Impact: An Indirect Dependency Awareness Tool For Software Development

Jordan Ell
University of Victoria
Victoria, British Columbia
jell@uvic.ca

Daniela Damian
University of Victoria
Victoria, British Columbia
danielad@cs.uvic.ca

***Abstract*—Awareness has been largely studied in the field of awareness research within software engineering. Many tools and techniques been been proposed and built in order to provide software developers and stakeholders with a greater sense of workspace and task awareness within their software projects. These techniques and tools have been largely focused on detecting *direct* conflicts which arise over a project's life time or have created an exploratory ground for stakeholders to use as a means of resolving self discovered direct or *indirect* conflicts. However, detecting and providing pertinent information regarding indirect conflicts has been largely ignored partially due to its inherently larger complexity than direct conflicts. Indirect conflicts arise when changes in one software artifact affect another. In this paper, we present Impact, a new task awareness tool directly aimed at both detecting and presenting indirect conflicts which arise inside of a software project. We introduce previous indirect conflict awareness attempts, our design and implementation, as well as describe Impact's potential through two evaluation studies.**

I. Introduction

Awareness is characterized as "an understanding of the activities of others which provides a context for one's own activities" [1]. The study of awareness and its tools has become an important topic of research in software engineering especially with the new importance of distributed work and collaboration. Awareness is generally associated with both technical and social dependencies that are created and evolve over a software project's life time. The study of these dependencies has become the primary focus of most awareness related research. Task awareness has become the most prevalent field of awareness research to understand how developers cope with these technical and social dependencies.

Tools have been created to attempt to solve task awareness related issues with moderate success [2], [3], [4], [5]. However, these tools have been designed to solve task awareness related issues at the direct conflict level. Examples of direct conflict awareness include knowing when two or more developers are editing same artifact, finding expert knowledge of a particular file, and knowing which developers are working in which files. Meanwhile, task awareness related issues at the indirect conflict level continue to be an issue which is largely unsolved by most coordination mechanisms [5].

Previous interviews and surveys conducted with software developers have show a pattern that developers of a software project view indirect conflict awareness as a high priority issue in their development [6], [7], [8], [9]. Examples of indirect conflict awareness include having one's own code effected by another developer's source code change or finding out who might be indirectly effected by one's own code change. Indirect conflicts arising in source code are inherently difficult to resolve as most of the time, source code analysis must be preformed in order to find relationships which are harmed by changes. While some awareness tools have been created with these indirect conflicts primarily in mind [8], [10], most have only created an exploratory environment which is used by developers to solve conflicts which may arise. These tools lack the ability to detect indirect conflicts that arise and alert developers to their presence inside of the software system. Some tools have started to work directly with solving indirect conflicts [11] but continue to be limited by their defined technical relationships.

Despite software developer's need for indirect conflict awareness tools detecting and alerting developers to arising indirect conflicts is still a major problem. Impact, a web based awareness tool, aims to solve this issue. In this paper, we describe Impact's generic design and implementation in order to both detect newly created indirect conflicts among software developers as well as alerting developers to these conflicts. By leveraging technical relationships inherent of software projects with method call graphs [12] as well as detecting changes to these technical relationship through software configuration management (SCM) systems, Impact is able to detect indirect conflicts as well as alert developers involved in such conflicts in task awareness. This process fulfills the technical and social dependency congruence found by Grinter et al. [13] which suggests to offer a smoother work flow.

The rest of this paper is organized as follows. First, we begin by discussing similar indirect conflict awareness tools which have partially solved the issues presented by this paper and how their workings can be applied to Impact. In the following section we describe a generic design and implementation of Impact as an awareness tool. We then discuss a preliminary evaluation of Impact followed by conclusions and future work.

II. Related Work

Although there is an abundance of Awareness tools developed in research today, only a handful have made an attempt to examine indirect conflicts. Here, we will outline three of the forefront projects in indirect conflicts and how these projects have influenced the decision making process in the design and implementation of Impact.

We first start with both Codebook [8] and Ariadne [10]. These projects produce an exploratory environment for developers to handle indirect conflicts. Exploratory pertains to the ability to solve self determined conflicts, meaning that once a developer discovers a conflict, they can use the tool as a type of lookup table to solve their issue. Codebook is a type of social developer network that relates developers to source code, issue repositories and other social media while Ariadne only looks at source code for developer to source code association. In this, developers become owners of source code artifacts. Both projects also use program dependency graphs [14] in order to relate technical artifacts to each other. These projects make use of method call graphs in order to determine which methods invoke others which forms the basis for linking source code artifacts creating a directed graph. While these projects can be great tools for solving indirect conflicts which may arise, by querying such directed graphs to view impacts of conflict creating code, they lack the ability to detect conflicts on their own.

