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

Anita Sarma, Larry Maccherone, Patrick Wagstrom, and Jim 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, a socio-technical dependency browser that utilizes cross-linked displays to enable exploration of 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

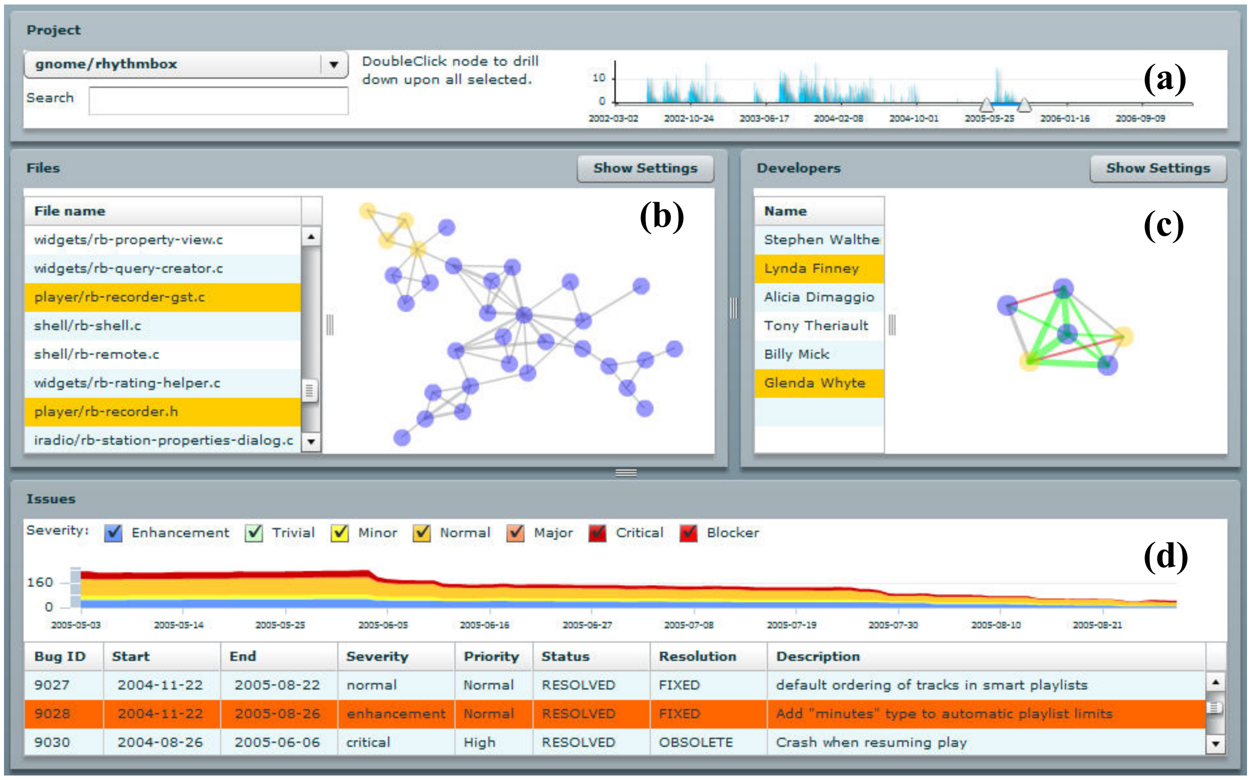
Development environments increasingly reflect the fact that artifacts, developers, and tasks are intrinsically bound together in a software project. While editing a file of source code, for example, many other artifacts are likely to be relevant. Considerable research effort has focused on using a variety 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 indicate 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 have intuitively known this for a long time, 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]



File name
widgets/rb-property-view.c
widgets/rb-query-creator.c
player/rb-recorder-gst.c
shell/rb-shell.c
shell/rb-remote.c
widgets/rb-rating-helper.c
player/rb-recorder.h
iradio/rb-station-properties-dialog.c

