Supporting Collaboration Awareness with Real-time Visualization of Development Activity

Michele Lanza, Lile Hattori
REVEAL @ Faculty of Informatics
University of Lugano, Switzerland

Anja Guzzi
Delft University of Technology
The Netherlands

***Abstract*—In the context of multi-developer projects, where several people are contributing code, developers must deal with concurrent development. Collaboration among developers assumes a fundamental role, and failing to address it can result, for example, in shipping delays. We argue that tool support for collaborative software development augments the level of awareness of developers, and consequently, help them to collaborate and coordinate their activities.**

In this context, we present an approach to augment awareness by recovering development information in real time and broadcasting it to developers in the form of three lightweight visualizations. Scamp, the Eclipse plug-in supporting this, is part of our Syde tool to support collaboration. We illustrate the usage of Scamp in the context of two multi-developer projects.

I. Introduction

In the development of software systems, teamwork has become the norm rather than an exception [1]. A team of developers working on the same project must deal with parallel development. A number of tools have been helping developers to coordinate parallel development, with Software Configuration Management (SCM) systems occupying a key position. With the increasing need for more effective solutions for parallel development, SCM systems have evolved from preventing concurrent editions on the same file [2] to allowing concurrent editions and offering support for merging them later [3].

The growth of global software development, in which teams of developers do not share the same office, but are rather spread over different locations, affects collaboration issues, such as *awareness*, *communication* and *synchronization* [4], [5]. This has a direct impact on parallel development, since an uncoordinated team – with lack of communication – tends to lose the notion of who is changing which parts of the system (awareness). Indeed, a drawback of the current mainstream SCM systems that becomes evident when awareness drops is the model of propagation of changes: only when a developer checks in his changes, his colleagues will have access to them. As a consequence, the number of concurrent changes to the same artifact tends to increase, resulting in merge conflicts and duplicated work.

Recently, there has been an increased interest in addressing such collaboration issues, which resulted in full-fledged collaborative software development environments, such as IBM's jazz.net [6] or Microsoft's CollabVS [7]. These are industrial environments, especially suited for distributed and collaborative software development. The first focuses on traceability of artifacts, and on clear division of activities to prevent synchronization problems, whereas the latter exploits the creation of various communication channels to promote coordination of activities among teams.

We argue that the key to promote coordination among de-located teams is increasing the level of awareness. We consider awareness as an understanding of the activities of others, providing a context for one's activities [8]. In co-located teams awareness is maintained mainly through interactions among developers, such as informal conversations, pair programming sessions, and expert assistance. When face-to-face communication is lost, the willingness of developers to help others and the ability to spot specialists drops dramatically [5], negatively impacting awareness.

In this context, we promote awareness by visually enriching a developer's Integrated Development Environment (IDE) with information of *who* is changing *what* in the system in real time (*i.e.,* by instantly propagating change information as the changes happen, instead of waiting for a developer to take the action to synchronize his code with the SCM system to check for changes). Indeed, current IDEs have surpassed the notion of being convoluted text editors, and have become environments in which developer assistance appears in many forms, such as refactorings [9] and code completion. We argue that an IDE enriched with visual cues about what is currently happening in the development of a system will contribute to increase developer's awareness, and help him to coordinate his activities with his team members. However, this does not come without a series of technical and conceptual challenges. One problem that needs to be addressed is how to display analysis results without competing with the main goal of developers: writing programs.

In this paper we present one such tool we have been building, called Syde [10]. Syde is an Eclipse plug-in that records *all* code changes performed in multi-developer projects. In a previous publication, we have shown that the recorded information is valuable, for example to refine the notion of code ownership [11]. Syde is a non-intrusive plug-in that silently records all changes and broadcasts them in real-time to all developers working on the same system. The advantage is that changes being performed elsewhere can be directly inspected and reacted upon. As part of our Syde project, we have now built a visual awareness system called Scamp [12]. Scamp processes the information provided by Syde and, using three custom visualizations made available in the Eclipse IDE, helps developers to understand and inspect the changes being performed on the system.

Our work focuses on offering workspace awareness through lightweight and non-intrusive techniques, not requiring the usage of overly complex environments (*e.g.,* jazz.net), but building on Eclipse, a state of the practice IDE. We implemented Scamp to offer three types of visual means to support the understanding of the recent changes being made to the system, superseding and complementing mainstream software configuration management (SCM) systems like CVS and SVN.

We tested Scamp in the context of two multi-developer projects that were developed for several weeks and report on our findings, which show that such "on-the-fly" aids succeed in augmenting awareness among developers, leading to a deeper understanding of the system.

Structure of the paper: In Section II we discuss previous work that relates to ours from the workspace awareness point of view. In Section III we detail our tool infrastructure composed of the Syde and Scamp Eclipse plug-ins. Scamp offers three custom views to augment awareness, the Word-Cloud view, the Buckets view, and the Decoration view. Scamp's views are then showcased in Section IV, where we also detail a qualitative evaluation performed in the form of interviews. We then discuss our findings and conclude in Section V.

II. Related Work

Recently, there has been a significant effort on crafting tools and techniques to promote team awareness in a collaborative context [13], [14], [15], [7], [16].

ProjectWatcher is an Eclipse plug-in that silently auto-commits changes to the source code at pre-defined time intervals [15]. The changes are mined and information is shown in the form of two visualizations: activity awareness shows past and current activities on project artifacts; proximity awareness shows the notion of distance among team members in terms of the systems structure and dependencies. Although these views are closely related to Scamp, no information is given on whether they are dynamically updated as developers work and the system evolves.

