From Technical Dependencies to Social Dependencies

Cleidson de Souza[1,2] Paul Dourish[1] David Redmiles[1] Stephen Quirk[1] Erik Trainer[1]

[1]Donald Bren School of Information and Computer Science
University of California, Irvine
Irvine, CA, USA – 92667

[2]Departamento de Informática
Universidade Federal do Pará
Belém, PA, Brazil – 66075

Abstract

This paper describes Ariadne, a Java tool for the Eclipse IDE, that links technical and social dependencies. Ariadne is based on the observation that technical dependencies among software components create social dependencies among the software developers implementing these components. We describe our approach for creating technical, socio-technical and social dependencies from a software project. We describe possible uses of our approach and tool, as well as, briefly discuss related work.

1. Introduction

One of the most important and influential principles in software engineering is the idea of information hiding proposed by Parnas [1]. According to this principle, software modules should be both "open (for extension and adaptation) and closed (to avoid modifications that affect clients)" [2]. Information hiding aims to decrease the dependency between two modules so that changes to one do not impact the other. Parnas had also recognized that information hiding also brings managerial advantages: by dividing the work in independent modules, it is also possible to assign the implementation of these modules to different developers that can work on them independently. However, a consequence of decomposing the system into pieces, is the eventual need to manage the integration of these pieces to create the whole software. This work of building the whole software from its parts requires a lot of coordination effort in both collocated and distributed projects [3] [4] [5]. The reconstruction of the system from its pieces is necessary during the development, testing, and maintenance phases. Despite the advantages of the modular decomposition, software engineering research has already found out that one module can not be implemented completely independently of its clients; somehow one module needs to know some of the requirements that its clients have [6].

These two aspects - the frequent need to recompose the software and the dependency among its components - suggest that software developers working on the implementation of these components need to interact regularly to ensure that their work is aligned, so the integration flows smoothly when necessary. In other words, because software components need to interact, this creates a similar need of interaction among the software developers implementing them. In short, technical dependencies between components create social dependencies between the software developers implementing them [3, 4]. Based on this observation, we argue that the source-code itself can be an important resource to identify social relationships that need to be maintained among software developers to facilitate the integration process. This is possible because the software itself contains information about the technical dependencies among pieces of software. This paper describes our approach and associated tool – Ariadne – that aims to uncover this relationship among technical and social dependencies. Ariadne is currently being developed at UC, Irvine and supports the analysis of Java programs to identify the technical dependencies. Authorship information of the software components is used to create the web of social dependencies. This is done by collecting information from a configuration management repository associated with the Java program.

The rest of the paper is organized as follows. The next section describes how from the technical dependencies existing in a software development effort, it is possible to uncover the social dependencies among the software engineers involved in the project. The following section describes our tool and some of its implementation details. After that, we describe some related work. Finally, conclusions and ideas for future work are presented.

2. From Technical to Social Dependencies

Our approach combines source-code analysis and collection of information about the software developers in order to appropriately identify the relationship between the technical and social dependencies. In this section, we will detail the technical aspects of the source-code analysis as well as the collection of (social) information about software developers.

Technical Dependencies

Our approach is strongly-based on the concept of dependencies. Dependencies among pieces of code exist because components, inevitably, make use of services provided by other components. For example, component *A* uses the services of component *B*, as a result, *A* depends on *B*. That is, in order to component *A* to be able to perform its functions, it relies on services provided by component *B*. A data structure containing all the dependency relationships of a program is called a *call-graph*, because it contains information about which components *call* the services provided by other components in the system. Figure 1 presents an example of a call graph of an open-source project hosted at sourceforge. A directed edge from a method A to another method B indicates a dependency from A to B. Because Java is an object-oriented language, the call-graph describes the relationship between methods being invoked by other methods in the context of their respective classes and packages.