Name
Stephen Walthe
Lynda Finney
Alicia Dimaggio
Tony Theriault
Billy Mick
Glenda Whyte

Bug ID	Start	End	Severity	Priority	Status	Resolution	Description
9027	2004-11-22	2005-08-22	normal	Normal	RESOLVED	FIXED	default ordering of tracks in smart playlists
9028	2004-11-22	2005-08-26	enhancement	Normal	RESOLVED	FIXED	Add "minutes" type to automatic playlist limits
9030	2004-08-26	2005-06-06	critical	High	RESOLVED	OBSOLETE	Crash when resuming play

Figure 1. Screenshot of Tesseract showing the four juxtaposed displays.

or in finding the right person with whom to communicate [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).
- Highlight the match (or the lack thereof) between the social and technical relationships.
- Cross-link and enable interactive exploration of these relationships and how they change over time.

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).

We evaluated Tesseract in three different ways. First, we demonstrated the feasibility of our technique by testing Tesseract with ten years of data from a large open source project. Second, we conducted informal user evaluations that illustrated the ease of use afforded by the tool. Finally, we verified Tesseract's suitability for the open source community by conducting interviews with open source developers, where we demonstrated the tool and solicited their feedback on the features and scenarios in which they envisioned using such a tool.

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.

The remainder of this paper is organized as follows. Section 2 motivates the design of Tesseract by building upon studies that reveal the necessity of a tool like Tesseract and lessons learned from existing tools. In Section 3 we present Tesseract followed by two usage scenarios. We then describe the underlying implementation technique of Tesseract and the design principles that guided our approach. We end this section with a discussion of the architecture of Tesseract. Section 4 discusses the three ways in which we evaluated Tesseract. We conclude in Section 5 with an outlook at future work.

2. Informing the design of Tesseract

We informed the design of Tesseract by building on (1) studies that reveal the problems of managing change and communication in teams and (2) lessons learned from tooling attempts that aim to address these problems. We then conducted interviews with open source developers to confirm these problems and better understand their nature in this particular environment. These interviews helped us identify the capabilities that must be present in a tool that will be validated with data from an open source project.

2.1. Problems of change and communication

We motivate our work from four studies. The first study by de Souza and Redmiles [9] observed two software development teams and found that the first team was highly disciplined and managed the impact of their changes by (1) rigorously broadcasting email of impending changes and (2) reading these emails to create an awareness of who was working on what. While this practice worked well, it involved significant effort in writing and reading the emails. The second 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 lead to duplication of information and overwhelmed the users.

The second study by Cataldo et al. [6] analyzed project data from a commercial software development team. They created social networks among developers based on the underlying dependencies among artifacts that were edited. They then compared this network (coordination requirements) with the communication patterns in the team and found that teams with high congruence – a match between the coordination requirements and communication patterns performed better. They also found that the coordination requirements evolved over time requiring developers to correctly identify the new relationships and people with whom to communicate.

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. They recommend that managers build an alignment matrix to align team interactions with the way components are associated via shared interfaces.

The final study by Gutwin et al. [18] investigated the collaboration mechanisms of open source developers. They found that text communications (mailing lists and text chats) were the primary mechanisms for maintaining awareness of who is doing what and finding experts. 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 the communications in different lists. Further, splitting communication between email, chat, and issue tracker caused duplication of information and lead to situations where developers missed important information.

These studies reflect different needs faced by developers and managers at large. Further, work by Gutwin et al. show that these needs are similar between commercial and open source projects. Tesseract uses the underlying analysis technique of Cataldo et al. to provide a socio-technical dependency browser that teams can use to align their interactions in the context of their current tasks. By automatically inferring the socio-technical networks from existing project data Tesseract should reduce the effort and errors associated with impact management.

