The Effects of Continuous Awareness on Distributed Software Development

Cong Chen
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas, USA
Email: congchen@utdallas.edu

Kang Zhang
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas, USA
Email: kzhang@utdallas.edu

***Abstract*—Awareness, the understanding of others' activities, has been widely accepted as a remedy to many collaboration difficulties in distributed organizations. Continuous Awareness (CA), a continuous support for integrated awareness information across space and time, aims to address the need of continuous awareness information in globally distributed software teams.**

This paper presents an comprehensive evaluation of a CA system, Team Radar, across multiple platforms. The result shows that Team Radar enables developers to receive continuous awareness information effectively and efficiently without the limitations of space and time. Our work provides insights into further exploration of awareness for distributed software development and the design and evaluation of similar tools.

***Keywords*—Distributed software development, virtual team, awareness, collaboration, experiment.**

I. Introduction

Software development is a typical team-based activity. Such teamwork requires intense collaboration among team members and other outside stakeholders. High levels of collaboration necessitate frequent communication between team members, spontaneous information processing, and mutual awareness of each other [1]. Awareness, "an understanding of the activities of others that provides a context for one's activities" [2], is important for efficient collaboration, as it "aids coordination of tasks and resources, and assists transitions between individual and shared activities." [3].

To foster competition and minimize cost, more and more software teams are becoming distributed. Team distribution incurs many difficulties to awareness, such as physical, temporal, and cultural barriers [4]. The loss of awareness not only harms team effectiveness and mutual trust [5], but also affects contributors' willingness and enthusiasm of work [6]. In such a setting, people have to take various alternative approaches to obtaining awareness. It has been argued that the key to promote collaboration in virtual teams is increasing the level of awareness and providing continuous information of ongoing changes [1], [7].

To address the need of continuous awareness information in distributed teams, we propose Continuous Awareness (CA), a continuous support for integrated awareness information across space and time [8]. CA emphasizes continuous awareness support in two aspects: continuous on the spectrum of awareness types and continuous across space and time.

A. Problem

Several challenges have to be addressed to realize CA, including supporting multiple awareness types and dealing with team distribution in space and time.

Software developers often need multiple types of awareness, such as presence awareness [9], workspace awareness [10]), and social awareness [11]), and these needs may change with the context of the work [4]. Unlike traditional paradigms of awareness research that often study single awareness type individually, CA requires providing integrated awareness information to meet users' changing needs. Although CA can be partially supported by many existing tools, such as instance-messaging (for presence awareness and communication) and synchronous online editors (e.g., Google Docs, for activity awareness), a more integrated CA system especially designed and customized for distributed software development is still lacking.

In modern software teams, team members may work at variable locations or in mobile. Much time can be spent in meetings, visiting customers, or moving between locations. At the same time, spontaneous meeting and group discussions continue to be an important factor in work [12]. Unplanned exchange of information can lead to spontaneous collaboration, such as code inspection, pair programming, and problem solving, which is one of the reasons collocated teams perform better than distributed teams [6].

B. Solution

To meet the awareness needs in physically and temporally distributed teams, we propose to realize CA on multiple platforms. Current awareness systems are limited to desktop platforms, which are insufficient for distributed collaboration across space and time. Mobile devices provide unbeatable flexibility than desktop or laptop computers, bringing information, availability, and efficiency to any stage and component of a software process. A mobile awareness approach can fill the information gap of desktop awareness tools and help construct a complete CA system.

We call awareness support on desktop platforms *desktop awareness*, and its support on mobile platforms *mobile awareness*. Being a relatively new concept, there is little validation for the effectiveness and efficiency of a CA system combining desktop awareness with mobile awareness. To address these

978-1-4799-4360-9/14 $31.00 © 2014 IEEE
DOI 10.1109/ICGSE.2014.28

knowledge gaps, we have created *Team Radar*, a complete CA system and evaluated its efficacy for providing continuous awareness information. Team Radar consists of *Team Radar Desktop*, a desktop awareness tool, and *Team Radar Mobile*, a mobile awareness tool.

C. Contributions

Our previous publications have presented the rationales and design of the system [13] and an evaluation of Team Radar Mobile [8]. This paper reports a comprehensive evaluation of the entire Team Radar system for presenting continuous awareness information under several predefined usage scenarios and system configurations. The experiment has found that workspace awareness offered by Team Radar Desktop allows developers to receive conflict warnings more quickly and accurately. Adding a mobile awareness tool and supporting online and offline modes enable developers to receive prompt notifications of the progress and activities of a project without the limitations of space and time. Consistent visualization for integrated awareness information ensures that users perform equally well on desktop and mobile awareness platforms.

The rest of the paper is organized as the following. Section II introduces the background of the research, especially continuous coordination. Section III briefly describes the features and design of the system. Section IV presents the experiments, followed by a discussion of the results in Section V and the implications and limitations of our work in Sections VI. Finally, we conclude the paper in Section VII.