Lighthouse is an Eclipse plug-in that aim at avoiding conflicts by propagating change events from Eclipse and SCM among workspaces, and showing them on a view of the emerging design representation of the system [13]. Lighthouse requires a side-by-side presentation of the design representation and the code, which is only feasible if developers work with two screens. Since Lighthouse requires a change in the developer's ways to program, it is likely to face adoption resistance by developers.

Palantír [14] and CollabVS [7] dynamically track code changes to give support for workspace awareness, although they do not focus on visually showing which artifacts are under edition. Rather, both add support for merge conflict detection and resolution, while CollabVS also focuses on communication. Similar to Scamp, Palantír adds decorations on the Eclipse package explorer to show when a java file has been recently changed. This visual cue is combined with a textual view that lists all the emerging conflicts with the goal of warning developers at a stage earlier than when they try to check in their code. On the other hand, CollabVS focuses on creating communication channels to help developers to cooperate, have joint working sessions, and resolve conflicts. In both cases developers do not have the notion of the amount of recent changes per artifact, nor on the order in which they happened.

FASTDash aims at augmenting developers' awareness by visually showing the code base. Information visually presented includes: which team members have source files checked out, which files are being viewed, and which classes and methods are currently under change [16]. Analogous to Scamp, FASTDash shows when two developers are working concurrently on a file, allowing them to preempt merge conflicts. However, FASTDash's visualization front-end is built separately from the IDE, and occupies the entire screen, which can also generate adoption resistance by developers.

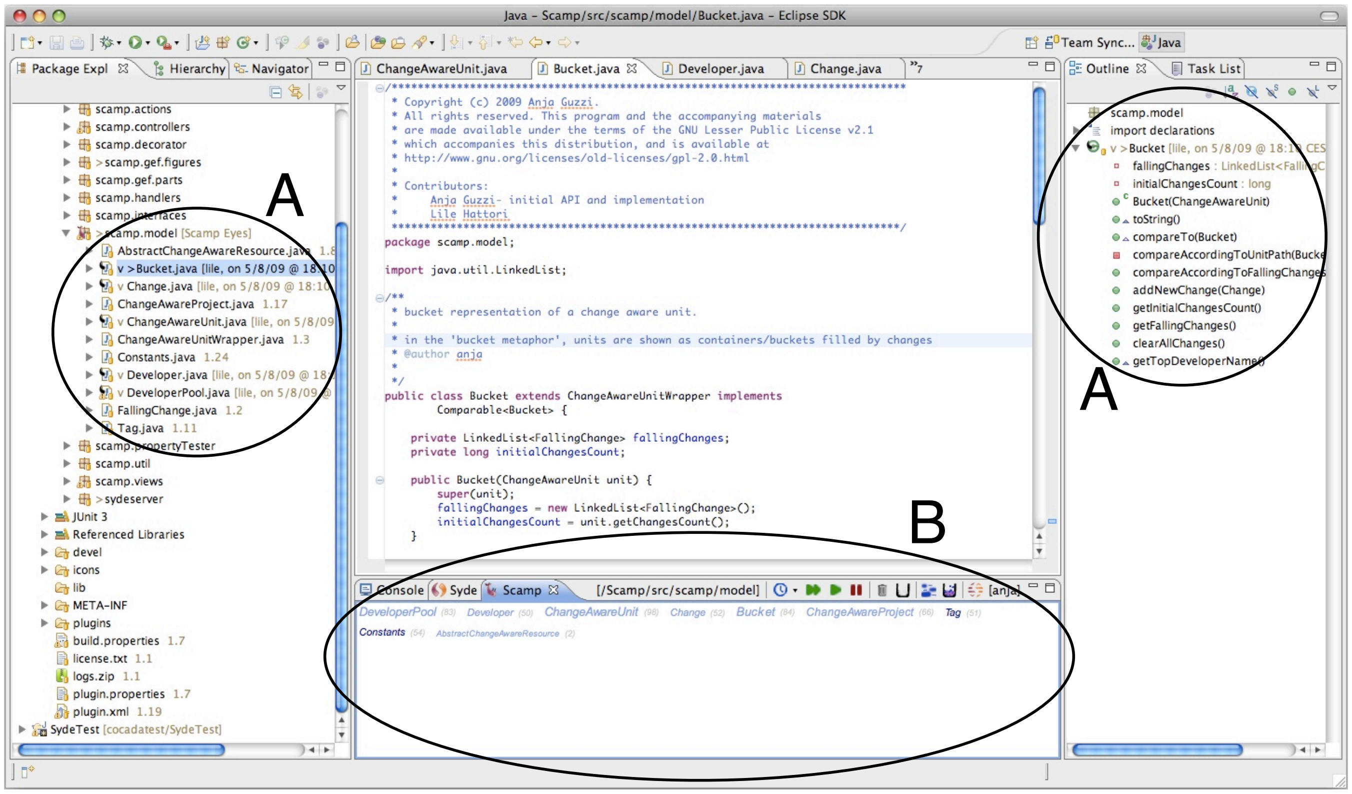
We believe that, in the context of collaborative development, it is important for developers to immediately know who is working on which artifacts. With this simple information, developers are able to coordinate their activities and avoid, among other problems, duplicated work and even conflicts. The challenge is to show this information using a representation of the system that is lightweight, non-intrusive, and easily understandable.

III. Tool Infrastructure: Scamp & Syde

Scamp is an Eclipse plug-in to promote workspace awareness that we recently developed in the context of our work on *synchronous development* [10]. The idea is to provide lightweight extensions to Eclipse to assist developers to collaborate. The main vehicle for our research is *Syde*, where Scamp is its first extension.

In Figure 1 we see the Eclipse IDE enriched with both Syde and Scamp. Syde, described in the next paragraph, is mostly invisible and non-intrusive, while Scamp manifests itself both in the Eclipse package explorer and the outline view (A), and uses the bottom space (B) to display three different types of visualizations.

