Tesseract: Interactive Visual Exploration of Socio-Technical Relationships in Software Development

Anita Sarma, Larry Maccherone, Patrick Wagstrom, and James Herbsleb
Institute for Software Research
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213 USA
{asarma, LMaccherone, pwagstro, herbsleb}@cmu.edu

Abstract

Software developers have long known that project success requires a robust understanding of both technical and social linkages. However, research has largely considered these independently. Research on networks of technical artifacts focuses on techniques like code analysis or mining project archives. Social network analysis has been used to capture information about relations among people. Yet, each type of information is often far more useful when combined, as when the "goodness" of social networks is judged by the patterns of dependencies in the technical artifacts. To bring such information together, we have developed Tesseract, an interactive exploratory environment that utilizes cross-linked displays to visualize the myriad relationships between artifacts, developers, bugs, and communications. We evaluated Tesseract by (1) demonstrating its feasibility with GNOME project data (2) assessing its usability via informal user evaluations, and (3) verifying its suitability for the open source community via semi-structured interviews.

1. Introduction

Development environments increasingly reflect the fact that artifacts, developers, and tasks are intrinsically bound together in a software project. While editing a file of source code, for example, many other artifacts are likely to be relevant. Considerable research effort has focused on using a variety of techniques such as static code analyses [1, 2], task definition [20], text analysis [7], and records of prior developer activity to identify these related artifacts [10] and make them easily accessible when they are likely to be useful.

There is also an increasing interest in understanding and using relationships among individuals in a team to improve software development. Research has focused on increasing awareness among developers about each other's relevant activities [17], and on using the social relations among developers to identify implicit teams [4] or to predict software defects [21]. Such efforts often draw on social network analysis (SNA).

So far, these two streams of research have mostly been separate from each other. Yet, both of these sets of relationships – the technical and the social – become much more useful when they are considered together. It is difficult, for example, to judge whether a given pattern of communication is adaptive or dysfunctional without understanding the dependencies in the tasks being undertaken by the communicators. For instance, developers who are modifying interdependent code modules but not communicating may introduce potential future integration problems. Research has shown that development work proceeds more efficiently when patterns of communication match the logical dependencies in the code that is being modified [6].

This match between the networks of artifacts and the networks of people, has a long history [19, 23], but has only recently become a focus of research in software engineering. Understanding and using analysis showing the degree of match, or congruence, between the social and technical aspects of a project is vital for supporting collaboration and coordination on software projects [5, 6]. While developers are intuitively aware of this, and software architects actively engage in social engineering while creating architectural design [15], we have relatively few tools or practices that provide socio-technical information in useful, actionable ways.

The need for such tools is reflected by findings from field studies, which have shown that developers find it difficult to decipher how their work binds them with that of others. Consequently, they spend a significant portion of their time in managing their changes [9, 24] or in finding the right person with whom to communi-

ICSE'09, May 16–24, 2009, Vancouver, Canada
978-1-4244-3452-7/09/$25.00 © 2009 IEEE

Figure 1. Tesseract UI showing four displays: (a) project activity pane with code commits (top) and communication (bottom), (b) file network, (c) developer communication network, and (d) issues pane.

cate [16]. Our own semi-structured interviews of open source developers confirm this need and have guided the design of Tesseract, a socio-technical dependency browser that is specifically constructed to:

- Simultaneously show the social as well as technical relationships among different project entities (e.g., developers, communication, code, and bugs).
- Cross-link and enable interactive exploration of these relationships and how they evolve.
- Highlight (mis)matches among technical dependencies and communication patterns of developers.

More specifically, Tesseract analyzes different project archives to determine the numerous socio-technical relationships in a project. It then visualizes these relationships via four juxtaposed displays that are cross-linked to enable in-depth, multi-perspective investigation for a user-selected time period (see Figure 1).

Constructing a system like Tesseract raises several questions such as, which dependency analysis best reflects technical dependencies (code-based analysis or files that are frequently committed together), which project entities really bind the social and technical aspects of a project, which social network analyses work best for clustering data, scalability, and the general effectiveness of the tool in helping developers understand socio-technical dependencies in a project. The aim of this paper is not to provide an absolute answer to all of the questions, but to serve as an initial investigation into the feasibility of creating a project exploration tool that focuses on the socio-technical space.

To do so, we fist demonstrated the feasibility of our approach by using Tesseract to analyze and visualize a large open source project archive. We then conducted formative evaluations of Tesseract through informal user evaluations that illustrated the ease of use afforded by the tool. Finally, we conducted interviews with open source developers, where we demonstrated Tesseract and obtained their feedback on its features and development scenarios where the tool would be useful.

The remainder of this paper is organized as follows. Section 2 motivates the design of Tesseract by building on case studies that reveal the need of an exploratory environment like Tesseract and lessons learned from existing tools. In Section 3, we present Tesseract followed by two usage scenarios. We end the section with a description of our design principles and implementation techniques. Section 4 discusses our formative evaluations of Tesseract and we conclude in Section 5 discussing potential future work.