II. Related Work

Our study on CA is based on two important trends of recent research: integrated and contextual awareness support and being continuous. Considering that developers need multiple types of awareness information, and such need often changes with time and location, researchers have proposed to identify users' dynamic information need and provide information based on current context. Holmes and Walker [14] proposed the notion of Developer Specific Awareness (DSA) and realized it in YooHoo, by which awareness information about code changes is automatically filtered based on a developer's own code and interests. Omoronyia et al. [4] suggested that more research is required for context awareness, a crosscutting form of awareness that concerns an individual's changing context in collaboration.

Most existing awareness tools aim to support single type of awareness information. CollabVS [15] attempts to integrate multiple awareness elements into workspace based on the context of the collaboration session as well as the role of the user. CollabVS has integrated many collaboration streams into the Microsoft Visual Studio IDE, such as a presence stream for presence awareness, a concurrent-access stream and a code-sharing stream for workspace awareness and conflict resolution, and a text-chat and audio/video stream for online chatting.

Compared with previous awareness studies, our work targets team distribution and uses mobile awareness platforms and asynchronous communication to deal with geographical and temporal separation. Our awareness tool also supports multiple types of awareness information but utilizes visualization techniques for consistent user experience on multiple platforms.

TABLE I: Awareness information supported by Team Radar [8].

Information type	Benefit	Benefited party
Project status	Monitoring project progress	Manager
Activeness of developers	Understanding developers' workload	Manager
Collaboration patterns	Understanding team organization and group dynamics	Manager
Developers' activities	Understanding work dependency Assisting expert locating and knowledge sharing	Developer
Overlapped work	Reducing merge conflicts	Developer

Continuous Coordination (CC) [16] aims at integrating heterogeneous tools to benefit from both formal and informal approaches. CA is an extension of CC focusing on continuous awareness support. We implement CA by integrating desktop awareness with mobile awareness and supporting synchronous and asynchronous communications. By studying the impact of CA on collaboration, we have also validated the CC paradigm and obtained experience with its implementation.

Evaluating awareness tools is challenging, considering the large number of participants needed and a complex collaboration scenario. Only a few awareness tools have been formally examined, such as the workspace awareness tool Palantír [10]. The evaluation of Team Radar adopts some strategies of previous experiment design, such as choosing a small software project that is likely to create conflicts [15] and reducing individual differences [10]. We have also innovated a new automatic conflict seeding technique that guarantees the consistence of the conflicts and saves human effort.

III. Team Radar

To contextualize the discussion, we briefly introduce the major features of Team Radar. More details of the system can be found in our previous publications [8], [13].

Realizing CA in distributed teams faces three major challenges: multiple types of information needs, geographical distribution, and temporal separation. To meet these challenges, Team Radar is designed with the following goals: (1) supporting multiple types of awareness for different roles and usage patterns in a software team, (2) working on desktop and mobile platforms, and (3) supporting synchronous and asynchronous communication.

Team Radar consists of a desktop client, Team Radar Desktop, and a mobile client, Team Radar Mobile. Team Radar Desktop can be used by any roles in a team who need unobtrusive continuous awareness of the team dynamics. Team Radar Mobile is especially useful for people who often work out of office or office hours (e.g., project managers) and also promotes unplanned collaboration. Team Radar supports both online and offline modes. The online mode enables users to receive awareness information in real-time. For projects across multiple time zones, the offline mode allows awareness information to be downloaded upon request and played asynchronously. Moreover, push notification on Team Radar

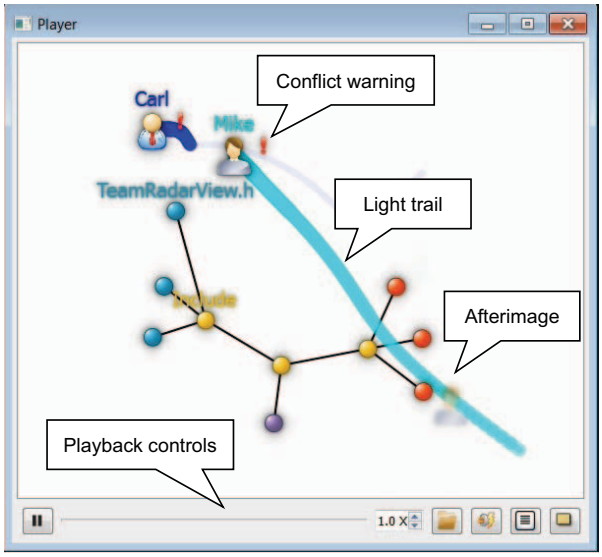
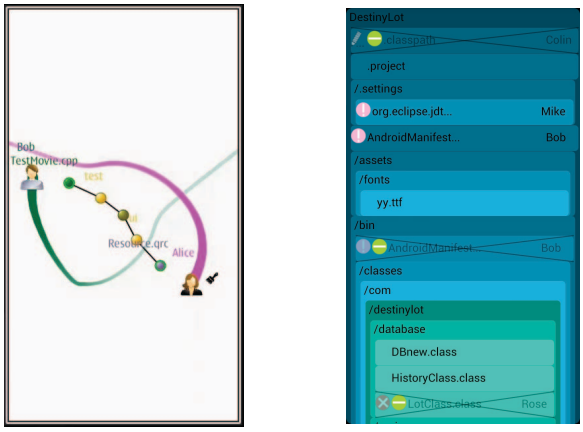


Fig. 1: The visualization on Team Radar Desktop.



