A Multiple View Environment for Collaborative Software Comprehension

Glauco de F. Carneiro
Computer Science Department
Salvador University (UNIFACS)
Salvador, Bahia, Brazil
glauco.carneiro@unifacs.br

Carlos F. R. Conceição
Computer Science Department
Salvador University (UNIFACS)
Salvador, Bahia, Brazil
carlos.conceicao@unifacs.br

José Maria N. David
Computer Science Department
Federal University of Juiz de Fora
Juiz de Fora, Minas Gerais, Brazil
jose.david@ufjf.edu.br

***Abstract*—Collaboration is an important issue for software comprehension activities which are performed in distributed development environments. Several studies have pointed to the relevance of visualization to provide support to these activities. Enriching visual metaphors with awareness elements can enhance collaboration in such environments. This paper presents a multiple view interactive environment to support collaborative software comprehension. A case study was carried out to analyze the effectiveness of the proposed environment considering that awareness elements are visually represented to support the collaborative software comprehension.**

Keywords- collaboration; software comprehension; software visualization; distributed development environments.

I. INTRODUCTION

Humans rely more on vision than all the other senses [25]. For this reason, the use of visual resources is relevant for software engineering. Software visualization uses perceptible cues to visually represent several software systems properties. The goal is to unveil patterns and structures that otherwise would remain hidden during software comprehension activities [24]. Software comprehension in distributed development environments requires collaboration support. Awareness plays an important role in software comprehension activities in a collaborative environment since it supports programmers to find meaningful information to their activities [11]. Supporting awareness in a distributed environment enables, for example, the identification of who is working in the project, what participants are doing, why they are doing, which artifacts they are manipulating and how their actions might impact others [16].

Visual resources have been used to support programmers to perform their activities in distributed development environments [1][2]. However, there are still some open questions in this area, specially related to awareness. In this paper we focus on two of these questions. The first is related to the inclusion of visual representation of awareness elements in integrated development environments (IDEs) in order to increase the effectiveness of software comprehension. The goal is to provide programmers with information related to what has been done in the context of a given project. The second question is related to the use of multiple view interactive environments as a mean to enhance software comprehension. The goal is to support awareness due to the use of three important concepts used in the information visualization domain: i) navigational slaving – multiple views systems should enable actions in one view to be automatically propagated to the others [22]; linking – multiple views systems should connect data in one view with data in the other views [17]; brushing – multiple views should enable corresponding data items in different views to be simultaneously highlighted [17].

A view is a particular visual representation of a data set. Complex data sets typically require multiple views, each revealing a different aspect of the data [19]. Multiple view systems have been proposed to support the investigation of a wide range of information visualization topics [20]. SourceMiner [3][28] is a multiple view interactive environment (MVIE) from which the collaborative environment was developed and now is described in this paper. It was implemented as an Eclipse IDE plug-in to interactively visualize Java projects, complementing the native views and resources provided by the IDE. It uses code as its main data source and provides a set of features to support programmers to configure the visual scenario that best fit a software comprehension goal. Examples of features to interact with the views are: (i) filters to visually present information that match filtering criteria; (ii) semantic and geometric zooming to better adjust views to the canvas; (iii) flexibility to arrange views in accordance with the preference of the programmer; and (iv) transparent navigation from the visual representation to the source code. SourceMiner has been used in different software engineering studies such as code smells identification [3] and characterization of strategies adopted by programmers in software comprehension activities [4].

The Collaborative SourceMiner [5] is a collaborative version of SourceMiner. It combines the use of a multiple view interactive environment with collaboration elements such as chat and bullets that inform which parts of the software have been analyzed by each programmer . This paper focuses on awareness support of the environment. The goal is to enhance software comprehension in distributed development, for this reason we use the term collaborative software comprehension.

