IEEE Access
practical innovations : open solutions

Received February 23, 2014, accepted April 1, 2014, date of publication May 6, 2014, date of current version May 16, 2014.

Digital Object Identifier 10.1109/ACCESS.2014.2322102

Source Code Revision History Visualization Tools: Do They Work and What Would It Take to Put Them to Work?

CHANG LIU, XIN YE, AND EN YE
School of Electrical and Engineering Computer Sciences, Ohio University, Athens, OH 45701 USA
Corresponding author: C. Liu (liuc@ohio.edu)

ABSTRACT Source code revision history visualization tools have been around for over two decades. Yet, they have not become a mainstream tool in a typical programmer's toolbox and are not typically available in intergraded development environments. So, do they really work? And if they do, what would it take to put them to work? This paper seeks to answer these two questions through experiments, surveys, and interviews. A source code history visualization tool named TeamWATCH was implemented to visualize subversion code repositories. Two comparative controlled experiments were conducted to evaluate the effectiveness of TeamWATCH. The experimental results showed that the subjects using TeamWATCH spent less time than subjects using the command-line subversion client and TortoiseSVN in answering the same set of questions regarding source code revision history. In addition, surveys and interviews were conducted to identify obstacles in adopting source code history visualization tools. Collectively, the results show that source code history visualization tools do bring value to programmers. Key obstacles to wider adoption in practice include nontrivial overhead in using the tools and perceived complexity in visualization.

INDEX TERMS Software engineering, version control repository, software development workspace awareness, software comprehension, source code revision history, software visualization.

I. INTRODUCTION

Source code revision history visualization tools have been around for over two decades. In 1992, the IEEE Transactions on Software Engineering published a paper by Eick, Steffen, and Sumner on Seesoft, a tool that visualized line-oriented software statistics, including version control history information [38]. Such tools seem to make sense because software developers work in an information-rich team environment. Quickly accessing overall project revision history and efficiently locating specific revision details when needed are essential in team collaboration. Then why are source code revision history visualization tools still not integrated in IDEs (Integrated Development Environments) today? We set out to find out why developers are not using such tools through experiments, surveys, and interviews. No experiments on whether such revision history visualization tools help improve developer efficiency have been reported in the literature. Seesoft is too old and not available for experiments. We designed and implemented TeamWATCH, a 3D code revision history visualization tool that complements traditional Subversion clients to facilitate the understanding of overall project revision history. Two user studies were conducted to evaluate TeamWATCH and test hypotheses that software source code revision history tools help improve developer efficiency. In addition, surveys and interviews were conducted to seek subjective opinions on this type of tools. The results are analyzed and summarized at the end of this paper, along with recommendations on how to facilitate the adoption of these tools in practice.

To put the discussion on software code revision history visualization tools into perspective, let's first take a look at the concept of history awareness.

A. WORKSPACE AWARENESS AND HISTORY AWARENESS

As reported by Vessey and Sravanapudi [1], software engineers spend about 70 percent of their time on cooperative activities; collaboration is essential for software development. Dourish and Bellotti considered awareness of each other's activities as what is critical for successful collaboration in CSCW (Computer-Supported Cooperative Work), and defined awareness in the context of shared workspace as ''an understanding of the activities of others, which provides a context for your own activity'' [2]. According to a two-month field study of ''collocated'' software development teams at Microsoft, Ko et al. found that developers frequently sought awareness information about artifacts such as ''how have

resources I depend on changed'', and awareness information about coworkers such as ''what developers' coworkers have been doing'' [3]. However, developers also identified such information as unavailable or difficult to obtain [3]. That is to say, even 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 similar studies on software developers at Microsoft [4]–[6]. This suggests that software development efficacy can be materially improved if software developers can obtain information relevant to their tasks more quickly and accurately through enhanced awareness, which is also true in other domains where human decision-making is partially based on situation awareness [7].

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 [2]. 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'' [8]. 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'' [9]. 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 paper is history awareness, which centers on source code and its revision history, as depicted by the shadow (the darker ovals) in Figure 1. Software history awareness is the understanding of revision history of source code, who committed what at what time, and why.

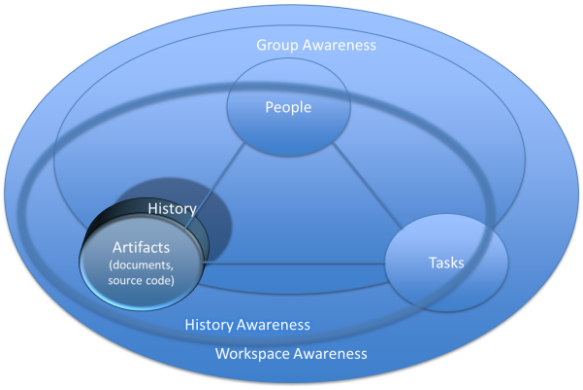


FIGURE 1. Differences among workspace awareness, group awareness, and software history awareness

According to Gutwin et al. [9]–[13], Pinelle et al. [14], and Storey et al. [15], 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 [16]. 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 [17]. Therefore, providing software history information efficiently to help maintain software history awareness not only benefits developers for their team development, but also helps newcomers become familiar with projects quickly.

History 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 history awareness is highly relevant in team software development.

B. PROVIDING TOOL SUPPORT FOR HISTORY AWARENESS

Given the definition above, it is reasonable to identify the following desired properties of history awareness tools to help developers effectively maintain heightened history awareness.

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 be capable to zoom into or center 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, this study introduces TeamWATCH (Team Workspace Awareness Toolkit and Collaboration Hub), which is a software revision history visualization tool that we implemented using a three-dimensional (3-D) city metaphor. TeamWATCH is designed to help developers and other project team members maintain history awareness. TeamWATCH extracts historical information of artifacts (such as files and folders), revisions, committing developers, and events (such as adding, deleting, and modifying

artifacts) from software projects' version control repositories. It visualizes such information using a 3-D city metaphor. 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 worked in the project, how many revisions has this project gone 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. Furthermore, because TeamWATCH extracts historical information directly from a repository's logs, it does not rely on any IDE (Integrated Development Environment) or any client software for access to the repository. The visualization runs on both personal computers (Windows and Mac OS) and mobile devices (e.g. iPhone and iPad), and makes it easier for developers to stay aware of project history by putting TeamWATCH on a second display.

There are no empirical results in the literature comparing the efficacy of source code history visualization or awareness tools with traditional version control clients that are widely used in practice. Therefore, two comparative experiments were conducted to evaluate the effectiveness of TeamWATCH. The first comparative experiment compares TeamWATCH and the command line Subversion [18] client using the same set of historical information questions based on the source code repository of the open-source Notepad++ project [19]. The second comparative experiment compares TeamWATCH and TortoiseSVN, a popular Subversion client with a GUI [20]. The results of these two experiments will also shed light on whether software history visualization tools in general are instrumental in software development.

The rest of this paper presents design and implementation of TeamWATCH in Sections 2, experimental design in Section 3, experimental result analysis in Section 4, a discussion of threats to validity in Section 5, survey and interview results in Section 6, related tools in Section 7, and a summary in Section 8.

II. DESIGN AND IMPLEMENTATION OF TEAMWATCH

As suggested by Shneiderman, "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" [21]. The design of the user interface of TeamWATCH was based on this mantra. TeamWATCH supports users to search for information through three steps: (1) gain perceptual experience and overview information of the whole project; (2) use filters to narrow down objects, then zoom in or out to locate the ROI; (3) get detailed information from specific objects if needed. Integration of the artifact filter, the author filter, and the revision filter into the user interface supports developers to efficiently find what they want through any combination of these filters. These three steps fulfilled Properties 1 and 2 in Section 1.2.

Based on the authors' earlier work in SecondWATCH [22], 3-D visualization metaphors of buildings and city blocks were used in TeamWATCH. In this visualization scheme, shapes of objects are used to distinguish types of artifacts. City districts (square blocks) represent folders; 3-D buildings (round columns, which are stacks of cylinders) represent files; floors of the buildings (cylinders) represent revisions. Similar to Ripley's Workspace Activity View, which uses a stack of cylinders to represent a workspace with each cylinder representing an artifact in it [23], TeamWATCH uses a stack of cylinders to represent a file with each cylinder representing a committed revision of this file. Because human has remarkable perceptual ability to detect changes in size, color, shape, movement, and texture for visual information [21], 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.

