ABSTRACT
In a multi-developer project, team collaboration is essential for the success of the project. When team members are spread across different locations, informal interactions are lost, having an impact on individual awareness of the activity of others. In this scenario, collaboration becomes a challenge. A number of works have tried to reestablish team awareness by sharing change information across developers’ workspaces. The main challenge of these approaches is to balance the tradeoff between offering useful information about the activity of others and avoiding information overload.

In this work, we address the challenge of enhancing group awareness and stimulating collaboration by providing relevant information to developers in a non-intrusive manner. The novelty of our approach is that we model changes as first-class entities to deliver precise change information to targeted developers.

Categories and Subject Descriptors
D.2.2 [Software Engineering]: Design Tools and Techniques—Distributed/Internet based software engineering tools and techniques; D.2.9 [Software Engineering]: Management—Programming teams

Keywords
Collaboration, awareness, assistance, visualization.

1. PROBLEM STATEMENT
Teamwork has become the norm rather than an exception in the development of software systems [17]. A team of developers working on a project must coordinate their activities and deal with parallel development. A key aspect of team collaboration is awareness: An understanding of the activities of others that provides a context for one’s activities [5]. Awareness can be seen as a means by which team members can become aware of the work of others that is interdependent with their current tasks, therefore enabling better coordination of teams [3]. This is mainly obtained through communication, where informal interactions play an important role, and by inspecting code changes on software configuration management (SCM) systems.

In the context of distributed software development, when teams of developers do not share the same office, they often struggle with communication problems, directly affecting team awareness and coordination. Herbsleb et al. assessed that when face-to-face communication is lost, the willingness of developers to help others and the ability to spot specialists drops dramatically [12]. In addition, Damian et al. observed that distributed teams are affected by communication breakdowns due to different organizational cultures and dynamic social networks, affecting awareness and generating broken integration builds [3].

When the level of awareness drops due to communication restrictions, alternatives to augment it should be considered. A significant effort has been made to recover the level of awareness by enhancing the coordination capabilities of SCM systems. A drawback shared by current SCM systems is their strategy of propagation of changes: Only when a developer checks in his changes, will his colleagues have access to them; and only when his colleagues synchronize their code with the repository, will they become aware of the new changes. A number of works have addressed this problem by providing real-time information of ongoing changes, and alerting developers of emerging conflicts [2, 11, 18, 19].

The main challenge faced by these approaches is how to balance the tradeoff between providing enough information about the activities of the team members, and avoiding overloading developers with irrelevant information. Our work tackles this challenge with the goal of enhancing group awareness and collaboration. To provide relevant information to developers in a non-intrusive manner, we are building an infrastructure – called Syde – that monitors ongoing changes on developers’ workspaces to multicast change and conflict information. Some of the collaboration aspects that we address are:

Change activities. It is important that developers have the notion of who else is changing the same artifacts or those artifacts directly related. Some of the problems that occur when they lose this notion are: duplicated work, merge conflicts, and broken builds. We propose to propagate in real-time change information of artifacts that are interesting to a developer, and show it through non-intrusive visualizations.

Conflict detection and resolution. SCM systems have a reactive behavior towards conflict detection, since a developer will only be aware of a conflict when he tries to check in his version of the code. We propose a proactive approach to the problem, in which developers are warned about emerging conflicts at early stages and can react accordingly.

Expert recommendation. Another aspect of awareness that is affected by the lack of informal interactions is the ability to spot specialists. We propose to mine Syde’s change history.
logs to identify and recommend experts of artifacts. Such a recommender can assist developers to look for help when they are struggling to understand a piece of code. In addition, it can help newcomers to quickly gain habitability of the code (the ability to understand the software’s construction and intentions and to change it comfortably and confidently[7]).

The fundamental difference between previous approaches and ours is that Syde encapsulates changes as first-class entities [15]. In other words, systems are modeled as abstract syntax trees (AST), and changes are operations on the AST. The entire process of capturing, processing and multicasting change information happens at the AST level, as opposed to current file-based approaches.

2. RELATED WORK

A significant number of tools have been developed to support collaborative development for globally distributed teams. Jazz.net [6] is a full-fledged platform, built on top of Eclipse or Visual Studio, that creates a new environment to support intra and inter-team collaboration, automation, and traceability of code, tasks and issues. Jazz offers significant support for team members to maintain a good level of awareness. However, it models the system under development as a file-based structure, losing the language dependent details about the changes that were performed.