 ISBN: 978-1-61208-230-1

This paper is structured as follows. In Section II we briefly review concepts related to collaborative software comprehension. Next, we present the proposed conceptual model of Collaborative SourceMiner. In Section IV we present a case study to analyze the effectiveness of the use of awareness elements in MVIEs to support collaborative software comprehension. Finally, in Section V, a discussion is presented, followed by conclusions about our work and avenues for future work

II. COLLABORATIVE SOFTWARE COMPREHENSION

In distributed software development environments, the geographic distance can hinder and limit the interaction opportunities due to the temporal distance [6]. It also hampers the understanding level of actions and efforts of group participants due to the cultural differences [7]. Moreover, the fact that the participants could have different native languages is a potential obstacle to communication [8]. According to Dix [9], two important aspects that benefit programmers in distributed development environment are: (i) explicit communication, where one programmer can inform others about his or her activities, and (ii) consequential communication where programmers can obtain useful information to accomplish activities by observing others´ actions.

The approach used in this paper is based on a collaboration model known as 3C+P (communication, coordination and cooperation plus perception) proposed in [17]. According to Fuks and Assis [12], awareness is the key element to support collaboration activities. However, the way these elements interact with each other depends on the project in which they have been used [18]. Awareness provides information to enhance collaboration due to the following: a) it enables the coordination of activities; b) it promotes the discussion of tasks through communication; c) it enhances interaction with others participants in the shared workspace through cooperation [10][13]. The workspace has an important role in collaboration activities [14]. Through the shared workspace participants can gain knowledge about group activities. This fact enhances awareness. The way awareness is supported in shared workspaces is essential in the cases where time and space need to be considered in the collaboration process definition [15].

In a distributed context, visual resources could also support awareness. These resources can be combined with collaboration elements (communication, coordination and cooperation) represented in the IDE to enhance software comprehension. This results in the proposed conceptual model that is discussed in the next section.

Researchers have already used visualization to support awareness. For example, Lanza et all. [26] proposed an approach to augment awareness by recovering development information in real time and broadcasting it to developers in the form of three lightweight visualizations. Treude and Storey [27] conducted a study about the use of a community portal by software project members. However, to the best of our knowledge, these researches do not consider examples of the use of awareness elements associated with collaboration elements in order to support software comprehension activities in MVIEs.

III. THE PROPOSED CONCEPTUAL MODEL

The proposed conceptual model was based on the definition of awareness presented in [11]. The main goal is to enable programmers from the same group to collaborate in a shared workspace and hence obtain knowledge to perform software comprehension activities. The conceptual model has as its start point the scenario illustrated in Figure 1. According to the figure, programmers perform software comprehension activities in different places. In the figure, the circle illustrates programmers accessing the source code (triangle) using the IDE (square). Considering this situation, we can conclude that collaboration occurs using resources (for example, chat on line) that are not integrated into the IDE. This scenario does not necessarily explore the potentiality of visual resources to support software comprehension activities. Moreover, a considerable cognitive effort will be needed due to the fact that the collaboration resources are not integrated into the IDE. This situation can also hinder convergence to perform a collaborative software comprehension.

Based on the scenario illustrated in Figure 1, we present the Figure 2 with the proposed conceptual model for collaborative software comprehension. The difference between Figure 1 and the part A of Figure 2 is that the IDE now has the Collaborative SourceMiner plug-in, represented by the red circle in the Figure 2. Moreover, the visual resources provided by the Collaborative SourceMiner have the goal to support awareness in a distributed software development. The views are enriched by awareness elements to enhance communication, coordination and cooperation.

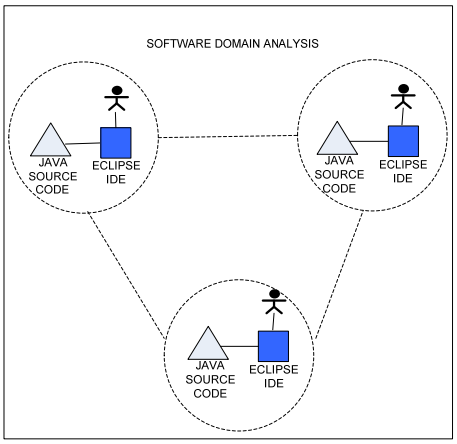

Figure 1. Collaborative Software Comprehension

For example, consider a set of classes that had its source code most frequently accessed by the members of a team while performing a given task. The result is that they can have its visual representation highlighted in the views. In this same example, a programmer can add a note to the visual representations of a class reporting information that needs to be considered relevant to the execution of the same task.