(a) Node-edge view (b) Treemap view

Fig. 2: The visualizations provided by Team Radar Mobile.

Mobile ensures that developers will not miss any important updates or collaboration opportunities. Table I summarizes the major types of awareness information Team Radar supports, as well as their benefits in practice.

The Team Radar system adopts a client-server architecture. The server side, Team Radar Server, is a central repository and relay of awareness information. On the client side, Team Radar Desktop is an awareness information monitor and viewer on the desktop, embedded into Qt Creator[1] as a plug-in. Team Radar Desktop supports all major desktop operating systems, including Windows, Mac OS, and Linux. Team Radar Mobile is a mobile awareness client with the same set of features as Team Radar Desktop, except that it does not capture awareness information. Team Radar Mobile supports two mobile operating systems: Symbian and Android.

Team Radar takes the following steps to process awareness information in a team: capturing, dissemination, analysis, and visualization. Team Radar Desktop monitors and captures events of interest in local IDEs and sends them to Team Radar Server, which broadcasts them to other registered clients. Team Radar Desktop and Team Radar Mobile use the same technique to analyze the received information, and present it with intuitive visualization. A set of analytical tools can mine the underlying patterns of collaboration, allowing managers to inspect daily activities, monitor progress, and analyze collaboration issues.

Figure 1 illustrates the animated visualization. Team Radar adopts a tree structure [17] to present the directory structure of a project. The tree is laid out aesthetically and automatically by a force-directed layout algorithm [18] to fully utilize screen space with minimal edge crossings. Non-leaf nodes represent directories and are connected by edges representing their directory relationships. Leaf nodes denote files and are colored by file types. Each online developer is shown as an icon. When a developer starts making changes to a file, her icon will fly close to the corresponding tree node indicating the artifact she is working on. When an icon moves, its afterimage stays, and a light trail shows its path. The accompanying tag shows a developer's current working mode (coding, debugging, etc.). When conflicting changes to an artifact occur, an exclamation mark will give developers an early warning of the potential merge conflict. All local events are stored in the repository of Team Radar Server as event scripts for users to retrieve and replay. As time intervals between events vary, in the offline mode, users are allowed to adjust the playback speed and navigate to certain event through the playback controls.

Team Radar Mobile shares most features with Team Radar Desktop. Making a mobile version of an application, however, is not a simple reimplementation of its desktop counterpart. It involves much attention to the specific requirements for mobile platforms. The characteristics of mobile platforms, such as smaller screen, limited computation power and battery life, and users' intermittent focus (contrasted to more continuous focus on desktop computers) [19] impose special requirements for mobile application design.

Team Radar Mobile is designed to meet the following requirements for mobile visualization we have identified [20]:

1) Maximizing the performance for mobile devices.
2) Utilizing screen space better.
3) Supporting multi-touch.
4) Suiting mobile users' intermittent use.
5) Minimizing power consumption.
6) Keeping style consistent with Team Radar Desktop.

The visualization on Team Radar Mobile is tailored for mobile platforms. As shown by Figure 2, it has two views: a node-edge view and a vertical treemap view. The node-edge view provides dynamic project information similar to its desktop counterpart. It uses a new multi-level force-directed layout algorithm considering screen boundary and shape [8], which utilizes screen space more efficiently and keep style consistency with the desktop version. The treemap view presents

[1]Qt Creator: a C++ IDE, http://qt-project.org

the static structure of the project with better navigation and screen utilization. Contrasted to the original two-dimensional fashion, it divides the screen vertically, leaving enough space for labeling. Additional information, such as modification status, who are working on the files, and conflict warnings, are annotated on the treemap.

IV. Evaluation

We conducted a controlled experiment to evaluate how Team Radar meets the requirements of CA in distributed teams: delivering continuous awareness information. More specifically, the experiment answers the following research questions.

- Q1: How does Team Radar (Desktop) promote workspace awareness for a physically distributed team?
- Q2: How does Team Radar (Desktop and Mobile) promote project and activity awareness for a temporally distributed team?
- Q3: Does the efficacy of the system differ on different platforms?

The first question is answered by an online experiment evaluating user performance for detecting conflicts arising in parallel development. The second question is answered by an offline experiment measuring user performance for perceiving awareness information relevant to project status and teammate activities. We answer the last question by comparing the user performance on Team Radar Desktop and Team Radar Mobile in the offline experiment.

A. Process

The experiment was conducted in three phases. First, there was a training session for the subjects to get familiar with the systems. Second, a pilot study involving a small (separate) group of participants refined the tasks and the questions, ensuring that the experiment environment was functional and the tasks could be completed within reasonable time. Any technical problems arose in the pilot study were solved before the experiment. Finally, we proceeded with the experiment.