A serious attempt at both detecting and informing developers of indirect conflicts is the tool Palantir [11]. Palantir monitors developers activities in files with regards to class signatures. Once a developer changes the signature of a class by modifying changes in name, parameters, return values of public methods etc., any workspace of other developers which are using that class will be notified. Palantir utilizes a push-based event model [15] which seems to be a favored collection system among Awareness tools. Sarma et al. [11] also develops a generic design for future indirect conflict awareness tools. However, Palantir falls short in its collection and distribution mechanisms. First, Palantir only considers ”outside” appearance of technical objects, being their return types, parameters, etc.. Secondly, Palantir only delivers detected conflicts to developers who are presently viewing or editing the indirect object while other developers who have used the modified class previously are not notified.

Impact is designed to handle the holes in the aforementioned projects. Impact will focus on the detection of indirect conflicts at an internal level of technical objects as opposed from object signatures and the distribution of these conflicts to all appropriate developers regardless of their current workspace activities. Impact is also designed around the successes these projects have had in the past with directed graphs as well as elements of collection, ownership, and distribution functionality.

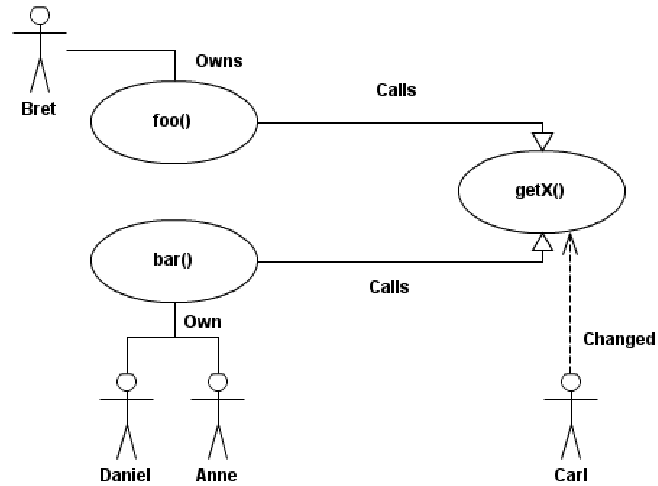

Fig. 1. Technical object directed graph with ownership

III. Impact

This section will proceed to give an outlined detail of Impact in both its design and implementation. The design of Impact was created to be a generic construct which can be applied to other indirect conflict awareness tools while the implementation is specific to the technical goals of Impact.

A. Design

Compared to tool design for direct conflicts, the major concern of indirect conflict tools is to relate technical objects to one another with a “uses” relationship. To say that object 1 uses object 2 is to infer a technical relationship between the two objects which can be used in part to detect indirect conflict that arise from modifying object 2. This kind of relationship is modeled after a directed graph [14]. Each technical object is represented by node while each “uses” relationship is represented by a directed edge. This representation is used to indicate all indirect relationships within a software project.

While technical object relationships form the basis of indirect conflicts, social communication is the ultimate goal of resolving such conflicts. This being the case, developer ownership must be placed on the identified technical objects. With this ownership, we now infer relationships among developers based on their technical objects “uses” relationship. Developer A, who owns object 1, which uses object 2 owned by developer B, may be notified by a change to object 2’s internal workings. Most, if not all, ownership information of technical objects can be extracted from a project’s source code repository (CVS, Git, SVN, etc.).

Finally, the indirect conflict tool must be able to detect changes to to the technical objects defined above and distribute information to the appropriate owners. Two approaches have been proposed for change gathering techniques: real time and commit time [15]. We propose the use of commit time information gathering as it avoids the issue of developers overwriting previous work or deleting modifications which would produce information for changes that no longer exist. However, the trade off is that indirect conflicts must be committed before detected, which results in conflicts being apart of the system before being able to be dealt with as opposed to catching conflicts before they happen. At commit time, the tool must parse changed source code in relation to technical artifacts in the created directed graph detailed above. Once technical objects are found to be changed, appropriate owners of objects which use the changed object should be notified. In Figure 1, X changes method (technical object) 1, which effects methods 2 and 3 resulting in the alerting of developers Y and Z.

With this three step design of: (i) creating directed graphs of technical objects, (ii) assigning ownership to those technical objects, and (iii) detecting changes within commit time and the distribution of information to appropriate owners, we believe a wide variety of indirect conflict awareness tools can be created or extended. The implementation of Impact in the following section will follow these three design guidelines.

B. Implementation

For Impact's implementation, we decided to focus on methods as our selected technical objects to infer a directed graph from. The "uses" relationship described above for methods is method invocation. Thus, in our constructed directed graph, methods represent nodes and method invocations represent our directed edges. In order to construct this directed graph, abstract syntax trees (ASTs) are constructed from source files in the project. The ASTs allow us to construct method call graphs from which the directed graphs can be constructed.