 ISBN: 978-1-61208-230-1

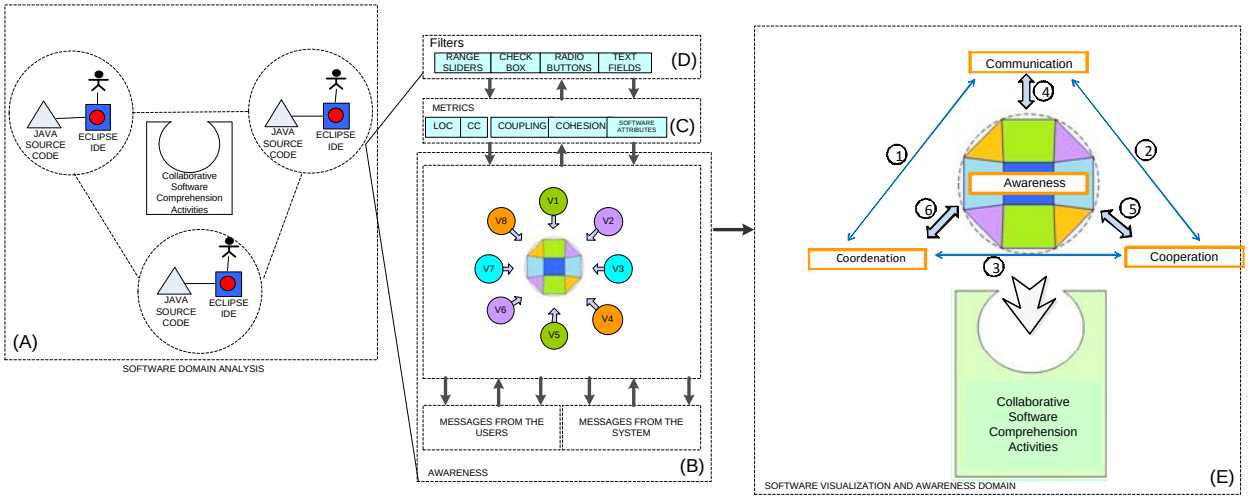

Figure 2. The Proposed Conceptual Model for Collaborative Software Comprehension

The fact that the representation of a set of entities is highlighted by awareness elements does not necessarily imply that these entities are relevant. These awareness elements are an initial suggestion of what should be analyzed by the group. For this end, the group can use the shared workspace to discuss and converge to the set of entities that are really of interest to the task at hand.

The combined use of multiple views enriched by awareness elements is conveyed in the part B of the Figure 2. Each view is represented by a colored circle (V1 to V8, for example). These views when used together and combined aim at providing features of a multiple view interactive environment (MVIE). The awareness elements are the result of information that programmers find useful to share with others from the same team (marked as messages from the user in the Figure 2 and implemented in Figure 6) and information regarding classes and methods accessed while performing a specific task (marked as messages from the system in the Figure 2 and implemented in Figure 7). These messages are represented in the views using visual attributes such as icons and colors that can vary in tonality depending on the type and numbers of messages related to a specific software entity (see Figure 7). The part C of the figure shows that software entities (packages, classes, methods, attributes and interfaces) obtained from the Abstract Syntax Tree (AST) are enriched by metrics such as size, cyclomatic complexity, and coupling. The model allows the inclusion of new metrics that appear to be relevant in the shared workspace. The part D illustrates that the interaction with the multiple views is supported by the filters, semantic and geometric zooms and other interaction resources.

In fact, the model considers the influence of coordination, cooperation and communication elements to enrich the shared workspace with awareness information. This is represented in part E of Figure 2, where the result is a visual scenario composed of multiple views and their corresponding awareness elements.

The model considers both synchronous and asynchronous interaction support. Interactions in Collaborative SourceMiner result from two types of messages: those from the user (Figure 6) and messages that are automatically collected by the system registering what programmers are doing in the IDE. The first one is the kind of messages that can be sent by the programmers to register information that is considered relevant to a specific task asynchronously. The second type of message is sent automatically by the Collaborative SourceMiner. The goal of this set of messages is to enrich the visual representation in the multiple views in order to contextualize programmers synchronously. When a programmer starts the execution of a task his or her actions are registered and automatically sent to a server, as illustrated in Figure 3.

