Tool Support for Collaborative Software Development based on Dependency Analysis

Jean M. R. Costa, Rafael M. Feitosa, Cleidson R. B. de Sousa

Abstract— **This paper presents RaisAware, a collaborative software development tool aimed at supporting the relationship between software architecture and coordination of software development activities. Our design is based on both dependency analysis of software development artifacts and software developers' activities. We describe the motivations behind this work, detail the design and implementation of RaisAware, and present an evaluation of the tool using open-source project data.**

Index Terms **— awareness, collaboration, collaborative software development, dependency analysis**

I. Introduction

Software development has been part of collaborative systems research since CSCW initial years [1]. Studies of collaborative software engineering, or CSE for short, have resulted both in theoretical contributions to the collaborative systems field [2] as well as in groupware tools to support this activity [3]. In the first case, Grinter [4], for instance, studied how software developers use configuration management tools to coordinate their work. Meanwhile, Halverson and colleagues [5] designed visualizations to support the management of change requests in large software development projects. In this paper, we are concerned with both types of approaches. More precisely, we use results of published field studies of software development teams to motivate the design of RaisAware, a collaborative software development tool. RaisAware explores the relationship between software architecture and work coordination, a long acknowledged [6-11], but unexplored [12] relationship. RaisAware's design explores this relationship by supporting dependency analysis of both software development *artifacts* and software developers' *activities*. Analysis of software development artifacts is based on software dependency analysis techniques [13] aimed at identifying dependencies among software components, while analysis of software development activities is based on co-change analysis [14] to identify historical patterns of developers activities regarding changes in the code. By representing different types of dependencies, RaisAware goes beyond previous collaborative software development tools like Palantír [15], ROSE [14], Ariadne [16], and CollabVS[39].

Jean M. R. Costa is with the Federal University of Pará, Belém, PA, Brazil, 66075.

Rafael M. Feitosa was with University of Pará, Belém, PA, Brazil, 66075. He is now with the Federal University of Rio Grande do Sul, Porto Alegre, RS.

Cleidson R. B. de Souza was with University of Pará, Belém, PA, Brazil, 66075. He is now with IBM Brazil, São Paulo, SP, Brazil, 04007, (e-mail: cleidson.desouza@acm.org).

The rest of this paper is organized as follows. Section 2 describes previous tools that are similar to RaisAware. Section 3 describes briefly some of the theoretical and empirical studies that motivated RaisAware. In particular, we use the impact management framework proposed by de Souza and Redmiles [12]. After that, we describe the software dependencies approaches used in RaisAware in section 3. In Section 4 we discuss some types of software dependencies and how they relate to coordination of activities. Section 5 presents our tool and its implementation details, while Section 6 presents the tool evaluation using open source data. Finally, section 7 presents our final comments and future work.

II. Collaborative Software Development Tools

As mentioned before, RaisAware is not the first tool to assist collaborative software development activities; there are other tools with this purpose. Researchers and practitioners have created tools to support this endeavor. One of the more traditional approaches is based on configuration management systems such as CVS and Subversion. These systems allow multiple developers to access a common set of artifacts and manage different versions of these artifacts. Furthermore, they also allow several developers to work in parallel. Parallel development means that two or more developers are modifying the same file at the same time [19]. This situation is difficult to handle because it requires developers to coordinate their work more carefully. In fact, parallel changes are generally correlated with faulty code [19].

Palantir [15] and Ariadne [16] are research tools to facilitate the coordination of developers' activities by facilitating the awareness [32] of developers' activities. Awareness is achieved by viewing the information provided in the IDE (Integrated Development Environment) of each developer. Each of these tools has only one type of dependency, while RaisAware extracts information from different types of dependencies (see details in section 4).

The Jazz environment [36] is a set of Eclipse plug-ins aimed at facilitating collaborative software development. Jazz does not make use of software dependency information, but presents information about artifacts' status in the developers' IDEs, such as whether the artifact was saved locally and the commit was not performed yet.

FASTDash [35] is a tool to facilitate the awareness of the team development activities based on a spatial representation of the project and on highlighting the relevant activities of members. This representation indicates, for example, which files are being viewed or which classes and methods are being modified at the time.

Finally, CollabSV [39] is a collaborative software development toool that uses a semi-synchronous model so that

asynchronous and synchronous activities can be easily chosen according to the task at hand. Furthermore, this tool uses different types of static analysis not supported by RaisAware.

The tools listed above are intended to assist the collaborative software development using sophisticated resources. However, another category of studies suggests that even simple tools can be effective in helping the coordination of software development activities. For instance, Fitzpatrick and colleagues [30] illustrate how the usage of a simple reporting system (a tickertape) can be appropriate, provided it has an adequate infrastructure for information exchange and developers have sufficient experience using the tool.

III. Previous Studies of CSE

RaisAware is inspired by results of field studies conducted with software development teams. In particular, our own previous work as reported by de Souza and Redmiles [12]. The result of this work resulted in an analytical framework for understanding the work of software developers in dealing with the effects of software dependencies on their activities. The framework is called impact management and is defined as: "… the work performed by software developers to minimize the impact of one's effort on others and, at the same time, the impact of others into one's own effort" [12]. There are three main aspects to impact management:

1. First, finding the network of people that might affect one's work and that might be affected by one's work, a concept that we call *impact network*. Identifying the impact network is the most important aspect of impact management, and, in fact, RaisAware's main functionality is to support this aspect;
2. Second, *forward impact management* is the work to assess the impact of one's own work on his respective impact network and inform the members of this network of such impact; and
3. Finally, *backward impact management* consists of assessing the impact of the work performed by developers in one's impact network on his work, and the appropriate actions to avoid such impact.

de Souza and Redmiles [12] illustrate the impact management framework using ethnographic data. However, this same framework can be used to explain different findings of previous studies of software development. We will detail some of these aspects below, since they are used to inform the design of our tool.

A common problem identified in the collaborative software engineering literature is that developers are not aware of the set of developers that can affect or be affected by a developer's changes, that is, developers are unaware of their impact network. They adopt several approaches to deal with this situation and to try to identify their network: some of approaches are technical (for instance, using a database [12] or consulting log files [2]) while other approaches are social (using their's and their manager's social network or assigning developers with the primary responsibility of communicating with other developers who provide software components to the team). Identifying the impact network becomes even more difficult because these networks shrink and expand as the development process progresses [12].