Syde: With Syde, we translated Robbes' change-based approach [17], supported by the SpyWare platform [18], into a multi-developer context: Syde automatically records *every change by every developer*, where the granularity of changes



```
/*******************************************************************************
 * Copyright (c) 2009 Anja Guzzi.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the GNU Lesser Public License v2.1
 * which accompanies this distribution, and is available at
 * http://www.gnu.org/licenses/old-licenses/gpl-2.0.html
 *
 * Contributors:
 *     Anja Guzzi- initial API and implementation
 *     Lile Hattori
 ******************************************************************************/
package scamp.model;

import java.util.LinkedList;

/**
 * bucket representation of a change aware unit.
 *
 * in the 'bucket metaphor', units are shown as containers/buckets filled by changes
 * @author anja
 *
 */
public class Bucket extends ChangeAwareUnitWrapper implements
        Comparable<Bucket> {

    private LinkedList<FallingChange> fallingChanges;
    private long initialChangesCount;

    public Bucket(ChangeAwareUnit unit) {
        super(unit);
        fallingChanges = new LinkedList<FallingChange>();
        initialChangesCount = unit.getChangesCount();
    }
```

Figure 1. Screenshot of the Eclipse IDE featuring the Scamp and the Syde plugins.

captured is the delta between two save actions. The result is a very detailed project history that can be mined.

Syde is a client-server application, where the client part is installed as part of the Eclipse workbench, while the server part resides on a central server. Syde collects data in a transparent and non-intrusive way: At every save action the server is notified, and notifications about each change that compiles successfully are immediately broadcasted to all clients. The notifications contain information about *who* changed *what* and *when*. We successfully used Syde to improve the notion of code ownership [11] thanks to the higher quality of the change information, compared to standard SCM systems. Syde does not replace, but complements standard SCM systems like CVS and SVN.

Implementation: Syde is implemented in Java. It is composed of a server and a client side (see Figure 2). The client (an Eclipse plug-in) collects information through the use of listeners. The information is then sent to the server's collector, and then broadcasted back to all clients through the notifier. The client's viewer (currently the Scamp plug-in) then allows developers to obtain real-time information about who changed what and when. A developer can obtain a more recent version of a file than the one present in the SCM by using the requestor. This component asks the distributor to send the most recent version to the client.

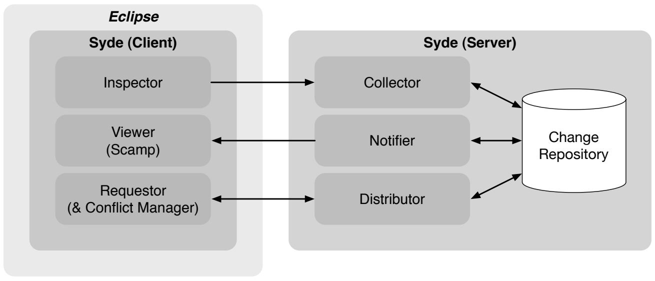


Figure 2. Architecture of Syde.

To display the change information on the client side, Scamp uses three different visual means, described next.

A. WordCloud View

A word cloud is a list of words, which are weighted, colored, and sorted according to specific metrics (not necessarily the same). This visual technique was first introduced by the popular photo sharing web site flickr[1], where words are used to tag pictures.

In Figure 3 we see the word cloud of the vocabulary of Scamp itself, where the size and the color of the words

[1]See http://www.flickr.com

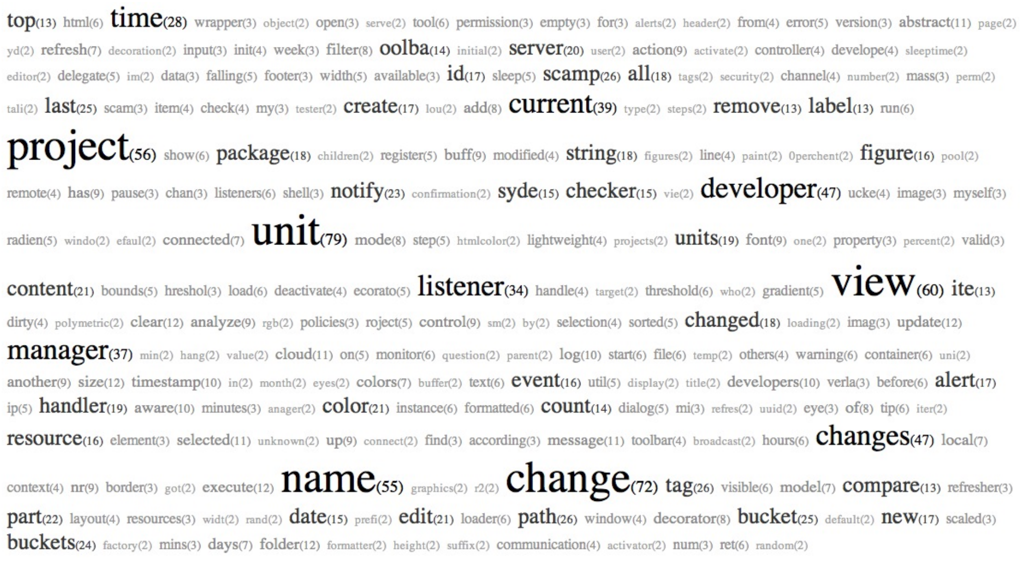


Figure 3. WordCloud of Scamp's vocabulary.

are mapped on the frequency of words in the Scamp source code. The number in parentheses near each word is the actual number of occurrences of the word.

In Scamp we display the names of the classes present in a project. The number of changes that have been performed on each class is used as size metric, while the order indicates the recency of the changes, with most recently changed classes at the top. Each word in the cloud is colored according to the developer who made the most recent change to the class in question. Clicking on a word will take the user to the source code.