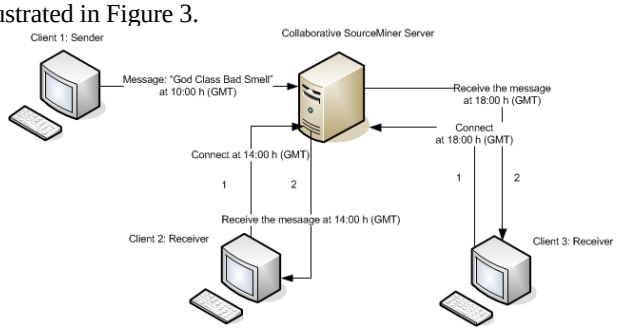

Figure 3. Implemented Topology of the Conceptual Model

A web service is available to receive and send messages from and to the Collaborative SourceMiner clients which are configured in the team. The client of this service is the IDE Eclipse with the Collaborative SourceMiner plug-in. The messages contain the following parameters: project, user, and, optionally, the activity in which the programmer is working. Before recording the message, Collaborative SourceMiner checks if the user who sent the message is in fact registered in the project.

 ISBN: 978-1-61208-230-1

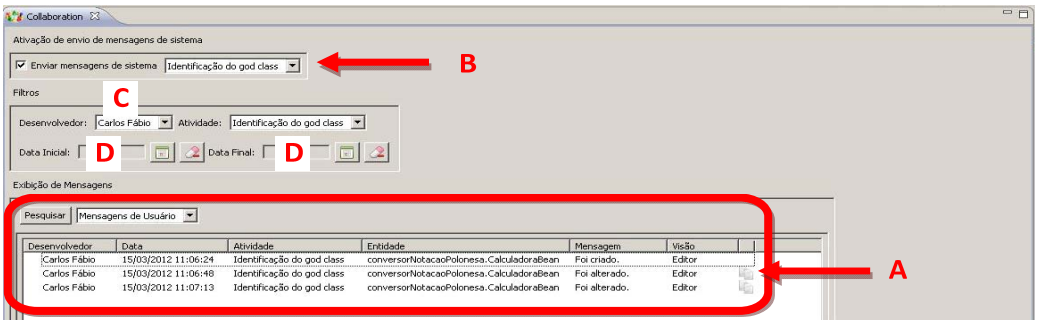

Figure 4. Messages from the System Registering Actions Executed by a Programmerc

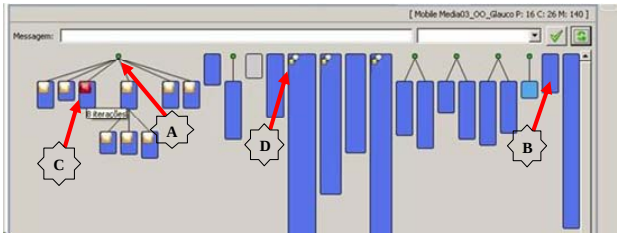

Figure 5. Polymetric View enriched by Messages from the System Registering Actions Executed by a Programmer

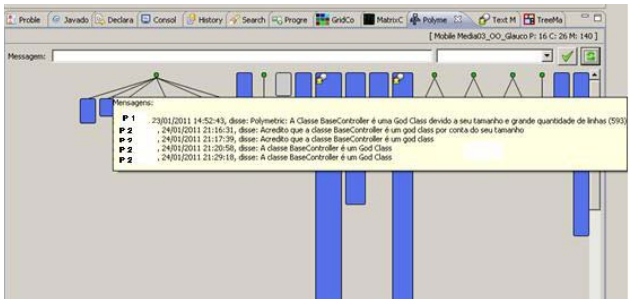

Figure 6. Communication among Participants through the Polymetric View

Figure 7. Treemap View displaying Icons to represent Messages from the System

Figure 4 displays examples of messages that are automatically registered (marked as A in the Figure). Each register has the following format: programmer, date/hour, activity, entity (class, interface, or method) and view. The messages can be filtered using as a parameter the programmer(s) that performed the actions (marked as C in the figure) and the task that it was associated with (marked as B in the figure).