B. Configurations and variables

We designed a set of configurations to simulate typical collaboration scenarios in software development. As many mainstream IDEs, Qt Creator supports various stages in software development, including UI design, code editing and navigation, refactoring, debugging, language reference, SCM, etc. Because Team Radar is based on Qt Creator, it is ideal to use Qt Creator as the baseline for comparison. The system can be used under three configurations: the original development environment provided by Qt Creator (O), Qt Creator with Team Radar Desktop (D), and Team Radar Mobile (M), as listed in Table II. Configuration O consists of Qt Creator 2.4.1 with default settings, a SVN plugin for Qt Creator, and a SVN server. The SVN plugin and the SVN server represent a traditional SCM system[2]. Configuration D adds a Team Radar Desktop plugin to Qt Creator, bringing workspace awareness, project awareness, and activity awareness to developers. Configuration M consists of Team Radar Mobile running on a Nokia C7 cell phone. Configurations O and D are on desktop platforms and were used for both online and offline experiment, while configuration M is especially designed for offline experiment.

[2]SVN: http://subversion.apache.org/

TABLE II: Experiment configurations.

Configuration	Tool set	Experiment
O	Qt Creator 2.4.1 SVN plugin for Qt Creator SVN server	Online Offline
D	Qt Creator 2.4.1 SVN plugin for Qt Creator SVN server Team Radar Desktop plugin for Qt Creator	Online Offline
M	Team Radar Mobile running on Nokia C7 with a 3.5" screen and 640 × 480 resolution	Offline

C. Subjects

A total of 31 subjects participated in the experiment voluntarily. The online experiment involved 10 graduate students from our department at our university. The offline experiment used 15 graduate students from the same institution and 6 professionals in software industry. Professional profiles of the subjects were gathered before the experiment to ensure that they have the required skills and experience. As the experiment tasks require only basic understanding of the C++ programming language and the Qt Creator IDE, and the subjects were given step-by-step instructions on what to do, we did not find significant difference in subjects' ability to perform the experiment tasks.

The experiment used a between subjects design. In the online experiment, the subjects were divided into a control group (configuration O) and an experiment group (configuration D), both of size 5. The offline experiment divided the subjects into three groups of size 7 using configurations O, D, and M respectively.

D. Experiment design

To make the experiment realistic, several key attributes were designed carefully: physical and temporal distribution of a team, a software project (*object system*) being developed in parallel, and change conflicts arising during the development. We used the online experiment and the offline experiment to simulate physical and temporal distribution respectively. We also created a fictional software project that is realistic, small, and leads to conflicts [15]. Finally, an automatic virtual conflict technique guarantees that conflicts are created consistently for all the subjects. The following subsections introduce these techniques in detail.

1) Online experiment: The goal of the online experiment is to test the efficacy of Team Radar as a workspace awareness tool for developers to identify teammates' activities and detect conflicts. It simulates a parallel development setting in which conflicts arise because of physical separation and lack of workspace awareness. In the experiment, a team of participants (*subjects*) worked on a software project (*object system*) simultaneously. They were asked to finish several predefined programming tasks involving modifying code and UI of the

Task 2: Move Method

1) Open *Customer.cpp* and locate method *Customer::statement()*
2) Find the switch statement and move it into a new method called *double getAmount(Rental aRental) const*
3) Modify the code of *Customer::statement()* and call *getAmount(each)* to get the value for thisAmount
4) Run the test
5) Commit the revision

A conflicting event was created when the subject modified *Customer.cpp* in Step 2. The event was sent to Team Radar Desktop and also committed to the SVN server. The subject could observe an early warning of the conflict in Team Radar or receive a merge error later from the SVN server when she committed the changes manually in Step 5.

Fig. 3: An example of a programming task.

TABLE III: Variables.

Experiment	Independent variable	Dependent variable
Online	System configuration (O and D)	Conflict detection rate Conflict detection time
Offline	System configuration (O, D, and M)	Answer correctness Question completion time

object system. Several conflicts were created automatically while the subjects were working in parallel. Although seated in the same room, the subjects were not allowed to talk to each other directly, simulating physical separation in a distributed team. Their changes to the code and interactions with the Qt Creator IDE were saved in Team Radar Server as event scripts for the offline experiment.

We made a tool to control and time the progress of the experiment. The subjects had to click a "start" button to read the instruction of a task and click "next" to start working on it. When they noticed a conflict, they would click "conflict" to note the time and location of the conflict. We measured the performance of the subjects for detecting the conflicts with (configuration D) and without Team Radar (configuration O). As shown in Table III, the independent variable is the configuration of the system (O and D), and the dependent variables are the number of subjects who detected the conflicts successfully and the time used for detecting the conflicts.

