Delft University of Technology
Software Engineering Research Group
Technical Report Series

Dynamic Analysis of Communication and Collaboration in OSS Projects

Martin Pinzger and Harald C. Gall

Report TUD-SERG-2010-017

TUD-SERG-2010-017

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
http://www.se.ewi.tudelft.nl/techreports/

For more information about the Software Engineering Research Group:
http://www.se.ewi.tudelft.nl/

Note: Accepted for publication in the book Collaborative Software Engineering, 2010, Springer.

Chapter 13
Dynamic Analysis of Communication and Collaboration in OSS Projects

Martin Pinzger and Harald C. Gall

Abstract Software repositories, such as versioning, bug reporting, and developer mailing list archives contain valuable data for analyzing the history of software projects and its dynamics. In this chapter, we focus on the analysis of the communication and collaboration in software projects and present an approach that works on software archives with social network analysis techniques. Our tool called STNA-Cockpit provides both, a meta-model to represent communication and collaboration and a graph visualization technique to interactively explore instances of the meta-model. These instances are reconstructed from CVS, Bugzilla, and mailing list data. In a case study with the Eclipse Platform Core project data we demonstrate that with STNA-Cockpit one can observe project dynamics for certain periods of time. This allows, for example, project managers to early identify communication bottlenecks, contributor and expertise networks, or to understand how newcomers can be integrated fast and efficiently into their team.

13.1 Introduction

Communication and collaboration among team members are key success factors for large, complex software projects. In addition to industry, examples of such projects can be found in the Open Source Software (OSS) community, for example, the Mozilla, Apache, Eclipse projects. OSS projects are of particular interest for communication and collaboration research because their developers rarely or never meet face-to-face.

Findings of previous research showed that OSS developers coordinate their work almost exclusively by three information spaces: the implementation space, the documentation space, and the discussion space [6]. Typically, in OSS projects a versioning system, such as, the concurrent versions system (CVS), provides the

M. Pinzger (✉)
Software Engineering Research Group, Delft University of Technology, Netherlands
e-mail: M.Pinzger@tudelft.nl

I. Mistrík et al. (eds.), *Collaborative Software Engineering*,
DOI 10.1007/978-3-642-10294-3_13, © Springer-Verlag Berlin Heidelberg 2010

backend of the implementation space. It keeps track of changes made to projected related files and corresponding versions. The World Wide Web is used as the primary documentation space. Because of the distributed and informal nature of OSS projects, discussions between project members, project associates, and users are done and tracked in mailing lists and bug reporting systems. This results in a representative data set that enables communication and collaboration analysis. The representative data in OSS projects as well as its public availability motivated us to develop the Socio-Technical Network Analysis (STNA)-Cockpit. However, our approach is not limited to OSS projects. It can also be applied in industrial settings in which such data is available.

STNA-Cockpit provides means and techniques to obtain a deeper insight into the communication and collaboration structure of software projects. In particular, we use STNA-Cockpit to address the following research questions:

- Who owns or is working on which components?
- Who are the key personalities (e.g., leading developers) in the project?
- Are there deviations in the developer contribution structure?

We address these questions by analyzing the communication and collaboration structure that is reconstructed from versioning archives (implementation space), bug tracking and mailing list archives (discussion space). We leave out the documentation space whose analysis is out of scope for this chapter.

In summary, the chapter makes three contributions, of which the first one is a meta-model for representing communication and collaboration in OSS projects. We briefly describe the set of techniques and tools for importing the data and further present the heuristics that are used by STNA-Cockpit to integrate the various data sources into the communication and collaboration network.

The STNA-Cockpit approach is our second contribution. STNA-Cockpit uses a graph-based visualization technique to analyze the communication and collaboration structure. Properties of the communication between developers and collaboration on software components are mapped to graphical attributes in the graph. This results in a number of graph structures that form *visual patterns* which indicate, for example, team organization, the key personalities in the project, or the owners of source code components. Furthermore, these patterns also indicate violations in the communication and collaboration structure, such as, isolated developers or alien commits. In addition, STNA-Cockpit provides facilities to dynamically browse the communication and collaboration network over time. It uses a sliding time window approach that allows the user to navigate back and forth in the project history. This enables the observation of changes in the communication and collaboration structure. For example, it shows how newcomers get involved in the project, or how leading developers hand over their job to their successors.

We demonstrate the benefits of our integrated meta-model and the STNA-Cockpit approach in a case study with the Eclipse Platform Core project. This is our third and last contribution. Results of the study show how STNA-Cockpit can be used to find out, for example, the roles of different developers, such as,

communicators and connectors, and how a new developer got socialized. The STNA-Cockpit approach proves useful to aid project leaders in observing and controlling the communication and collaboration structure in software projects and can provide an integral part in collaborative software engineering.

The remainder of the chapter is structured as follows: The next section presents related work of social network analysis in the software engineering domain. Section 13.3 describes the concepts for modeling communication and collaboration in OSS projects and the techniques to reconstruct them from raw data available for OSS projects. Section 13.4 introduces the STNA-Cockpit approach and its graphical language. The evaluation of STNA-Cockpit with the Eclipse Platform Core project is presented in Section 13.5. And, in Section 13.6 we draw conclusions and outline future work.

13.2 State-of-the-Art in Socio-technical Network Analysis

The public availability of project data made OSS projects to one of the most studied subjects in the software engineering research community. In [7] Karl Fogel presents a number of guidelines to manage and the technical infrastructure to run OSS projects. In the context of this chapter, the technical infrastructure of OSS projects is of particular interest. It basically consists of a versioning system, bug tracking system, and mailing lists for the communication and co-ordination of work. Communication between developers and users takes place in discussion forums and the bug tracker. Topics range from bug fixes, feature requests, to hints for the installation and usage of an application. The source code typically is managed with a versioning system, such as, CVS or subversion. They keep track of changes in the source files and project documents and are also used to mark software releases. While the "large" projects, such as, Eclipse and Mozilla provide their own infrastructure many OSS projects are hosted by development web sites, such as, SourceForge.net. Recent research results and emerging opportunities in OSS development are presented by Scacchi in [20]. We would like to refer the reader to this publication to get a deeper insight into OSS development.