Another possibility to filter messages is by the period in days that it occurred (marked as D). These data aim at characterizing actions performed by programmers while performing a specific task. When a software entity is modified, an icon is presented. Different versions of this entity can be shown if the programmer click in this icon. In Figure 5 the polymetric view [3] illustrates the inheritance hierarchy of entities (classes and interfaces) of a software system. It portrays inheritance relationships between the software entities (class/interface) as a forest of round rectangles.

Originally proposed for this purpose, polymetric views help to understand the structure and detect problems of a software system in the initial phases of a reverse engineering process [3]. Interfaces are represented as green circles (arrow A) and classes as blue rectangles (arrow B). In the same figure, arrows C and D indicate icons that represent messages that can be relevant to the understanding of a specific entity. In this case, arrow C indicates messages from system while arrow D indicates messages from the users. The icons can vary in tonality to highlight software entities with which programmers most interacted. This is related to the coordination support. It has the goal to indicate entities that at first glance are somehow related to the activity performed by the group. This can, for example, motivate the group to know which entities programmers with more experience were interacting with. In this case, specific pieces of code are relevant when the team knows that experienced programmers worked on them.

This can promote faster convergence for the identification of these parts of the code that are probably related to the software comprehension activity. Moreover, this scenario can also be used to stimulate interactions among participants so that they can make a decision. The difference between this scenario and the one described in Figure 1 is that now collaboration occurs through the use of the Collaborative SourceMiner shared workspace. Another difference is that the collaborative software comprehension is based in a multiple view interactive environment.

As already discussed, visual resources have the potential to support collaborative software comprehension. This potential can be better exploited when the views are enriched

 ISBN: 978-1-61208-230-1

with information from every participant. For example, the way the shared workspace has been used by group members of the group (part B of conceptual model presented in Figure 2) encourage the sharing of knowledge.

Another example is related to the importance of expert programmers in a team. They can lead the convergence to the strategy to be applied in a given activity. This fact can occur when the shared workspace indicates software entities that expert programmers have selected to perform the activity. This enables other programmers to analyze the same entities through the same views as suggested and registered by the experts. Moreover, this situation motivates interaction among programmers so that decisions can be taken together and the group has access to a wider pool of ideas and possibilities regarding the activity to be performed then they would have when working alone. The expected result is the combined use of collaboration elements (communication, coordination and cooperation) in a multiple view interactive environment to support software comprehension activities. This is represented in Part E of Figure 2.

IV. THE CASE STUDY

A case study was conducted to analyze the following research question: "How awareness elements provided by Collaborative SourceMiner support software comprehension considering that programmers work collaboratively in a distributed environment?"

Null hypothesis: Awareness elements provided by Collaborative SourceMiner do not effectively support the identification of code smells considering that the participants work collaboratively in a distributed environment.

Alternative hypothesis: Awareness elements provided by Collaborative SourceMiner effectively support the identification of code smells considering that the participants work collaboratively in a distributed environment.

Six participants took part in the study. They worked in two groups of three participants each. This number of participants offered a reasonable tradeoff between the cost of the study and detailed qualitative analysis and the generalizability of the results. To be eligible for inclusion, participants were required to have the following skills: experience with the object-oriented programming Java; and in the use of the Eclipse IDE. This experience was verified by asking them to fill in questionnaire forms. No current member of our research groups took part in this study. They were all volunteers and no compensation was provided for their participation in this study.

Prior to the study tasks, the participants were required to complete a tutorial session on how to use the multiple views approach implemented by Collaborative SourceMiner. In this training session, the participants had 24 hours to familiarize themselves with the tool. They were asked to analyze a program, called Health Watcher [21], and to answer 28 basic questions regarding the tool functionalities. During the tutorial session, the second author of this paper was available online (email and chat) to provide complementary guidance and detailed explanation on how to use Collaborative SourceMiner. After the tutorial participants were asked to execute the code smells identification.