Once the directed graph is constructed, we must now assign ownership to our technical objects (methods) as per our design. To do this, we simply query the source code repository. In our case we used Git as the source code repository, so the command *git blame* is used for querying ownership information. (Most source code repositories have similar commands and functionality.) This command gives authors of source code per line which can is used to assign ownership to methods. If a method has 10 lines and developer A has written 3 while developer B has written 7, then ownership is assigned 30% and 70% respectively.

To detect changes to our technical objects (methods), we simply use a commit's *diff* which is a representation of all changes made inside a commit. We can use the lines changed in the *diff* to find methods that have been changed. This gives cause of potential indirect conflicts. We now find all methods

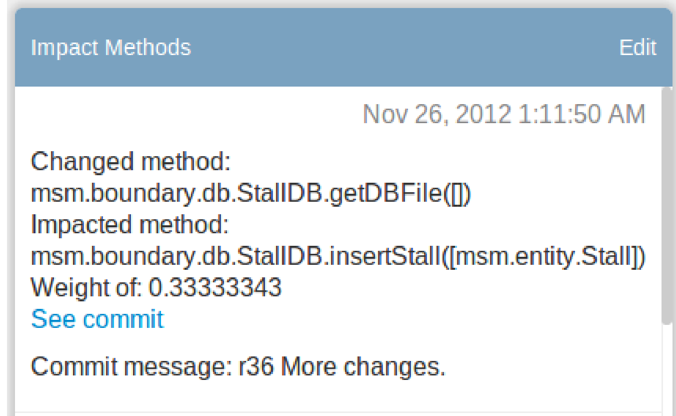

Fig. 2. Impact's RSS type information feed.

in our directed graphs which invoke these changed methods. These are the final indirect conflicts.

Now that the indirect conflicts have been found, we use the ownership information of our technical objects to send alerts to those developers involved in the indirect conflict. All owners of methods which invoke those that have been changed are alerted to the newly changed method. This can been seen in Figure 2, the user interface of Impact. Here, in an RSS type feed, the changing developer, time of change, changed method, invoking methods, and commit message are all displayed. This list is auto updating as new commits are pushed to the central git repository. The information provided also comes with a simple weight of the percent changed of changed method multiplied by ownership of the invoking method. This allows developers to filter through high and low changes affecting their own source code. Developers are now able to be alerted of detected indirect conflicts and solve them by communicating the the developer who authored the change.

IV. Evaluation

To fully evaluate both the generic design of detecting and resolving indirect conflicts as well as Impact, extensive testing and evaluation must be performed. However, we feel that a simple evaluation is first needed to assess the foundation of our design and claims: does Impact improve developer awareness with indirect conflicts and help the resolving of such issues?

We performed two user case studies to address our question where we gave Impact to two small development teams. Each team was free to use Impact at their leisure during their development process, after which interviews were conducted with lead developers from each development team. The interviews were conducted after each team had used Impact for three weeks.

We asked lead developers to address two main concerns: do indirect conflicts pose a threat at the method level, and did Impact help raise awareness and promote quicker conflict resolution for indirect conflicts. Our two interviews largely supported our initial claims of indirect conflicts posing a serious threat to developers, especially in medium to large teams or projects as opposed to the small teams which they were apart of. Our interviews also pointed out that method use can be a particularly large area for indirect conflicts to arise. However, both interviewees also pointed out that any technical object which is used as an interface to some data construct or methodology, database access for instance, can be a large potential issue for indirect conflicts. Interview response to Impact was also largely positive, as both interviewees said that Impact helped raise awareness among their teams with what other developers are doing as well as the influence it has on their own work. However, both interviews showed Impact to have information overload. Both interviews showed that while all method changes were being detected, not all are alert worthy. One lead developer suggested to only alert developers to indirect conflicts if the internal structure of a method changes due to modification to input parameters or output parameters. In other words, the boundaries of the technical objects (changing how a parameter is used inside the method, modifying the return result inside the method) seem to be more of interest than other internal workings.

These two studies have show that our design and approach to detecting and alerting developers to indirect conflicts appear to be on the correct path. Impact as a tool has laid the foundations for future work in detecting indirect conflicts as well as alerting developers, although more thought must be given as to what constitutes a meaningful change inside our selected technical objects.

V. Conclusion and Future Work

In this paper, we have presented the issues that arise from indirect conflicts in present awareness tools. We have proposed a generic design for the future development of awareness tools in regards to handling indirect conflicts. We have presented a prototype awareness tool, Impact, which was designed around our generic awareness approach. Impact was evaluated on a small scale, showing its future potential as well as highlighting its current weaknesses.