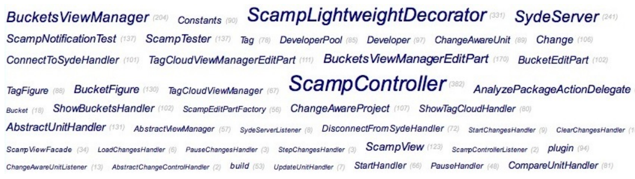


Figure 4. WordCloud of Scamp's changes.

It is possible to choose the time window of past change events. For example, a developer might be interested in seeing which classes changed for the past week, or month. In such case, classes that underwent heavy work will stand out from the others, helping developers to spot where the major changes happened. Choosing a smaller time window – one day, for example – allows developers to focus on what is happening in the present.

In this case, the constant reordering of the words to maintain them ordered chronologically, and changing of color based on the last developer to change a class, allows developers to quickly spot when concurrent work is taking place: If a developer is changing a class, but its word is painted with someone else's color, he immediately knows that someone else is also changing it.

In Figure 4 we see an example of this modified word cloud. All class names have the same color, as Scamp was mostly developed by one person only. This view depicts the development process at a certain moment in time, in this case the developer was focused on changing the classes `BucketsViewManager`, `Constants`, `ScampLightweightDecorator`, and `SydeServer`. The most labor-intensive classes, in terms of number of changes, are `ScampLightweightDecorator` and `ScampController`, as is denoted by their large size.

Although this view is a good means to depict which are the entities of recent interest, it does not display to what extent the various developers contribute to it. The buckets view is devised to display this information.

B. Buckets View

The Buckets View's name comes from the fact that source entities, in our case classes, are displayed as "buckets", which are progressively filled with single changes depicted as small squares. The color of each change denotes the developer responsible for it. Changes follow a chronological order, thus older changes are at the bottom of the bucket, while newer changes appear at the top. Each bucket has the corresponding class name colored according to the developer who owns the code. Ownership in this case is defined as the developer who has performed the greatest number of changes [11]. The time window can also be adjusted according to the developer's will, and the filling patterns of each bucket reflects the effort that each developer spent on it.

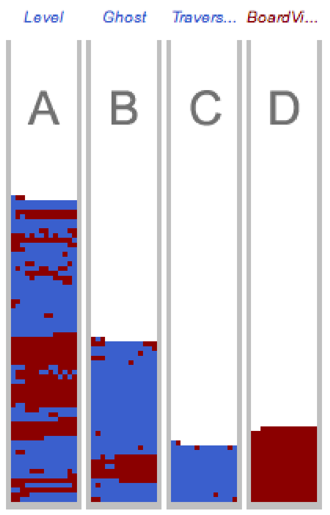


Figure 5. Example patterns of buckets.

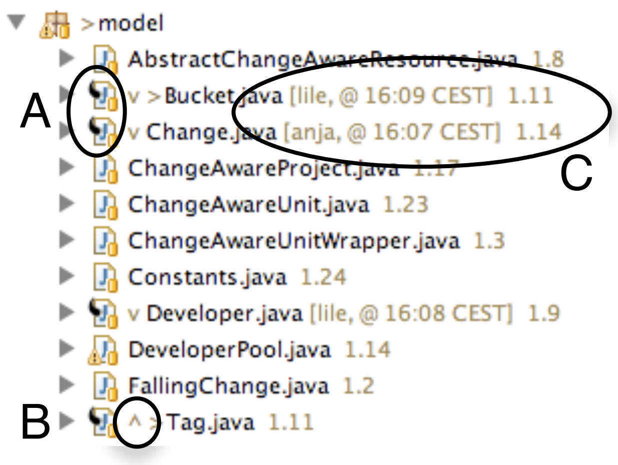

>model
AbstractChangeAwareResource.java 1.8
v >Bucket.java [lile, @ 16:09 CEST] 1.11
v Change.java [anja, @ 16:07 CEST] 1.14
ChangeAwareProject.java 1.17
ChangeAwareUnit.java 1.23
ChangeAwareUnitWrapper.java 1.3
Constants.java 1.24
v Developer.java [lile, @ 16:08 CEST] 1.9
DeveloperPool.java 1.14
FallingChange.java 1.2
^ >Tag.java 1.11
A
B
C

Figure 6. Package explorer decorations.

In Figure 5 we see four patterns that the buckets view can reveal. In case A this class was repeatedly changed by two developers (denoted by the two colors). However, in the middle of the recent lifetime the red developer was most active. In the case of B the situation is the same, but by looking at the height of the bucket, we noted that, considering this time span, file B was modified much less than A. In the case of C the blue developer was the responsible one, with some changes being performed by red, while in the case of D the red developer is the sole responsible for the file.

Both the Word cloud view and the Buckets view occupy the bottom space of the Eclipse user interface. However, sometimes developers prefer to maximize the text editor. To take this into account we also added decorations on the package explorer.

C. *Package Explorer Decoration*

Scamp provides a decoration in the form of small annotations within the Eclipse package explorer, where the files of the project are displayed. If a developer using Syde and Scamp is changing a file, its representation in the package explorer is annotated in three different ways to express that "something is going on" with that file. Scamp's decorations are (1) an overlay icon, (2) an arrow, and (3) a textual annotation, respectively marked as A, B, and C in Figure 6.

The overlay icon (A) denotes the files that have been changed by anyone using Syde and Scamp since the developer has started his working session.

The arrow (B) is placed between the file icon and the file name. The arrow goes up ($\wedge$) if the file has been changed by the user himself, and goes down ($\vee$) if the last person changing the file is someone else.

If someone else is the last person having changed a file, an annotation (C) is displayed after the file name, showing who made the change (username) and the timestamp.