This study relies on a software product line, called MobileMedia (MM) [22] that manipulates photo, music, and video on mobile devices, such as mobile phones. It has about 4 KLOC distributed in 18 packages and 50 classes. We selected MobileMedia due to several reasons. First, its Java implementation is available. Second, its key concerns were previously identified by the developers and mapped to the source code [22].

We relied on two experts to build a reference list for each analyzed code smell (Feature Envy - FE, God Class - GC, and Divergent Change - DC[23]). The experts are researchers that participated in the development, maintenance, and assessment of the target system. The goal was to detect actual instances of each code smell in versions 3 to 7 of MobileMedia.

We collected direct and indirect data based on questionnaires answered by the participants and provided by an instrumentation system. The questionnaires described the MobileMedia main functionalities, the code smells with examples, and the tasks to be performed by the participants. Participants were asked to list classes suspected of manifesting code smells as well as the strategies they use to identify them. They were also asked to describe which of the Collaborative SourceMiner resources, such as views, concerns, filters, and colors, they found helpful to perform the task at hand. A logging functionality of Collaborative SourceMiner automatically records data describing the environment usage at a fine-grained detailing level. This functionality sends the data automatically to the server (see Figure 3) and is used to monitor how frequently a view or a feature of the tool is used, the transitions among views, and the time each action happened. The goal is a better understanding of the participants' strategies based on their recorded actions.

Two important roles of this study were the coordinator and programmer. The first had the following responsibilities: register the project to be analyzed, the activities to be performed and the participants of the study. The goal was to configure the environment for the study. The coordinator did not perform any of the software comprehension activities. Each team had 48 hours to perform the asked tasks. Each group was asked to identify the code smells Feature Envy, God Class and Divergent Change [23] in a software system called Mobile Media.

Table 1 presents the values of precision (p) and recall (r) of the identification of code smells of each participant. The precision metric quantifies the rate of correctly identified code smells by the number of detected code smell candidates. Recall quantifies the rate of correctly identified code smells by the totally number of actual code smells. PA1, PA2, and PA3 represent the participants 1, 2 and 3 from the first group. PA4, PA5, and PA6 represent the participants 1, 2, and 3 from the second group. The values of PA5 were not considered in the study due to the fact that he did not answer the questionnaires.

 ISBN: 978-1-61208-230-1

TABLE I. PRECISION AND RECALL IN CODE SMELLS IDENTIFICATION

	PA1		PA2		PA3		PA4		PA6	
	r	p	r	p	r	p	r	p	r	p
GC	0,9	1,0	0,8	0,9	0,9	0,7	0,2	0,7	0,2	0,7
DC	0,2	0,6	0,1	0,4	0,1	0,1	0,1	0,1	0,1	0,4
FE	0,4	0,3	0,1	0,1	0,2	0,2	0,2	0,4	0,2	0,6

The analysis of the data from Table 1 shows that participants from the first group had greater variation of precision and recall than participants from group 2. The lesser variation of precision and recall of group 2 can be justified by the collaboration among participants. All the participants obtained higher values and lesser variation of precision and recall in the God Class identification when compared with the other two code smells. The analysis of the messages provided by the Collaborative SourceMiner and the questionnaires show that participants used the communication feature (internal chat) provided by the proposed plug-in to indicate the code smells candidates.

Messages among participants revealed evidences of communication during the execution of the asked tasks. Figure 6, for example, shows evidences of this communication where PA1 and PA2 comment the case of the class BaseController as a God Class candidate. PA2 also informed us in the questionnaires that s/he had clicked in the icons in the views to read messages sent by others about the relationship of specific software entities and the asked task.