The various data sources available from OSS development web sites formed the input to several studies of organization, communication, and co-ordination aspects in software projects. For example, Crowston et al. [5] used data from developer mailing lists and online forums of three active projects to analyze co-ordination mechanisms of OSS communities. The analysis is based on the Co-ordination Theory Approach framework [15]. They found similarities between OSS groups and reported practices of proprietary projects in the co-ordination mechanisms used to manage task-task dependencies. Differences were found in the co-ordination of task-actor dependencies. In particular, "self-assignment" was the most common mechanism used in OSS projects. Later on, Howison et al. [11] took a closer look at the dynamics of the social structure by applying social network analysis over time. They used data obtained from the SourceForge.net bug tracking repository. Results

of their analyses showed that most of the participants are involved in the project for only a short period while few participants are involved for longer periods.

Similar to our approach, Ducheneaut [6] analyzed the socialization of newcomers to the OSS community of Python, showing that the integration of a new member is not only depending on her technical skills but also on her ability to learn how to participate and to build an identity for that her ideas will get accepted and integrated. He combines the social network built from the mailing list archive with the material structure based on CVS log. To visualize the project's evolution he implemented the OSS Browser, which provides a dynamic view of the social network, built on the Conversation Map of Sack [18].

Sack et al. [19] continued this research field with an analysis across the three information spaces that build the socio-technical network: discussion, implementation, and documentation. They tried to answer the questions how power is distributed, how links evolve between people, and how the cognitive activity of discussions is influenced by the social and governance structures of the project. Mails, CVS logs, and enhancement proposals of the Python project served as data basis. Similarly, Bird et al. presented a study in which they analyzed the process by which people join open source projects [3]. Results support their hypotheses that the rate of immigration is non-monotonic, and that technical skill and social reputation has an impact on becoming a developer. In our approach we reuse several of the ideas presented by these approaches

Several other studies used data from OSS projects to analyze communication and co-ordination aspects. For example, Ghosh showed that many open source projects hosted at SourceForge.net are organized as self-organizing social networks [9]. Similarly, Xu et al. studied the development community at SourceForge.net and classified contributors into project leader, core developer, co-developer, and active user [21]. Huang et al. used version histories to identify core and peripheral development teams [12]. Ohira et al. used social networks and collaborative filtering to support the identification of experts across projects [17]. Lopez et al. explored statistics and social network properties of the development community at SourceForge.net to find collaborations and topological properties [14]. In particular, they found small world phenomenon and scale free behaviors and also that weakly associated but contributing co-developers and active users may be important factors in open source software development.

Network visualization is a well-researched field and there exist a number of sophisticated frameworks and tools to visualize social networks, for example, Pajek [2] or Net Draw which is an integral part of the social network analysis tool Ucinet [4]. While these tools can visualize various kinds of social networks including also socio-technical networks, none of them takes into account evolution. Similar to our approach, Ogawa et al. [16] presented a visualization technique to analyze the evolution of the communication and collaboration activities of software projects. They used data from CVS repositories and mailing list archives. The visualization is based on combining the repository view and the mailing list view via people. The repository is represented using the Windows Explorer tree visualization and the mailing lists are displayed as clusters within Sankey diagrams.

Aberdour [1] addressed the question on how to achieve OSS quality by comparing best practices of OSS development with closed-source software development. He reported that high-quality OSS relies on having a large and sustainable community that has to be fully understood by the community members. The final guidelines to high-quality OSS imply high code modularity, rapid release cycles and many bug finders. His findings on quality justify our aim at providing means for a better understanding of software project dynamics, in both open and closed source software projects.

13.3 Modeling Communication and Collaboration in OSS Projects

In this section, we outline the motivation and present the means and techniques to analyze the interactions of a software project team. The main focus is on the question about the inner life of the project that consists of people playing different roles and of the products they develop. The collaborative interaction among the project members is reflected in the organization and has an influence on the project's outcome and its environment. The social structure of a community, based on communication, is combined with collaboration information representing working teams. This integration enables to further investigate the activities going on inside the project. The developed means and techniques are based on analyses of OSS communities, but they can be adapted to commercial projects as well.

13.3.1 Communication in OSS Projects

Open source software projects typically have no formal organization and preassigned command and control structure. Team members can join projects and contribute as they wish. This demands organizational instruments to share and exchange information. Bug tracking systems, such as Bugzilla, are used to manage bug reports and development tasks. Internet mailing lists are instruments to address information to a dynamically changing community. A mailing list has a list of subscribers receiving the messages processed by the reflector address. We assume that most of the core developers of the community interact using such designated tools. This section shows how we derive a model of the communication in OSS projects from Bugzilla and mailing list data.

13.3.1.1 Deriving Communication Paths from Mail Traffic

Communication in mailing lists happens on a subject/topic between a sender and at least one receiver at a certain point in time with a given content. Discussions arising from an initial mail can be grouped into threads – mails referring to the same subject are kept together. Within mailing list threads, the messages can grow in a dendritic

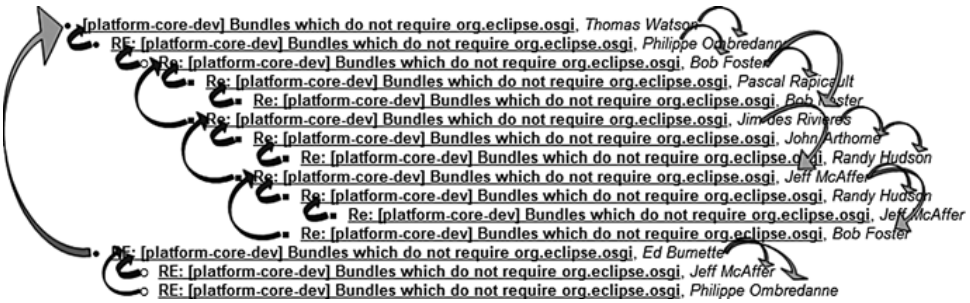


Fig. 13.1 Communication paths extracted from the Eclipse Platform Core developer mailing list

way. Figure 13.1 shows an excerpt of a mail thread from the online mailing list archive of Eclipse Platform Core.

A mail addressed to a mailing list is processed by the reflector and sent to all subscribers. This means that the *To:* address is always the mailing list address itself; hence, there is no explicit receiver address. In our example this address is *platform-core-dev*. To model the communication path between sender and receiver, the receiver needs to be reconstructed from subsequent answering mails. The identification of the sender is given by the *From:* field which is denoted by the name on the right side of a message. For determining the receivers of emails we analyze the tree structure of a mail thread and compute the *To:* and *Cc:* paths.