One of the important findings from de Souza's and Redmiles' study is the observation that software developers engage in impact management because they are aware of the details of the software architecture. Experienced developers know that some software components are sources of dependency to several other components and use this knowledge to guarantee the smooth flow of work and avoid impacting their colleagues. In other words, software components with a higher degree of dependencies have to be handled more carefully because of their larger "potential" for impacting other components in the architecture, and as a consequence, their effect on the coordination of developers' activities. Furthermore, experienced software developers change their activities accordingly when dealing with these components.

Another important aspect of software development work in general is the avoidance to parallel work [15, 17], that is, developers adopt strategies (e.g., "speeding up" their work [18]) to avoid engaging in parallel development with their colleagues. As mentioned in the previous section, parallel development means that two or more developers are changing the same file. This situation is difficult to handle because they require extra-coordination among developers and, often, require some work to be re-done.

IV. Software Dependencies and the Coordination of Activities

The relationship between the structure of a software organization and the software structure has been discussed since Conway has originally proposed it [6]. In general, researchers have associated the structure of the software with the technical dependencies between the components i.e., it describes software dependencies between components. A body of empirical evidence indicates that these dependencies, as suggested initially by Conway [6] and Parnas [7], play a major role in the coordination of the work. Accordingly, one approach that researchers have adopted is to use dependency analysis techniques to understand the coordination of development work [11, 16, 20]. This section describes the two most used approaches to identify component dependencies, both of them implemented in RaisAware.

A. Dependencies among software artifacts

In software engineering, different data structures can be used to allow the explicit representation and manipulation of a program's dependencies. RaisAware uses system dependency graphs (SDGs) to represent information about several procedure calls and their parameters and return types [21]. SDGs are used to construct *call graphs* that "summarize the dynamic invocation relationships between procedures. The nodes of the call graph are the procedures in the program. An edge (*pl*, *p2*) exists if procedure pl can call procedure p2 from some *call site* within *pl*. Hence, each edge may be thought of

as representing some call site in the program" [22]. A call graph, then, reveals the potential dynamic structure of a software system, i.e., it potentially unveils dependencies among software developers responsible for the software components [23]. For instance, assume that a software component *a* depends on another software component *b*, and that *a* is being developed by *developer A* and *b* is being implemented by *developer B*. If *a* depends on *b*, we similarly find that *developer A* depends on *developer B*. That is, these software developers need to coordinate and communicate to guarantee the smooth flow of work [24-27].

B. *Dependencies among developers' activities*

The view based solely on the explicit relationships among software development artifacts is too narrow, however. For instance, software connectors are used by software architects to model interactions between components [28], but these interactions are *not* captured by analyzing dependencies between software artifacts [29]. In general, dependencies between developers' activities can be extracted by mining software repositories and identifying evolutionary coupling between components, that is, coupling between components that are not necessarily identified through dependency analysis. This is done by creating association rules between files changed together, that is, files that are co-changed are likely to be coupled, even though traditional dependency analysis of these files does not indicate such coupling. Because co-changes are built out of historical activities (changes to be more precise) from software developers, we call them dependencies among developers' activities in contrast to dependencies among software artifacts. As we will describe in the next section, RaisAware supports both types of dependencies.

Zimmerman and colleagues [14] introduced two measures to identify the degree of evolutionary dependencies among artifacts:

- **Support**: it indicates how many times a artifact has been modified with another;
- **Confidence**: it indicates the changes ratio, i.e., how often two files were changed together.

In RaisAware, an artifact is said to depend on another if the support value is at least 7 and the confidence (or trust) value is at least 35%. These values are similar to those proposed by Zimmerman [14].

C. *Dependencies in RaisAware*

RaisAware uses both types of dependencies, among software artifacts and among software developers' activities, going one step further than other tools as Palantir [15] and Ariadne [16]. Furthermore, we created an infra-structure in which it is possible to add new types of software dependencies as we see fit. For instance, Dewan and Hedge (2007) use interface implementation as a way of identifying dependency between software components. While this is not currently implemented in RaisAware, we can easily add this new feature. Finally, RaisAware allows one to identify his impact network, a feature that is not available in any of these previous tools. RaisAware is described in more details in the next section.

V. RAISAWARE

This section describes the RaisAware tool. Initially, we present its main features according the theoretical framework of impact management. Then, we present the tool architecture and the description of its operation.

A. *Identification of the Most Dependent Artifacts*

As discussed earlier, experienced developers have knowledge about the software architecture of the system being built (e.g., files that have a high degree of dependency) and use this knowledge to adjust their activities [18]. On the other hand, inexperienced developers have no such knowledge, so their activities more prone to errors. To facilitate the work of these developers, RaisAware provides a mechanism for identifying the most dependent files. In order to identify these files, an analysis is made about the data contained in the matrix that contains information about the dependencies in order to find the 15% of files with more dependencies.

The most dependent artifacts are visualized using *Decorators*, i.e., graphical representations applied over the corresponding files icons, as can be seen in Figure 1. In the figure, icons with *Decorators* are circumvented by circles. The idea is that when a developer notices a *Decorator* over the file icon that he is planning to modify, he will notice that his change can affect the work of several other developers, so he will be aware that the file he is planning to modify needs to be handled more carefully [18].

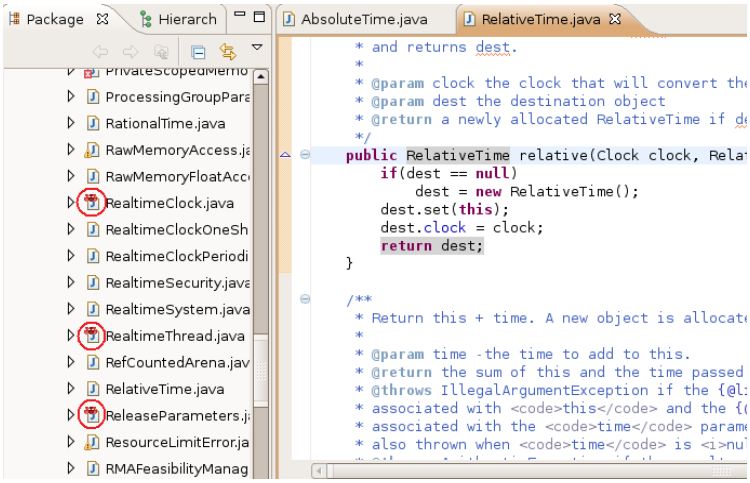

Fig. 1. Decorators used to identify the most dependent files in a view of Eclipse.