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 column represents the location of a file in the project. Information of revisions was mapped onto the vertical Z coordinate. The more cylinders in a column, the more revisions this file has gone through (revisions are ordered by the committed time with the latest revision on top).

Table 1 summarizes the mapping between the metaphors and relevant project historical information. Some elements of this mapping were also used in CodeCity built by Wettel and Lanza [24], [25], and sv3D (source viewer 3-D) built by Marcus et al. [26]. 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

TABLE 1. Mapping between the metaphors and the underlying project.

Project information	*Attributes of the 3-D metaphor*
Layout of a project (artifacts: files and folders)	Coordinates on a horizontal surface (X, Y)
Revisions of a file	Height (Z)
Author	Color
Status (deleted or not)	Transparency
Files versus folders	Shape (Cylindrical column versus blocks)

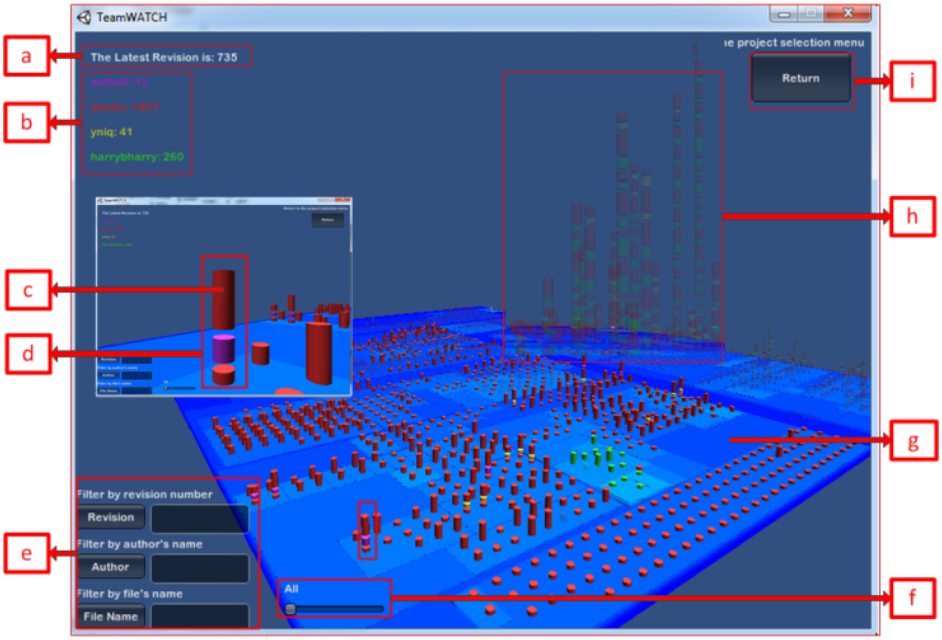


FIGURE 2. TeamWATCH visualization (a) Revision statistics; (b) Developer statistics; (c) A single cylinder; (d) A stack of cylinders; (e) Filters; (f) Time Slider; (g) Blue district; (h) Transparent cylinders representing deleted files; (i) Return to the main menu.

code (each file is represented by a container in which each cylinder represents a line of code).

In addition to the 3-D 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. Overall statistical information about the project and the developers are displayed on the top left corner of the GUI. Top bottom of the GUI shows filters based artifact names, author names, and revision numbers.

As shown in Figure 2, TeamWATCH visualizes a software project as a 3-D city. The city districts (blue blocks) represents folders (Figure 2g). A stack of cylinders in a column (Figure 2d) represents a file. Each cylinder (Figure 2c) represents a revision of this file. Transparent objects (Figure 2h) represent deleted files that no longer exist in the latest revision. The GUI also shows revision statistics (Figure 2a) and developer statistics (Figure 2b) 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 have 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 3a), users can look at only certain files. Using both the file filter and the author filter together, users can get information of how often and when a developer modified certain files (Figure 3b). More information of a revision of a file (Figure 3c) 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.

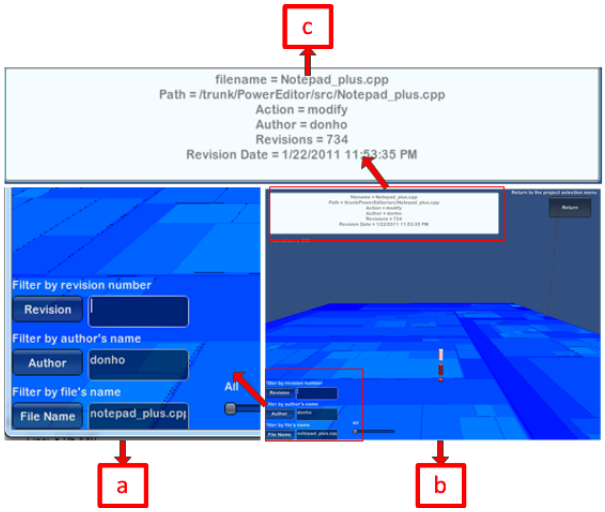


FIGURE 3. Filtering by both the author name and the file name; (b) Filtered visualization result; (c) Path, action, author, revision date and time of a file's specific revision.

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.

Five real-world open-source projects were visualized in TeamWATCH. They were Notepad++ [19], jEdit [27], Firebird [28], Hugin [29], and OpenNMS [30]. Descriptions and screenshots of these visualizations as well as TeamWATCH software and user guide are available on the VITAL Lab website [31].

TeamWATCH is implemented in a client-server architecture, as shown in Figure 4. Arrows represent information flow. The server side is implemented as a Java Web service. The Extractor on the TeamWATCH server extracts project's historical log information from a repository through SCM (Software Configuration Management) clients such as SVN command in Subversion [18]. Specifically for the experiments in this study, TeamWATCH only supported Subversion. The Extractor sends Subversion commands such as ''svn log –v ≫ some_documenst.txt'' to the Subversion client, and obtains a log file of the project. The log file contains information such as author, revision date and time, and files that were added, modified, or deleted for every revision of the project. The input to the Analyzer is the log information extracted by the Extractor. The Analyzer extracts file information, author information, revision information, and event information from project's log file, formats this information, and sends it 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 Table 1, using the Quantum Treemap algorithm [32]. 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 log 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.

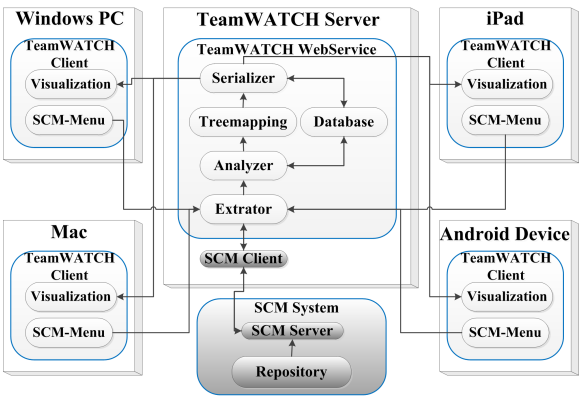


FIGURE 4. The TeamWATCH Architecture.

The TeamWATCH client was implemented as a stand-alone application using a 3-D game engine and development tool named Unity 3D [33]. The client side of TeamWATCH consists of two components: the SCM menu, and the awareness information viewer. The SCM menu allows users to either select an existing (as a built-in) visualized project offline or connect to the TeamWATCH server to load and visualize another project. The TeamWATCH server provides the client side with the serialized data of the project so that the TeamWATCH client can visualize it. The visualization component in the client side obtains information from the Serializer on the server side, and presents the final visualization for users. Because TeamWATCH visualizes historical information based on a project's logs, which can be checked out through a Subversion client, it does not rely on any IDE to collect information. The server side of TeamWATCH is implemented as a web service, and the client side is implemented as a stand-alone application running on multiple platforms including Windows PCs, Mac computers, and iOS devices[1], with which developers can set up a 2nd display to display awareness information. This helps fulfill Property 5 as described in Section 1.2. This study chose not to implement the TeamWATCH client directly as an Eclipse plug-in, because visualization requires large space to display but as an awareness tool, TeamWATCH is most fitted to stay in view constantly but mostly not in focal view. It would be desirable, though, to implement an Eclipse plug-in[2] to inform TeamWATCH in which files developers are interested so that Property 3 in Section 1.2 can be fulfilled as well.