Figure 13.1 illustrates the two paths in our example thread whereas gray arrows denote the *To:* path and light gray arrows the *Cc:* path. A gray arrow is established between an initial mail and its replies. For example, Philippe Ombredanne is first replying to the mail of Thomas Watson, so in this case Philippe Ombredanne is the sender and Thomas Watson is the receiver of the mail. To derive *Cc:* receivers we consider the person answering a mail as an intended receiver of this mail. In case this person is already the *To:* receiver (as it applies with the mails number 3–5 between Bob Foster and Pascal Rapicault) no additional path is derived, because we assume that a mail is not sent to a person twice.

For importing the data from the mailing lists archives we extended the iQuest tool. iQuest is part of TeCFlow,[1] a set of tools to visualize the temporal evolution of communication patterns among groups of people. It contains a component to parse mailing lists and import them into a MySQL database. Our extension aims at including the *follow-up* information of mails to fully reconstruct the structure of a mail-thread. The sample thread shown above consists of 15 mails that result in 25 communication paths.

13.3.1.2 Deriving Communication Paths from Bug Reports

The second source outlined for modeling communication paths is a bug tracking repository, such as, Bugzilla. Bugzilla users create reports and comments and give

[1] http://www.ickn.org/ickndemo/

answers to former editors or commentators. Within a Bugzilla bug report, a person can play different roles: (1) the reporter that opens and describes a problem; (2) the assigned developer who takes over the ownership or current responsibility; (3) a developer on the *Cc:* list who wants to be kept informed; or (4) a person that comments on the report.

Similar to mailing lists, communication in bug reports consists of a sender, at least one receiver, a time stamp, the subject, and the content of the message. For the reconstruction of communication paths we consider two actions: creating a bug report and writing a comment. The communication emerging from report creation is the assignment of the task to the assignee by the reporter and the notification of the persons registered as *Cc:* The subject of the communication is the short description and the content is the long description of the bug report.

Comments result in further communication. Each commentator addresses their comment to the reporter, the assignee, and all former commentators. This approach differs to the one of Howison et al. [11] where only a communication to the immediate previous poster was assumed. The subject of communication is denoted by the short description and the content by the comment. Regarding communication with *Cc:* addressees, we assume that if somebody is concerned he or she will get involved as a commentator.

We use the Bugzilla importer of Evolizer [8] to obtain the bug report data from Bugzilla repositories. Given the URL of the Bugzilla repository the importer downloads the bug reports in XML format, parses them and stores the results into the Evolizer database. We next query each bug report from the database and reconstruct the communication paths as illustrated before. Regarding our example we reconstructed 36 communication paths, including three *Cc:*'s. In general, we expect more communication paths in bug reports than in mailing lists archives.

13.3.2 Collaboration in OSS Projects

To model the collaboration in a project, we need to know who is or was working on which component of the system. Versioning repositories, such as CVS, provide details about code revisions that enable to derive this information. The minimal information required is the author of the modification and the affected file. For each revision the time stamp of the CVS commit, the corresponding commit message and the extent of the file modification (number of lines added and deleted) are extracted from the CVS log. We use the Evolizer CVS importer plug-in to obtain the CVS log information from online repositories. The importer parses the CVS log of each source file and stores the extracted information into the Evolizer database.

In addition to the collaboration of developers on source files, we are interested in the ownership of source files. This enables the analysis of the interaction between the developer and the owner of a file, and, in particular, how the communication between the two proceeds. Girba et al. propose a measurement for the notion of code ownership by evaluating the CVS log [10]. They define the owner of a source file as being the developer that contributed the most code lines to it. For each source files

revision and author the difference between the number of lines added and number of lines deleted is computed. The sum of these deltas presents the contribution of a developer.

We extend the approach by Girba et al. by also taking into account the initial files size that refers to the initial contribution of the first developer. Experiments with CVS information from the Eclipse project showed that, when taking into account the initial number of lines of code, the number of owner changes is reduced by around 88%. With this we can more realistically reflect code ownership relationships.

13.3.3 Integrating Communication and Collaboration Data

The *person* is the central entity in communication and collaboration data as obtained from mailing lists, Bugzilla and CVS data. Therefore, we use the person to link the three data sources to obtain a consistent view on the communication and collaboration in software projects. The underlying data sources, however, have different approaches regarding the identification and characterization of a person. The personal information appearing within CVS logs, bug reports, or emails are the name of the person, the email address and the CVS user name. The objective of the integrated data model is to unify this person information so that each person is represented by exactly one entity in the model. Figure 13.2 depicts the meta-model to represent the integrated CVS, Bugzilla, and mailing list data.

The person entity is in the center of the model and links CVS with Bugzilla and mailing list data. The possible roles of a person are highlighted by arc labels which are author of source code contributions, owner of source files, reporter of a bug, assignee, and person on the *Cc:* list, and commenter of a bug report, and sender, receiver of an email. Furthermore, we establish a link between Issues and affected source file revisions.

In the virtual world of the Internet it is easy to create different identifiers for a single person. For CVS, Bugzilla, and mailing lists archive data this concerns the use of different email addresses. For example, *Chris McGee* uses *cbmcgee@ca.ibm.com*, *jeffl@informaldata.com* and *sirnewton_01@yahoo.ca* as his email addresses. The mapping of these addresses to a single person is a non-trivial task.

We follow a semi-automatic approach: For each person entity extracted from an email, CVS log, or bug report, the matching algorithm first looks up the email address in the database. If a person with the same email address exists, the person is assigned to the corresponding revision, issue, comment, or email. If not, the email address is analyzed to extract the person's name. Our algorithm assumes that a name consists of at least two words and that they are separated by a dot or underscore within an email address prefix. In some cases such a name cannot be derived from the prefix, because, for example, it denotes an email distributor address, an alias, or a nickname. In case the name could be extracted, the algorithm searches the corresponding person in the object model. For this our algorithm uses the Levenshtein string similarity measure [13]. If a person object with a similar name is found in the object model, the new email address is added to the list of email addresses of this