2. Informing the design of Tesseract

We informed the design of Tesseract by building on: (1) case studies that highlight problems of manag-

ing change and communication in software teams, and (2) lessons learned from coordination tools that attempt to address these problems. Our interviews with open source developers helped us better understand the nature of these problems and identify tool features that would be useful in their development environment.

2.1. Problems of change and communication

We motivate our work through four case studies. The first study by de Souza and Redmiles [9] observed two software teams to understand how they managed the impact of their changes. The authors observed that one team was highly disciplined and managed change by: (1) rigorously broadcasting email of impending changes and (2) reading email to create an awareness of "who was working on what". While this practice worked well, it involved significant effort in writing and reading email. The other team was larger and less disciplined when informing others of changes. This resulted in team members having much difficulty in identifying the impact of their changes on others and vice versa. Further, the use of multiple databases for maintaining development and communication records caused duplication of information and overwhelmed the users.

The second study by Cataldo et al. [6] analyzed project data of a large software development team to understand the coordination needs arising because of underlying technical dependencies in work. They compared social networks created because of technical dependencies among artifacts (coordination requirements) with the communication patterns in the team. It was found that teams with high congruence (match between the coordination requirements and communication patterns) took less time to complete tasks. They also found that the coordination requirements evolved over time, requiring developers to correctly identify the new relationships and people with whom to communicate [5].

The third study by Sosa et al. [27] uses a similar technique as Cataldo et al. to find the degree to which team interactions match the way components are coupled because of shared interfaces (alignment matrix). They found that developers and engineers were unlikely to be aware of interface changes, especially when such changes occurred across system or organizational boundaries. Their recommendation to overcome these problems is that managers create an alignment matrix to align their team interactions with the way components are associated via shared interfaces.

Finally, Gutwin et al. [18] investigated collaboration mechanisms in open source development. They found that text communications (mailing list and IRC chat) were the primary mechanisms for maintaining awareness of "who is doing what" and expertise location. The community was sufficiently disciplined to generate and maintain public communication archives, thereby enabling all developers on the lists to become peripheral participants in each others' conversation. However, developers often found it difficult to remain up-to-date with all communications in the different lists. Further, splitting communication among email, chat, and issue trackers caused information duplication and situations where developers missed important information.

These studies reflect the different needs faced by developers and managers. Further, the work by Gutwin et al. shows that these needs are similar between commercial and open source projects. We developed Tesseract, an interactive exploratory environment to enable developers to investigate the complex relationships among code, developers, communication, and bugs and, thereby, help them align their interactions with the technical dependencies in their work.

2.2. Current tool support

There exists considerable tool support for analyzing technical dependencies in a project via code analysis techniques [2]. Similarly, there has been extensive work on capturing the social structure of an organization based on the interactions among individuals [28]. While each analysis type provides important information about the project, it can be much more useful when combined together. Emerging research following this premise has produced a suite of tools. For instance, tools in the computer-supported cooperative work community attempt to help developers become aware of ongoing project activities as they occur [17]. The hypothesis is that developers can better coordinate their work with that of others' by leveraging the contextual awareness of ongoing changes in the project [3, 12]. Tools such as Tukan [26], CollabVS [11], and Palantír [25] perform basic code analysis to identify dependencies among artifacts and warn developers when these dependencies are changed by another.

Project exploration tools follow a similar approach, but have slightly different goals. Many of these tools use the socio-technical relationship model created from code contributions, visitations, or dependencies to help a developer with their work. Expertise Browser [22], for instance, uses information about developers' past code contributions (both frequency and lines-of-code committed) to determine their expertise. However, it was noted that Expertise Browser was often used for other purposes such as obtaining an overview of pro-

ject activities or finding recent changes to a particular code module. Team Tracks [10] attempts to familiarize new developers with the code base by flagging parts of code that have been frequently visited (signifying their importance) and parts that are visited in succession (depicting their logical coupling). Ariadne [8] uses call graph analysis to identify artifacts dependencies, which are combined with information of code contributions to model socio-technical relations in a project.

In summary, current tools use different techniques to decipher code relationships, information of which is then used to facilitate social interactions in a team. However, these tools typically: (1) use a single data source – primarily the code archive, (2) do not compare or contrast the model created by technical dependencies with communication patterns, and (3) do not allow interactive exploration of the underlying socio-technical model. Further, the Expertise Browser study indicates that developers and managers are eager to investigate the different relationships existing in their projects to better inform their work.

To the best of our knowledge, CodeSaw [13] is the only tool that considers communication records in addition to code archives. Both of which are displayed juxtaposed in a time series. While such a display helps a user link peaks in contributions with spikes in email communications (or lack thereof) to discern development practices, it fails to map social interactions with specific technical dependencies.