In future work we are planning to conduct interviews and surveys with software developers to confirm that indirect conflicts pose a threat to their projects as well as discovering what constitutes a valid change inside of a given technical object. With these two improvements, we plan to justify our generic design of indirect conflict tools further as well as improve the ideas of detecting valid and significant changes to technical objects.

References

[1] P. Dourish and V. Bellotti, "Awareness and coordination in shared workspaces," in *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, ser. CSCW '92. New York, NY, USA: ACM, 1992, pp. 107–114. [Online]. Available: http://doi.acm.org/10.1145/143457.143468

[2] P. F. Xiang, A. T. T. Ying, P. Cheng, Y. B. Dang, K. Ehrlich, M. E. Helander, P. M. Matchen, A. Empere, P. L. Tarr, C. Williams, and S. X. Yang, "Ensemble: a recommendation tool for promoting communication in software teams," in *Proceedings of the 2008 international workshop on Recommendation systems for software engineering*, ser. RSSE '08. New York, NY, USA: ACM, 2008, pp. 2:1–2:1. [Online]. Available: http://doi.acm.org/10.1145/1454247.1454259

[3] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson, "Fastdash: a visual dashboard for fostering awareness in software teams," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '07. New York, NY, USA: ACM, 2007, pp. 1313–1322. [Online]. Available: http://doi.acm.org/10.1145/1240624.1240823

[4] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb, "Tesseract: Interactive visual exploration of socio-technical relationships in software development," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 23–33. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2009.5070505

[5] H. Khurana, J. Basney, M. Bakht, M. Freemon, V. Welch, and R. Butler, "Palantir: a framework for collaborative incident response and investigation," in *Proceedings of the 8th Symposium on Identity and Trust on the Internet*, ser. IDtrust '09. New York, NY, USA: ACM, 2009, pp. 38–51. [Online]. Available: http://doi.acm.org/10.1145/1527017.1527023

[6] D. Damian, L. Izquierdo, J. Singer, and I. Kwan, "Awareness in the wild: Why communication breakdowns occur," in *Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on*, aug. 2007, pp. 81 –90.

[7] C. A. Halverson, J. B. Ellis, C. Danis, and W. A. Kellogg, "Designing task visualizations to support the coordination of work in software development," in *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, ser. CSCW '06. New York, NY, USA: ACM, 2006, pp. 39–48. [Online]. Available: http://doi.acm.org/10.1145/1180875.1180883

[8] A. Begel, Y. P. Khoo, and T. Zimmermann, "Codebook: discovering and exploiting relationships in software repositories," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 125–134. [Online]. Available: http://doi.acm.org/10.1145/1806799.1806821

[9] A. Schröter, J. Aranda, D. Damian, and I. Kwan, "To talk or not to talk: factors that influence communication around changesets," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, ser. CSCW '12. New York, NY, USA: ACM, 2012, pp. 1317–1326. [Online]. Available: http://doi.acm.org/10.1145/2145204.2145401

[10] E. Trainer, S. Quirk, C. de Souza, and D. Redmiles, "Bridging the gap between technical and social dependencies with ariadne," in *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, ser. eclipse '05. New York, NY, USA: ACM, 2005, pp. 26–30. [Online]. Available: http://doi.acm.org/10.1145/1117696.1117702

[11] A. Sarma, G. Bortis, and A. van der Hoek, "Towards supporting awareness of indirect conflicts across software configuration management workspaces," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ser. ASE '07. New York, NY, USA: ACM, 2007, pp. 94–103. [Online]. Available: http://doi.acm.org/10.1145/1321631.1321647

[12] A. Lakhotia, "Constructing call multigraphs using dependence graphs," in *Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ser. POPL '93. New York, NY, USA: ACM, 1993, pp. 273–284. [Online]. Available: http://doi.acm.org/10.1145/158511.158647

[13] R. E. Grinter, "Recomposition: Coordinating a web of softwaredependencies," *Comput. Supported Coop. Work*, vol. 12, no. 3, pp. 297–327, Jul. 2003. [Online]. Available: http://dx.doi.org/10.1023/A:1025012916465

[14] S. Horwitz and T. Reps, "The use of program dependence graphs in software engineering," in *Proceedings of the 14th international conference on Software engineering*, ser. ICSE '92. New York, NY, USA: ACM, 1992, pp. 392–411. [Online]. Available: http://doi.acm.org/10.1145/143062.143156

[15] G. Fitzpatrick, S. Kaplan, T. Mansfield, A. David, and B. Segall, "Supporting public availability and accessibility with elvin:

Experiences and reflections," *Comput. Supported Coop. Work*, vol. 11, no. 3, pp. 447–474, Nov. 2002. [Online]. Available: http://dx.doi.org/10.1023/A:1021226206564