2.2. Current tool support

There exists considerable work in analyzing technical dependencies in a project via sophisticated 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 provides important information about the project, they can be much more useful when combined together. Emerging research following this premise has produced a suite of tools. Tools in the computer-supported cooperative work community, for instance, attempt to enable a developer become aware of ongoing project activities as they occur [17]. The hypothesis is that developers can better coordinate their changes with those of others by incorporating information about ongoing changes with their own knowledge about the project [3, 12]. Tools such as Tukan [26], CollabVS [11] and Palantír [25] perform some basic code analysis to identify dependencies, changes to w-hich are then used to flag problems.

Tools such as Expertise Browser [22], Team Tracks [10], and Ariadne [8] follow a similar approach, but have slightly different goals. These tools use the socio-technical relationship model created from code contributions, visitations, or dependencies to help a developer with their work. Expertise Browser, for instance, uses code contributions (frequency as well as lines of code committed) of a developer to determine their expertise. However, it was noted that users often attempted to use Expertise Browser for purposes other than finding experts, such as obtaining an overview of project activities or finding recent changes to a particular code module. Team Tracks 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 performs code analysis to identify artifacts dependencies, which it then combines with information of code contributions to model the socio-technical relations.

To summarize, each tool uses a different way to decipher code relationships which are then used to facilitate social interactions by promoting awareness of ongoing changes, enabling identification of experts, or helping new developers understand their project. 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 study of usage of Expertise Browser team reveals that developers and managers are eager to investigate the socio-technical relationships for getting a better understanding of their project so as to better inform their work.

To the best of our knowledge, CodeSaw [13] is the only tool that actually 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 does not create a socio-technical dependency mapping.

3. Tesseract

Tesseract analyzes different project archives, such as change management systems, bug repositories, and email discussion forums to determine the socio-technical relationships, which are then graphically displayed via four juxtaposed panes enabling users to easily explore the data set. Tesseract also uses the analysis technique proposed by Cataldo et al. [6] to show the user where the set of technical dependencies is congruent with the social interactions, and where it is not. The four display panes of Tesseract are:

- The Project activity pane (Figure 1(a)) displays the overall activities in a project in 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 dependencies as a file-to-file network, which is created by linking files that are frequently changed together. The number of times two files are committed together is represented by the thickness of the edges in the network. A textual listing of the file names is provided to allow quick identification of specific files by name.
- The Developers network pane (Figure 1(c)) displays relationships among developers. Two developers are considered related if they either edited the same artifact or interdependent artifacts. The edges in this network are colored (green vs. red) to show when developers have communicated via either email or the bug repository (e.g., comments or activities in Bugzilla). The thickness of the edges is derived from 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 in a number of 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, clicking on a particular developer highlights all the files which that developer had edited in the given time frame. Second, hovering over a node in the 'Files' or 'Developers' panes causes two actions: (1) display of additional information about the node and (2) highlighting of all neighbors (any other nodes with an edge to the hovered over node). 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 only the developers she personally knows in the 'Developers' pane to find whether any of her acquaintances have expertise who could provide help with her current task.

3.1. Usage Scenarios

Tesseract allows the investigation of a particular event or identification of past development patterns. Here we present two example scenarios about each

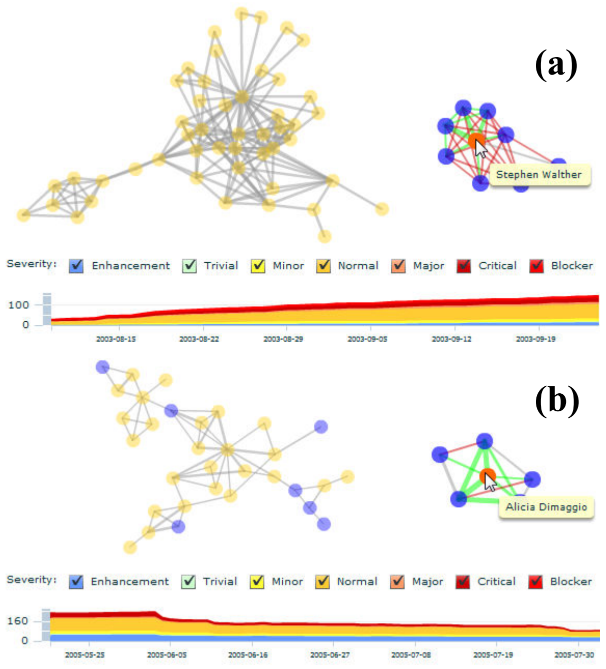


Figure 2. Contrasting development patterns.

activity. Both scenarios reflect real project data from a GNOME project with developer names anonimyzed.