IV. Case Study

We present 2 multi-developer projects that have been developed with the assistance of Scamp. Both projects are group projects developed in the context of the "Programming Fundamentals 2" course given at our faculty. The projects lasted approximately 5 weeks.

A. *The jArk Project*

This project counted at the end 12 packages, 83 classes, 315 methods, for a total of 7,200 lines of Java code. The number of SVN commits in this case was 300, while the number of changes recorded by Syde was 60,166.

In Figure 7 we see the WordCloud view of this project taken during April 2009. This view tells us which classes have changed the most, *i.e.*,`Game` and `GamePanel`, and the most recent changes, *i.e.*,`RifleBullet`, `Vaus`, and `LaserVaus`. There is a distinct pattern visible here: all the most recent changes were performed by the same developer (orange), who is currently the owner of many of those classes, while the red developer worked on many classes, but not so recently.

Figure 8 displays the Buckets view of the last weeks of development before the project ended. The class `Game` is still the main focus of the developers, but development now is equally divided between two developers (red and orange). The classes `Ball` and `Level` are experiencing many changes. We also see a pattern of many identical buckets. These are all sibling classes with a common superclass. The pattern shows a current shortcoming of Syde: if a superclass is changed and compiled, Eclipse automatically recompiles all subclasses, even if they did not change.

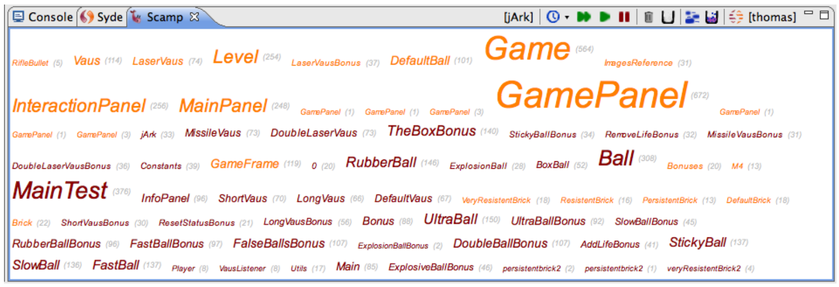


Figure 7. WordCloud view of the jArk project.

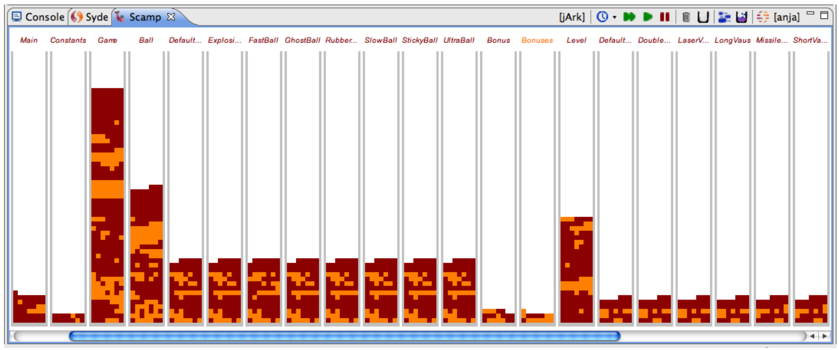


Figure 8. Buckets view of the jArk project.

B. The PacMan Project

This project counted at the end 5 packages, 59 classes, 382 methods, for a total of 3,978 lines of Java code. The number of SVN commits in this case was 170, while the number of changes recorded by Syde was 35,066.

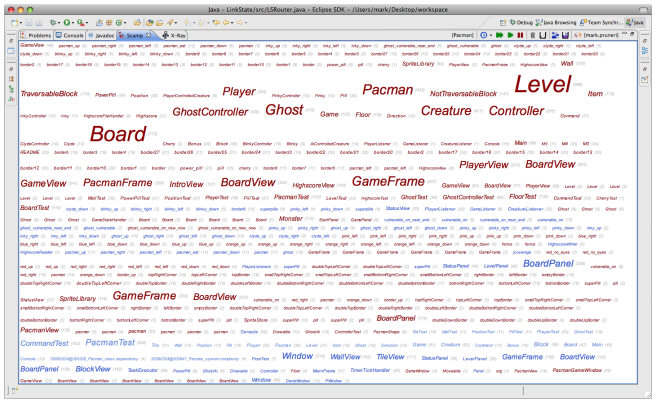


Figure 9. WordCloud view of the PacMan project.

In Figure 9 we see the WordCloud view depicting the changed entities during the last month of the project. The developers confirmed that the most visible classes, such as `Board`, `Level`, `Ghost`, etc. were indeed the central domain classes in the system. The view reveals also that this project is, at this stage, undergoing many changes on entities that have seldom changed, denoted by their small size (= few changes), while being at the top of the cloud (= changes are recent). The developers also mentioned that some of the classes, such as `GameFrame` had been moved from one package to another. The class name is visible in more than one place of the cloud, with the top word corresponding to the newest path of this class.

As the development evolves, the old words will go to the bottom of the cloud until they disappear from the view. One possibility here is to grey out the name to imply that the old words do not have a link to the class anymore.

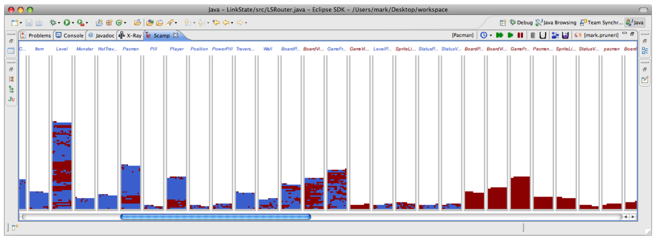


Figure 10. Buckets view of the PacMan project.