B. *Identification of the software components most impacted*

This feature allows the user to visualize the files that will be impacted according either to his source-code modifications or to source-code modifications from other developers. There are two ways to view these files using RaisAware: (i) clicking the right mouse button on the file icon (viewing files impacted or impacting this file) or (ii) through the project icon (looking at

all files and their corresponding impacts), by choosing "View Technical Network".

To generate this view, the files that are being changed by the software developer are initially identified. These files are compared with files changed by other developers, since information about the activities of other developers are also stored locally when sent by other Workspaces (see the description of the Architecture in the next section). When RaisAware identifies that two developers are changing the same file, i.e., that they are engaged in parallel development, it creates two nodes representing the same file and an edge connecting these files to indicate a potential conflict between the developers. After that, RaisAware searches in the matrices that contain the dependency information about which files are impacted. One of these matrices contains dependency information created from analysis of software artifacts, while the other contains dependency information about developers' activities created from co-changes. Once these files are identified, it is verified whether they are being changed remotely. If so, it is created an edge to represent that a file modified by a developer can affect or be affected by another file modified by another developer. RaisAware uses a visualization framework called Prefuse to display this type of information.

Figure 2 shows an example of a visualization from RaisAware. Parallel development (direct conflict) is represented by thicker edges, while dependencies among artifacts (indirect conflict) are indicated by directed edges, and finally, dependencies among developers' activities (co-changes) are represented as non-directed edges in the network. In Figure 2, the file *Clock* is involved in a direct conflict, i.e. while a developer is editing the file *Clock* another developer is simultaneously editing the same file. So, due to the parallel development of this file, two files *Clock* are shown in the figure. *AbsoluteTime* and *Clock* present an example of indirect conflicts and, finally, the edge between *RealTimeClock* and *Clock* illustrate an example of co-change dependency.

Figures 3 and 4 show other visualizations from RaisAware. In Figure 3 there is a menu on the right corner with a checklist, which can be used to filter the information being displayed, presenting only the nodes and edges corresponding to the selected types of dependencies. There is also a tooltip with detailed information about the file involved in the direct conflict (*Clock*). Thus, it is possible to identify, for example, that another user is editing this file in parallel. In Figure 4 there is a tooltip indicating the reasons for the existence of an indirect conflict between the file *Clock* and the file *HighResolutionTime*. Providing the rationale for the dependency information is essential for the developer to understand the reason why he might need to coordinate his work with another developer.

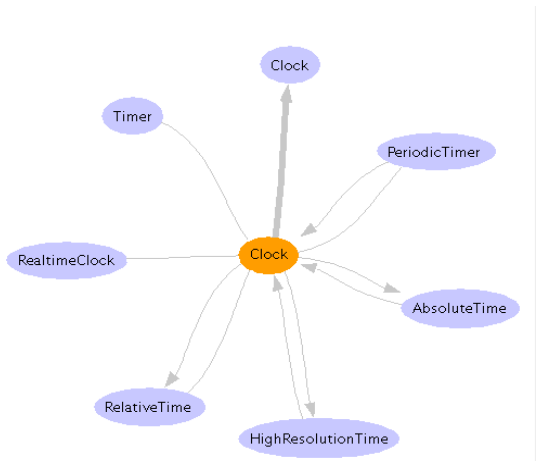

Fig. 2.

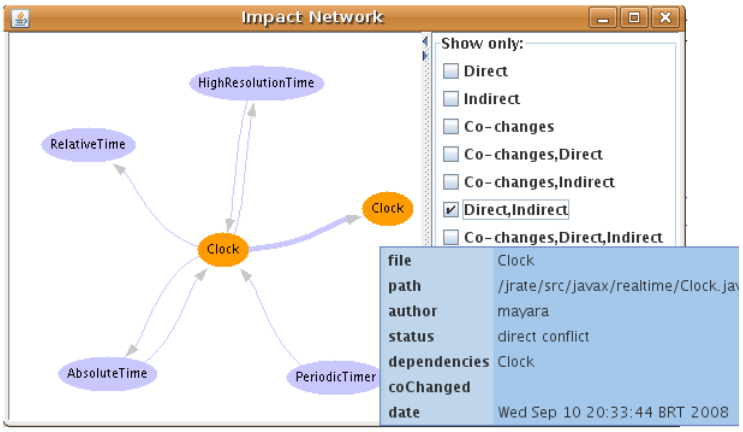

Fig. 3. Filter "Direct,Indirect" dependencies is activated. In addition, details about the file Clock are shown.

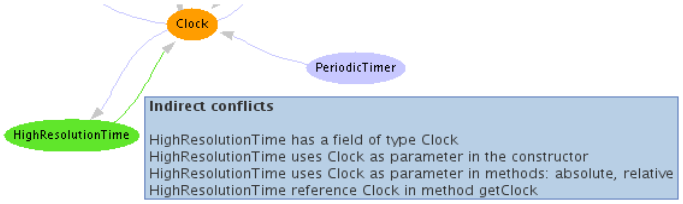

Fig. 4. Explanation about the existence of an indirect conflict

C. *Identification of the impact network*

Based on the information about files impacting other developers and being impacted by others, RaisAware "translates" this information into the impact network. To be more precise, RaisAware replaces files by software developers' names to generate the impact network, that is, the list of software developers impacting and being impacted by the changes performed by the software developer.

Figure 5 below presents an example of an impact network. Similarly to the visualization of files, parallel development is represented as thicker edges, artifact dependencies are indicated

as directed edges, and developer activities' dependencies are represented as undirected edges in the impact network.