3. Tesseract

Tesseract provides an interactive exploratory environment which developers can investigate relationships across code, bugs, email and bug discussions, and developers' contributions. It consists of four cross-linked panes:

- The *Project activity* pane (Figure 1(a)) displays the overall activities in a project (code commits at the top and communication at the bottom) as a time series display. It allows users to select a time period for their investigation, which is reflected in all other panes.
- The *Files network* pane (Figure 1(b)) displays artifact associations as a network, which is created by linking files that are frequently changed together. The thickness of the edges indicates the number of times pairs of files have been committed together and can be thresholded by the user. Textual listing of the file names allows quick identification of specific files by name.
- The *Developers network* pane (Figure 1(c)) displays developers and links among them. Two developers are linked if they either edited the same artifact or interdependent artifacts. The edges in this network are colored green when there exists a requirement to communicate and developers have done so via either email or the bug repository (e.g., comments or activities in Bugzilla); otherwise the edges are colored red. The thickness of the edges is based on the number of times developers communicated. Similar to the file network, a textual listing of the developer names is provided.
- The *Issues* pane (Figure 1(d)) displays defect or feature related information as a stacked area chart, as well as, in a detailed listing.

Tesseract enables exploration of project data and its underlying relationships in multiple ways. First, clicking on an entity (graph element or line in the textual lists) will highlight that entity and will also show all related entities in the other panes. For example, selecting a node in the developer network highlights all the files and bugs with which the developer has been involved during the given time frame. Second, hovering over a node in either the '*Files*' or '*Developers*' network, displays additional information about the node and highlights its neighbors. Third, a user can pan (background drag), zoom (wheel), and move individual nodes in the graph. Fourth, search functionality allows users to quickly find an entity when they know its full or partial details. Finally, Tesseract allows users to change the perspective of their investigation by drilling down on specific artifact(s) and developer(s). For instance, a user might drill-down to view a subset of artifacts to find which other developers have edited them.

3.1. Usage Scenarios

Tesseract is designed to facilitate the investigation of a particular event or identification of development patterns. Here we present two example scenarios of each type. The first scenario reflects a common practice by new developers and was mentioned multiple times in our interviews. The second scenario presents an interesting insight into the research hypothesis on congruence [5]. Both examples reflect real project data from a GNOME project with anonimyzed developer names.

Investigating an event:
Interactive exploration of the underlying socio-technical space in a project allows a developer to draw upon their (possibly incomplete or incorrect) memory and either confirm it, refute it, or supplement it with adaptive analysis of only the portions of the data set that they consider relevant.

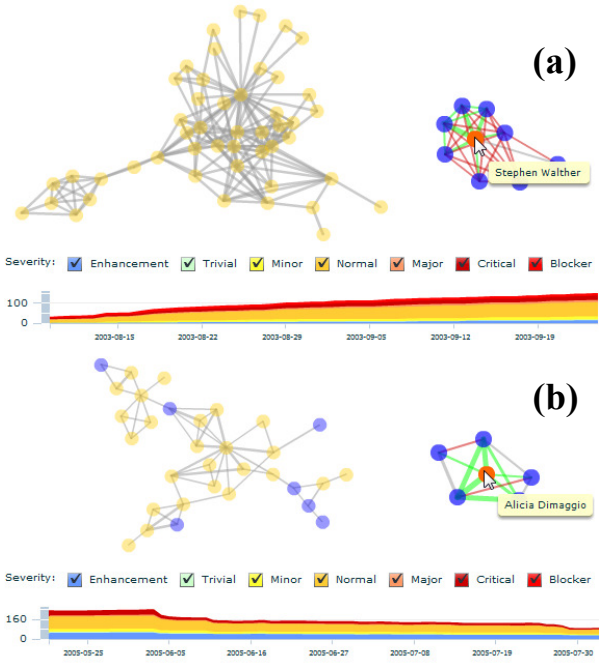

Figure 2. Contrasting development patterns.

Assume a hypothetical case where Billy Mick, a new developer, is assigned to fix a particular bug regarding the display of "minutes remaining" in a playlist. From reading the mailing lists, he remembers that Glenda Whyte had worked on a related feature – adding "minutes" to the product. He decides to investigate that feature to obtain a better understanding of the files and people that were involved. To do so, Billy, changes the time frame in Tesseract (Figure 1(a)) to a time of an earlier activity spurt. He finds the feature that Glenda had added (Bug ID 9028 in Figure 1 (d)) and selects it. This highlights the two developers and four artifacts that are associated with that bug id (shown in yellow in Figure 1 (b) & (c)).

He realizes that his bug fix would at least involve these four files. Additionally, he notices that another developer – Lynda Finney, who is not part of the core group – actually worked on three of those files in relation to this feature. Because of the red line between Glenda and Lynda, Billy realizes that these developers have not communicated with each other in the selected time frame. He supposes that Alicia Dimaggio, the current project lead, may have mediated between Glenda and Lynda, as both developers have green edges with Alicia. To ensure that he has a complete picture of development activities regarding the feature, Billy decides to contact Lynda.

Deciphering patterns:

Figure 2 provides two snapshots of project history, each presenting the file network, developer network, and issues data during two distinct time periods. These periods were chosen because they contained high bursts of activities. We can make the following observations from Figure 2(a), which shows the earlier activity burst: (1) Stephen Walther is the primary contributor having changed literally every file; (2) while Stephen is in contact with most other developers (green lines from Stephen to other developers), very few developers are communicating among themselves (red lines); (3) the file network is densely connected; and (4) this time period is marked with a continuously increasing list of open issues.