In Figure 10 we see a Buckets view of the last month of development of the PacMan project. Some patterns are visible, such as collaboratively edited files (exhibiting a blue-red pattern), heavily edited files (the tallest buckets), or files edited by one developer only (on the right side).

C. Qualitative Evaluation

The low number of projects and people participating in those projects (a maximum of 2 people changed these systems) does not permit us to perform any statistical evaluation, which is however part of our future work. We present here a qualitative evaluation, performed in the form of interviews, using the questionnaire listed in Table I.

No.	Question
1	How often did you use Syde and Scamp?
2	Were the Scamp views always visible?
3	Did you usually work physically close to your team mate(s)?
4	Were you aware of what the other(s) were doing by looking at the information shown through the views?
5	Was the constant refreshing of the views disturbing?
6	When you noticed that the other(s) were working on the same class did you (a) talk to each other, (b) wait and stop editing, (c) ignore and continue editing, (d) rush to commit changes?
7	How often did you commit your code?
8	How often did you have to merge code that was already committed?
9	How were you aware that certain file commits would generate merge conflicts?
10	When you were not physically close, was Scamp more useful than when you were working nearby and if yes, why?
11	What is the most useful feature of Syde/Scamp and why?
12	Which improvements can you suggest?
13	Classify the following statements on a scale from 1 (= strongly disagree) to 7 (= strongly agree) a. It was easy for me to see what my co-worker was doing b. it was not useful to see what my team mate(s) were doing c. I liked seeing my team mate(s) presence even when I did not have any direct benefit such as conflict detection d. I was not comfortable with others seeing information about my activity
14	Classify the following improvements in terms of usefulness on a scale from 1 (= completely useless) to 7 (= very useful) a. Create a filter to allow developers to choose which file types and which folders they are interested in b. Allow a developer to compare the most up-to-date version of a file with his version c. Add an option to automatically login to Syde and load the Scamp views (a.k.a. auto-connect) d. Show the status (online/offline) of each developer e. Provide a mechanism to resolve merge conflicts automatically f. Show who is responsible for each method inside a class that is being edited

Table I
QUESTIONNAIRE USED FOR THE INTERVIEWS WITH THE DEVELOPERS.

The interviews were conducted individually, and on a face-to-face basis and lasted each approximately 30 minutes.

Q1: How often did you use Syde and Scamp? Three developers reported that they used it most of the time, with some exceptions when they occasionally forgot to activate Scamp, especially when they were doing small changes. One developer reported that he underwent an initial phase of acceptance, during which he more or less consciously forgot to use the tools. With time, he got used to the new tool and incorporated it on his "development process".

Q2: Were the Scamp views always visible? Most developers answered positively; only in some cases they minimized the views at the bottom of the IDE to obtain a larger text area. We guess this is the case when they were inspecting some large piece of code.

Q3: Did you usually work physically close to your team mate(s)? One pair of developers worked in pair programming the first couple of weeks, after that they worked mostly from home, coordinating their activities mostly through the Scamp views, instant messaging, and as a last resort through the use of SVN. The other pair worked mostly side-by-side, in a setup where one could actually see the other's screen.

Q4: Were you aware of what the other(s) were doing by looking at the information shown through the views? The answers in this case were mostly a clear "yes". In some borderline cases where too much happened at once, the developers talked to each other to clarify.

Q5: Was the constant refreshing of the views disturbing? The answers were a clear "no" in three cases. One of the developers reported that he felt a little bit disturbed, but this feeling disappeared as soon as he concentrated on programming.

Q6: When you noticed that the other(s) were working on the same class did you (a) talk to each other, (b) wait and stop editing, (c) ignore and continue editing, (d) rush to commit changes? The reactions were unanimously that the developers talked to each other. Any other answer would have implied a delay.

Q7: How often did you commit your code? The answers varied from one a day to several times a day. This is fairly in line with common practice.

Q8: How often did you have to merge code that was already committed? This seems to have happened more in the beginning of the project. Later on the developers, also through the use of the Scamp views, started to have a smoother integration of the individual efforts.

Q9: How were you aware that certain file commits would generate merge conflicts? The developers who worked separately mostly used the Scamp views to preempt conflicts, which then also would spawn forth discussions through instant messaging. Even for the pair working together, Scamp was useful to avoid emerging conflicts in some situations. However, for them it was easier to coordinate their activities by talking to each other constantly.

Q10: When you were not physically close, was Scamp more useful than when you were working nearby and if yes, why? The pair that worked separately answered yes, and explained that the Scamp views were very helpful to have clarification discussions.

Q11: What is the most useful feature of Syde/Scamp and why? Two developers said that it was useful to see which files the other person was editing, and also to work on something else in case he saw that a certain file was being worked on. The other developer said that it was useful to have the possibility to see "live" who was doing what, thus also minimizing redundancies and helping to have focused discussions about the imminent things to be done. One developer really appreciated the WordCloud view, because he could quickly spot which were the most important classes (those that they have put most of their effort on) of the system within a certain period of time.

Q12: Which improvements can you suggest? One developer mentioned "auto-connect", i.e., Syde and Scamp should automatically start instead of needing to be switched on. This is a good sign, the two tools had become part the working environment, and having to repeatedly explicitly switch them on was even seen as a nuisance. Two developers mentioned that it would be good to see even finer-grained real-time change information at the level of single lines of source code. We are currently working on this. The developer who really appreciated the WordCloud view suggested to add a scroll bar, and to limit the growth of words to a maximum threshold. This reinforces the assumption that this view was heavily used throughout the project development.

Q13: Classify the following statements according to the scale: 1 = strongly disagree, 2 = disagree, 3 = partially agree, 4 = neutral, 5 = partially agree, 6 = agree, 7 = strongly agree. This question is extracted from the questionnaire of a previous study conducted in the context of CollabVS [7].