III. EXPERIMENT DESIGN

Though software revision history visualization tools have been reported in the literature (e.g. Seesoft [38]), there was no user study on whether software history visualization tools helped improve developer efficiency. To answer the question of whether and how TeamWATCH helps facilitate the understanding of project revision history as compared with tradition SVN[3] clients (i.e. 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 (a SVN client with GUI).

In each experiment, subjects from both groups were asked to answer the same set of questions, which were based on

[1]It is also compatible with Android devices, although an Android TeamWATCH client has not been released yet.

[2]Such a plug-in was implemented in SecondWATCH but has yet to be integrated into TeamWATCH.

[3]The term ''SVN'' in this paper refers to the Apache Subversion tool.

TABLE 2. Null hypotheses and alternative hypotheses.

Null hypothesis	*Alternative hypothesis*
$H1_0$: Using TeamWATCH does not improve the accuracy of searching software history information.	$H1_a$: Using TeamWATCH improves the accuracy of searching software history information.
$H2_0$: Using TeamWATCH does not reduce the time used in searching software history information.	$H2_a$: Using TeamWATCH reduces the time used in searching software history information.
$H3_0$: Users do not consider TeamWATCH helpful in software history information exploration subjectively.	$H3_a$: Users consider TeamWATCH helpful in software history information exploration subjectively.

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.

A. RESEARCH QUESTIONS AND HYPOTHESES

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

1) Does the use of TeamWACH 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?
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) In searching of what kind of software history information does TeamWATCH perform significantly better than traditional SVN clients, and why?
5) In searching of what kind of software history information do traditional SVN clients perform significantly better than TeamWATCH, and why?

The null hypotheses and the alternative hypotheses that correspond to the first three research questions are described in Table 2.

The first three research questions would be answered through quantitative analyses. The fourth and fifth questions would be answered through qualitative analyses.

B. 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 Team WATCH over a baseline in software history information exploration. Therefore, TeamWATCH and the chosen baseline are the two levels of the independent variables.

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 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 [20], 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. TortoiseSVN version 1.7.12 released on April 4th 2013 supporting Subversion 1.7.9 was used in the experiment.

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 important complementary material for qualitative analysis of why subjects are satisfied or unsatisfied with TeamWATCH, the command-line SVN client, or TortoiseSVN.

C. 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 [34].) (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 experiment.) Therefore, the Notepad++ project was a suitable project for this experiment on the effectiveness of TeamWATCH on improving efficiency of software history exploration for small software teams.

D. SUBJECTS

In the two controlled experiments, Computer Science undergraduate and graduate students aged from 19 to 35 were recruited to serve as the subjects. 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 system 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 a SVN client, they all chose the command-line SVN client.

In the second controlled experiment, nineteen CS undergraduate and graduate students were recruited and randomly assigned to two groups. 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 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 become more familiar with the project after they work on it in 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.

E. TASK

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 [8]–[13] 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 [14] 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 3. ''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. These experimental questions were tested before the actual experiment to make sure that they were understandable and reasonable in the context of the project.

Separately, Sillito et al. [35] described 44 types of code-based questions programmers asked in software development, they also observed programmer behaviors around

TABLE 3. Questions that subjects were asked to answer on an open-source project, notepad++.

Specific Questions	**Category**		**Result** (Which tool was better, TeamWATCH or a traditional Subversion client?)
Q1. List all authors who have contributed to the Notepad+ project, then identify who made the most contributions.	Understanding a subgraph	Who	TeamWATCH
Q2. List all authors who have contributed to file *Notepad_plus.cpp*, then identify who last revised this file.	Expanding focus points		TeamWATCH
Q3. Who created the */trunk/PowerEditor/src/tools/NppShell* folder, and in which revision and at what time?	Expanding focus points	Who & When	TeamWATCH
Q4. Roughly how many deleted files are there in the repository? 0, 5, 10, 20, 50, 100, 700, 1500?	Understanding a subgraph	What	TeamWATCH
Q5. What happened in revision 460?	Finding a focus point		SVN command and TortoiseSVN
Q6. How many files were modified in the latest revision, and when did the last revision happen?	Expanding focus points	What & When	TeamWATCH performed worse than SVN command but better than TortoiseSVN
Q7. Is it true that "harrybsharry" and "donho" used to work together closely by modifying the same files, but not any more?	Understanding a subgraph	Where	TeamWATCH
Q8. Which folders was "harrybharry" last working on?	Expanding focus points		TeamWATCH
Q9. When was file *WordStyleDlg.cpp* last modified?	Expanding focus points	When	TortoiseSVN
Q10. When did "yniq" last commit a revision?	Expanding focus points		No significant difference
Q11. Which file went through the most revisions?	Understanding a subgraph	How	TeamWATCH
Q12. Who at which revision modified the file */trunk/PowerEditor/src/Notepad_plus.cpp*? List the author's name and revision number.[4]	Expanding focus points		No significant difference
Q13. List one subfolder that all files inside went through only one revision.[5]	Understanding a subgraph		TeamWATCH
Q14. List three files that were edited by both "donho" and "yniq".[5]	Understanding a subgraph		TeamWATCH
Q15. List all authors who have committed works in folder */trunk/PowerEditor/src/tools/NppShell/src/bitmap*.[5]	Expanding focus points		No significant difference
Q16. List one subfolder that most files inside were frequently edited by more than one developer.[5]	Understanding a subgraph		TeamWATCH

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 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 3, 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 subgraph'' category that needs overall information of the project.

Alwis et al. [36] 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 3 fall under the evolution category as defined by Alwis et al.

[4]Question 12 was used only in the first experiment.
[5]Question 13 to 16 were used only in the second experiment.

The rightmost result column of Table 3 is based on the results from the experiment, which are discussed later in this section.

F. PROCEDURE

Two separated 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 TeamWATCH. 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 3. 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.

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 3. 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 3.4 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 Question 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.

IV. EXPERIMENT RESULTS

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.

A. ANALYSIS OF CORRECTNESS AND TIME

To better understand the difference between TeamWATCH and traditional Subversion clients, comparisons of mean correctness of answers per person to each question and mean time per person used to answer each question were performed. Figure 5 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 5, 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,

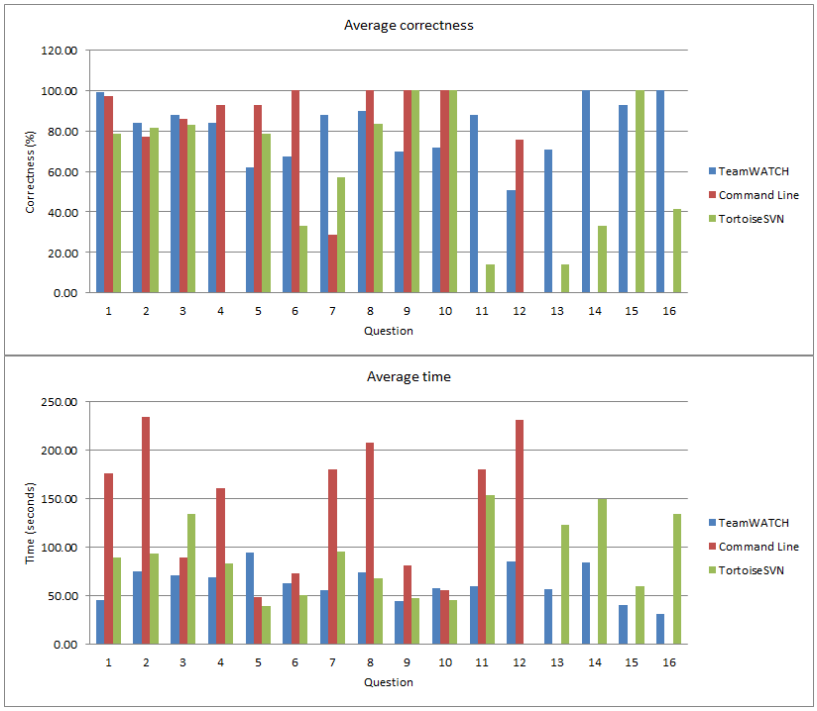


FIGURE 5. Comparison of the average correctness and average time spent for each question.[6]

TABLE 4. Comparison of average correctness and average time in the first experiment.