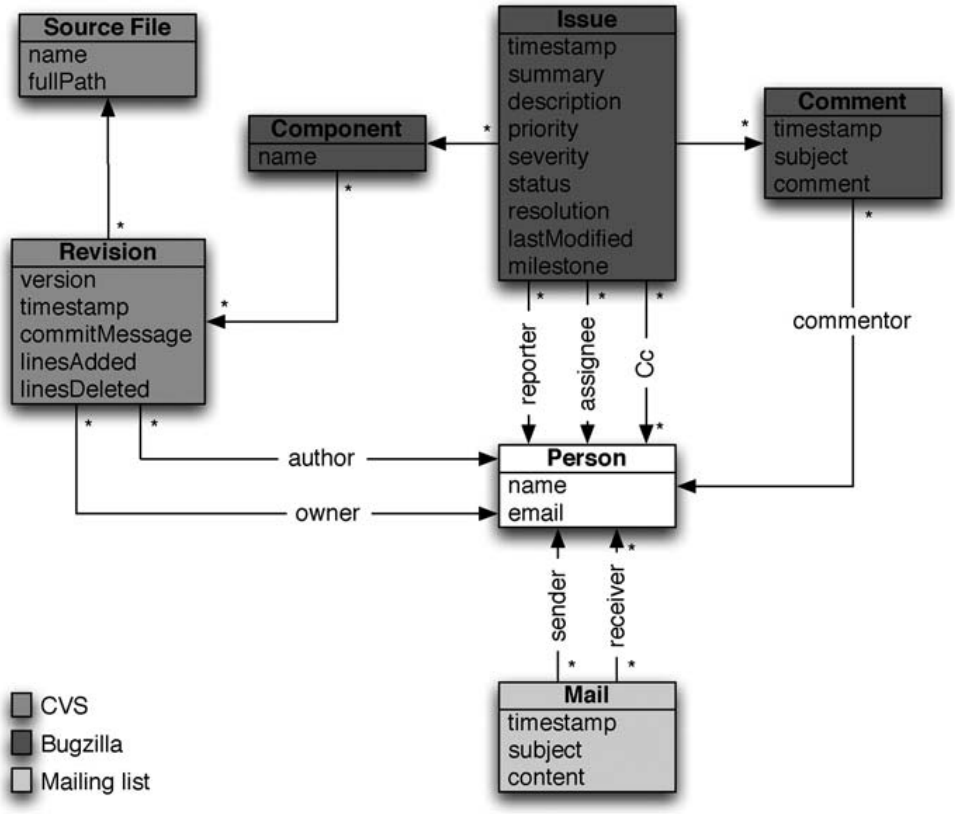


Fig. 13.2 Integrated model for representing communication and collaboration data in OSS projects

person. In every other case the person is assumed to be unknown and a new person entity is added to the database.

While this algorithm works fine for person information obtained from Bugzilla and mailing lists, there are problems with matching persons obtained from CVS log data. Typically, the author stored in CVS logs indicates the CVS user name, but not the real name of a person. Because of the high number of false matches, the mapping of these persons is done manually.

In addition to the information of a person, email addresses contain domain information that, for example, denotes the business unit of a developer. We use this information to assign developers to teams. We obtain email addresses that have been generated with *MHonArc*.[2] The problem is that MHonArc provides a spam mode which deters spam address harvesters by hiding the domain information of email addresses. For example, the email address of *Chris McGee* is displayed as

[2] http://www.mhonarc.org/

cbmcgee@xxxxxxxxx. In such cases our matching algorithm searches the alternative email addresses of a person to reconstruct the missing domain information. We furthermore do a manual inspection of the results to assure the correctness of the matching's.

13.4 STNA-Cockpit

The objective of STNA-Cockpit is to enable an understandable perception of the project's set-up and to illustrate its dynamics by exploring the evolution of the communication and collaboration structure interactively. The user can either investigate a particular period in time or move through time by shifting the observation period forward and backward. Figure 13.3 shows a sample view of a socio-technical network graph as created by STNA-Cockpit for the Eclipse Platform Core project.

Various graphical features are utilized to convey information concerning the communication and collaboration structure. Basically, nodes in the graph represent persons or work packages. Edges illustrate the communication between people or the collaborations of developers on work packages. In the following, we present the various graphical features and visual patterns used by STNA-Cockpit.

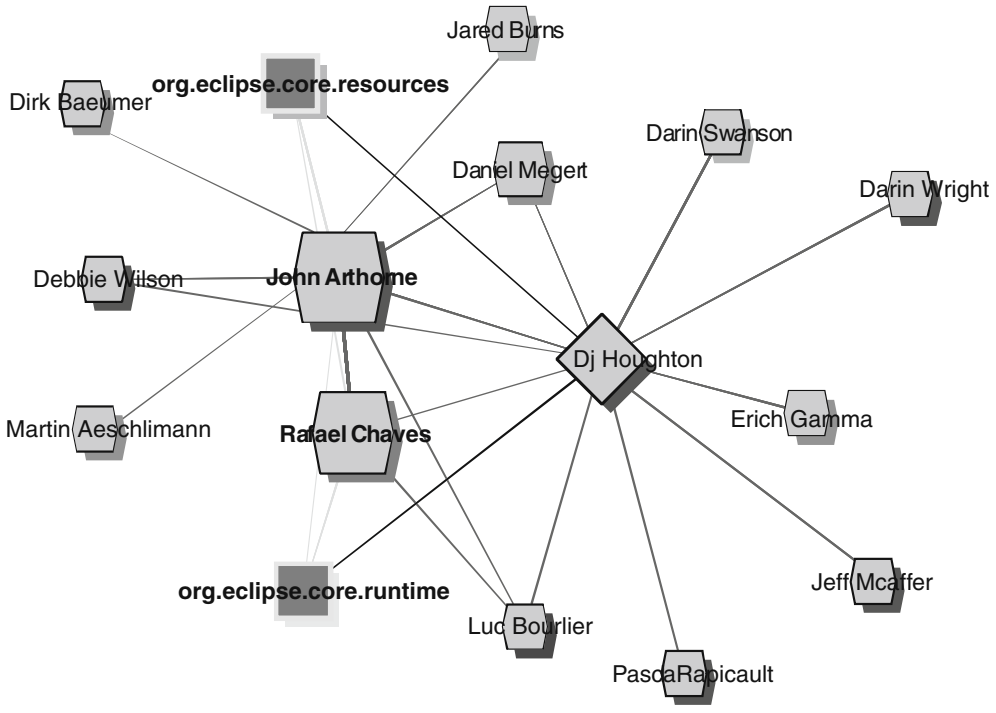


Fig. 13.3 Socio-technical network graph of the Eclipse Platform Core project

13.4.1 Actors

An actor can play different roles within the project: project member, source file owner, new source file owner. A project member is illustrated by a gray actor node and labeled with the person's name. The color of the shadow of a node illustrates the domain (i.e., business unit) to which the actor belongs. The default shape is a hexagon and the border color is always black (see Fig. 13.4a). The size of an actor node is proportional to the number of incoming and outgoing communication paths. The bigger the node is the more this actor has communicated with other team members. The owner of a source file is illustrated by shaping the node as diamond (see Fig. 13.4b).