In their survey, these questions were classified as "presence (awareness) stream related questions". They applied this survey after conducting a user study with 16 participants grouped into pairs, were each pair had 60 minutes to implement a given task, and could freely use CollabVS to accomplish it.

We summarized the answers indicated by the developers in our case study, and the mean value of the answers given by the CollabVS users in Table II.

From the answers to the first question, there is a large gap between the opinions of the two pairs. This is due to the fact that developers 3 and 4 worked all the time side-by-side, a situation in which Syde and Scamp have little added value, whereas developers 1 and 2, who worked remotely, had the opportunity to benefit from information broadcast by our plug-ins to increase their awareness on the other's activities.

Answers	Dev1	Dev2	Dev3	Dev4	CollabVS
It was easy to see what my team-mate(s) were doing.	7	6	2	3	5.88
It was not useful to see what my team mate(s) were doing.	2	3	2	2	2.22
I liked seeing my team mate(s) presence even when I did not have any direct benefit such as conflict detection.	5	4	6	7	5.56
I was not comfortable with others seeing information about my activity.	2	3	1	2	2.00

Table II
PRESENCE (AWARENESS) RELATED QUESTIONS EXTRACTED FROM [7].

The results obtained from the first pair of developers are similar to the results in CollabVS study, which emphasizes the need for awareness support in cases where developers lose face-to-face communication.

For question 2, even though the second pair was not sure whether the plug-ins helped them to maintain a high level of awareness, they still found the available information useful, which complies both with the first pair and the CollabVS results.

As far as the aspect of privacy is concerned, the answers are indicating that the developers were not bothered by the fact that someone else could see what they were doing, also meeting the CollabVS results. In fact, one can argue that any instant messaging application like Skype or iChat is displaying the same amount of privacy information, and people have gotten used to them.

Q14: Please classify the following improvements in terms of usefulness according to the scale: 1 = completely useless, 2 = useless, 3 = probably useless, 4 = neutral, 5 probably useful, 6 = useful, 7 = very useful. We summarized the answers indicated by the developers in Table III.

The answers indicated by all developers show that there is a wish for having more functionality available, which is a strong argument in favor of tools like Syde and Scamp.

Observations: Overall, it seems that the developers greatly appreciated the additional information that the Scamp views provided them, without being bothered by their presence.

A fact that struck us when analyzing the data and conducting the interviews was that the developers changed their behavior after using Scamp for some time. While at the beginning Scamp's views were ignored, with time the

Answers	Dev1	Dev2	Dev3	Dev4
Create a filter to allow developers to choose which file types and which folders they are interested in.	6	7	7	7
Allow a developer to compare the most up-to-date version of a file with his version.	6	6	7	7
Add an option to automatically login to Syde and load the Scamp views (a.k.a. auto-connect).	7	7	7	7
Show the status (online/offline) of each developer.	6	6	5	5
Provide a mechanism to resolve merge conflicts automatically.	6	7	3	7
Show who is responsible for each method inside a class that is being edited.	5	6	6	6

Table III
ANSWERS GIVEN TO QUESTION 14

developers started not only to look at them to understand what was going in on, especially when they were physically distant, but also started to react on the events happening in the views. The reactions usually came in the form of instant messages going back and forth, asking for explanations. This is much in line with our intentions about Syde and Scamp, which is not only to augment awareness, but also to modify the behavior of programmers to increase efficiency and minimize conflicts. This indeed happened in both projects, where both Syde and Scamp became second nature.

D. Reflections

While monitoring the two projects, a number of patterns emerged from the views offered by Scamp.

The WordCloud view highlights, at every moment during development, which classes are of interest. At first sight, one may think that for a developer this view may be less interesting, as opposed to the Buckets view, because usually a developer knows on which part of the system people are working. However, this is not true, and without the help of Scamp, developers only know that something has changed only after it has been committed to the repository. According to the feedback given by the developers of our case study, the information on WordCloud was fundamental to spot when a source code started to be edited concurrently, which immediately drove them to coordinate their work by contacting each other through instant messaging.

The developers also saw great benefits in the Buckets view and the decorations. The Buckets view is a good means at supporting the notion of "real time": at each and every change by anyone working on the project, the buckets are updated. This helps to instantaneously preempt conflicts that would emerge at the level of an SCM commit: if a bucket quickly receives many, differently colored items it is obvious that a problem is in the making. The drawback of the Buckets view is that its visual language needs to be internalized, while the understanding of the other two views is more immediate and intuitive.

The decoration is certainly the least intrusive view, and the developers appreciated that it smoothly merged into the development environment without occupying any extra screen space. Of all views, this is probably the one that would have the least acceptance problems, even by non-visual individuals.

E. Threats to Validity

Although this initial case study shows that Scamp's lightweight and non-intrusive views were able to increase the level of workspace awareness, the limited size of the projects and number of developers involved prevent us from drawing strong conclusions.

A number of other characteristics about the developers might have influenced the results. First of all, working in pairs, they only have one communication path to keep track of. In addition, one of the groups reported that they worked physically together most of the time, sometimes even doing pair programming. In the latter case, Scamp's views are not useful, and indeed they deactivated them.

Another noteworthy aspect is that the developers in question were fairly new to the Eclipse IDE, *i.e.,* they had not settled yet on a specific way to use the IDE, and accepting an addition like the two plug-ins was not problematic. It is probable that more expert programmers will oppose more resistance to changing their ways of working, but this is a reality that any recommender system must face.

Lastly, the qualitative evaluation could have been biased by the fact that the authors of these tools are the same persons who evaluate their course work. However, it is important to point out that the professor responsible for the course "Programming Fundamentals 2" is not involved in this work, and that the students volunteered themselves to use the tool knowing that this would not influence whatsoever their grades on the course.