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.

Assume a hypothetical case where Billy Mick, a core developer, is asked to fix a particular bug regarding the display of “minutes remaining” in a playlist. He vaguely remembers that Glenda Whyte, another core developer had worked on a feature to add the “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 he remembered Glenda adding (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 makes an assumption that that Alicia Dimaggio, the current project lead, may have mediated between Glenda and Lynda as they both have green lines to Alicia. To ensure that he gets a complete picture of the feature he decides to contact Lynda before starting work.

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 DiMaggio is the primary contributor; (2) Alicia is communicating with other contributors and there is sufficient communications among the other core contributors; (3) the file network is less dense 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 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. This 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 software projects activities that involve code and issue or bug tracking are automatically archived by the SCM or issue tracking system. Project mailing list or other communication records are not always available for commercial projects. But, open source projects maintain and make publicly available a rich history of their communication records. Tesseract relies upon such prior collection of project data.

Extracting and Cross-linking: Different projects use different individual 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 Tracs 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 incorporate an additional extraction and cross-linking step.

Depending upon the tools and practices of a particular project, data that links different project entities may appear in a variety of forms in the project databases. 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 partial cross-link data can be obtained by virtue of team practices. For example, when team practices require the bug id to be listed in the commit log for a change set that fixes that particular bug. We note that developers have to expend additional effort in creating these links across data, but we believe that the benefits provided by Tesseract will out weigh these costs. For example, mentioning the bug id in email communication subjects or ensuring that discussion are recorded in the email forums and not taken offline. Teams can also provide richer data sources by archiving and maintaining chat logs or discussions taking place via blogs or wikis.

However, the data necessary for the primary socio-technical analysis of Tesseract is easily available. Tesseract can perform its analyses as long as there are ways to identify an individual across multiple databases and discover which files were changed with which commits.

The extractor component pulls the data from its original location and cross-links data where applicable. This cross-linked data is then stored in a small set of XML files. The DTD of which is straight forward and enumerates all the information we need to analyze and link the myriad relationships between artifacts, developers, and bugs or issues.

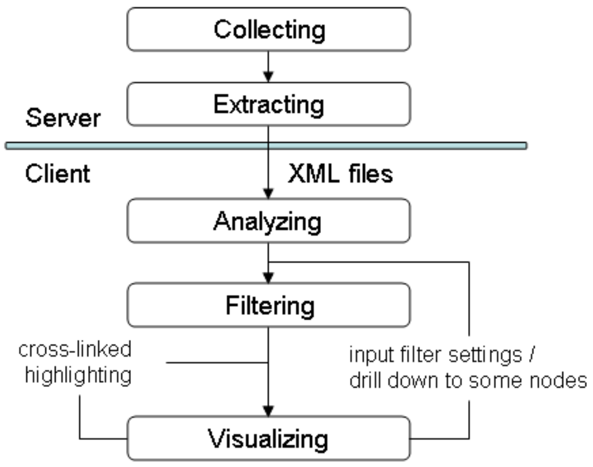

Collecting
Extracting
Server
Client
XML files
Analyzing
Filtering
cross-linked highlighting
input filter settings / drill down to some nodes
Visualizing

Figure 3. Information flow for Tesseract.

The Extractor component is specific to a particular set of tools and practices and may need re-implementation for different projects. But once this step is performed, the rest of Tesseract is independent of the underlying data collection.

Analyzing: The XML files generated by the extractor are analyzed on the rich web client to discern (1) relationships among code, developer, and bugs, (2) communication patterns among developers, and (3) the match between the technical dependencies and the social interactions.