2) Object system: Ideally, the object system should be an actual software project. Choosing a real project, however, would complicate the experiment by introducing several other variables, such as the subjects' knowledge in the application domain and expertise in programming. The object system we used was a fictional software project adapted from an example in [21], a seminal work on refactoring. The example is a simple movie rental management system, demonstrating how a professional developer would improve the code of a software project through a series of disciplined refactorings. The subjects were asked to perform some of the representative refactorings, including "Extract Method", "Moving Method", "Replace Temp with Query", "Renaming", etc. Each refactoring was used as a programming task, consisting of multiple smaller steps.

We carefully rewrote the program in C++ from the original Java code, balancing simplicity and reality. The program consists of 19 files, 11 classes, and approximately 400 lines of code. Although small in size, the project has the skeleton of a complete software system, including UI, business logic, data persistence, and test code, and adopts typical structure and design of a real software system.

To ensure that all the subjects can finish the programming tasks at consistent pace, we explained the design and code of the system and demonstrated all the programming tasks to the subjects during the training session. The subjects were also provided with detailed written instructions on the steps to perform.

3) Automatic virtual conflict: When designing an experiment for a workspace awareness tool, it is important to ensure that the occurrences of conflicts are consistent across all the subjects. This is challenging, because individuals change the code at their own paces, and a conflict may not occur for a team or may occur at different times for different teams. Previous studies relied on carefully selected programming tasks to create conflicts in parallel changes [15] or added a virtual teammate to introduce conflicts manually at appropriate time [10]. Our pilot study had found that it is too tricky to control the timing of seeding conflicts manually, because the subjects may perform a programming task in different steps, and the order of the steps is often unpredictable.

In addition to selecting programming tasks that are likely to cause conflicts, we invented an *automatic virtual conflict* technique to create conflicts consistently. The virtual conflicts were triggered automatically by the subjects' actions. Each subject was told that she would work in a team with 2 other members. When she made a certain change to the code (called *trigger event*, such as renaming a variable), as determined by her programming task, Team Radar Server would receive the trigger event and respond to her with a *virtual event* describing a conflicting action from a virtual teammate (e.g., renaming the same variable differently). Being unaware of the virtual teammate, the subject would consider that the conflict was caused by a real teammate. The triggers and the responding virtual events were both predefined. No matter in what steps and order the subjects performed a programming task, they always experienced the same set of conflicts, and the contexts (precedent events) of the conflicts remained consistent.

The virtual conflicts were created in both the Team Radar system and the SVN system, so that if the subjects failed to

TABLE IV: Experiment tasks.

Task	Tested features				
	Conflict warning	Work dependency	Project progress	Expert locating	Developer activeness
1. Find the most conflicted file	√	√			
2. Find who conflicted with Mike	√	√			
3. Find the most active developer					√
4. What phase is the project most likely at (e.g, coding and testing)?			√		
5. List the files Mike has edited				√	
6. Tell who has edited rental.cpp				√	

notice the conflicts in Team Radar, they could still receive merge error messages from the SVN system. Note that we could only detect when a subject was about to change the code (e.g., using opening a file as a trigger event) but could not determine when she would commit the change to the SVN server. Because of this limitation, all the virtual conflicts were created before the subjects committed the changes to the SVN repository. In a SVN system, the developer who commits the conflicting change later receives the merge error. For the control group using only the SVN system, the subjects could always receive merge conflicts from the SVN server, because they committed the revisions after the virtual teammate.

Each subject performed 7 programming tasks. 7 conflicts were seeded in 5 tasks, leaving 2 tasks with no conflicts. Figure 3 gives an example of a programming task with a seeded virtual conflict. For each task, the subjects had to finish several steps of the refactoring, run the test code, and commit the changes to the SVN repository. The subjects must ensure that the code passes compiling and the test but were not required to resolve the conflicts.

4) Offline experiment: The offline experiment simulates an asynchronous collaboration scenario in a temporally distributed team and studies how Team Radar promotes project awareness and activity awareness in such a setting. After the subjects in the online experiment finished changing the code, another group of subjects received notification of the updates and were asked to download previously stored awareness events, review the visualization for them, and answer questions about what awareness information was perceived. There were 55 events, and the visualization lasted 47 seconds (idle time between events was shortened). The subjects were allowed up to 180 seconds for each question. Table IV lists the questions, which cover all the information aspects of Team Radar (see Table I for details).

We compared user performance under 3 configurations. Configuration O displays awareness information textually on Qt Creator, while configurations D and M visualize awareness information on Team Radar Desktop and Team Radar Mobile respectively. The independent variable is the configuration of the system (O, D, and M), and the dependent variables are the correctness of the answers and the time spent on the questions (see Table III).

V. Results and findings

We quantitatively measured the performance differences between different groups by performing an Analysis of Variance (ANOVA) on each metric.

TABLE V: Conflict detection rate.

Group	Conflict							Mean	SD
	1	2	3	4	5	6	7		
O	5	5	5	5	5	5	5	5.00	0
D	5	5	5	5	5	4	5	4.86	0.35

TABLE VI: Average conflict detection time (second).