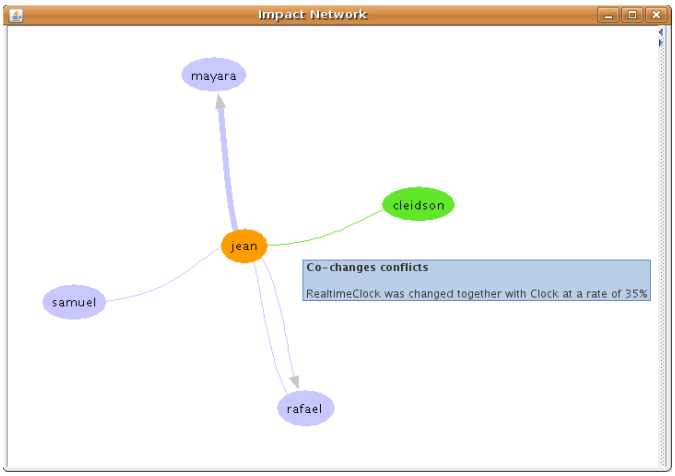

Fig. 5. Example of a social impact network

The visualization of the impact network and of the dependencies among artifacts is interactive: when the user holds the mouse over a node for a few seconds different types of information are displayed as a tooltip including name of the developer, date modified, file impacted and reasons for impacts. Likewise, when the user holds the mouse over an edge, it is shown the type of dependency between the nodes and the reason for this edge to exist. Figure 5, for example, displays the edge connecting the node *Jean* with the node *Cleidson*. This edge represents a dependency based on activities (co-changes), since the files *RealTimeClock* and *Clock* (which are being modified by developers *Jean* and *Cleidson*) were modified together.

D. Tool architecture

RaisAware has been developed using the Java programming language and it is implemented as a plug-in for the popular IDE (Integrated Development Environment) Eclipse. As a plug-in, it is possible to use many resources already available in the IDE. For instance, it is possible to monitor events on each instance of Eclipse being used. Thus, whenever a developer starts an action that the tool identifies as relevant, such as starting the modification of a source code file, it fires an event that, once processed by RaisAware, transmits the obtained information to all developers.

RaisAware's architecture is shown in Figure 6. Arrows represent the information flow while rectangles indicate the software components. The upper block of each instance containing Visualization, I / O, Parser and Communicator are the components of RaisAware that were implemented, and the others blocks identify the external components. The description of the components and how they interact with each other is discussed in the next section.

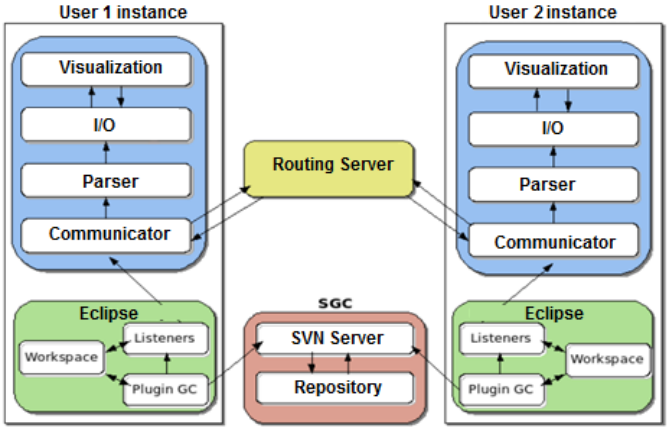

Fig. 6. Architecture of the RaisAware tool.

A. Setup

The first step to use RaisAware is to register a project on the tool, i.e., to select the project from which RaisAware will collect data and transmit them as events. Once the project is registered, the tool collects the dependency information aout the artifacts. Basically, two threads are initiated at this point: one, locally, retrieves dependencies based on the software architecture, and another, remotely, identifies dependencies from the activities of developers. We describe next how the data obtained by these threads is used.

Dependency extraction from the software architecture

To collect the dependencies from the software architecture, a feature provided by Eclipse called *ASTParser* is used. This feature allows, among other things, to identify the references between the source code files of a Java project. Once the user registers a particular project in RaisAware, the dependencies for all Java classes from this project are calculated, and with this information, RaisAware creates a call-graph representing these dependencies. This call-graph is represented as a matrix {Class X Class}, called call-graph matrix, which is stored in the project root directory as a CSV file.

Dependency based on developers' activities

As mentioned before, the co-changes method [14] is used to collect the dependencies based on developers' activities. This method allows the discovery of evolutionary dependencies between artifacts based on the frequency they were modified together. To identify whether an artifact has been modified alongside with others, data is obtained from the configuration management repository in which the project artifacts are stored. All files that are submitted together in each commit are analyzed by the Configuration Management plug-in (GC component plug-in) that allows data access and retrieval from configuration management systems. In the current implementation the Subclipse plug-in is used as the GC plug-in allowing access to Subversion repositories. In addition, an integrated Eclipse plug-in allows RaisAware to access CVS repositories. Once the data is obtained, it is created a matrix {File X file} called co- changes matrix, which indicates how many times each file was changed together with other files. This matrix is used during the computation of support and

confidence measures and is also stored in the project root directory as a CSV file.

Once the two matrices, the call-graph matrix and the co-changes matrix, are generated RaisAware is ready to collect and transmit information to other developers. This process begins in the Workspace of the developer: when he performs any action, events are triggered and are captured by Workspace Listeners. These listeners are interfaces integrated into the Eclipse Workspace that are able to identify the occurrence of events. When these events are triggered, the Communicator component of RaisAware, which implements the Listeners, captures these events and forwards them to the Notification Server component. This component is in charge of sending the events it receives with their associated information (e.g., file name, the developer who changed the file, modified date, etc) for all developers who are using RaisAware in the project on which the event has been captured. The Notification Server component has been implemented using Avis, an event routing service.

In another developer's workspace running another instance of RaisAware, the Communicator component is responsible for obtaining the events transmitted from the Notification Server. Once the events are obtained, they are passed to the Parser component, which interprets the information contained in the events to identify the file name obtained in the dependencies matrix. If the file name is found, it processes the data to identify the dependencies and sends the result to the I/O component. This component generates an XML file containing information of all files being modified. Finally, the View layer is responsible for presenting graphical user interfaces related to the impact network and other aspects. This is done by parsing the XML file.