The current version of Tesseract analyzes dependencies among artifacts based on how often they are committed together. 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. For our purposes, we consider two files to be dependent when they are committed together a user specified number of times (threshold). Further, we visually represent the significance of this coupling by showing files that were frequently committed together with thicker edges in the file network (weighting).

This method of deducing dependencies among artifacts based on when they were changed together works better in situations where the call site is separated from the target with a network connection as in remote procedure calls, or when the dependency is transmitted by an event bus. Caltaldo, et al. have, in fact, validated that this technique of calculating coordination requirements, is a better indicator of "who needs to coordinate with whom" in a team than techniques that employ static analysis [5].

Next, Tesseract calculates the communication behavior of the project 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 in 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, which is the match between coordination requirements and communication behavior.

Filtering: To help manage information overload, each of Tesseract's four panels has some controls that allow the user to adjust the amount of information that is used. 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 the last 'n' weeks or in investigating a particular time period, say a past release.

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 to be configurable enables a user to fine tune the density of the file-to-file graph. For example, having a relatively low threshold of 3 will show a denser network than say a threshold of 10. Additionally, such configuration allows the user to tune out noise in the data that may be generated when non related artifacts are erroneously committed. Such noise will not occur when the threshold is set to a reasonable limit.

There are similar filtering controls for the Developers and Issues panes which allow the user to specify that only a subset of the available data be considered. This allows users 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 for each kind of information, each of which have been already been discussed. A point to note is that Tesseract considers the analysis and filter settings when rendering a visualization of project data.

3.3. Design Rationale

The following design considerations guided our approach:

- **Decoupling data collection from consumption.** Tesseract decouples the data collection from data consumption. This allows the tool to be easily adapted to different projects which may use alternative repositories or may already have archived data in a specific format. 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 discern file dependencies. However, static analysis of code might provide additional insight, or at least an alternative view, into file dependencies. Similarly, Tesseract presently uses three sources of communication records and two distinct heuristics to discern social relationships. Projects might have additional data (wiki edits, web logs) readily available that can be used to determine social relationships. To address such additional data archives as well as to prepare for possible future enhancements, Tesseract is purposefully architected to allow the use of different kinds of heuristics on different kinds of data sources to create associations among project entities.
- **Easy substitution of visualization components.** Tesseract currently uses a force directed network layout to display networks. It uses a bar graph for overall project activity and a stacked area graph for bug data. The underlying analysis is cleanly separated from the visualization components to allow different kinds of graphical displays.

3.4. Architecture

The architecture of Tesseract, as seen in Figure 4, reflects these 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

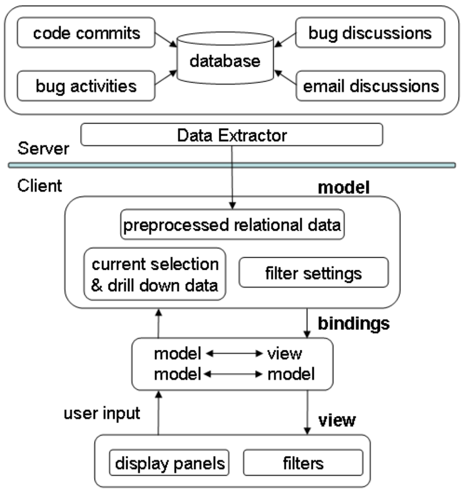


Figure 4. Tesseract architecture.

filter settings, and the selection state of the tool that includes the entities which 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 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

To evaluate Tesseract we first demonstrated that it can successfully analyze data from a real life project. In our case we utilized more than ten years of history of the GNOME project, a large open source desktop project [14]. We then assessed the usability of Tesseract via informal user evaluations that required participants to perform a given number tasks using one of the GNOME projects. This study helped us identify bugs and features, most of which has helped shape the next version of the tool. However, our study participants, while having expertise in software engineering and usability, lacked the context of contributing to open source projects and could not provide information on the usefulness of Tesseract. To overcome this problem, we demonstrated Tesseract to experienced open source developers and obtained their feedback to further improve the tool.