Question	Correctness (%)						Time (seconds)					
	TeamWATCH		SVN cmd		U	P	TeamWATCH		SVN cmd		U	P
	n	Mean	n	Mean			n	Mean	n	Mean		
Q1	13	**98.46**	7	97.14	42.5	0.648	13	**46.64**	7	176.00	3.0	0.006
Q2	13	**81.54**	7	77.14	43.5	0.863	13	**73.10**	7	234.40	3.0	0.007
Q3	13	76.92	7	**85.71**	39.0	0.497	13	**69.20**	7	89.60	20.5	0.581
Q4	13	80.77	7	**92.86**	34.5	0.273	13	**77.60**	7	161.00	8.0	0.037
Q5	13	61.54	7	**92.86**	23.0	0.045	13	96.30	7	**49.20**	13.0	0.141
Q6	13	57.69	7	**100.00**	14.0	0.005	13	**60.00**	7	73.50	10.5	0.178
Q7	13	**84.62**	7	28.57	20.0	0.015	13	**37.55**	7	180	0	0
Q8	13	92.31	7	**100.00**	42.0	0.463	13	**52.30**	7	208.00	2.0	0.005
Q9	13	76.92	7	**100.00**	35.0	0.179	13	**43.80**	7	81.60	9.0	0.05
Q10	13	61.54	7	**100.00**	28.0	0.065	13	**44.00**	7	56.20	15.0	0.317
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.

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 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.

[6]Question 12 evaluates only TeamWATCH and SVN commands. Questions 13 to 16 evaluate only TeamWATCH and TortoiseSVN.

TABLE 5. Comparison of average correctness and average time in the second experiment.

Question	Correctness (%)						Time (seconds)					
	TeamWATCH		TortoiseSVN		U	P	TeamWATCH		TortoiseSVN		U	P
	n	Mean	n	Mean			n	Mean	n	Mean		
Q1	12	**100**	7	78.57	24	0.016	12	**45.25**	7	89.43	24.5	0.138
Q2	7	**88.57**	12	81.67	39	0.783	7	**79.43**	12	93.42	31	0.352
Q3	12	**100**	7	82.86	30	0.057	12	**73.67**	7	134.14	18	0.042
Q4	12	**87.5**	7	0	0	0	12	**60.42**	7	83.43	24.5	0.138
Q5	12	62.5	7	**78.57**	31	0.305	12	92.42	7	**39.43**	15	0.022
Q6	7	**85.71**	12	33.33	20	0.032	7	69.43	12	**50.92**	27.5	0.219
Q7	12	**91.67**	7	57.14	31	0.224	12	**76.67**	7	95.57	32.5	0.421
Q8	7	**85.71**	12	83.33	41	0.894	7	115.86	12	**68.08**	22	0.089
Q9	7	57.14	12	**100.00**	24	0.016	7	**47.14**	12	47.83	41	0.932
Q10	12	83.33	7	**100.00**	35	0.266	12	73.83	7	**45.43**	23	0.108
Q11	12	**91.67**	7	14.29	9.5	0.001	12	**84.92**	7	153.29	15	0.021
Q13	12	**70.83**	7	14.29	17	0.017	12	**57.17**	7	123.29	12	0.011
Q14	7	**100**	12	33.33	14	0.006	7	**84.29**	12	150	16	0.027
Q15	7	92.86	12	**100**	36	0.190	7	**40.71**	12	59.75	30.5	0.331
Q16	7	**100**	12	41.67	17.5	0.013	7	**31.14**	12	134.42	0	0

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.

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 4 and Table 5, 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 statistical significant differences were highlighted. The dark grey color indicates that the treatment group did better than the control group in answering a question; the light grey color indicates the reverse situation.

1) CORRECTNESS – RESEARCH QUESTION 1

As indicated by the results of the first experiment in Table 4, 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 5, 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.

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 4. In the second experiment, as shown in Table 5, 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.

3) TASK ANALYSIS – RESEARCH QUESTION 4 AND 5

As shown in Table 4 and Table 5, 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 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 near-by object was recorded instead. This identified an area to improve for this particular TeamWATCH implementation. But it was not nec-essarily 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 ex-periment 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 5.

For all seven questions in the ''Understanding a subgraph'' category, Q1, Q4, Q7, Q11, Q13, Q14, and Q16 in Table 3, TeamWATCH subjects performed better than SVN commands and TortoiseSVN. This result shows that TeamWATCH is more suitable for answering questions that required higher-level understanding of the project repository.

Furthermore, the sixteen questions in Table 3 are all related to project evolution, which was reported by Alwis et al. as not being well supported by current tools [36]. 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.

B. ANALYSIS OF THE RESULTS ON SATISFACTION

In the first experiment, subjects in the treatment group were generally satisfied with using TeamWATCH to complete the

TABLE 6. Mean satisfaction of the two groups in the first experiment.

Question	TeamWATCH			SVN Commands			U	P
	n	Mean	Standard Deviation	*n*	Mean	Standard Deviation		
1	13	**4.55**	0.69	7	2.20	0.84	1.0	0.002
2	13	**4.45**	0.69	7	2.60	0.55	1.5	0.002
3	13	3.17	1.53	7	**4.20**	0.45	19.0	0.207
4	13	**3.55**	1.21	7	2.80	0.84	17.5	0.241
5	13	**4.18**	0.60	7	4.00	1.22	27.0	0.949
6	13	**3.55**	1.29	7	3.40	0.55	23.5	0.635
7	13	**3.83**	1.27	7	1.20	0.45	3.5	0.004
8	13	**3.58**	1.24	7	3.40	0.55	24.5	0.543
9	13	**4.33**	1.23	7	4.00	0.71	18.5	0.183
10	13	2.45	1.29	7	**4.60**	0.55	4.5	0.008
11	13	**3.91**	1.45	7	1.00	0.00	2.5	0.003
12	13	**4.00**	1.48	7	2.60	1.52	12.0	0.047

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.

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 6, where n is the number of participants in each group and U is the Mann-Whitney U test statistic.

Subject satisfactions in the second experiment, as shown in Table 7, coincide with the result in 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).

1) SATISFACTION – RESEARCH QUESTION 3

Results show that the treatment group in the first experiment was more satisfied for 10 out of 16 questions (significantly for Q1, Q2, Q7, Q11, Q12, Q13, Q14, Q16, and nonsignificantly for Q9, and Q15). 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.

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, *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 angels 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 –ql grep yniq".

TABLE 7. Mean satisfaction of the two groups in the second experiment.

Question	TeamWATCH			TortoiseSVN			U	P
	n	Mean	Standard Deviation	n	Mean	Standard Deviation		
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	40.5	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.

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.

C. 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 are not experienced in traditional Subversion clients.

Nineteen out of thirty-two (59.4%) subjects in both the experiments agreed or strongly agreed with the statement ''TeamWATCH 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.

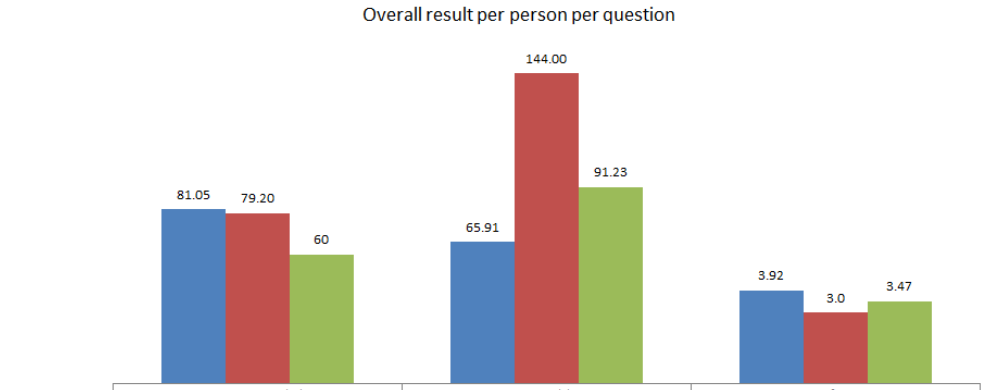


FIGURE 6. Comparison of overall mean correctness, time, and satisfaction for all groups.