VI. Evaluation

The evaluation of RaisAware has been made indirectly. In other words, it was not possible to evaluate the tool effectiveness neither in a real environment nor in an experiment. We chose to simulate situations using the tool from real data, and, by doing so, we were able to evaluate the design of the tool, especially the visualizations that we created. All these aspects are discussed in this section.

A. Methodology

In order to evaluate the RaisAware we used the approach described in [37]. More specifically, Ren and colleagues [37] describe how they evaluated a tool that supports change impact analysis in Java programs. The approach consists of (i) obtaining details of the commits done in a software development project during the period of one year, (ii) partitioning the data into weeks, and (iii) analyzing the commits that took place during the intervals of each week so that these commits are considered as parallel activities. In other words, this approach assumes that if commits were made by two or more developers within a 1-week interval, parallel activities possibly occurred during this period and, consequently, direct conflicts. The analysis of the modified source code files during the commits and of the entire codebase allows the identification of indirect conflicts. Finally, the historical analysis of the commits allows the identification of co-changes conflicts. In summary, using this approach the RaisAware tool could be evaluated taking into account the conflicts (direct, indirect and co-changes) that potentially occurred in each week of the project analyzed.

During the evaluation of the tool, we did not have access to data from corporate software development projects, therefore we chose to analyze a free software project since data from these projects is widely available. The project used was MegaMek, a network game developed in Java that is registered at SourceForge.net site since 2002, with the participation of 33 developers in its implementation since then. This project was chosen because it is one of the most actives in the community, which is important for our evaluation due to the high likelihood of conflicts.

Since our tool focus on conflicts among software developers, it is necessary to ensure that the commits in the MegaMek project have been made by more than a single developer. In our case, we wanted to identify time intervals in which development activities in MegaMek were performed by several developers. To identify the time interval to be used, we used the TransFlow tool [38], which allows one to visualize information about the commits in a software project.

Figure 7 shows an image generated by TransFlow for the first year of development of MegaMek. In the figure, each square is a commit performed by a developer, and the color of the square is used to distinguish the developers who performed the commits. The squares are plotted using Cartesian coordinates to plot two-variable values. In the case of Figure 7, the X-axis is a time axis that corresponds to the number of commits performed, while the Y-axis indicates the number of new classes added to the project. Based on this chart one can identify the periods in which many developers actively worked on the project. For example, the rectangle labeled "1" indicates a period in which a single developer committed in the project since all squares have the same color. Meanwhile, the rectangle labeled "2" indicates a period in which several different developers made commits, since the squares have different colors. Based on this analysis, we decided to use the entire first year of the project (periods 1 and 2) because we could illustrate the lack of conflicts (in period 1) and its existence (in period 2) when other developers joined the project.

In order to get data from the configuration management repository of Megamek and calculate the conflicts that occurred during its development, we implemented a software module that allows RaisAware to identify the occurrence of direct, indirect and co-changes conflicts for each week of development. Once conflicts are identified, data from these conflicts (number of conflicts, week number, filenames, etc) are exported in CSV format so that they can be analyzed and plotted in electronics spreadsheets.

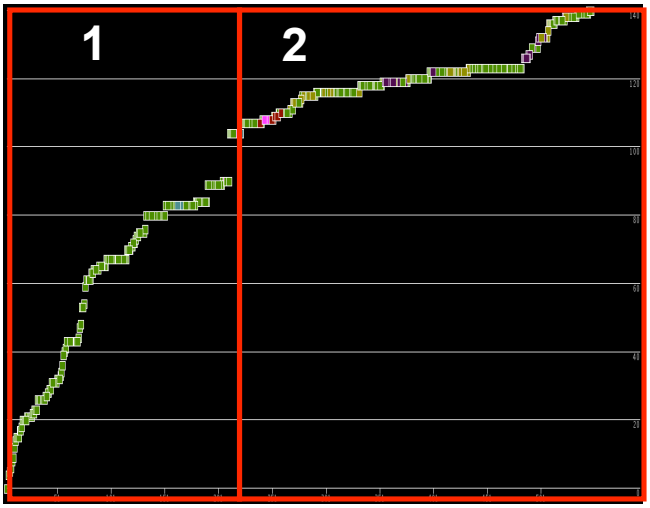

Fig. 7. Visualization generated by the tool Transflow for the first year of development of MegaMek

B. Results

The results obtained from our analysis can be seen in the graphs below. For simplicity reasons, we will now refer to the impact network as the *social* impact network because it refers to software developers, and the dependencies among files as the *technical* impact network, since it focuses on the technical aspects of the software development activity. The data presented in this section can be viewed for the entire project or only considering the impact network, either from a technical or social point of view.

Initially, Figure 8 shows the occurrence of direct, indirect and co-changes conflicts for each week from the technical point of view. It should be noted that the values for the conflicts are equal to zero in the beginning of the project, which corresponds to the period of Figure 7 when only one developer worked on the project. Only after 10 weeks, i.e., in week 11, conflicts begun to emerge: the period "2" in Figure 7 when other developers started to participate in the project. Furthermore, it is possible to notice that there were weeks in which the number of files involved in conflicts had increased substantially. For example, in the 39th week over 45 potential conflicts were identified. This can be justified by the amount of commits performed during this week, and also by the number of developers who have submitted code during the period. Figure 9 shows an image created by TransFlow that shows the commits that were performed during this particular week. In the figure, the X-axis represents the number of commits that occurred, while the Y-axis indicates the number of new classes added to the project. It can be seen in the visualization that 25 commits were made, a much higher number than the average number of commits for each week of the year (13). Moreover, the number of developers who worked during the 39th week is another factor to be considered, since 4 developers committed files during the week: twice the average of developers who committed files in each week of the year analyzed.

Figure 8 shows the number of files potentially involved in conflicts for every week of the year. This is the total number of files that may conflict in any given week, not the number of files modified by a particular developer. Thus, Figure 8 shows a super-set of the possible files that would be affected or would affect a single developer. To assess the specific number of files that could be in conflict with a developer's work, the following analysis was performed: for each file identified in a given week, we identified the number of files that are connected to this file in the impact network, i.e., the degree of the file for that week. Thus, the average degree of each file involved in possible conflicts was calculated for each week. The results are shown in Figures 10 and 11: they present the results for each week of the year for indirect conflicts and for co-changes, respectively.