When we investigate the second time period, as shown in Figure 2(b), we find that: (1) Alicia Di-Maggio is now the primary contributor; (2) she is communicating with other contributors and there is sufficient communications among the other core contributors; (3) the file network is sparse and displays a discernable structure; and (4) the list of open issues is decreasing.

These two contrasting patterns do not necessarily imply any causal relationships between communication patterns and/or a denser file network and/or an increase in open issues, but certainly provide interesting insights into the project that merits further investigation. Readers can investigate these scenarios further through our tool that is available at http://crc.maccherone.com/ tesseract/.

3.2. Information flow

Figure 3 presents the information flow underlying Tesseract. We have specifically designed Tesseract to separate the data collection and extraction from the analysis and visualization. The former functionalities are carried out at the server side, while the latter are part of a rich web client. Designing Tesseract as a web application removes the need for installing any software on the client side, which makes it easy for managers to quickly use the tool, as well as, making it feasible for adoption by the open source community.

Collecting: Best practices for most open source and distributed development projects use three major tools to manage software development: a source code management system (SCM), one or more project mailing lists, and a common bug or issue tracking database. Most activities involving code and issues are archived by either SCM systems or issue trackers. Open source projects typically maintain and make publicly available a rich history of their communication records. Tesseract relies upon such prior collection of project data. We note that in commercial software development, project mailing list or other communication records

may not be available. In such conditions, the developer networks will either be partial or unavailable.

Extracting and Cross-linking: Different projects use different systems for their code and bug archival. For example, a project may use CVS instead of Perforce as their configuration management system or may use Trac instead of Bugzilla for their issue/defect tracking. In order to ensure that Tesseract is able to work with a wide set of projects as well as data that is already archived by researchers, we employ an additional extraction and cross-linking step.

Depending upon the tools and practices of a particular project, data linking different project entities may exist in different forms in the project archives. In many cases, these links are explicit. As in the case of associating which developers have committed which files. In other cases, the links may need to be deduced heuristically, as in the case of identifying associations among artifacts. Finally, in some cases, team practices call for the recording of cross-links in data. For example, team practices commonly call for the bug-id to be listed in the commit log of a change set that fixed a particular bug. A critical requirement of Tesseract is the ability to identify individuals across multiple databases and discover which files were changed with which commits. We note that in some cases, additional effort may be needed to create these links, but we believe that the benefits provided by Tesseract will outweigh these costs. Creating richer information archives, by capturing discussions via email, chat, or wikis can help the team build a better understanding of their project.

Tesseract uses an extractor component (see Figure 4) to pull project data from its original locations and cross-link it where applicable. This cross-linked data is then stored in a small set of XML files. The schema of which is straight forward and enumerates the information we need to analyze and link the different relationships between artifacts, developers, and bugs or issues. We note that the extractor component is specific to a particular set of tools and practices and may require re-implementation for different projects. But once this step is performed, the rest of Tesseract is independent of the underlying data collection process.

Analyzing: The XML files generated by the extractor are analyzed on the rich web client to identify (1) relationships among code, developer, and bugs, (2) coordination requirements among team members that may arise because of the underlying technical dependencies in their work, (3) communication patterns among developers, and (4) the match between the coordination requirements and the communication patterns.

We treat artifacts that are frequently committed together to be logically associated with each other. We do so because artifacts that are edited or created as part of a particular task are often checked-in together as a change set [29]. Such practices have become norms in most organizations and software teams. This method of deducing dependencies works in situations where code analysis might be ineffective at identifying dependencies, including when: (1) the project contains code in different programming languages, (2) the call site is separated from the target with a network connection as in remote procedure calls, or (3) when the dependency is transmitted by an event bus. Caltaldo, et al. have, in fact, validated that this technique of calculating artifact association, is a better indicator of "who needs to coordinate with whom" in a team than techniques that employ static analysis [5]. The coordination requirements among developers are calculated based on the underlying technical dependencies of artifacts that the developers have edited.

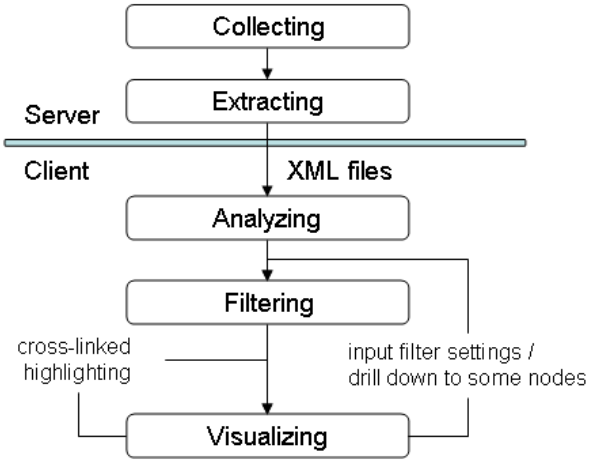

Figure 3. Information flow for Tesseract.