Group	Conflict							Mean	SD
	1	2	3	4	5	6	7		
O	355	130	389	389	83	509	509	337.83	156.81
D	135	108	144	305	86	146	229	164.59	70.65

A. Online experiment

One of the expected benefits of using Team Radar for distributed development is increased workspace awareness of parallel activities. To test this hypothesis, we compared the user performance for identifying conflicts with and without Team Radar. We measured conflict detection rate and time for the effectiveness and efficiency of the system respectively.

For each group, a total of 35 conflicts were created during the online experiment (5 subjects and 7 conflicts per subject). Table V reports the conflict detection rate, measured as the number of subjects who detected a conflict correctly. The subjects in the control group (O) detected all the conflicts when they committed the revisions to the SVN repository manually and found their (virtual) teammates had made conflicting changes to the files. The experiment group (D) detected 34 conflicts, leaving one undetected. We hypothesize that it is because there were two conflicts emerging at almost the same time and the subject failed to notice one of them. ANOVA does not find significant difference between the detection rates of the two groups (p=0.34).

Table VI presents the average time used for detecting the conflicts. ANOVA shows that the experiment group spent significantly less time than the control group (with p=0.03 at 95% confidence level). The subjects in the experiment group could often notice the change conflicts immediately when they arose, while the control group had to wait until the changes were committed manually to the SCM repository. It is interesting to note that Team Radar did not improve the performance for detecting Conflict 5, which was seeded in a simple task taking only a few minutes. In such a short task, the benefit of early awareness may be lost, because the subjects can receive the merge failure message from the SCM system quickly when they commit the revision. This confirms the

TABLE VII: Correctness of the answers to the questions.

Group	Conflict						Mean	SD
	1	2	3	4	5	6		
O	5	5	5	5	6	7	5.50	0.76
M	6	7	6	4	6	7	6.00	1.00
D	6	6	7	5	7	7	6.33	0.75

TABLE VIII: Average completion time of the questions (in second).

Group	Conflict						Mean	SD
	1	2	3	4	5	6		
O	164	118	131	173	106	97	131.50	28.29
M	127	89	53	114	134	64	96.83	30.68
D	101	77	58	89	87	61	78.83	15.37

effectiveness of practitioners' strategy of making frequent and small commits for reducing the risks of merge conflicts [22].

The online experiment aims to answer research question Q1. By simulating a parallel development scenario in a physically distributed team, we measured the effectiveness and efficiency of Team Radar Desktop for delivering workspace awareness information. Compared with the non-awareness approach, increasing workspace awareness can effectively shorten the time to notice conflicting actions while achieving comparable detection rate. Team Radar Desktop can help developers not only take actions earlier to avoid costly merge resolution, but also perceive relevant information with less effort.

B. Offline experiment

One of the key features of a CA system is allowing temporally distributed members to coordinate effectively. The offline experiment concerns how Team Radar delivers various types of awareness information asynchronously to both desktop and mobile platforms.

Table VII presents the correctness of the subjects' answers to the questions, calculated as the number of subjects who answered a question correctly. The performance of group D and M are slightly better than that of group O, and group D better than group M.

In addition to improving the correctness of the answers, Team Radar also shortened average completion time of the questions. As shown in Table VIII, group D finished the questions with the least time, group M was in the middle, and group O was the slowest.

Table IX reports the ANOVA significance test result for configurations O compared to D, O compared to M, and D compared to M. The differences of correctness between the three groups are not significant, while the completion time between groups O and D is significant (with p=0.02 at 95% confidence level) and the completion time between O and M and D and M are not significant.

TABLE IX: ANOVA result (p value) for offline experiment.

	Correctness	Completion time
O-D	0.29	**0.02**
O-M	0.63	0.15
D-M	0.81	0.54

The offline experiment is designed to answer questions Q2 and Q3. By comparing the performances of group D and group O, we can tell that Team Radar Desktop delivers project and activity awareness information more efficiently than the original IDE, and the effectiveness of the system is also slightly improved.

Team Radar Mobile is an essential component of our CA implementation. Due to the limitations of the computation power, screen size, and navigation method, Team Radar Mobile did not perform as effectively and efficiently as Team Radar Desktop, but the difference is not significant. With the advance of mobile computing techniques, this difference may be further diminished in the future. Also note that group M beat group O in almost all the tasks and questions. It is obvious that Team Radar Mobile offers much more freedom than Team Radar Desktop, thus we believe that Team Radar allows developers to monitor project status and coworkers' activities asynchronously and enables a temporally distributed team to collaborate across different time zones.

VI. Discussions

This section summarizes the implications of our research on the theoretical study of awareness and software development practice, as well as the limitations of current work.

A. Theoretical Implications

Current research on awareness has evolved from providing single type of awareness support to offering integrated and contextual awareness services. Developers' changing work location and time in distributed teams, however, are not addressed enough by current study. CA fills this gap by providing an integrated awareness service that supports geographically and temporally distributed teams. One of the major contributions of this paper is the concept of CA and the evaluation of its effectiveness.