Indeed, the continuous adoption of language independent SCM systems in the context of team-based development has influenced the creation of solutions to overcome the workspace isolation enforced by them [17]. However, these solutions continuously use the file as the smallest entity that can change, ignoring the fact that a file can contain methods that are used in different functionalities. As an example, two developers might be working on different methods of a file for distinct functionalities, which in practice does not create any conflict, while a file-based approach could erroneously indicate a potential problem.

Palantir [18] and ProjectWatcher [19] are Eclipse plug-ins that monitor developers’ workspaces to immediately propagate change related information. Palantir focuses on detecting merge conflict. When conflicts emerge, it visually notifies developers and shows the severity of the conflict. ProjectWatcher records ongoing changes on a shadow CVS repository at fixed time intervals to visually inform developers who is working on which files. In both cases, change information is captured at the file level, although Palantir informs the user of conflicts at the code entity level.

Lighthouse [2] and FASTDash [1] use visualizations to show which classes are being changed in real-time. Both show change information at object-oriented level (classes, methods, field), thus going beyond previous file-based approaches. However, the downside of these approaches is that the visualizations are not integrated into the IDE. Hence, if a developer wants to see both the code and changing model of the system, he needs to either have two screens, or switch between windows.

Lastly, CollabVS [11] enriches Visual Studio IDE, and TUKAN [20] enriches Visual Works with both communication and change track extensions. They track changes at source code level and visually shows them to developers upon request. CollabVS is heavily focused on offering different communication channels, such as integrated instant messaging (IM) and videoconferencing services. TUKAN, on the other hand, focuses on showing to developers who else is working on closely related artifacts.

The demand for workspace awareness is becoming urgent as globally distributed team collaboration becomes the state of the practice. Although the solutions discussed above increase workspace awareness by working around some of the limitations imposed by largely adopted SCM systems, they lack on either providing enough fine-grained change information, or on maintaining a non-disturbing approach. We claim that current file-based and language independent SCM models that are adopted by current workspace awareness tools are the cause of these limitations.

3. SYDE

Syde is our infrastructure to promote collaboration awareness. It is a client-server application, where clients are Eclipse plug-ins. Syde tracks changes at the AST level for Java files and at file level for other file types. In this section, we concentrate on describing the novelty of our approach, which is the process of inspecting, collecting and distributing change information of object-oriented artifacts in a multi-developer environment.

Overview. Figure 1 shows the architecture of Syde. The client is a collection of plug-ins, where the Inspector is the one responsible of capturing the changes. The Viewer is composed of plug-ins responsible for showing interesting change information to developers. The Requestor is another plug-in that communicates with the server to acquire entities changed in other workspaces. On the server side, the Collector is the one responsible for receiving the changes and detecting differences between implementations across workspaces. After these differences are detected, the Notifier distributes them to targeted clients. The Notifier only shares change information, but if a developer wants to request the change itself, the Distributor is the module responsible for retrieving it.

Figure 1: Syde Architecture.

Changes at the AST level. To leverage changes at the AST level, Syde models the software system to be monitored as an AST, where nodes are packages, classes, methods and fields. Each node is composed of properties, e.g., the arguments of a method. Changes are modeled as operations on the AST [16]. Current operations modeled by Syde are: addition of a node; deletion of a node, or a branch of the tree that if the node is not a leaf; movement of a node or branch; and property change, addition and deletion.

Syde extends Spyware’s change-centric model [15] from a single-developer approach to a multi-developer approach. To accomplish that, Syde’s server stores one AST for each developer, which reflects exactly the state of the system at a developer’s workspace. Figure 2 illustrates the change flow from a developer’s workspace to the server and back. The Inspector is silently triggered whenever a file under edition is compiled. The saved file is decomposed into an AST branch, and compared with the current AST model in the plug-in. The detected changes are transformed into tree operations and sent to the server. This process is completely transparent to the developer.

On the server, the Collector updates the developer’s AST with the new change and the Notifier is triggered to create a notification of the change and send it to potentially interested (targeted)
developers. The second step is to check for emerging conflicts. The Collector compares the modified AST with other developers’ ASTs to check whether a new conflict has emerged, and if positive, a conflict alert is created.

At the moment, Syde detects structural conflicts [14]: those related to the node operations previously listed. The conflicts are classified into two categories: yellow, when there are structural differences between two versions of a node, but none of these versions were checked in the SCM system; red, when there are structural differences between two versions of a node, and one of them was checked in the SCM system. Syde keeps track of the version of each node, by inspecting the corresponding file’s version on the adopted SCM system (e.g., Subversion). The state of a conflict is stored, and at every change on one of the entities involved in the conflict, it is updated. The involved developers are kept up-to-date with the conflict’s state until it is solved, and the visual alert is removed from their workspaces.