Next, Tesseract calculates the communication behavior of the team, which is simply the social network of developers as determined by their communication records. For our purposes, we analyze email communication, comments about a bug as available in the bug tracker, and work performed and submitted through the bug tracker. We consider the latter two sources as a record of communication since OSS developers often discuss an idea or leave notes for each other in the bug tracking system. Finally, we calculate congruence – the match between coordination requirements and communication behavior by using the algorithm proposed by Cataldo et al. [6].

Filtering: To help manage information overload, each of Tesseract's four panels provides controls that allow the user to adjust the amount of information that is presented. For instance, the Project Activity pane includes a *time slider* from which a user can select a particular time period that they want to investigate. Often a user

might be interested in investigating a specific time period, such as a past release or a period of time when she was away from the project.

The Files pane has, among others, a threshold for determining the number of times a file must be committed together before it is considered linked. Making this threshold configurable enables a user to fine tune the density of the file-to-file graph. For example, having a relatively low threshold of three will show a denser network than say a threshold of ten. Additionally, such configurations allow the user to filter noise in the data, which may be generated when non related artifacts are committed by coincidence. Developers can set appropriate thresholds based on their project culture and rigor.

There are similar filters for the Developers and Issues panes which allow users to select only a subset of the data. Such filtering enables users to have fine-grained control over their investigations and allows them to configure the tool to best fit their team's practices.

Visualizing: The last step in the process is visualizing the socio-technical relationships in the project. We have chosen appropriate graphical representation (e.g., networks, time series display, area charts) for different kinds of information, each of which have been already been discussed.

3.3. Design Rationale

The following design considerations guided our approach:

- **Decoupling data collection from consumption.** Tesseract decouples data collection from data consumption. This allows it to be easily adapted to different projects, which may use alternative repositories or may already have archived data in specific formats. For Tesseract to work with these different projects all that needs to change is the data collection part as in the former case or the data extractor part as in the latter case – the rest of Tesseract remains the same.
- **Easy substitution of linkage heuristics.** Currently, Tesseract uses commit logs to determine file associations. However, static analysis of code might provide additional insights, or an alternative view into file dependencies. Similarly, Tesseract presently uses three sources of communication records and two distinct heuristics to identify social relationships. Projects might have additional data (e.g., chat, wiki edits, web logs) available that can be used to determine social relationships. To address such additional data archives, as well as, prepare for possible future enhancements, Tesseract is purposefully architected to allow the use of different heuristics on different data sources.
- **Easy substitution of visualization components.** Tesseract currently uses a force directed network layout to visualize file and developer networks. It uses a bar graph for overall project activity and a stacked area graph for bug data. Each visualization widget can be substituted with other appropriate visualization techniques. Tesseract's architecture with its clean separation of analysis and visualization is specifically geared towards experimentation with different kinds of graphical displays.

3.4. Architecture

The architecture of Tesseract, as seen in Figure 4, reflects our design considerations. Section 3.2 provided a description of the data collection and extraction part of Tesseract. Here, we describe the overall design pattern of the rich web client, which comprises the analysis and visualization components.

Model: The data model stores three general categories of data: pre-processed relational data, user-specified filter settings, and the selection state of the tool, which includes entities that are currently selected and/or highlighted

View: The different user interface (UI) components (e.g., bar chart, stacked area chart, graph visualization) are specified declaratively. We use third party visualization widgets for each of these UI components.

Bindings: Bindings are also specified declaratively. Bindings exist between model data and view components, as well as, among model components as is the case when a user configuration changes the dependency

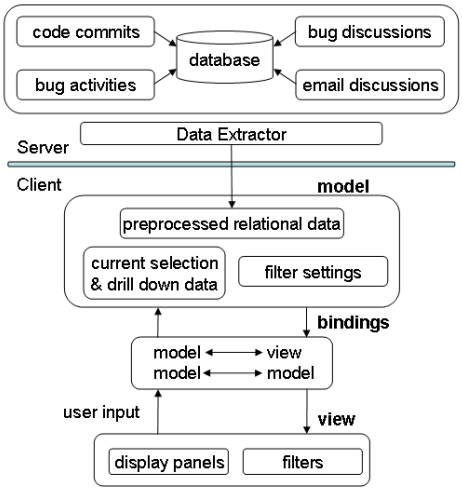

Figure 4. Tesseract architecture.

determination analysis. In this case, the change in the settings is sensed and the bound model components are automatically recalculated. Similarly, some of the more complex analysis is done with a series of separate calculations where the output of one calculation is consumed by the next. This pipe and filter approach is accomplished by binding the output of the first to the input of the next. In this way, the "controller" from a traditional model-view-controller meta-pattern is spread out among all of the "on-changed" events of the model objects.

4. Evaluation

We performed formative evaluations of Tesseract with two objectives in mind. First, we wished to investigate the usability of cross-linked displays providing multiple perspectives of a software project. Second, we wanted to understand the kinds of interactions and investigations that developers would perform with an exploratory tool such as Tesseract. Towards this goal, we first instrumented Tesseract with data from a large scale open source project, which validated the feasibility of our technique. We then performed informal lab based experiments for usability testing, which helped us identify bugs and features that have been incorporated in the next version of the tool. We then interviewed experienced open source developers to obtain feedback on the kinds of information that developers needed for their day-to-day work and how they would use Tesseract.