He suggested 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.

D. OVERALL RESULTS

Figure 6 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 but 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, in 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 were more satisfied with TeamWATCH than subjects using SVN commands and TortoiseSVN.

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 any more'' (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 1.2.

V. THREATS TO VALIDITY

A. INTERNAL VALIDITY

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.

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 in-formation. 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.

3) TASK

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.

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.

B. EXTERNAL VALIDITY

1) SUBJECTS

Subjects in both the control group and the experiment group are Computer Science students. Their experience in software development is different from real-world professional software developers, who are the ultimate target audience of this prototype tool. These subjects might also have different understanding of the experimental project and have different experience in various kinds of 3-D games. The experimental results from these subjects may not be generalized to a large population of industry practioners in real world.

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 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 searching for source code historical information than conventional SVN clients.

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.

4) TASK

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) TEAMWATCH

TeamWATCH cannot 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 to software history awareness in general could be a threat to the validity of the general result. However, better tool designs will likely produce even better results than what has been shown with TeamWATCH.

VI. SURVEY AND INTERVIEW

So far, the experiments showed that source code revision history visualization tools such as TeamWATCH were indeed instrumental to developers. But why have not any tools like this been integrated in IDEs and made available to developers on a daily basis? What would it take for developers to actually adopt these tools in practice? To shed some light on these

TABLE 8. Interview details.

Issues raised by the interviewee	Nature of the issues
"Change in a business environment is always challenging. There is always resistance to change. Management buy-in may be difficult to obtain."	Cost/benefit tradeoff
"There is also a financial aspect to this. Depending on the cost for license, different decisions may be made. In addition, training cost should be considered, too. Maintenance cost of the tool may also become an issue."	Cost/benefit tradeoff
"It is desirable to view different layers other than just code. In addition to relationship between commits, it will be useful to achieve a higher level of understanding of the project."	Description of desired feature
"Technical managers are often interested in knowing how big the code is and how many commits there are."	Description of desired feature
"What should the metrics for productivity be? Is it number of commits per month alone? Probably not."	Alignment with business process
"Another issue may be developer time. There may be no window of free time slot for developers to do anything like this when often they are barely keeping up with deadlines."	Cost/benefit tradeoff
Management may not want to take any risks.	Cost/benefit tradeoff

questions, in 2013, we conducted a survey among 33 computer science senior undergraduate students and graduate students. In addition, we interviewed an experienced professional software project manager.

A. AN INTERVIEW WITH A SOFTWARE PROFESSIONAL

After TeamWATCH was demonstrated to a software professional with over 15 years of experience in software development, software testing, and software project management, he expressed strong interests in using the tool. He also believed that the adoption of such a tool would benefit a software development team. He was not certain if a software development organization would incorporate this tool right away, though. In the interview, he discussed a number of concerns as listed in Table 8. It was clear that that while he saw a few additional desired features on top of what he had seen in the demo, the main concerns were cost/benefit tradeoff and whether the benefit would be much greater than both actual and perceived cost in time and money.

B. A SURVEY OF 33 SENIOR AND GRADUATE COMPUTER SCIENCE STUDENTS

In a survey of 33 Computer Science senior undergraduate and graduate students who had already learned about TeamWATCH through demonstrations and presentations, when asked "If you were to manage a software project that involves 10+ developers and 10k+ lines of source code for 3+ years with potential personnel turnover, would you adopt a source code revision history visualization tool like the ones shown in class?", 32 or 97% of them answered yes; only 1 or 3% chose "no". When asked why, most of them (31 out of 32) offered concrete scenarios in which they thought this tool may help them as developers, e.g. "I can see what we developed more and in the case that there was little or no development; we can look over that portion," "It would help see who did the most work, etc." "Yes, because it will give me a lucid idea about the working of the team."

When asked an open-ended question "Why do you think modern IDEs such as Visual Studio, XCode, Eclipse do not currently provide source code revision history visualization features?", subjects speculated on vague reasons such as "Fairly new concept but I think it will catch on," "It will take a lot of memory/space?", or even privacy concerns, which are not really an issue because TeamWATCH only visualizes what is already accessible. However, later, when a list of concrete factors was provided and the subjects were asked to rank what "could be potential obstacles in adopting such code visualization tools", the subjects showed their concerns as listed in Table 9. It was clear that among the provided list of potential obstacles, the subjects were more worried about the visualization itself (and therefore the utility of the tool) than other factors such as cost and management buy-in.

C. SUMMARY OF THE INTERVIEW AND THE SURVEY

Both the professional developer in the interview and the computer science students in the survey considered the tool useful and desirable. Further discussions and questions revealed that the experienced developer wanted more elements in the visualization but was concerned by cost/benefit tradeoff and

TABLE 9. Potential obstacles in adopting source code visualization tools and their rankings by subjects.

Provided list of potential obstacles	**Rank** (the smaller the number, the higher the rank)
The visualization may be difficult to understand	2.69
Visualization may be irrelevant and distract programmers in some cases	3
Developers do not have time for yet another tool	3.75
The visualization may cause misunderstanding	3.93
Lack of integration with IDE	4.06
Visualization could be relevant but may slow down programmers in some cases	4.3
Cost in training	4.45
Extra hardware requirement for 3D visualization	4.71
Management buy-in	5
Potential computer performance hit by visualization	6.3

management buy-in, whereas the student subjects were more concerned about understanding the current visualization and less about management buy-in, something they had less experience with.

VII. RELATED SOFTWARE TOOLS

TeamWATCH is not alone in using a 3D metaphor in visualization or attempting to provide awareness information about a software project. TeamWATCH is similar to several types of tools in some aspects. A quick discussion of those tools can help us understand where TeamWATCH or source code revision history visualization tools stand in general.

A. SOFTWARE PROJECT AWARENESS TOOLS

Software project awareness tools typically retrieve information from several information sources, which is consistent with what developers do without awareness tools. According to Gutwin et al. [12], developers in open-source projects tend to maintain both 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 chats, 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 the ''maintain'' field in the source tree, version control repository logs, issue trackers, help from senior developers, and project document. They also post queries to the project mailing list. In short, open-source developers maintain awareness by manually ''pulling''
information from several information sources. This was also true for commercial software developers as found by de Souza, Redmiles, and Dourish [37]; Ko et al. [3]; LaToza et al. [6] and Biehl et al. [5].

Many tools have been built to help developers maintain awareness in team software development. Some of these tools such as Seesoft [38], Evospace [39], [40], COOP/Orm [41], BSCW [42], Xia [43], Augur [44], and Rationalizer [45] provide awareness of activities based on information currently available in the repository. Among these tools, Seesoft is one of the earliest tools visualizing historical information. It visualizes line-oriented statistics of source code [38], [46]. Seesoft visualizes historical information about authorship, timeline, and description of revisions of projects in version control systems. Different from Seesoft, TeamWATCH aims to provide file-based information such as ''who and at which revision modified a file'' rather than line-based information such as ''who revised a source code line''.