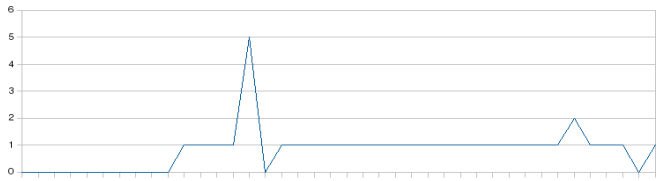

Fig. 10. Average number of impacted files by indirect conflicts for each file

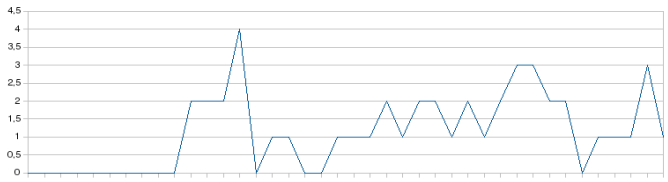

Fig. 11. Average number of impacted files by co-changes conflicts for each file

Figure 12 presents data of the potential conflicts identified in the year, but now considering the social impact network. Again, it should be noted that up to 10 weeks, no conflict was identified, since only one developer was active in the project.

Finally, the mean and standard deviation for the number of files (or people) possibly involved in a conflict was calculated for each type of conflict. Results are presented in Table 1. It is possible to verify that the average conflict by co-changes (considering the technical point of view) is larger than the other two types of conflicts, i.e., the amount of files on the impact network based on the co-changes method are usually larger than the values observed in direct and indirect conflicts. Moreover, is possible to notice that the number of affected files (technical impact network) is always greater than the corresponding number of developers (social impact network).

TABLE 1
VALUES OF MEAN AND STANDARD DEVIATION FOR THE 3 TYPES OF CONFLITS

	Technical		**Social**	
	Mean	Standard Deviation	Mean	Standard Deviation
Direct conflicts	3,63	3,24	2,17	0,79
Indirect conflicts	5,9	3,13	1,87	0,51
Co-Changes conflicts	8,63	6,7	2,07	0,94

C. Discussion

The results presented in the previous section suggest that RaisAware was able to identify potential conflicts in the project Megamek during the period of analysis. It is important to notice that all the graphs do not indicate any kind of conflict for the first 10 weeks of the project. This result is consistent with the analysis of Figure 7 that indicates that in these weeks there was only one developer making changes to the code and performing commits to the repository: there were not other developers on the project, so potential conflicts could not be identified.

Without considering the initial 10 weeks, it is possible to observe that the remaining weeks have very different values for each type of conflict when compared. For example, considering the values for the 39th week of the technical conflicts chart, the value is equal to 27 for the co-changes conflicts, 15 to direct conflicts and 6 to indirect conflicts. This result is interesting *per se*, since there is not a general consensus about which method of conflict identification is better [33]. Despite that, as the RaisAware's original goal is to assist software developers in coordinating their work during conflicts, the developer with his own experience using the tool would filter what information would be most useful.

An important aspect to be analyzed based on the results is whether the visualization offered by RaisAware is effective. Looking at the values from the technical impact network, it is possible to notice that a lot of potential conflicts are identified considering all the modified files in each week. Thus, the graph presentation would not be appropriate, since graphs with too many elements would not be easy to interpret due to the overlapping of nodes and edges. This would make the task of identifying conflicts more complex, possibly requiring user intervention (moving nodes and edges in the graph) in order to investigate a particular conflict.

On the other hand, when looking at Figures 10 and 11, we can see that the average number of files impacted by each file is not greater than 5 (Figure 10) or 4 (Figure 11). Thus, the impact network for each file that a user is modifying does not contain many elements, which suggests that a visualization based on graphs of impacted files might even be appropriate. In short, further research is necessary to establish the best visualization possible. To obtain a definitive answer to this question, we intend to conduct a controlled experiment [34] with undergraduate students. Similarly, we can say that the visualization is appropriate to present the social impact networks, as the graph in Figure 12 does not have high values: the maximum value is 9, which occurs only once. This suggests a small quantity of elements in the graph of the social impact network and, consequently, it is possible to conclude that the visualization used in RaisAware is effective for displaying information about software developers.

Since RaisAware visualizations are somewhat effective, we can argue that the software developers using our tool are likely to be able to identify their respective impact networks, technical or social. From a technical point of view, the developer, when visualizing the dependencies between files that are being modified, has the necessary information to coordinate his activities with other developers to ensure that the changes in the code of the project will not lead to a problematic situation. The social impact network allows the identification of developers who may be impacted by the developer's code or whose code might impact him. Thus, it indicates with whom a given developer needs to coordinate his activities, something seen as problematic in previous studies of software developers' work [15] [18]. RaisAware therefore facilitates the work of software engineers involved in coding activities, and can be used as a complement for other approaches for impact management [12].

VII. Final Remarks

This article presented RaisAware, a tool to support collaborative software development that is integrated in the popular Eclipse IDE. The theory in the literature that motivated the tool implementation was presented, in addition to the description of the tool architecture and the techniques used by RaisAware to identify dependencies.

The purpose of the tool is to support the impact management as the framework proposed by de Souza and Redmiles [12] rather than focusing in support for strategies that are specific to certain contexts. The tool provides real-time data on the impacts of activities undertaken by developers, thus the tool allows the users to be aware of their impact networks [12], allowing them to adopt strategies that do not disrupt the work of their colleagues and to prevent others from affecting their work. RaisAware was evaluated using an approach based on real data from a free software project. The results suggest that the views generated by RaisAware could be effective in a context similar to the analyzed project. However, further research is still necessary to confirm that.

One aspect that needs to be investigated in future studies is the influence of different

Acknowledgments

This research was supported by the Brazilian Government under grants CNPq479206/2006-6, CNPq 473220/2008-3 and by the Fundação de Amparo à Pesquisa do Estado do Pará (FAPESPA) through "Edital Universal N.° 003/2008".