4.1. Use of GNOME project data

To be useful, Tesseract must collect, analyze, and cross-link extensive data from software projects. We tested the feasibility of building Tesseract by collecting and analyzing approximately ten years of data from the GNOME project [14]. Project source code and mailing list archives are freely available and were downloaded from public archives. All together more than 1,000 developers made nearly 2.5 million changes to files grouped into 480,000 commits. We worked with project administrators to obtain a copy of the complete bug database for the project, which contained 790,000 comments on 200,000 bugs, reported by 26,000 different people.

This data was loaded into a large database with a single schema that integrates each of these data streams. Like many open source projects these data streams were not seamlessly integrated with one another, making it difficult to associate files, bugs, email messages, and individual users. We worked with members of the community, and utilized information from norms and practices, such as referencing bug numbers in source code commit messages, to link together all the elements. The most difficult part of cross-linking the GNOME data was in normalizing user names across databases. Being open source, individuals often used different aliases for each system (SCM, bug tracker, email). While, a large part of the normalization process was automated, we needed to speak to individuals to makes sense of the remaining subset (about 10%). Such inconsistencies in user names are unlikely to be an issue in commercial projects making the underlying analysis and cross-linking effort easier.

Our system, currently, contains a complete and integrated data set that links together personal identities, individual files, source code commits, email messages, bugs and bug discussions. The database is not limited to a single project or ecosystem. While we currently have the most robust data for the GNOME project, we also have done substantial explorations of the Eclipse ecosystem and had similar success in linking entities. Furthermore, the system is extensible enough that additional data sources such as blogs and chat logs can be added as and when they are made available.

4.2. Usability studies

We performed a small set of usability experiments to evaluate whether users could understand and apply the cross-linked displays to perform a given set of project exploration tasks. We recruited four graduate students for the study. Participants were asked about their background and given a brief tutorial on tool usage. They were then given one hour to perform a set of five tasks that involved a particular GNOME project (Project Rbx).

- Task 1: identify a set of developers who you can go for help with your task (a set of file names that would be required for the task was provided).
- Task 2: identify how much these files have changed and by whom in the past two releases.
- Task 3: identify the key contributors in the project in the above time frame.
- Task 4: identify the contributions of a particular developer (name provided) and comment on their communication network (email, bug discussion, congruence).
- Task 5: what factors would you consider before determining whether project Rbx should be incorporated in your application? What information can you derive from Tesseract?

These tasks were purposely designed to require the use of two panels per task. Our main objective was to eva-

luate the effectiveness of the different visualization layouts for presenting each kind of information and the usability of cross-linked panels. One of the researchers was present in the room as an observer. Participants were asked to think aloud and their interactions with the tool were recorded via screen capture software.

We found that all participants performed consistently and provided similar answers to Tasks 1 through 4, in the allotted time. We found that participants had difficulty understanding the concept of "congruence" and typically simplified the concept by relying on the color coding "green" to be good and "red" to be bad. Alternatively, they switched to only viewing the communication network (email or bug or a combination) to solve Task 4. Answers to Task 5 varied widely among participants as different people used different heuristics, such as number of developers, current number of bugs open and their severity, total number of bugs, levels of activity. Keeping Task 5 open ended allowed us to identify the different kinds of information that users might require and provided us pointers to other data sources that might be useful for Tesseract to capture.

We terminated our studies after four runs because we found that participants were using similar processes to perform their tasks, which provided consistent results. Further, we needed to fix two bugs that were identified during the study and implement several feature requests that would greatly enhance the usability of the tool. In particular, the implementation of the search capabilities and textual listing of active developers and files was a request from users in this study. We decided to stop the usability testing to implement the requested changes.

4.3. Experienced developer feedback

A key objective of our formative evaluations was to understand how real-life developers would use an exploratory environment, such as Tesseract, for their day-to-day use. The fact that we had instrumented Tesseract with open source project data motivated us to interview open source developers and demonstrate Tesseract to them. Specifically, we conducted a series of interviews with five developers experienced in both open source and distributed software development. These developers had experience working in the software domain from four to thirteen years and had been involved with open source development from two to eight years. None of these developers had experience in the specific GNOME project that we used during the tool demonstrations. This was a deliberate choice, as we did not want past experience or personal information about the project to influence the developer in their investigations.

During the course of the interview we asked these developers about what role(s) they held on their project and their typical day-to-day activities. We then demonstrated different features of Tesseract, always using data from the same project in GNOME for consistency. After presenting the features and providing a brief explanation, we solicited free-form feedback on the features and scenarios in which they envisioned using Tesseract.