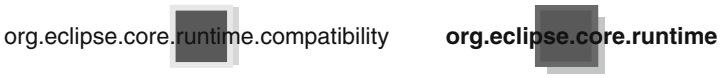
A node label with a frame indicates that the developer took over the ownership of a source file in the corresponding work package. The change of the ownership comes along with an alien commit that is represented by drawing the actor name in bold face (see Fig. 13.4c).

(a) Project member (b) Owner (c) Owner change

Fig. 13.4 Shapes and graphical features to represent actors in the STNA-Cockpit graph

13.4.2 Work Packages

A work package is illustrated by a gray rectangle. The default color of the border is light gray and the default color of the shadow is also light gray (see Fig. 13.5a). The border color indicates the number of bug reports that have been associated with committed revisions (i.e., the number of commits that contained a bug report number in the commit message). The color gradient is from light to dark gray. The darker the color the fewer commits referenced bug reports (see Fig. 13.5b). In addition, the shadow of a node indicates the total number of problems reported for the work package. The color gradient ranges also from light to dark gray, whereas dark gray indicates a work package that has been affected by many problems. Similar to actor



(a) Work package (b) Commit-problem links

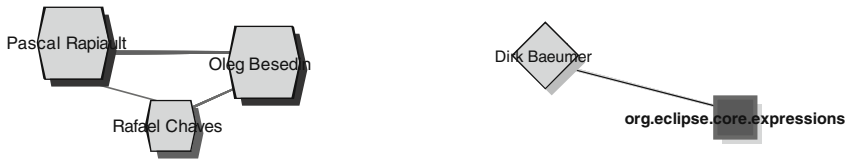
Fig. 13.5 Shapes and graphical features to represent work packages in the STNA-Cockpit graph

nodes, changes in the ownership of source files are indicated by drawing the label of the affected work package in bold face.

13.4.3 Communication and Collaboration

Ties (i.e., edges) in the graph either represent an interaction that occurs or is valid within the selected observation period. Possible interactions are: sending an email to a mailing list, opening a bug report, commenting on a bug report, committing source code changes to the versioning repository, and owning source code.

The communication between actors is colored gray. The width of an edge indicates the amount of communication between the associated actors (see Fig. 13.6a). A commit of source code changes to the versioning repository is indicated by a gray edge between the developer and the work package the modified source file belongs to (see Fig. 13.6b). Also for these edges, the width indicates the number of commits. In case of an alien commit, the font of the two node labels denoting the developer and affected work package are changed to bold face as described above. The ownership of source files contained by work packages is represented by a black edge between owners and work packages.



(a) Email and bug communication (b) Developer contribution

Fig. 13.6 Shapes and graphical features to represent communication and collaboration in the STNA-Cockpit graph

Applying these patterns to the network graph of Fig. 13.3 we can see that most of the communication has been between John Arthorne, Dj Houghton, and Rafael Chaves. The represented communication was on bug reports solely. During the selected observation period these three authors committed changes to source files contained by the two packages *org.eclipse.core.resources* and *org.eclipse.core.runtime*. Dj Houghton owns source files in these packages while John Arthorne and Rafael Chaves performed alien commits.

13.5 Communication and Collaboration in the Eclipse Project

In this section we demonstrate STNA-Cockpit by applying it to the Eclipse Platform Core project data. In particular, we demonstrate how STNA-Cockpit can be used to answer the following questions:

- Who owns or is working on which source code package?
- Who are the key personalities in the Eclipse Platform Core project?
- Can we identify shortcomings in the communication and collaboration structure in the project, meaning alien commits?

We use the Eclipse Platform Core project as an example to illustrate the benefits of our integrated data model and the STNA-Cockpit approach. Analysis results are interpreted in the context of this project and should not be generalized. The following section briefly outlines the Eclipse Platform Core project and the data sources we used in the case study.

13.5.1 The Eclipse Platform Core Project

Eclipse.org is an open source community whose projects are focused on building an integrated and extensible development platform. The Eclipse Project is the top-level project dedicated to providing a robust, full-featured, commercial-quality, and freely available platform for the development of integrated tools. In this case study, we focus on the Eclipse Platform Core component that is a main component of the Eclipse Platform project. In January 2007, the Eclipse Platform project comprised 18 mailing lists, 34 different classified Bugzilla components and more than 350 plug-ins. To know which part of the source code is affected by a discussion within an email or bug report the different data sources had to be mapped. Table 13.1 shows the set of the plug-ins, mailing lists, and Bugzilla components that concern Eclipse Platform Core. The mapping was obtained from the Eclipse Platform Core project website.[3]

In total the source code of Eclipse Platform Core component consists of 17 plug-ins. Communication between the developers of the component takes place in the mailing list *platform-core-dev*. In Bugzilla, two components were used to report problems and enter change requests for the Eclipse Platform Core project. In a first

Table 13.1 Plug-in sources, mailing list, and Bugzilla components of the Eclipse Platform Core project

Name	Plug-ins	Mailing list	Bugzilla
Platform.Core	org.eclipse.core.contenttype org.eclipse.core.expressions org.eclipse.core.filesystem.* org.eclipse.core.jobs org.eclipse.core.resources.* org.eclipse.core.runtime.* org.eclipse.core.variables	platform-core-dev	Platform.Runtime Platform.Resources

[3] http://www.eclipse.org/eclipse/platform-core/

step, we retrieved the CVS, Bugzilla, and mailing list data of mentioned data sources up to November 2006. From CVS we retrieved 7,479 change log entries from 997 source files. 7,907 bug reports have been imported from Bugzilla and 102 emails were retrieved from the *platform-core-dev* mailing list.