References

[1] Schmidt, K. and Simone, C. Coordination mechanisms: Towards a conceptual foundation of CSCW systems design. Journal of Computer Supported Cooperative Work, 5 (2-3). 155-200.

[2] McDonald, D.W. and Ackerman, M.S., Just Talk to Me: A Field Study of Expertise Location. in Conference on Computer Supported Cooperative Work (CSCW '98), (Seattle, Washington, 1998), 315-324.

[3] Schümmer, T. and Haake, J.M., Supporting Distributed Software Development by Modes of Collaboration. in Seventh European Conference on Computer Supported Cooperative Work, (2001), 79-98.

[4] Grinter, R.E., Using a Configuration Management Tool to Coordinate Software Development. in Conference on Organizational Computing Systems, (Milpitas, CA, 1995), 168-177.

[5] Halverson, C.A., Ellis, J.B., Danis, C. and Kellogg, W.A. Designing task visualizations to support the coordination of work in software development Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work, ACM, Banff, Alberta, Canada, 2006.

[6] Conway, M.E. How Do Committees invent? Datamation, 14 (4). 28-31.

[7] Parnas, D.L. On the Criteria to be Used in Decomposing Systems into Modules. Communications of the ACM, 15 (12). 1053-1058.

[8] Morelli, M.D., Eppinger, S.D. and Gulati, R.K. Predicting Technical Communication in Product Development Organizations. IEEE Transactions on Engineering Management, 42 (3). 215-222.

[9] de Souza, C.R.B., Redmiles, D., Cheng, L.-T., Millen, D. and Patterson, J., Sometimes You Need to See Through Walls - A Field Study of Application Programming Interfaces. in Conference on Computer-Supported Cooperative Work, (Chicago, IL, USA, 2004), ACM Press, 63-71.

[10] MacCormack, A., Rusnak, J. and Baldwin, C.Y. Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code Harvard Business School Working Papers, Harvard University, Cambridge, MA, 2004, 40.

[11] Cataldo, M., Wagstrom, P.A., Herbsleb, J.D. and Carley, K.M. Identification of Coordination Requirements: implications for the Design of Collaboration and Awareness Tools 20th Conference on Computer Supported Cooperative Work, ACM Press, Banff, Alberta, Canada, 2006.

[12] de Souza, C.R.B. and Redmiles, D. An Empirical Study of Software Developers' Management of Dependencies and Changes. International Conference on Software Engineering, Leipzig, Germany, 2008, ACM, 241-250.

[13] Lakhotia, A., Constructing call multigraphs using dependence graphs. in 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, (Charleston, South Carolina, USA, 1993), ACM Press, 273-284.

[14] Zimmermann, T., Weibgerber, P., Diehl, S. and Zeller, A. Mining Version Histories to Guide Software Changes. IEEE Transactions on Software Engineering, 31 (6). 429-445, 2003.

[15] Sarma, A., Noroozi, Z. and van der Hoek, A., Palantír: Raising Awareness among Configuration Management Workspaces. in Twenty-fifth International Conference on Software Engineering, (Portland, Oregon, 2003), 444-453.

[16] Trainer, E., Quirk, S., de Souza, C.R.B. and Redmiles, D., Bridging the Gap between Technical and Social Dependencies with Ariadne. in Eclipse Technology Exchange, (San Diego, CA, 2005).

[17] Grinter, R.E. Recomposition: Coordinating a Web of Software Dependencies. Journal of Computer Supported Cooperative Work, 12 (3). 297-327.

[18] de Souza, C.R.B., Redmiles, D.F. and Dourish, P., "Breaking the Code", Moving between Private and Public Work in Collaborative Software Development. in International Conference on Supporting Group Work, (Sanibel Island, Florida, USA, 2003), 105-114.

[19] Perry, D.E., and, H.P.S. and Votta, L.G. Parallel Changes in Large-Scale Software Development: An Observational Case Study. ACM Transactions on Software Engineering and Methodology, 10 (3). 308-337, 1998.

[20] Valleto, G., Helander, M., Ehrlich, K., Chulani, S., Wegman, M. and Williams, C. Using Software Repositories to Investigate Socio-technical Congruence in Development Projects Workshop on Mining Software Repositories, ACM Press, Minneapolis, 2007.

[21] Aho, A.V., Sethi, R. and Ullman, J.D. Compilers: Principles, Techniques and Tools. Addison-Wesley, 1986.

[22] Callahan, D., Carle, A., Hall, M.W. and Kennedy, K. Constructing the Procedure Call Multigraph. IEEE Transactions on Software Engineering, 16 (4). 483-487.

[23] de Souza, C.R.B., Froehlich, J. and Dourish, P., Seeking the Source: Software Source Code as a Social and Technical Artifact. in ACM Conference on Supporting Group Work, (Sanibel Island, FL, USA, 2005), ACM Press, 197-206.

[24] Grinter, R.E., Recomposition: Putting It All Back Together Again. in Conference on Computer Supported Cooperative Work (CSCW'98), (Seattle, WA, USA, 1998), 393-402.

[25] Sosa, M.E., Eppinger, S.D., Pich, M., McKendrick, D.G. and Stout, S.K. Factors that influence Technical Communication in Distributed Product Development: An Empirical Study in the Telecommunications Industry. IEEE Transactions on Engineering Management, 49 (1). 45-58.

[26] Sosa, M.E., Eppinger, S.D. and Rowles, C.M. Identifying Modular and Integrative Systems and Their Impact on Design Team Interactions. ASME Journal of Mechanical Design, 125. 240-252.

[27] Sosa, M.E., Eppinger, S.D. and Rowles, C.M. The Misalignment of Product Architecture and Organizational Structure in Complex Product Development. Management Science, 50 (12). 1674-1689.

[28] Perry, D.E. and Wolf, A.L. Foundations for the Study of Software Architecture. ACM SIGSOFT Software Engineering Notes, 17 (4). 40-52, 1992.

[29] Bass, M., Mikulovic, V., Bass, L., Herbsleb, J. and Cataldo, M. Architectural Misalignment: An Experience Report IEEE/IFIP Working Conference on Software Architecture, 2007.