All our interviewees found the ability of viewing and exploring linkages among different project entities extremely interesting and useful. In particular, interviewees suggested they would use the file-to-file linkages to investigate which files are changed together and the ripple effects of changes. Most developers particularly liked our heuristics for file association (i.e. files that are changed together are linked)

> "*The implicit dependency stuff, that, I think could be really useful in and of itself. So things that which end up being changed together but don't necessarily have an inheritance relationship, or compositional -- knowing that, I've changed this thing it looks like something in isolation, but in reality whenever someone changes something here, these thirty other things change because of some ripple effect, that would be useful…*"

Interviewees also showed considerable interest in the linkages between files and developers. They foresaw using such links to answer questions such as: (1) which developers are interested in which files, (2) who is contributing what, (3) who should I talk to, and (4) who has made a particular change. They also suggested that this feature could be useful for quickly updating oneself with information of what had occurred in the project while they were away. Developer largely felt that finding such information currently requires significant efforts in reading large amounts of email or communicating with numerous people

> "*It's usually just talking to people about what happened, going back to the CVS and trying to see what happened with the file changes [is] kinda fruitless.*" The developer then mentioned how Tesseract could prove useful in this situation. "…f*rom a grunt developer standpoint, the file listing and cross reference of who has worked before – that would be very, very nice."*

Some interviewees suggested that the developer-to-developer linkages could serve as a means of creating an awareness of which developers work closely – information that is missing in their distributed work set-

tings. As already observed by Gutwin et al. [18], we found that most (senior) developers relied on an implicit knowledge of their project as created from meticulously keeping up-to-date with the different mailing lists. They thought that the developer linkages would only be marginally useful for their every day work. It was interesting to note that interviewees who were managers felt differently, and considered these linkages to be extremely useful. They foresaw using the congruence information provided by Tesseract to align the communication patterns in their team.

> *"this [developer pane] is a project manager view. What I know is, I am this person, three people have red flag and one person has green flag. My dashboard says you need to talk to [developer] because he made these changes…*".

Most developers agreed that Tesseract would greatly benefit new developers and managers. Without being asked explicitly, three developers volunteered that they would use Tesseract if they were to start working on a new project and four developers mentioned the tool to be particularly useful for managerial purposes.

In addition to confirming the need for capabilities for a tool like Tesseract, these interviews provided us with insightful feedback, which will help us improve the tool and can make it a likely candidate for adoption by this community.

5. Conclusions and Future Work

We have developed Tesseract to enable interactive exploration of the complex, evolving relationships among different project elements. Our work builds upon the recent history of socio-technical tools by:

- Showing the feasibility of creating a general project browser tool that considers both technical dependencies as well as social interactions. This significantly extends the capabilities of other tools like Ariadne [8] and Expertise Browser [22], which only consider technical records to assist social interactions. Further, Tesseract provides a generalization of their intended capabilities. Tesseract is designed for investigating relationships among code, communication records, bugs, and developers over time.
- Embedding the theoretical foundations of congruence established by Cataldo et al. – one of a number of possible retrospective analysis techniques – in a tool to help developers better align their communication patterns with coordination requirements.
- Enabling interactive exploration of the complex socio-technical space by novel use of cross-linked views that support selecting, highlighting, searching, drilling-down, and filtering.

Our informal usability studies and feedback from open source developers illustrate that Tesseract is relatively easy to use and valuable for new developers or managers who have to yet create a good mental mapping of the project. Further, Tesseract can help experienced users to investigate a problem when they have an incomplete or incorrect knowledge of that event.

We intend to further refine Tesseract based on user recommendations, such as: (1) hierarchically grouping files based on packages, functionality, or architecture, (2) providing additional context of changes, (3) allowing developers to specify when they have communicated with another developer by means other than that currently captured by the tool, and (4) explore different analysis techniques, such as adding temporal considerations to our calculation of congruence or adapting social network analyses to software engineering.

We also plan to conduct other lab-based studies to observe whether different development modes (e.g., code design, implementation, debugging) require different kinds of information gathering and investigations. Additionally, we plan to have Tesseract adopted by a live project and/or communication channel and identify the different, live usage patterns by capturing user interactions. Such a study will help us further refine our understanding of the kinds of information that are sought by developers during their development process. Finally, our ultimate goal is to have Tesseract serve as a front-end tool for project explorations.

6. Acknowledgements

This effort is partially funded by the NSF grant number IIS-0414698 and IIS 0534656, and the Software Industry Center and its sponsors, particularly the Alfred P. Sloan Foundation. Effort also supported by a 2007 Jazz Faculty Grant.

7. References

[1] R. Arnold and S. Bohner, *Software Change Impact Analysis (Practitioners)*, 1 ed.: Wiley-IEEE Computer Society Pr, 1996, p. 392

[2] R. S. Arnold, "The Year 2000 problem: Impact, Strategies and Tools", Software Evolution Technology, Inc. Tech. Report February 1996.

[3] J. Biehl, M. Czerwinski, G. Smith, et al., "FASTDash: A Visual Dashboard for Fostering Awareness in Software Teams", *SIGCHI conference on Human Factors in computing systems*, San Jose, California, USA, 2007, pp. 1313-1322.