B. PROGRAM COMPREHENSION TOOLS USING 3-D CITY METAPHORS

H. Gall, M. Lanza [40] and P. Dugerdil [47] led the EvoSpaces project that helped developers with their software analysis and maintenance by providing a visualization of the evolving software system, using multi-dimensional navigation spaces. In the overarching EvoSpaces project, R. Wettel [24], [25], [48]–[50] developed the CodeCity project, which utilized a 3-D city metaphor to represent classes, methods, and packages. CodeCity mainly concentrates on helping users

understand software project structure and intentions rather than providing awareness information such as a developer's history activities on specific files, or events happened on artifacts during a specific time period. Similar with Langelier's approach [51], CodeCity maps packages to districts, and classes in a package to buildings in the relevant district. In CodeCity, the height of a building represents the number of methods in the class (by utilizing a boxplot-based mapping technique), the size represents the number of attributes, and the color represents the number of source code lines. Wettel et al. extend their approach by mapping design problems to colors [50], and introduce a coarse-grained level and a fine-grained level of visualization for helping software evolution analysis [48]. Even though users can step forward and backward in time, one scenario of the visualization shows only the software structure at that time. It does not help gain insight of the project evolution in one scene, nor does it help obtain information of relationship among developers, artifacts and revisions. They perform a controlled experiment [25] showing the advantage of the visualization of CodeCity against required data in an Excel sheet in answering a set of program comprehension related questions. Even though TeamWATCH utilizes a city metaphor as well, it was mapped to different concepts to visualize the historical relationship among developers, artifacts, and revisions to help maintain what was referred to history awareness, which are both the overall comprehension of the software evolution and the details of each revision (for example, who at what time reconstructed the project, how many and what files were revised, added, or deleted by a developer in a specific revision, etc.).

Besides CodeCity, in the EvoSpaces project, S. Boccuzzo [52]–[54] designed the CocoViz tool, which integrated audio into a 3-D common space metaphor to represent concepts such as code smells or design erosion. CocoViz was focused on supporting the understanding of software structures and evolution, and helping answer common software comprehension questions. The unifying EvoSpace tool combined successful concepts from both CodeCity and CocoViz, and had additional features such as the visualization of execution traces. Boccuzzo et al. [54] categorized common software comprehension tasks into five categories, which were Functionality, Relationship of Code Entities, Features and their Implementation, Architecture and Design, and Testing. They introduced an automated workflow to help locate software entities based on these comprehension tasks. Their proposed workflow was implemented in their CocoViz tool. In their workflow, users first select a comprehension task such as finding code smells or finding siblings of a method, and they can finally obtain a customized visualization of entities such packages and classes relevant to the selected task. Even though a similar 3-D visualization is used in TeamWATCH, the semantics of the TeamWATCH city metaphor differs significantly from the metaphor in the EvoSpaces project. The EvoSpaces project mainly focuses on program comprehension tasks in terms of packages and classes, whereas TeamWATCH aims to help developers maintain awareness on the revision history.

C. WORKSPACE AWARENESS TOOLS FOR MONITORING POTENTIAL CONFLICTS

Some tools such as Palantir [55], [56], JAZZ [57], FASTDash [5], SecondWATCH [22], Workspace Activity Viewer [23], and War Room [58] visualize up-to-date information of ongoing changes in developers' local workspaces. Palantir, Workspace Activity Viewer, CollabVS [59], and Celine [60] provide filter mechanism to lower developers' cognitive load. That is, developers only see the changes they are interested in. Jazz, CollabVS, and State Treemap [61] provide communication mechanisms. Palantir, CollabVS, and TUKAN [62] support indirect conflict detection through dependency analysis. Palantir, JAZZ, FASTDash, War Room, CollabVS, Celine, and State Treemap mainly focus on current information regarding ongoing changes to artifacts, not historical revision information. Only SecondWATCH and Workspace Activity Viewer present awareness information in a 3-D form. SecondWATCH, a tool developed by the authors before TeamWATCH, was designed to include an Eclipse plugin to help developers maintain workspace awareness by monitoring teammates' local workspace activities. Unlike SecondWATCH, which focuses on providing developers' local workspace awareness information, TeamWATCH focuses on improving efficiency of searching through historical information. Both SecondWATCH and Workspace Activity Viewer show historical awareness information in detailed views, e.g. a snapshot of all changes to artifacts at a particular time. TeamWATCH does that, and also provides statistical information such as the total revision number, the list of contributors, and contributions of every developer to support quick understanding of the overview of project revision history.

D. OTHER RELATED SOFTWARE VISUALIZATION FRAMEWORK

Langelier et al. [51] introduced a visualization framework for quality analysis of large-scale software system. In their city metaphor, districts represented packages; buildings represented classes; the size and color of each building represented software quality characteristics such as CBO (Coupling Between Objects) and WMC (Weighted Methods per Class). Their visualization aims at helping analyze software quality such as which classes are associated with each other, while TeamWATCH aims to help identify history information such as who at what time revised which specific file.

Marcus et al. [26], [63] designed sv3D, in which files were mapped to containers; lines of texts from source codes were mapped to cylinders in each container; colors represented control structures; and the height of a cylinder represented the nesting level. Alam et al. [39] depicted a 3-D virtual city in which the larger the building, the larger the size of the file.

TABLE 10. Comparison of select awareness tools.

Tool Name	Seesoft	CodeCity	FASTDash	Palantir	TeamWATCH
Main focus	Source code statistics	Software structure and contents on class and package level	Team members' current activities	Monitoring changes in all artifacts	Historical source code revision
Awareness information	History awareness	Mainly focuses on program comprehension rather than awareness	Workspace awareness	Workspace awareness	History awareness
Developer information	The line color represents the programmer who wrote each line.	Developer information is not mapped on the 3-D city metaphor	A photo of programmer is shown when a file is opened.	Programmers involved in indirect conflicts are shown in an extra tab.	The color of a revision represents its author; all authors' names and summaries of their contributions are shown on a corner.
Artifact information	Columns represent files; colored rows represent source code lines; colors represent associated properties.	Districts represent packages; buildings represent classes; properties of a class are mapped to the height, size, and color of the building	Files being operated on in other teammates' workspaces are highlighted. Each file's current status was represented by how it is highlighted.	Artifacts with direct and indirect conflicts are marked with blue and red triangles.	Each file is represented by a set of stacked cylinders; each blue block represents a folder. Transparent cylinders represent deleted files.
Revision information	The color on the scale represents the age of the code; the number of times a file has been changed is shown under the color scale.	In coarse-grained level, users can step forward and backward to different scenarios at different spot; in fine-grained level, colors of districts and buildings indicate their ages	Artifacts' revision information is not shown on FASTDash visualization.	The current revision number is shown behind each artifact's name.	Each cylinder represents a revision of a file; the revision number is shown on a pop-up window; The total number of revisions is shown on the top-left corner.
Graphical representations	2D graphical line-oriented view	3-D city view	2D graphical file-oriented view	Unobtrusive extensions of the Eclipse package explorer view, and an extra tab in Eclipse	3-D city view
Implementation mechanism	Standalone application	Standalone application	Visual Studio plug-in	Eclipse plug-in	Standalone application

However, neither tools considered the evolution and revision of a project's source code. They just provided visualizations of the project's current structure.

Steinbrückner and Lewerentz [64] described a three-staged visualization approach which consisted of a ''logical primary model'' decomposing a system into subsystems, a ''geometric secondary model'' representing the decomposition from the first stage into a hierarchical street layout, and a ''tertiary models'' mapping and visualizing software properties into city metaphors for different scenarios. In their visualization scenarios, streets down the hills indicated growing subsystems; towers in the high land surrounded by more contours represented subsystems with more revisions; the size and color of a tower represented its properties. Although it helps comprehension of software evolution, the models miss reconstruction information. Users could easily observe deleted models from empty plots, but they could not figure out what were deleted and whether the models were reconstructed at other plots. Besides, the models do not show detailed information for each revision in their visualization. As the authors stated, their visualization mainly focuses on helping users create a mental map of system for product properties and process events localization. Again, our approach concentrates on maintaining history awareness, gaining overall comprehension and detailed information of history relationship among developers, artifacts, and revisions. Besides the overall reconstruction information, TeamWATCH visualization provides detailed information such as the developer's name, revision number, commit time, its path, and event (add, delete, or modify) for a specific revision of a file.