4.1. Use of GNOME project data

To be useful, Tesseract must analyze and cross-link extensive data from software engineering processes. We tested the feasibility of building Tesseract by collecting and analyzing approximately ten years of data from the GNOME project. 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.

All of 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 generally not 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. Together this system provides a complete and integrated data set that forms the core of Tesseract's data analysis capabilities and 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 recruited four graduate students for our usability 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.

- Task 1: identify a set of developers who have expertise in a given set of files.
- Task 2: identify the files which have changed and by whom in a given time period.
- Task 3: identify the key contributors in the project.
- Task 4: identify the contributions of a particular [central] developer and comment on their communication network.
- Task 5: how would you determine whether the project under investigation is a good project to incorporate in your application?

These tasks were designed to evaluate how participants used and understood the different features of Tesseract. 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 similarly and were able to correctly complete Tasks 1 to 4 in the given time. The answer to Task 5 varied among participants as different people used different heuristics (number of developers, current number of bugs open and their severity, total number of bugs, levels of activity) which led them to different results. We also 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.

We stopped our evaluations after four studies because we found that participants were performing tasks similarly giving us consistent results and were unable to provide additional insights since they lacked the context of having actually worked on the GNOME project that was being used for the evaluation. Further, we also wanted to implement the feature requests that would enhance the usability of the tool and fix two bugs that were discovered during the study.

The feedback from this study has shaped some of the features described earlier. In particular, the implementation of the search capabilities and textual listing of active developers and files was a result of this study. We have to yet implement two additional suggestions – moving a group of nodes together and maintaining a constant layout even when users change settings. It was not possible to address these two issues with the third party graphing component that we are currently using. We are in the process of creating our own graph layout component which will address these needs.

4.3. Experienced developer feedback

Given that we prototyped and tested Tesseract using data from the GNOME project, we wanted to verify the need for its capabilities in the open source community. To do so, 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. During the course of the interview we asked these developers about their role in the project and their typical day-to-day activities. We then demonstrated the 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 such a tool.

We found that all 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 method of linking artifacts based on logical coupling (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. "*…from 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 settings. 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 to developer linkages would only be marginally useful for their every day work. It was interesting to note that interviewees who were manager 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 also provided us with insightful feedback, which will help us improve the tool and can make it a likely candidate for adoption by this community. Some of the recommendations that we plan on implementing in the next prototype are: (1) hierarchically grouping files based on packages, functionality, or architecture, (2) providing additional context about changes, (3) hooking Tesseract to a live project and/or communication channel, and (4) allowing developers to specify when they have communicated with another developer by means other than that captured by the tool.

5. Conclusions and Future Work

We have developed Tesseract to enable interactive exploration of the socio-technical relationships in a project. 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 and Expertise Browser which only consider technical records to assist social interactions. Further, Tesseract provides a generalization of their intended capabilities. Ariadne is designed to manage impact of changes and Expertise Browser is targeted at locating experts. Tesseract can be used for both of these purposes, and more.
- 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 achieve a better match between the coordination requirements and communications in their own projects.
- Enabling interactive exploration of the complex socio-technical space by judicious 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 extend Tesseract in two future directions. First, we wish to deploy Tesseract for live projects and making it an integrated front-end for a the different tools that are currently used by . This would require Tesseract to provide additional functionalities such as source browsing, version comparison, and attribution. We note that capturing communication records and maintaining entity cross-references – data needed by Tesseract – requires additional effort from users. However, we believe that the benefits provided by the live use of Tesseract will encourage the diligence necessary for such data to be captured. Of course, we need to validate this claim by actual adoption and use by development teams. Second, we plan to explore different analysis techniques, such as adding temporal considerations to our calculation of congruence or adapting social network analyses to the software engineering domain.

6. Acknowledgements

This effort is partially funded by the NSF grant number IIS-0329090, 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, (to appear).

[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.