After importing the data, the Evolizer database contained 2,581 persons, 101 email and 11,081 Bugzilla communication paths. 2,536 person entities were imported from the Bugzilla data, 132 are mailing list users of which 73 have been matched to Bugzilla users. Contributions to the source code were from 27 developers. All of them have been mapped to Bugzilla users. Because we were mainly interested in the communication and collaboration of Eclipse developers, we concentrated our analysis on the 27 developers. While all these developers participated in Bugzilla reporting; only 14 of them wrote emails to the mailing list.

In the following we show a number of applications of STNA-Cockpit and the benefits of our integrated data model and visualization approach.

13.5.2 Ownership and Alien Commits

Assume a scenario in which a project manager wants to find out that owns or is working on which plug-in of the Eclipse Platform Core project, and whether there have been violations in the developer contribution structure. For this, the project manager selects the observation period and has STNA-Cockpit draw the collaboration graph that represents only the CVS information. We did this for the Eclipse Platform Core project for the time period from 14th to 28th February 2005 and obtained the collaboration graph depicted in Fig. 13.7.

The black edges in the graph denote the ownership of source files at that time. For example, the graph shows that Jeff Mcaffer, Pascal Rapicault, and Dj Houghton are the owners of source files of the *org.eclipse.core.runtime.compatibility* plug-in. In general, the graph shows several owners of source files per plug-in. Most interesting, however, is that John Arthorne contributed to this plug-ins, though; he is not an owner of source files of any of this plug-ins. All his contributions were so called alien commits that are indicated by the bold labels of the nodes representing John Arthorne and the modified plug-ins. The dark gray border of work packages further indicates that almost zero of the commits reference a Bugzilla bug report. Moreover, the shadows of two rectangles are painted in dark gray indicating that the two corresponding plug-ins were affected by a high number of problems. In summary, such a view provides the project manager with an overview about the commit and bug reporting activities within the selected observation period. Alien commits might indicate shortcomings in the code or team organization, depending on whether or not strict code ownership has been followed in a project.

13.5.3 Communicators

STNA-Cockpit can aid project managers in identifying the key personalities in her project. The *communicator* is such a key personality who knows where the

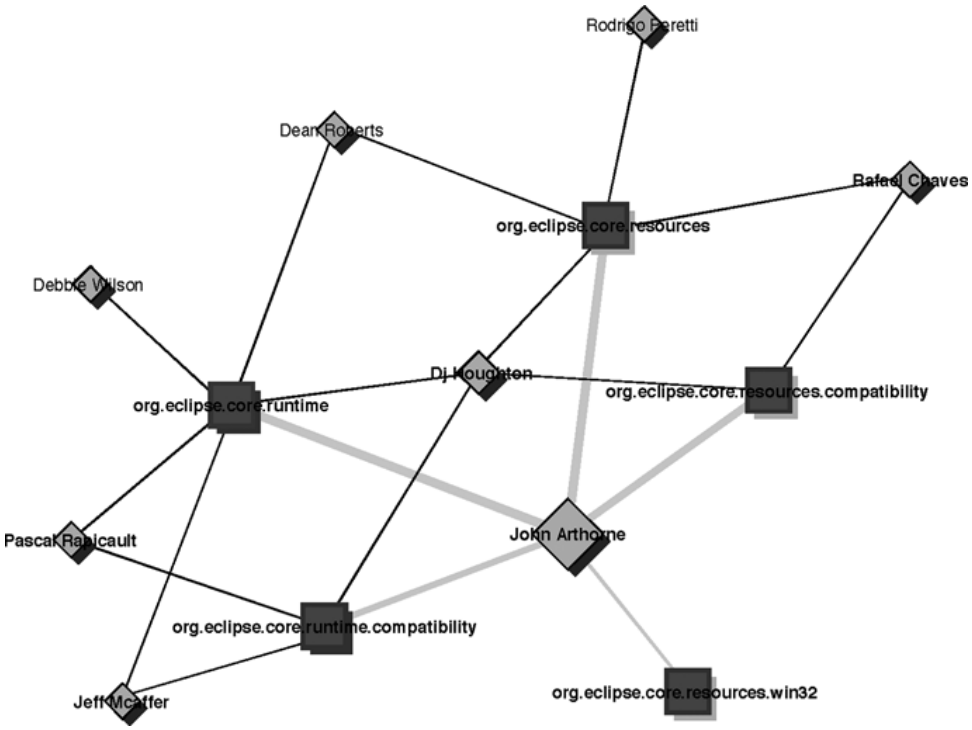


Fig. 13.7 Collaboration in the Eclipse Platform Core project observed in the time from 14th to 28th February 2005

information ideally gets processed. Figure 13.8 illustrates the communication via the developer mailing list and Bugzilla data over 21 months. The amount of communication (i.e., the number of communication paths reconstructed from bug reports and emails) is illustrated by the width of edges. The wider the edges of a person's node are, the more this person communicated with other developers.

The graph in Fig. 13.8 shows the core development team whose members frequently communicate with each other. Rafael Chaves, Dj Houghton, Jeff Mcaffer Thomas Watson, John Arthorne, and Pascal Rapicault form the core team. They are the communicators who keep the network together and play an important role within the project. Interesting is that they all belong to either the group *@ca.ibm.com* or *@us.ibm.com* as indicated by the shadows of rectangles representing these developers. Another highly connected group is formed by the Swiss team (*@ch.ibm.ch*) whose members are represented by the nodes on the right side of the graph. Almost each developer of the Swiss team is in touch with the US team; however, Markus Keller and Daniel Megert turn out as the main communicators between the two teams during that time.