[4] C. Bird, D. Pattison, R. D'Souza, et al., "Chapels in the Bazaar? Latent Social Structure in OSS", in *16th ACM SigSoft International Symposium on the Foundations of Software Engineering,* Atlanta, GA, 2008, (to appear).

[5] M. Cataldo and J. Herbsleb, "Communication Networks in Geographically Distributed Software Development", *Computer Supported Cooperative Work*, San Diego, California, USA, 2008, pp. 579-588.

[6] M. Cataldo, P. Wagstrom, J. D. Herbsleb, et al., "Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools", *ACM Conference on Computer Supported Cooperative Work*, Banff, Alberta, Canada, 2006, pp. 353-362.

[7] D. Cubranic, G. C. Murphy, J. Singer, et al., "Hipikat: A Project Memory for Software Development", *IEEE Transactions on Software Engineering,* vol. 31, June 2005, pp. 446-465.

[8] C. R. B. de Souza, S. Quirk, E. Trainer, et al., "Supporting Collaborative Software Development through the Visualization of Socio-Technical Dependencies", *International ACM SIGGROUP Conference on Supporting Group Work* Sanibel Island, FL, 2007, pp. 147-156.

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

[10] R. DeLine, A. Khella, M. Czerwinski, et al., "Towards Understanding Programs through Wear-Based Filtering", *ACM Symposium on Software Visualization*, St. Louis, Missouri, 2005, pp. 183-192.

[11] P. Dewan and R. Hegde, "Semi-Synchronous Conflict Detection and Resolution in Asynchronous Software Development", *Conference on European Computer Supported Cooperative Work*, Limerick, Ireland, 2007, pp. 159-178.

[12] J. Froehlich and P. Dourish, "Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams", *International Conference on Software Engineering*, Edinburgh, UK, 2004, pp. 387-396.

[13] E. Gilbert and K. Karahalios, "CodeSaw: A Social Visualization of Distributed Software Development", *Human-Computer Interaction – INTERACT*, 2007, pp. 303-316.

[14] GNOME - The Free Software Desktop Project. http://www.gnome.org/.

[15] R. E. Grinter, "Recomposition: Putting It All Back Together Again", *ACM conference on Computer supported cooperative work*, Seattle, Washington, USA, 1998, pp. 393-402.

[16] R. E. Grinter, J. D. Herbsleb, and D. E. Perry, "The Geography of Coordination: Dealing with Distance in R&D Work", *ACM Conference on Supporting Group Work*, Phoenix, AZ, 1999, pp. 306-315.

[17] C. Gutwin and S. Greenberg, "The Effects of Workspace Awareness Support on the Usability of Real-Time Distributed Groupware," *Transactions on Computer-Human Interaction* vol. 6, September 1999, pp. 243-281.

[18] C. Gutwin, R. Penner, and K. Schneider, "Group Awareness in Distributed Software Development", *ACM conference on Computer Supported Cooperative Work*, Chicago, Illinois, USA, 2004, pp. 72-81.

[19] J. Herbsleb and R. E. Grinter, "Splitting the Organization and Integrating the Code: Conway's Law Revisited", *Proceedings of the 21st international conference on Software engineering*, Los Angeles, CA, USA, 1999, pp. 85-95.

[20] M. Kersten and G. C. Murphy, "Using Task Context to Improve Programmer Productivity", *Fourteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Portland, Oregon, USA, 2006, pp. 1-11.

[21] A. Meneely, L. Williams, W. Snipes, et al., "Predicting Failures with Developer Networks and Social Network Analysis", *ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, Altanta, GA, 2008, (to appear).

[22] A. Mockus and J. Herbsleb, "Expertise Browser: A Quantitative Approach to Identifying Expertise", *International Conference on Software Engineering*, Orlando, FL, 2002, pp. 503-512.

[23] D. L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules," *Communications of the ACM,* vol. 15, 1972, pp. 1053-1058.

[24] D. E. Perry, H. P. Siy, and L. G. Votta, "Parallel Changes in Large-Scale Software Development: An Observational Case Study," *ACM Transactions on Software Engineering and Methodology,* vol. 10, 2001, pp. 308-337.

[25] A. Sarma, Z. Noroozi, and A. van der Hoek, "Palantír: Raising Awareness among Configuration Management Workspaces", *Twenty-fifth International Conference on Software Engineering*, Portland, Oregon, USA, 2003, pp. 444-454.

[26] T. Schümmer and J. M. Haake, "Supporting Distributed Software Development by Modes of Collaboration", *Seventh European Conference on Computer Supported Cooperative Work*, 2001, pp. 79-98.

[27] M. E. Sosa, S. D. Eppinger, and C. R. Rowles, "The Misalignment of Product Architecture and Organizational Structure in Complex Product Development," *Management Science,* vol. 50, December 2004, pp. 1674-1689.

[28] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications (Structural Analysis in the Social Sciences)*, 1 ed.: Cambridge University Press, 1994, p. 857.

[29] W. D. Weber, "Change Sets versus Change Packages: Comparing Implementations of Change-Based SCM", *Seventh International Workshop on Software Configuration Management*, 1997, pp. 25-35.