[30] Fitzpatrick, G., Mansfield, T. and al., e., Augmenting the workaday world with Elvin. in 6th European Conference on Computer Supported Cooperative Work, (Copenhagen, Denmark, 1999), Kluwer, 431-450.

[31] Fitzpatrick, G., Marshall, P. and Phillips, A. CVS integration with notification and chat: lightweight software team collaboration Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work, ACM, Banff, Alberta, Canada, 2006.

[32] Dourish P. and V. Bellotti. Awareness and Coordination in Shared Workspaces. Conference on Computer-Supported Cooperative Work (CSCW '92), Toronto, Ontario, Canada, ACM Press, 1992.

[33] Kagdi, H. and Maletic, J. I. "Combining Single-Version and Evolutionary Dependencies for Software-Change Prediction". In Proceedings of the Fourth international Workshop on Mining Software Repositories. International Conference on Software Engineering, 2007.

[34] Wohlin, C., P. Runeson, et al. "Experimentation in Software Engineering: An Introduction", Kluwer Academic Publishers, 2000.

[35]Biehl, J. T., Czerwinskim M., Smith, G., and Robertson, G. G. FASTDash: a visual dashboard for fostering awareness in software teams. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2007

[36] Cheng, L. Hupfer, S. Ross, S. and Patterson. "Jazzing up Eclipse with collaborative tools". In *Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology Exchange*, 2003.

[37] Ren X., Shah, F. Tip, F., Ryder, B. G. and Chesley, O."Chianti: a tool for change impact analysis of java programs," in OOPSLA '04: Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 2004.

[38] Costa, JMR, Santana, Francisco W., de souza, Cleidson R.B. "Understanding Open Source Developers Evolution Using Transflow". In: Collaboration Researchers International Workshop on Groupware, 2009.

[39] Dewan, P. Hegde R.: Semi-Synchronous Conflict Detection and Resolution in Asynchronous Software Development. In Proceedings of the European Conference on Computer-Supported Cooperative Work 2007: 159-178.

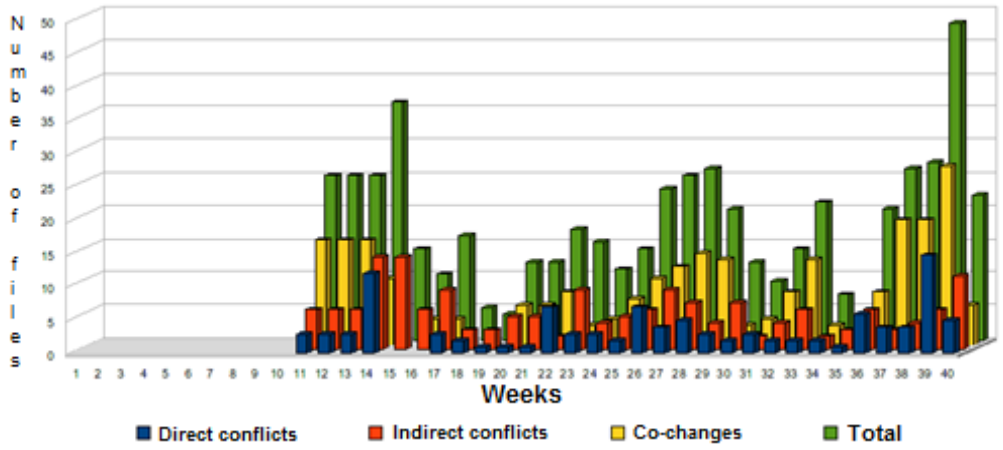

Fig. 8. Total number of conflicts from a technical point of view, i.e., regarding the conflicts between files.

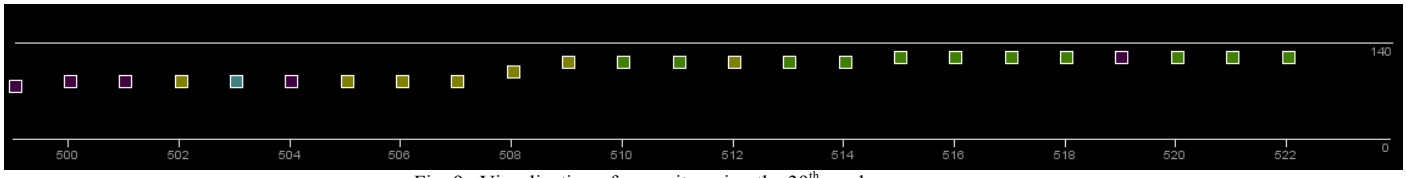

Fig. 9. Visualization of commits uring the 39th week

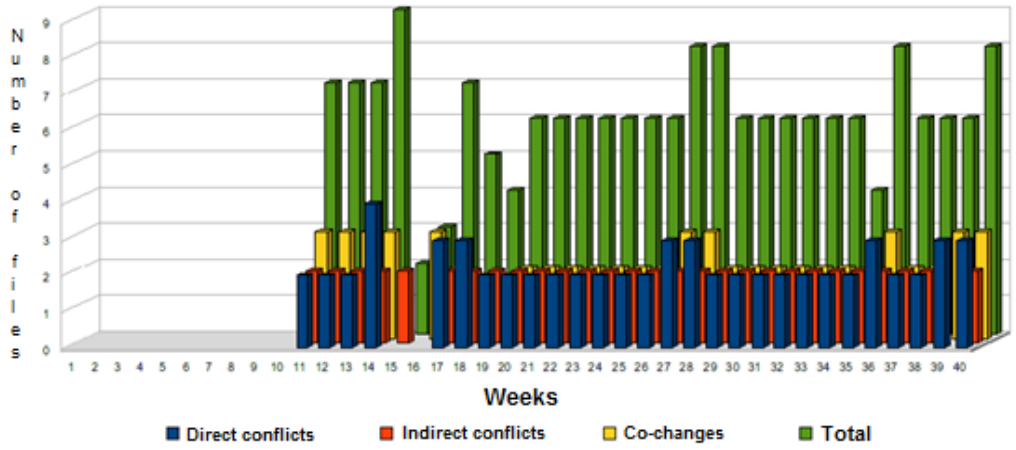

Fig. 12. Total number of conflicts from a social point of view, i.e., the size of the social impact network.