Few existing awareness systems meet all the requirements of CA, particularly in globally distributed settings. Our work on Team Radar has explored several ways to address the major challenges of maintaining CA in distributed teams. Specifically, we have proposed to extend awareness support to mobile platforms to tackle geographical and temporally distribution. We have also investigated techniques for visualizing integrated awareness information on multiple platforms consistently. Team Radar is among the few awareness tools that support asynchronous communication, which allows temporally distributed teams to coordinate more conveniently. Our initial exploration can inspire more research on new ways to support CA.

Our study has also evaluated whether CA benefits a distributed team with continuous and flexible access to awareness information. Compared with previous work, our evaluation considers team distribution in space and time, covers multiple platforms, and examines multiple types of awareness services.

The online experiment has shown that workspace awareness offered by Team Radar enables developers to detect conflicts effectively and more efficiently than traditional SCM systems. The use of Team Radar can reduce the risk of merge conflicts and improve developers' confidence in making changes. Team Radar Mobile offers performance and experience comparable to those of Team Radar Desktop and thus complements the deficiency of desktop platforms with more flexibility and portability.

Our practice in mobile awareness is based on general guidelines for mobile awareness design. Our experiment has proved that the experiences from mobile CSCW can be applied to awareness for software development and has contributed design lessons for mobile awareness for distributed software development.

B. Practical Implications

Our study on CA may also provide practical suggestions for the design and evaluation of similar tools. To our knowledge, Team Radar Mobile is the first mobile awareness tool dedicated to software development. Our experiment has shown that using consistent visualization ensures that users receive similar experience on multiple platforms. We have explored techniques for adapting a visualization from desktop to mobile platforms with limited screen space and computational power. Our experience with visualizing awareness information on mobile platforms could benefit the design of similar tools in this area.

Ensuring that conflicts arise consistently for all the participants is critical for a successful workspace awareness experiment. Previous studies relied on carefully chosen programming tasks to create conflicts or introduced conflicts manually at fixed time. We have created a new automatic virtual conflict technique that has better control over the timing of the conflicts and reduces manual effort. During the online experiment, all subjects received the same set of conflicts in the same order and context. The automatic virtual conflict technique can be applied to other similar experiments.

C. Limitations

The experiment was conducted in a controlled lab setting. Although we made the environment as close to the real world as possible, the limitations of the experiment design may affect the generalizability of our findings.

Our current study relies on the assumption that mobile platforms offer more flexibility than desktop platforms and fit developers' changing work location and time better. The experiment shows that our mobile awareness tool achieves similar performance for delivering awareness information compared with the desktop tool, but whether such a tool is equally useful in a real software team still needs to be verified through an empirical study.

Most of the participants of the experiment were graduate students, with at least 5 years' experience in C++ programming, SCM, and IDEs. Such a subject group may only represent typical profiles of fresh graduates in software industry. Thus our findings may not apply to senior developers in the real world, who have more experience with handling workspace conflicts and obtaining awareness information. Our experiment, however, has provided an initial opportunity to understand the nature of CA and laid out foundations for future study.

Due to the limitation of time, the subjects in the experiment experienced frequent change conflicts, and such intensity may not be seen in an actual software project. This design places a higher burden for detecting conflicts equally on both the experiment and control groups, and the advantages of Team Radar should also exist in the real world.

Although the conflicts in the online experiment were created automatically, the timing of them was predefined. In actual settings, the rise of conflicts may be more random, affected by team organization, work division, and developers' commit strategy. As confirmed by our experiment, making small and frequent commits can reduce the chance of conflicts but requires more manual attention. The actual benefit of workspace awareness may increase or decrease depending on the commit strategy used. In either case, workspace awareness offered by Team Radar can help reduce the risk of conflicts or save manual effort.

Finally, in order to compare the performance of two conflict notification mechanisms (traditional SCM and workspace awareness), we made the control group to be able to receive all merge errors from the SCM system. In actual settings, only the developer who commits the conflicting revision later gets the error. This design is biased in favor of the control group, and the Team Radar approach should perform even better than the traditional approach in a real setting.

VII. Conclusions

This paper has presented an evaluation of a continuous awareness (CA) system, Team Radar. Awareness is widely accepted as an effective approach to promoting collaboration, especially in distributed organizations. Existing research on awareness, however, usually focuses on single awareness type and platform and does not address awareness challenges of global distribution. We propose CA as an extension of CC for the need of continuous and integrated awareness information in geographically and temporally distributed teams. We have realized CA in Team Radar through the cooperation of multiple information platforms and synchronous and asynchronous communication modes.

The effectiveness and efficiency of our approach to CA have been evaluated by a controlled experiment. We have found that with increased workspace awareness offered by Team Radar, developers receive warnings of potential merge errors more quickly and accurately. Adding a mobile awareness tool and supporting online and offline modes enable developers to receive prompt notifications of the progress and activities of a project without the limitations of space and time. By visualizing integrated awareness information consistently, users perform equally well with desktop and mobile awareness tools. These results indicate that the design and implementation of Team Radar have met the requirements of CA in physically and temporally distributed teams. Our research can inspire further exploration of new ways to achieving CA and has provided lessons for designing and evaluating CA systems.