V. CONCLUSIONS AND FUTURE WORK

Collaborative and distributed software development is a growing phenomenon. This is pushed on the one hand by the surge of outsourcing and offshoring of projects, where development teams distributed across the globe develop the same system. On the other hand, this is nothing new and is a standard practice in open-source development since years.

Such phenomenon is expected to have a number of benefits compared to in-site software development: increased productivity, cost reduction, sharing of knowledge, etc. However, a number of collaboration issues that are trivial to solve when teams are co-located, become problematic when distance separates teams.

In this paper, we presented a lightweight and non-intrusive approach to augment team awareness, *i.e.,* to increase the level of developers knowledge about what is happening in the project in terms of implementation changes. We aimed at overcoming some of the drawbacks caused by the lack of face-to-face communication, such as the loss of who is knowledgeable about a class. The case study indicates that developers benefited from the increased level of awareness. They were able to avoid duplicated work, and to reduce the number of merge conflicts.

As future work, we plan to enrich the capabilities of our Syde and Scamp plug-ins in different dimensions. First of all, we will apply real time reverse engineering on the code to show to developers the emerging design of the system. The great challenge of this approach is how to visually represent the system in limited screen space remaining non-intrusive. We also plan to enrich the event notifications by directly alerting developers when conflicts might be emerging. Finally, Syde and Scamp will become more interactive, helping developers to discover who they should look for to ask about the functionalities of a class, or helping them to resolve merge conflicts. In the long term, Syde and Scamp should silently collect change information to not only increase developers awareness, but also to assist them to coordinate their work, and thus, proactively collaborate.

Acknowledgment

We gratefully acknowledge the financial support of the Swiss National Science foundation for the project "REBASE" (SNF Project No. 115990). We thank the students participating in the projects and their willingness to participate to the interviews. We thank Romain Robbes and Alberto Bacchelli for providing helpful comments.

References

[1] A. Sarma, G. Bortis, and A. van der Hoek, "Towards supporting awareness of indirect conflicts across software configuration management workspaces," in *Proceedings of ASE 2007 (22nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE CS Press, 2007, pp. 94–103.

[2] W. F. Tichy, "Rcs – a system for version control," *Software Practice and Experiece*, vol. 15, no. 7, pp. 637–654, 1985.

[3] J. Estublier, D. Leblang, A. van der Hoek, R. Conradi, G. Clemm, W. Ticky, and D. Wiborg-Weber, "Impact of software engineering research on the practice of software configuration management," *ACM Transactions on Software Engineering and Methodology*, vol. 14, no. 4, pp. 383–430, 2005.

[4] R. Sangwan, M. Bass, N. Mullick, D. Paulish, and J. Kazmeier, *Global Software Development Handbook*. Auerbach Publications, 2006.

[5] J. Herbsleb, A. Mockus, T. Finholt, and R. Grinter, "Distance, dependencies, and delay in a global collaboration," in *Proceedings of CSCW 2000 (ACM Conference on Computer Supported Cooperative Work*. ACM Press, 2000, pp. 319–328.

[6] R. Frost, "Jazz and the eclipse way of collaboration," *IEEE Software*, vol. 24, no. 6, pp. 114–117, 2007.

[7] R. Hegde and P. Dewan, "Connecting programming environments to support ad-hoc collaboration," in *Proceedings of ASE 2008 (23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE CS Press, 2008, pp. 178–187.

[8] P. Dourish and V. Bellotti, "Awareness and coordination in shared workspaces," in *Proceedings of CSCW 1992 (ACM conference on Computer-supported Cooperative Work)*. ACM Press, 1992, pp. 107–114.

[9] M. Fowler, *Refactoring - Improving the Design of Existing Code*. Addison-Wesley, 1999.

[10] L. Hattori and M. Lanza, "An environment for synchronous software development," in *Proceedings of ICSE 2009 (31st ACM/IEEE International Conference on Software Engineering - New Ideas and Emerging Results Track)*. IEEE CS Press, 2009, pp. 223–226.

[11] ——, "Mining the history of synchronous changes to refine code ownership," in *Proceedings of MSR 2009 (6th IEEE Working Conference on Mining Software Repositories)*. IEEE CS Press, 2009, pp. 141–150.

[12] A. Guzzi, "Supporting collaboration awareness in multi-developer projects," Master's thesis, University of Lugano, Jun. 2009.

[13] I. da Silva, P. Chen, C. V. der Westhuizen, R. Ripley, and A. van der Hoek, "Lighthouse: Coordination through emerging design," in *Proceedings of ETX 2006 (OOPSLA Workshop on Eclipse Technology eXchange*. ACM Press, 2006, pp. 11–15.

[14] A. Sarma, D. Redmiles, and A. van der Hoek, "Empirical evidence of the benefits of workspace awareness in software configuration management," in *Proceedings of FSE 2008 (16th ACM SIGSOFT International Symposium on Foundations of software engineering*. New York, NY, USA: ACM Press, 2008, pp. 113–123.

[15] K. A. Schneider, C. Gutwin, R. Penner, and D. Paquette, "Mining a software developers local interaction history," in *Proceedings of MSR 2004 (1st International Workshop on Mining Software Repositories*, 2004, pp. 106–110.

[16] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson, "FASTDash: a visual dashboard for fostering awareness in software teams," in *Proceedings of CHI 2007 (25th SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2007, pp. 1313–1322.

[17] R. Robbes, "Of change and software," Ph.D. dissertation, University of Lugano, Switzerland, Dec. 2008.

[18] R. Robbes and M. Lanza, "Spyware: A change-aware development toolset," in *Proceedings of ICSE 2008 (30th ACM/IEEE International Conference in Software Engineering)*. ACM Press, 2008, pp. 847–850.