Another interesting finding concerns the environment via which the developers communicated. Most of the communication was via Bugzilla bug reports indicated by the gray edges. Only the core team also used the mailing list to discuss Eclipse

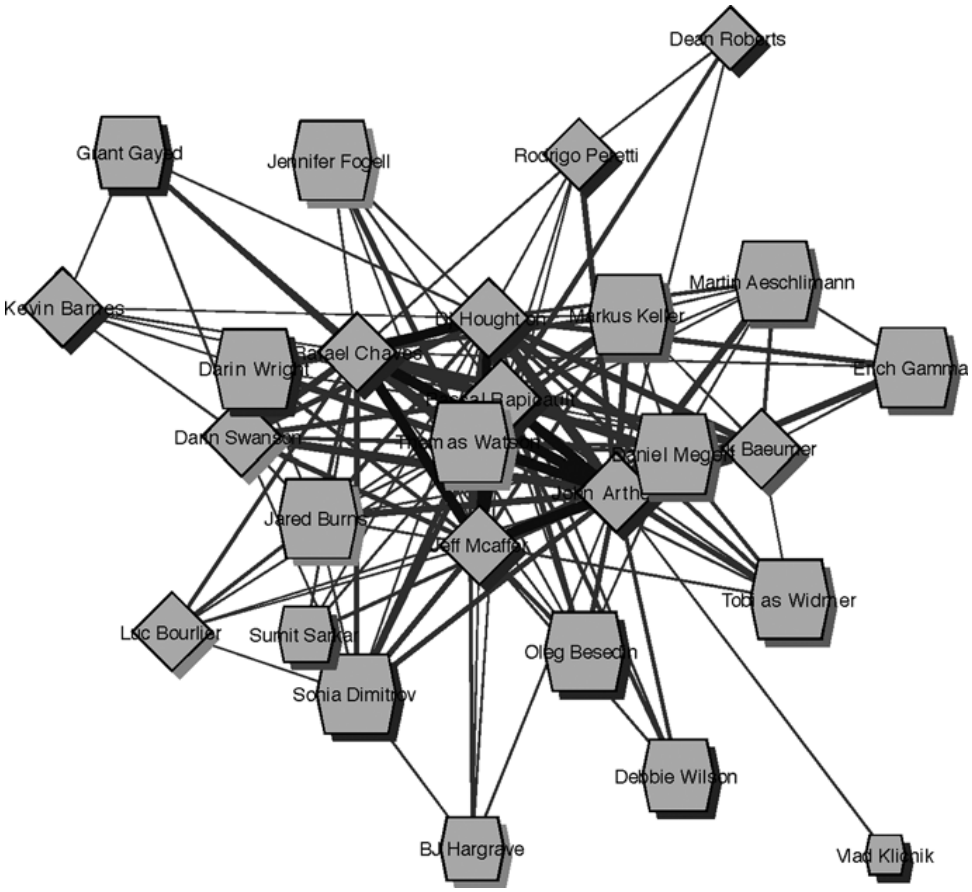
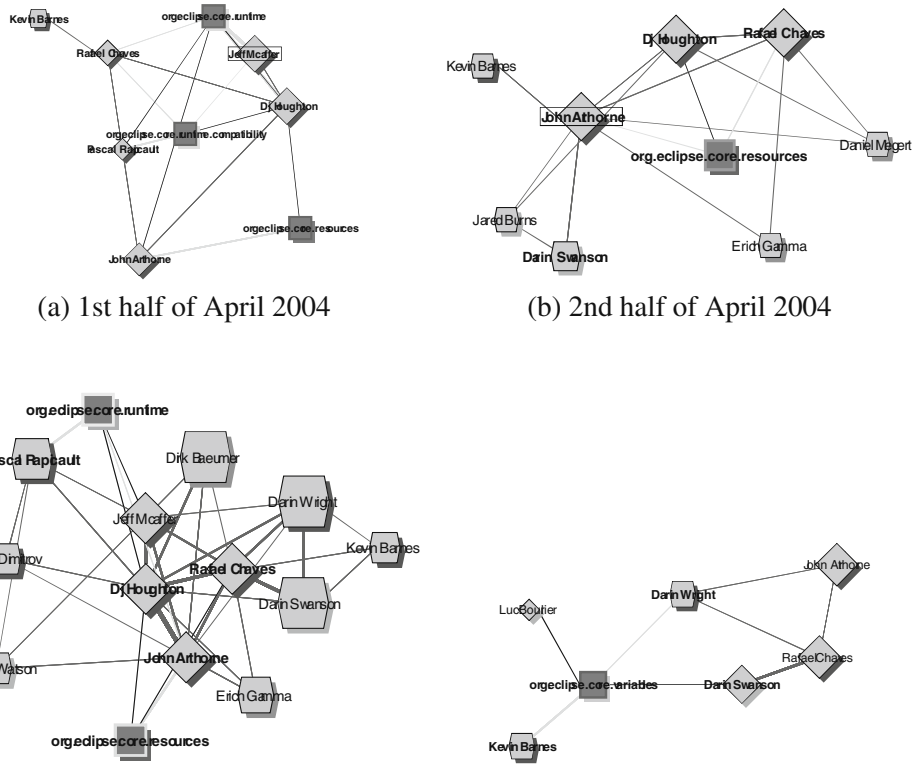


Fig. 13.8 Communicators of the Eclipse Platform Core project as from May 2004 to February 2006

Platform Core relevant issues. Such findings are of particular interest when new ways of communication are considered.

13.5.4 Project Dynamics

Newcomers should be integrated fast into development teams to rapidly increase productivity and foster synergy among team members. With STNA-Cockpit the project manager can observe how newcomers actually are integrated into their teams. For this, the project manager selects the starting observation period and uses the time-navigation facility of STNA-Cockpit to investigate the evolution of the communication and collaboration network over time. The graph animation allows the project manager to observe how the newcomer behaves concerning communication and collaboration with other team members. In particular, she looks for communication paths that tell her the newcomer gets actively involved



(a) 1st half of April 2004 (b) 2nd half of April 2004

(c) 1st half of June 2004 (d) 2nd half of June 2004

Fig. 13.9 Socialization of Kevin Barness in the Eclipse Platform Core project

in discussions on the developers mailing lists and bug reports. In addition, she observes whether the newcomer contributes to the plug-ins and finally takes over responsibility of portions of the source code.

Consider the following scenario in which Kevin Barness is entering the US team *@ca.ibm.com* of the Eclipse Platform Core project in April 2004. Figure 13.9 depicts various snapshots taken from the network created for subsequent points in time. Kevin Barness is starting as a developer in the Eclipse Platform Core team at the beginning of April 2004. His first action is to get in touch with some key personalities of the project, namely Rafael Chaves and John Arthorne. His first contacts are visualized by the graphs depicted by (Fig. 13.9a, b). In the following weeks he communicates also with other project members to get more involved into the project (see Fig. 13.9c), namely Darin Wright and Darin Swanson. As (Fig. 13.9d) illustrates, Darin Wright is a developer and Darin Swanson the owner of the files that are going to be modified by Kevin. Rafael Chaves seems to play the role of the connector who introduces the new developer Kevin Barness to the responsible persons. According to the graph, he is communicating with two senior developers.