REFERENCES

[1] J. Espinosa, S. Slaughter, R. Kraut, and J. Herbsleb, "Team knowledge and coordination in geographically distributed software development," *J. Manage. Inf. Syst.*, vol. 24, no. 1, pp. 135–169, 2007.

[2] P. Dourish and V. Bellotti, "Awareness and Coordination in Shared Workspaces," in *Proc. 1992 ACM Conf. Computer-supported Coop. Work*. ACM, 1992, pp. 107–114.

[3] C. Gutwin and S. Greenberg, "Workspace Awareness for Groupware," in *Proc. Conf. Companion on Human Factors in Computing Systems: Common Ground*. ACM, 1996, pp. 208–209.

[4] I. Omoronyia, J. Ferguson, M. Roper, and M. Wood, "A Review of Awareness in Distributed Collaborative Software Engineering," *Software: Practice and Experience*, pp. 1107–1133, 2010.

[5] S. L. Jarvenpaa and D. E. Leidner, "Communication and trust in global virtual teams," *Organization Science*, vol. 10, no. 6, pp. 791–815, 1999.

[6] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter, "Distance, Dependencies, and Delay in a Global Collaboration," in *Proc. 2000 ACM Conf. Computer Supported Coop. Work*. ACM, 2000, pp. 319–328.

[7] M. Lanza, L. Hattori, and A. Guzzi, "Supporting Collaboration Awareness with Real-time Visualization of Development Activity," in *Proc. 14th IEEE European Conf. Software Maintenance and ReEng.* IEEE Computer Society, 2010, pp. 207–216.

[8] C. Chen, W. Tao, and K. Zhang, "Continuous awareness: A visual mobile approach," *Journal of Visual Languages & Computing*, vol. 25, no. 5, pp. 390 – 401, 2013.

[9] A. E. Milewski and T. M. Smith, "Providing Presence Cues to Telephone Users," in *Proc. 2000 ACM Conf. Computer Supported Coop. Work*. ACM, 2000, pp. 89–96.

[10] A. Sarma, D. Redmiles, and A. Van der Hoek, "Palantír: Early detection of development conflicts arising from parallel code changes," *IEEE Trans. Software Engineering*, vol. 38, pp. 889–908, 2012.

[11] F. Calefato, F. Lanubile, N. Sanitate, and G. Santoro, "Augmenting Social Awareness in a Collaborative Development Environment," in *Proc. 4th Int'l Workshop on Social Software Eng.* ACM, 2011, pp. 39–42.

[12] L. E. Holmquist, J. Falk, and J. Wigström, "Supporting Group Collaboration with Inter-Personal Awareness Devices," *J. Personal Technologies*, vol. 3, pp. 13–21, 1999.

[13] C. Chen and K. Zhang, "Team Radar: Visualizing Team Memories," in *Proc. 6th Int'l Conf. Evaluation of Novel Approaches to Software Eng.*, 2011, pp. 114–120.

[14] R. Holmes and J. R. Walker, "Customized Awareness: Recommending Relevant External Change Events," in *Proc. 32nd Int'l Conf. Software Engineering*. ACM, 2010, pp. 465–474.

[15] P. Dewan and R. Hegde, "Semi-synchronous Conflict Detection and Resolution in Asynchronous Software Development," in *Proc. 10th European Conf. Computer-Supported Coop. Work*. Springer, 2007, pp. 159–178.

[16] D. Redmiles, A. van der Hoek, B. Al-Ani, T. Hildebrand, S. Quirk, A. Sarma, R. Silva Filho, C. de Souza, and E. Trainer, "Continuous Coordination: A New Paradigm to Support Globally Distributed Software Development Projects," *Wirtschaftsinformatik*, vol. 49, pp. S28–S38, 2007.

[17] A. H. Caudwell, "Gource: Visualizing Software Version Control History," in *Proc. ACM Int'l Conf. Companion Object Oriented Programming Systems Languages and Applications Companion*, 2010, pp. 73–74.

[18] P. A. Eades, "A Heuristic for Graph Drawing," *Congressus Numerantium*, vol. 42, pp. 149–160, 1984.

[19] J. C. Tang, N. Yankelovich, J. Begole, M. Van Kleek, F. Li, and J. Bhalodia, "ConNexus to Awarenex: Extending Awareness to Mobile Users," in *Proc. SIGCHI Conf. Human Factors in Computing Systems*, 2001, pp. 221–228.

[20] C. Papadopoulos, "Improving Awareness in Mobile CSCW," *IEEE Trans. Mobile Computing*, vol. 5, no. 10, pp. 1331–1346, 2006.

[21] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999.

[22] J. Estublier and S. Garcia, "Process Model and Awareness in SCM," in *Proc. 12th Int'l Workshop on Software Configuration Management*. ACM, 2005, pp. 59–74.