PA2 also mentioned that used the filters presented in Figure 4 to analyze the messages of interest to the task. According to PA2, “when I noticed that PA1 interacted several times with the class UnavailablePhotoAlbumException, I analyzed the class in more details in order to verify if it should be a Feature Envy candidate. However, after this analysis I concluded that it was not a Feature Envy occurrence”. PA1 registered in the questionnaire that “the comments from the other participants helped me to identify certain particularities in the classes and methods of the analyzed software system that I would not be able to identify without collaboration”. PA3 informed that: “The messages, especially the ones from PA1, were of great relevance to guide me in the execution of the asked tasks. PA4 also mentioned that: “the indication of the BaseController class was in accordance with the suggestion of PA6”. This comments provided by the participants show initial evidences that enriching the visual representations with Information provided by the participants of a group contributed to the convergence of which should be done in the asked tasks.

Due to the values of precision and recall obtained by PA1 and the comments registered in the questionnaires, there are initial evidences that PA1 guided PA2 and PA3 in the tasks execution using the collaboration resources provided by the Collaborative SourceMiner.

Based on the analysis of the research questions analysis, we present the observations as follows. Observation 1: during the execution of the asked activities programmers collaborated among themselves and to some extent converged in the indication of code smells. Observation 2: there are initial evidences that participants considered the actions and comments of others from the same group to decide how to proceed in the asked activities. Observation 3: there are initial evidences that participants adopted similar strategies to identify code smells, hence they collaborate while performing the asked tasks. These observations and the results obtained in the study presented in this paper show evidences that the alternative hypothesis is true.

A. *Threads to Validity*

The use of only one object (Mobile Media) as well as its size and complexity are far inferior when compared with typical software systems. However, MobileMedia has already been used in other studies to characterize the use of software visualization tools. An important limit to the generalizability of our findings comes from the fact that we have based our observations on the analysis of the behavior of only five subjects. However, as already mentioned, the number of participants accepted in the study was based on a tradeoff between the cost of the study and qualitative analysis of the results to derive the observations.

V. CONCLUSION AND FUTURE WORK

This paper presented a multiple view environment to support collaborative software comprehension. The coordinated views integrated into the IDE provide mechanisms that enable the use of awareness to perform software comprehension activities in a distributed development. The results of the study presented in this paper show initial evidences about how programmers use Collaborative SourceMiner to perform code smells identification. Moreover, the results showed how the features provided by the proposed environment enable the use of awareness elements to perform software comprehension activities. Another important result was the use of visual representation of software entities combined with awareness elements in the context of software comprehension. Differently from other studies like the ones presented in [26] and [27], the focus of this paper was to present initial evidences on how programmers could interact and collaborate to foster software comprehension using the visual metaphors available in SourceMiner. We are planning the inclusion of other mechanisms of cooperation, communication and coordination in the Collaborative SourceMiner to support software comprehension activities in a distributed development.

A version of CollaborativeSourceMiner is available at [28] as well as instructions to configure its environment.

 ISBN: 978-1-61208-230-1

REFERENCES

[1] Biehl, J.T., Czerwinski, M., Smith, G., and Robertson, G.G. (2007) "Fastdash: a visual dashboard for fostering awareness in software teams". In: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM 1313-1322.

[2] de Souza, C.R.B., Quirk, S., Trainer, E. and Redmiles, D. Supporting Collaborative Software Development through the Visualization of Socio-Technical Dependencies. ACM Conference on Supporting Group Work, ACM Press, Sanibel Island, FL, 2007.

[3] Carneiro, G., Silva, M., Mara, L., Figueiredo, E., Sant'Anna, C., Garcia, A., and Mendonca, M. Identifying Code Smells with Multiple Concern Views. In proceedings of the 24th Brazilian Symposium on Software Engineering (SBES), 2010.

[4] Fernandes, J. M.; Carneiro, G. Strategies and Profiles of Novice Programmers while Identifying Code Smells. In: IX Brazilian Workshop on Software Maintenance (WMSWM 2012), Fortaleza/CE. In portuguese.

[5] Conceição, C.F.R. Analyzing the Use of Awareness Elements to Support Software Comprehension Activities in a Distributed Development Environment. Master Thesis. Computer Science Department. Salvador University (UNIFACS), 2012. In portuguese.

[6] Gerfalk, P. J. Fitzgerald B. Flexible and distributed software processes: old petunias in new bowls? Communications of the ACM, v. 49, n.10, p.26-34, 2006.