Another example of project dynamics concerns members leaving the project. This is particularly interesting for OSS projects in which there exists no official commitment or contract between members and the OSS project. In general, members are free to join and leave an OSS project. STNA-Cockpit can be used to early recognize such situations by observing the developer's contributions to mailing list forums, bug report discussions, and source code. Knowing such changes in advance helps the project leaders to take proper actions, such as, to find active members to take over the work of the leaving person.

13.5.5 Summary

We demonstrated the application of STNA-Cockpit to analyze the communication and collaboration structure of the Eclipse Platform Core project. Concerning collaboration we showed how the tool can be used to find out which developers have worked on which plug-ins during a selected observation period. Violations, in particular, alien commits to plug-ins, were highlighted. The visualization of the communication structure allows the project manager to observe the roles of developers in her project. For example, in the Eclipse Platform Core project, we identified the communicators that represent the developers most active on mailing lists and Bugzilla. They represent the right information source to obtain status reports and also to get newcomers involved into the project. The sliding time window approach of STNA-Cockpit was used to find out about project dynamics. For example, we found newcomers joining the Eclipse Platform Core project, as well as, developers leaving the project. These findings underline the value of our integrated communication and collaboration data model and visualization techniques as implemented by STNA-Cockpit.

13.6 Conclusions and Future Work

Software repositories, such as versioning, bug reporting, and developer mailing lists contain valuable data for analyzing the communication and collaboration structure of software projects. We presented a meta-model to represent communication and collaboration in OSS project and showed how an instance of such a model can be obtained from CVS, Bugzilla, and mailing list data. We also introduced our STNA-Cockpit tool to interactively explore the integrated model by means of graph visualizations. With this tool the user can observe project dynamics in a software project at any point in time and over time using the data provided by Evolizer.

Getting awareness of communication and collaboration in a project can be very valuable for the project manager: (1) understanding how newcomers can be integrated fast and efficient; (2) knowing the key contributors and communicators in the different teams; and (3) being able to replace or add expertise holders in project phases and in software parts. Of course all this requires the data to be available in a

processable form but the key issue for that is the identity management. As long as email and Bugzilla identities can be matched to people such an analysis can work mostly automated. Such mapping data for identities should be easy to keep up-to-date and then can be fed into tools, such as, STNA-Cockpit that then can compute the communication and collaboration network of a project automatically. The time-window browsing further allows one to zoom into particular phases of the project and learn about collaboration patterns of developers.

Still, there are limitations of the current approach that are due two facts. First, in many OSS projects such identity mapping data does not exist and has to be reconstructed with quite some manual effort. Second, the analysis of collaboration patterns is not yet reflected on software releases, features or software phases, such as, testing or refactoring. But from our analysis we have seen a great potential of investigating communication and collaboration data for project steering ranging from the role of a developer to the role of a project manager.

References

1. Aberdour M (2007) Achieving quality in open source software. IEEE Software 24: 58–64.
2. Batagelj V, Mrvar A (2003) Pajek – Analysis and visualization of large networks. Graph Drawing Software, Springer, pp. 77–103.
3. Bird C, Gourley A, Devanbu P, Swaminathan A, Hsu G (2007) Open borders? Immigration in open source projects. Proceedings of the International Workshop on Mining Software Repositories, IEEE Computer Society Press.
4. Borgatti SP, Everett MG, Freeman LC (2002) Ucinet for Windows: Software for Social Network Analysis. Harvard, MA: Analytic Technologie.
5. Crowston K, Wei K, Li Q, Eseryel UY, Howison J (2005) Co-ordination of free/libre open source software development. Proceedings of the International Conference on Information Systems, Association for Information Systems, pp. 181–193.
6. Ducheneaut N (2005) Socialization in an open source software community: A Socio-technical analysis. Computer Supported Cooperative Work 14: 323–368.
7. Fogel K (2005) Producing Open Source Software: How to Run a Successful Free Software Project. Sebastopol, CA: O'Reilly Media.
8. Gall HC, Fluri B, Pinzger M (2009) Change analysis with evolizer and change distiller. IEEE Software 26: 26–33.
9. Ghosh RA (2003) Clustering and dependencies in free/open source software development: Methodology and tools. First Monday 8(4).
10. Girba T, Kuhn A, Seeberger M, Ducasse S (2005) How developers drive software evolution. Proceedings of the International Workshop on Principles of Software Evolution, IEEE Computer Society Press, pp. 113–122.
11. Howison J, Inoue K, Crowston K (2006) Social dynamics of free and open source team communication. Proceedings of the International Conference on Open Source Software, Boston, Springer, pp. 319–330.
12. Huang SK, Min Liu K (2005) Mining version histories to verify the learning process of legitimate peripheral participants. Proceedings of the International Workshop on Mining Software Repositories, ACM Press.
13. Levenshtein VI (1966) Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics Doklady 10: 701–710.
14. Lopez-Fernandez L, Robles G, Gonzalez-Barahona JM (2004) Applying social network analysis to the information in cvs repositories. Proceedings of the International Workshop on Mining Software Repositories.

15. Malone TW, Crowston K (1994) The interdisciplinary study of co-ordination. ACM Computing Surveys 26: 87–119.
16. Ogawa M, Ma KL, Bird C, Devanbu P, Gourley A (2007) Visualizing social interaction in open source software projects. Proceedings of the International Asia-Pacific Symposium on Visualization, IEEE Computer Society Press, pp. 25–32.
17. Ohira M, Ohsugi N, Ohoka T, Matsumoto K (2005) Accelerating cross project knowledge collaboration using collaborative filtering and social networks. Proceedings of the International Workshop on Mining Software Repositories, ACM Press.
18. Sack W (2001) Conversation map: An interface for very large-scale conversations. Journal of Management Information Systems 17: 73–92.
19. Sack W, Detienne F, Ducheneaut N, Burkhardt JM, Mahendran D, Barcellini F (2006) A methodological framework for socio-cognitive analysis of collaborative design of open source software. Computer Supported Co-operative Work 15: 229–250.
20. Scacchi W (2007) Free/open source software development. Proceedings of the Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering, ACM Press, pp. 459–468.
21. Xu J, Gao Y, Christley S, Madey G (2005) A topological analysis of the open source software development community. Proceedings of the Proceedings of the Annual Hawaii International Conference on System Sciences, IEEE Computer Society Press.

TUD-SERG-2010-017
ISSN 1872-5392