Delivering awareness information. Once change and conflict information are captured and persisted on the server, they can be accessed both on the fly and for history analysis. The Syde Viewer can be extended to show this information on Eclipse’s workbench. At the moment, Syde features the Scamp plug-in [8], and the Conflict plug-in. Figure 3 shows a screenshot of these views in action: visual cues that are added in the package explorer to indicate that a class has been changed by someone else (A); a word cloud view showing which are the most recently changed classes and who was the last one to edit each one (B); and the conflict view showing emerging conflicts (C).

4. METHODOLOGY

We are carrying out our research by iteratively prototyping, evaluating, and reasoning about the evaluation results to address the drawbacks on the next iteration.

In the first iteration, we built the infrastructure support for capturing change information at file level every time a developer saves a file [9]. This information was then used to compute code ownership – who is the most knowledgeable about which files –, and compared it with previous approaches that use exclusively history logs from SCM systems. The change history provided by Syde reflects more accurately the amount of effort a developer spent on a file. Thus, we showed that Syde is capable of refining ownership classification, especially when there is a high variability on the frequency with which developers check in their changes [10].

In the second iteration, we enriched Eclipse’s workbench with visualizations to promote awareness. To evaluate the tradeoff between the usefulness of the visual cues and the level of distraction they caused, we conducted a study with a group of students at our University. We selected four students to use Syde during the development of a 5-weeks project in the context of their Programming Fundamentals 2 course. At the end of their projects, we conducted a survey to evaluate their experience with the visualizations. Overall, the students agreed that Syde helped them to maintain a high level of awareness, but suggested some improvements, such as allowing them to choose which artifacts they want to track [13].

Currently, we are working on the third iteration, which is to enrich Syde’s infrastructure to support notification of changes at object-oriented level, and conflict detection. We plan to evaluate this prototype by conducting a case study with a team of practitioners. The plan is to let them use Syde for a period of time, while constantly collecting feedback from them through questionnaires. Their feedback should indicate whether awareness promoted by Syde helps developers to accomplish the tasks faster and to avoid merging conflicts when checking in their changes on the SCM.

In the last iteration, we will add assistance features, such as expert recommendation (based on our previous study on code ownership), support for conflict resolution, and filtering options (to allow a developer to choose which notifications and for which artifacts of the system he is interested in). The final evaluation will be conducted with practitioners, where they will use Syde in the context of their work for a predefined period of time. The goal of this evaluation is to measure the usefulness of Syde, which will be done through a survey divided into weekly questionnaires.

5. CONCLUSIONS

We presented our ongoing work to promote team awareness and stimulate collaboration in the context of global software development. The novelty of our approach is that Syde models changes as first-class entities and thus, is capable of delivering awareness information at object-oriented level.

The contributions we have made so far are:

Porting a single-developer change-centric approach [15] to multi-developer projects. We model object-oriented systems in Java as ASTs and changes in these systems as tree operations. Syde server combines this model with information extracted from the adopted SCM of a project to precisely inform targeted developers which entities have been changed.

Use of historical data collected by Syde to refine evolution analyses. More specifically, we used historical change information to refine the notion of ownership by measuring the actual effort that each developer has made.

Integration of lightweight visualizations into an IDE to deliver awareness information. Syde plug-ins feature a number of differ-
ent visualizations that give developers the notion of who is editing which files in real-time, and also, who has changed which files in the recent past. This feature should be improved to be as non-disturbing and yet informative as possible.

Some of the future contributions we foresee are:

*Support for structural conflict detection and resolution of ongoing changes.* There are two direct benefits of this feature. The first is that a developer knows at early stage when his changes are conflicting with someone else’s changes and can take actions before the conflicts become too complex. The second is that a developer can actually request from Syde the part of the code that is under conflict, which will offer structural merge support [4], rather than the classic textual merge offered by mainstream SCM systems.

*Integration of developer assistance on the IDE.* The change history of Syde is rich in details, and can be used to refine the results of evolution analyses classically performed with SCM history logs. We plan to integrate these analyses with the IDE to provide up-to-date and detailed information related to the system’s evolution, such as expert recommendation, change recommendation, and support for change playback. The expected contribution of this feature is to port classic post-mortem analyses, which were mainly used by maintainers, to a continuously and integrated process that can deliver useful information to developers while they work.

### 6. ACKNOWLEDGMENTS

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project “REBASE” (SNF Project No. 115990).

### 7. REFERENCES