[7] Casey V. Leveraging or exploiting cultural difference? In: IEEE International Conference on Global Software Engineering (ICGSE 2009), Limerick, Ireland: IEEE Computer Society, 2009. p. 8-17.

[8] Carmel, E.; Tjia, P. Offshoring Information tecnhnology: sourcing and outsourcing to a global workforce. Cambridge: Cambridge University Press, Cambridge, U.K., 2005.

[9] Dix, A.; Finlay, J.; Abowd, G.; and Beale, R. Human-Computer Interaction, Prentice Hall. 1993.

[10] Ellis, C. A.; Gibbs, S. J.; and Rein, G. L. Groupware - Some Issues and Experiences. Communications of the ACM, v. 34, n. 1, p. 38-58, 1991.

[11] Dourish, P.; Bellotti, V. Awareness and coordination in shared workspace. Conference on Computer-Supported Cooperative Work. pp. 107-114, Toronto, Canada, Nov. 1992.

[12] Fuks, H.; Assis, R. L. Facilitating perception on virtual learningware-based environments. The Journal of Systems and Information Technology. Edith Cowan University. Austrália, v. 5, n. 1, p. 93-113, 2001.

[13] Gutwin, C.; Greenberg, S. A descriptive framework of workspace awareness for real-time groupware. Journal of Computer-Supported Cooperative Work. Issue 3-4, p. 411-446, 2002.

[14] Gutwin, C. Workspace awareness in real-time distributed groupware. 1997. PhD Thesis. Department of Computer Science, University of Calgary, 1997.

[15] Omoronyia, J. Ferguson, M. Roper, and M. Wood. A Review of Awareness in Distributed Collaborative Software Engineering. Software Practice and Experience, 40 (12). November 2010. pp. 1107-1133.

[16] Storey,M. Theories, Tools and Research Methods in Program Comprehension: Past, Present and Future. Software Quality Journal, 2006.

[17] Fuks, H.; Raposo, A.; Gerosa, M.A.; Pimentel, M.; and Lucena, C.J.P. The 3C Collaboration Model. The Encyclopedia of E-Collaboration, Ned Kock (org), 2007, pp. 637-644.

[18] Fuks, H., Raposo, A., Gerosa, M.A., Pimentel, M., Filippo, D., and Lucena, C.J.P. Inter- and Intra-relations among Communication, Coordination and Cooperation. In IV Brasilian Symposium on Collaborative Systems, Rio de Janeiro – RJ. 2007, pp. 57-68. (In Portuguese).

[19] Pattison, T. and Phillips, M. View Coordination Architecture for Information Visualization. In Proceedings of the Australian Symposium on Information Visualization, 2001, Sydney, Australia. pages 165-171.

[20] M. Baldonado, A. Woodruff, and A. Kuchinsky. Guidelines for Using Multiple Views in Information Visualization. In ACM AVI 2000; Palermo, Italy. 110-119.

[21] Greenwood, P. On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study. ECOOP, Germany, 2007.

[22] Figueiredo, E. Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability. ICSE, May 2008.

[23] Fowler, M. Refactoring: Improving the Design of Existing Code. Addison Wesley, 1999.

[24] Petre, M. Mental imagery and software visualization in high-performance software development teams. J. Vis. Lang. Comput., v. 21, n. 3, p. 171-183, 2010.

[25] Ware, C. Information Visualization, Second Edition: Perception for Design (Interactive Technologies). 2. ed.Morgan Kaufmann, 2004.

[26] Lanza, M., Hattori, L., and Guzzi, A. Supporting Collaboration Awareness with Real-time Visualization of Development Activity. In Proceedings of the 14th IEEE European Conference on Software Maintenance and Reengineering (CSMR), pp. 207 - 216. IEEE CS Press, 2010.

[27] C. Treude and M.-A. Storey. Effective Communication of Software Development Knowledge Through Community Portals. In Proceedings of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE '11). ACM, New York, NY, 91-101.

[28] SourceMiner. A Multiple View Interactive Environment Implemented as an Eclipse Plug-in. Available at http://www.sourceminer.org.

 ISBN: 978-1-61208-230-1