian sites, but there are a couple of seemingly odd ones. Swiss sites end in .ch, while South African ones end in .za. Some U.S. sites have followed this international convention (such as well.sf.ca.us).  
2. CVS question: how many revisions have Chapter 2 went through so far?  
3. In Chapter 2, switch the universities (i.e. the content in the bracket right after the email addresses) that tomg@umn.edu and tomg@umn.edu represent |
| 1. CVS question: who has contributed most number of revisions to the Chapter 10?  
2. In Chapter 10, remove the section 10.7 The all-knowing Oracle, rename the section 10.8 FYI to 10.7, and then update the Table of Contents accordingly | 1. CVS question: who has contributed most number of revisions to the Chapter 10?  
2. Insert the following texts right after section 10.7 to become section 10.8, and rename the original section 10.8 to section 10.9, and then update the Table Of Contents accordingly  
10.8 WHEN THINGS GO WRONG |
* You get back an error message that your fax could not be delivered. With TPC, that could mean one of two things. Either you tried sending a fax to an area not covered by TPC or you made a mistake converting the fax number into a TPC address. Double-check both the list of TPC coverage areas and the address you created.