E. DISCUSSION

Different from the tools discussed above, TeamWATCH was designed as a revision history awareness tool that complements traditional version control clients to provide users with fast understanding of the overall project revision history. Table 10 shows side-by-side comparison between TeamWATCH and the four most representative ones of the related tools discussed above, namely Seesoft, CodeCity,

FASTDash, and Palantir. These four tools were selected because they were higher-profile, more accessible, more similar to TeamWATCH, or better described in the literature. In the table, the awareness information row lists the type of awareness each tool focuses on; the developer information row covers how each tool provides information about developers' current and past activities, the artifact information row refers to how artifacts' current status or historical information are presented; the event information row summarizes how each tool shows events such as adding, modifying, and deleting artifacts; the revision information row shows how revision numbers and revision dates associated with artifacts are displayed. As shown in Table 10, Seesoft focuses on providing source code lines historical statistics. CodeCity, as a representation of the software structure visualization tools, concentrates on visualizing software structure and relationship among classes, packages, and properties of classes. To help program comprehension and reverse engineering, both CodeCity and CocoViz introduced their 3-D city metaphors, the appearances of which are similar to the metaphor used by TeamWATCH, although with different meanings. FASTDash and Palantir aim at monitoring developers' and files' current status to help increase workspace awareness. Compared with these tools, TeamWATCH helps maintain project history awareness by facilitating efficient search and understanding of software historical repository information through an intuitive 3-D visualization, and provides history information on source code files level. The focus of TeamWATCH is to 1) supply users with source file level awareness information about project revision history, and 2) provide an intuitive user interface to help search project revision information efficiently.

With all the differences listed above in mind, it is important to note that the general principle of ''awareness through visualization'' behind TeamWATCH and these awareness tools is the same. Among these tools, only Seesoft and TeamWATCH focus on source code history awareness. The experiment results presented in this paper provide much needed empirical data to justify further research and development as well as the adoption of such tools in practice.

VIII. SUMMARY

Software revision history information, which is frequently requested and searched by software developers, is essential in collaborative team developments. This paper discusses whether source code revision history visualization tools are beneficial to developers in the context of software history awareness.

TeamWATCH, a 3-D source file and revision visualization tool that helps developers explore software history information efficiently is presented. TeamWATCH aims at providing an intuitive 3-D visualization of software history information from source control repositories so that developers can obtain overview information of a project quickly to maintain software history awareness. The visualization in TeamWATCH was designed to support the natural cognitive process of understanding the overview first, zooming in to an area of interest later, and searching for details when necessary. In TeamWATCH, different types of software history information such as developer, artifact, revision, and event information were mapped into different metaphors in a 3-D view. Filters were integrated into the GUI to allow dynamic selection and filtering of the visualization. Detailed information about each item was displayed on demand. Statistical history information was presented directly on the user interface.

Experimental results from two controlled experiments evaluating the efficacy of TeamWATCH, compared with SVN commands and TortoiseSVN, provided support to the general hypothesis that appropriate history awareness tool support improves the efficiency of history information exploration in team software development. Specifically, 3-D visualization of the complete software revision history by TeamWATCH effectively improved developer comprehension of source code revisions and project evolution. It was clear that TeamWATCH was superior when it came to questions that required better comprehension of the big picture. Because of that, TeamWATCH subjects achieved better results in all ''subgraph''-type questions in Table 3.

To seek further insight on why this type of tools is not currently widely used by developers in practice, an interview and a survey were conducted, which revealed concerns on cost/benefit trade-off, management buy-in, and intuitiveness of the visualization. Even though virtually all subjects wanted to use tools like this, further efforts addressing these concerns will be necessary to make the argument for adopting this type of tools in practice more convincing. Designing these tools as an awareness tool that stays outside the focal view of developers may help lower the cost for developer adoption in terms of developer attention and overhead. (TeamWATCH was designed to be an awareness tool running on a 2nd monitor, but it was used as the primary tool in the experiments, not as an awareness tool.) Making the visualization both more intuitive and richer may help smooth out the learning curve and add utility to the tool.

ACKNOWLEDGMENT

The authors would like to thank Ohio University undergraduate and graduate students who participated in this study.

REFERENCES

[1] I. Vessey and A. P. Sravanapudi, ''CASE tools as collaborative support technologies,'' *Commun. ACM*, vol. 38, no. 1, pp. 83–95, Jan. 1995.

[2] P. Dourish and V. Bellotti, ''Awareness and coordination in shared workspaces,'' in *Proc. CSCW*, 1992, pp. 107–114.

[3] A. J. Ko, R. DeLine, and G. Venolia, ''Information needs in collocated software development teams,'' in *Proc. 29th ICSE*, 2007, pp. 344–353.

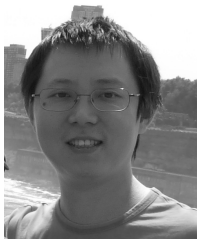
[4] A. Begel, Y. P. Khoo, and T. Zimmermann, ''Codebook: Discovering and exploiting relationships in software repositories,'' in *Proc. 32nd ICSE*, 2010, pp. 125–134.

[5] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson, ''FASTDash: A visual dashboard for fostering awareness in software teams,'' in *Proc. CHI*, 2007, pp. 1313–1322.

[6] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: A study of developer work habits," in *Proc. 28th ICSE*, 2006, pp. 492–501.
[7] M. R. Endsley, "Toward a theory of situation awareness in dynamic systems," *Human Factors, J. Human Factors Ergonomics Soc.*, vol. 37, no. 1, pp. 32–64, 1995.
[8] C. Gutwin and S. Greenberg, "Workspace awareness for groupware," in *Proc. CHI*, 1996, pp. 208–209.
[9] C. Gutwin, R. Penner, and K. Schneider, "Group awareness in distributed software development," in *Proc. CSCW*, 2004, pp. 72–81.
[10] C. Gutwin and S. Greenberg, "The effects of workspace awareness support on the usability of real-time distributed groupware," *ACM Trans. Comput.-Human Interact.*, vol. 6, no. 3, pp. 243–281, Sep. 1999.
[11] C. Gutwin and S. Greenberg, "The mechanics of collaboration: Developing low cost usability evaluation methods for shared workspaces," in *Proc. WETICE*, 2000, pp. 98–103.
[12] C. Gutwin, M. Roseman, and S. Greenberg, "A usability study of awareness widgets in a shared workspace groupware system," in *Proc. CSCW*, 1996, pp. 258–267.
[13] C. Gutwin and S. Greenberg, "A framework of awareness for small groups in shared-workspace groupware," in *Proc. CSCW*, 2002, pp. 411–446.
[14] D. Pinelle, C. Gutwin, and S. Greenberg, "Task analysis for groupware usability evaluation: Modeling shared-workspace tasks with the mechanics of collaboration," *ACM Trans. Comput.-Human Interact.*, vol. 10, no. 4, pp. 281–311, Dec. 2003.
[15] M. D. Storey, C. Davor, and D. M. German, "On the use of visualization to support awareness of human activities in software development: A survey and a framework," in *Proc. SOFTVIS*, 2005, pp. 193–202.
[16] T. D. LaToza and B. A. Myers, "Hard-to-answer questions about code," in *Proc. PLATEAU*, 2010, pp. 8:1–8:6.
[17] B. Dagenais, H. Ossher, R. K. E. Bellamy, M. P. Robillard, and J. P. de Vries, "Moving into a new software project landscape," in *Proc. 32nd ICSE*, 2010, pp. 275–284.
[18] (2013, Apr. 18). *Apache Subversion: Enterprise-Class Centralized Version Control for the Masses* [Online]. Available: http://subversion.apache.org/
[19] (2013, Apr. 18). *Notepad++: A Free Source Code Editor and Notepad Replacement* [Online]. Available: http://notepad-plus-plus.org/
[20] (2013, Apr. 18). *TortoiseSVN: The Coolest Interface to (Sub)Version Control* [Online]. Available: http://tortoisesvn.net/
[21] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *Proc. IEEE Symp. VL*, Sep. 1996, pp. 336–343.
[22] E. Ye, L. A. Neiman, H. Q. Dinh, and C. Liu, "SecondWATCH: A workspace awareness tool based on a 3-D virtual world," *in Proc. 31st ICSE-Companion*, May 2009, pp. 291–294.
[23] R. M. Ripley, A. Sarma, and A. van der Hoek, "A visualization for software project awareness and evolution," in *Proc. 4th IEEE VISSOFT*, Jun. 2007, pp. 137–144.
[24] R. Wettel and M. Lanza, "Program comprehension through software habitability," in *Proc. 15th IEEE ICPC*, Jun. 2007, pp. 231–240.
[25] R. Wettel, M. Lanza, and R. Robbes, "Software systems as cities: A controlled experiment," in *Proc. 33rd ICSE*, 2011, pp. 551–560.
[26] A. Marcus, L. Feng, and J. I. Maletic, "3D representations for software visualization," in *Proc. SOFTVIS*, 2003, pp. 27–ff.
[27] (2013, Apr. 18). *jEdit: Programmer's Text Editor* [Online]. Available: http://www.jedit.org/
[28] (2013, Apr. 18). *Firebird: True Universal Open Source Database* [Online]. Available: http://www.firebirdsql.org/
[29] (2013, Apr. 18). *Hugin: Panorama Photo Stitcher* [Online]. Available: http://hugin.sourceforge.net/
[30] (2013, Apr. 18). *The OpenNMS Project* [Online]. Available: http://www.opennms.org/
[31] (2013, Apr. 18). *VITAL Lab: Virtual Immersive Technologies and Arts for Learning Laboratory* [Online]. Available: http://vital.cs.ohiou.edu/
[32] B. B. Bederson, B. Shneiderman, and M. Wattenberg, "Ordered and quantum treemaps: Making effective use of 2D space to display hier-archies," *ACM Trans. Graph.*, vol. 21, pp. 833–854, Oct. 2002.
[33] (2013, Apr. 18). *Unity: Game Engine* [Online]. Available: http://unity3d.com/
[34] (2013, Apr. 18). *The Notepad++ Download Statistics* [Online]. Available: http://sourceforge.net/projects/notepad-plus/files/stats/timeline
[35] J. Sillito, G. C. Murphy, and K. De Volder, "Asking and answering questions during a programming change task," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 434–451, Jul. 2008.
[36] B. de Alwis and G. C. Murphy, "Answering conceptual queries with Ferret," in *Proc. ACM/IEEE 30th ICSE*, May 2008, pp. 21–30.
[37] C. R. B. de Souza, D. Redmiles, and P. Dourish, "'Breaking the code', moving between private and public work in collaborative software development," in *Proc. GROUP*, 2003, pp. 105–114.
[38] S. G. Eick, J. L. Steffen, and J. Sumner Eric E., "Seesoft-a tool for visualizing line oriented software statistics," *IEEE Trans. Softw. Eng.*, vol. 18, no. 11, pp. 957–968, Nov. 1992.
[39] S. Alam, "Evospaces: 3D visualization of software architecture," in *Proc. SEKE*, 2007, pp. 500–505.
[40] M. Lanza, H. Gall, and P. Dugerdil, "EvoSpaces: Multi-dimensional navigation spaces for software evolution," in *Proc. 13th Eur. CSMR*, Mar. 2009, pp. 293–296.
[41] B. Magnusson and U. Asklund, "Fine grained version control of configurations in COOP/Orm," in *Proc. SCM-6 Workshop Syst. Configuration Manag.*, 1996, pp. 31–48.
[42] W. Appelt, "WWW based collaboration with the BSCW system," in *Proc. SOFSEM*, 1999, pp. 66–78.
[43] X. Wu, A. Murray, M. Storey, and R. Lintern, "A reverse engineering approach to support software maintenance: Version control knowledge extraction," in *Proc. 11th WCRE*, Nov. 2004, pp. 90–99.
[44] J. Froehlich and P. Dourish, "Unifying artifacts and activities in a visual tool for distributed software development teams," in *Proc. 26th ICSE*, May 2004, pp. 387–396.
[45] A. W. J. Bradley and G. C. Murphy, "Supporting software history exploration," in *Proc. 8th MSR*, 2011, pp. 193–202.
[46] T. Ball and S. G. Eick, "Software visualization in the large," *Computer*, vol. 29, pp. 33–43, Apr, 1996.
[47] P. Dugerdil and S. Alam, "Execution trace visualization in a 3D space," in *Proc. 5th ITNG*, 2008, pp. 38–43.
[48] R. Wettel and M. Lanza, "Visual exploration of large-scale system evolution," in *Proc. 15th WCRE*, 2008, pp. 219–228.
[49] R. Wettel and M. Lanza, "Visualizing software systems as cities," in *Proc. 4th IEEE VISSOFT*, Jun. 2007, pp. 92–99.
[50] R. Wettel and M. Lanza, "Visually localizing design problems with disharmony maps," in *Proc. 4th SOFTVIS*, 2008, pp. 155–164.
[51] G. Langelier, H. Sahraoui, and P. Poulin, "Visualization-based analysis of quality for large-scale software systems," in *Proc. 20th ASE*, 2005, pp. 214–223.
[52] S. Boccuzzo and H. C. Gall, "CocoViz with ambient audio software exploration," in *Proc. IEEE 31st ICSE*, May 2009, pp. 571–574.
[53] S. Boccuzzo and H. C. Gall, "Software visualization with audio supported cognitive glyphs," in *Proc. IEEE ICSM*, Sep./Oct. 2008, pp. 366–375.
[54] S. Boccuzzo and H. C. Gall, "Automated comprehension tasks in software exploration," in *Proc. 24th IEEE ASE*, Nov. 2009, pp. 570–574.
[55] A. Sarma, Z. Noroozi, and A. van der Hoek, "Palantir: Raising awareness among configuration management workspaces," in *Proc. 25th ICSE*, May 2003, pp. 444–454.
[56] A. Sarma, D. Redmiles, and A. van der Hoek, "Palantir: Early detection of development conflicts arising from parallel code changes," *IEEE Trans. Softw. Eng.*, vol. 38, no. 4, pp. 889–908, Jul. 2012.
[57] S. Hupfer, L. Cheng, S. Ross, and J. Patterson, "Introducing collaboration into an application development environment," in *Proc. CSCW*, 2004, pp. 21–24.
[58] C. O'Reilly, D. Bustard, and P. Morrow, "The war room command console: Shared visualizations for inclusive team coordination," in *Proc. SOFTVIS*, 2005, pp. 57–65.
[59] R. Hegde and P. Dewan, "Connecting programming environments to support ad-hoc collaboration," in *Proc. 23rd IEEE/ACM ASE*, Sep. 2008, pp. 178–187.
[60] J. Estublier and S. Garcia, "Process model and awareness in SCM," in *Proc. 12th Int. Workshop Softw. Configuration Manag.*, 2005, pp. 59–74.
[61] P. Molli, H. Skaf-Molli, and C. Bouthier, "State Treemap: An awareness widget for multi-synchronous groupware," in *Proc. 7th Int. Workshop Groupware*, 2001, pp. 106–114.
[62] T. Schummer and J. M. Haake, "Supporting distributed software development by modes of collaboration," in *Proc. 7th ECSCW*, 2001, pp. 79–98.

[63] A. Marcus, D. Comorski, and A. Sergeyev, "Supporting the evolution of a software visualization tool through usability studies," in *Proc. 13th IWPC*, May 2005, pp. 307–316.
[64] F. Steinbruckner and C. Lewerentz, "Representing development history in software cities," in *Proc. 5th SOFTVIS*, 2005, pp. 193–202.

CHANG LIU received the Ph.D. degree in information and computer science from the University of California at Irvine, Irvine, CA, USA, in 2002. He is currently an Associate Professor of Computer Science with Ohio University, Athens, OH, USA, where he is the Founding Director of the VITAL Laboratory. His research focuses on software engineering and immersive technologies. Sponsors of his projects include the National Science Foundation and the U.S. Environmental Protection Agency.



XIN YE is currently pursuing the Ph.D. degree with the School of Electrical Engineering and Computer Science, Ohio University, Athens, OH, USA. He received the master's degree in communication and information systems from the Beijing Institute of Technology, Beijing, China, in 2006.

EN YE is currently pursuing the Ph.D. degree with the School of Electrical Engineering and Computer Science, Russ College of Engineering and Technology, Ohio University, Athens, OH, USA. His research focuses on using 3-D virtualization created on top of game engines (e.g., Unity3D) and online virtual worlds to enhance collaboration in software development. He also developed 3-D online learning games to enrich education.