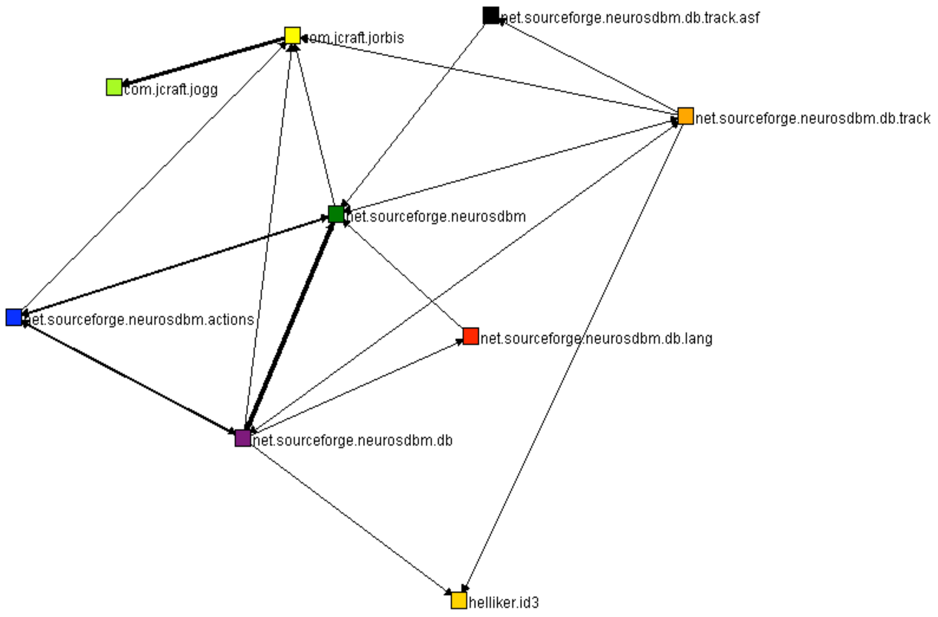
net.sourceforge.neurosdbm.db.track.asf
com.jcraft.jorbis
com.jcraft.jogg
net.sourceforge.neurosdbm.db.track
net.sourceforge.neurosdbm
net.sourceforge.neurosdbm.actions
net.sourceforge.neurosdbm.db.lang
net.sourceforge.neurosdbm.db
helliker.id3

Figure 1: An example of a call graph

By describing dependencies in the source-code, this graph potentially describes dependencies among software developers responsible for those software components. Using the previous example, where component *A* depends on component *B*, assuming that *A* is being developed by *developer a* and *B* is being implemented by *developer b*, since *A* depends on *B*, we similarly find that *developer a* depends on *developer b*. Hence, to be able to describe these social dependencies, it is necessary to populate the call-graph with social information, i.e., information about which software developer wrote which part of the code. This is explained in the next section.

Socio-Technical Dependencies

Authorship information about each node of the call-graph can be extracted from configuration management (CM) repositories, which usually store revision information describing each and every change made to the software system being analyzed [7]. Typical revision information for each change includes: the changes applied to the software, date and time of these changes, the author of the changes, the files where the changes were applied. Combining information from the call-graph with authorship information from the CM repository can then create a "social call-graph", that describes which software developers depend on which other software developers for a given piece of code. Figure 2 presents an example of a "social call-graph" from an open-source project. A directed edge from package A to B indicates a dependency from A to B. Directed edges between authors and packages indicate authorship information. Every node of the call-graph might have different options for the associated authorship information: for example, in a company, one might decide to use information about the last person who committed changes in the file because the last committer is sometimes considered an expert on it [8], in another company, ownership architectures could be used since it expresses the relationship between developers and source code [9]. The information of a "social call-graph" can be translated into a two-mode network. The "social-call graph" diagram presented in Figure 2 was created using our tool, Ariadne. This tool is described in more details in the following section.

Social Dependencies

Because of the information that they integrate, "social call-graphs" could easily be used to generate sociograms describing the dependency relationship among software developers *without* depicting dependencies among software components. Figure 3 below presents an example of such situation. Currently we are investigating the use of social networks algorithms to assess potential coordination problems in the software development process. For example, one could generate technical recommendations about how to reorganize the source code, or provide managerial recommendations about how to change the division of labor to minimize the coordination effort of some developers that have to deal with too many dependencies. However, in order to do that, we need a tool that is able to construct and analyze "social call-graphs" and social network graphs from software development projects. In the next section, we describe Ariadne, a tool that addresses this issue.

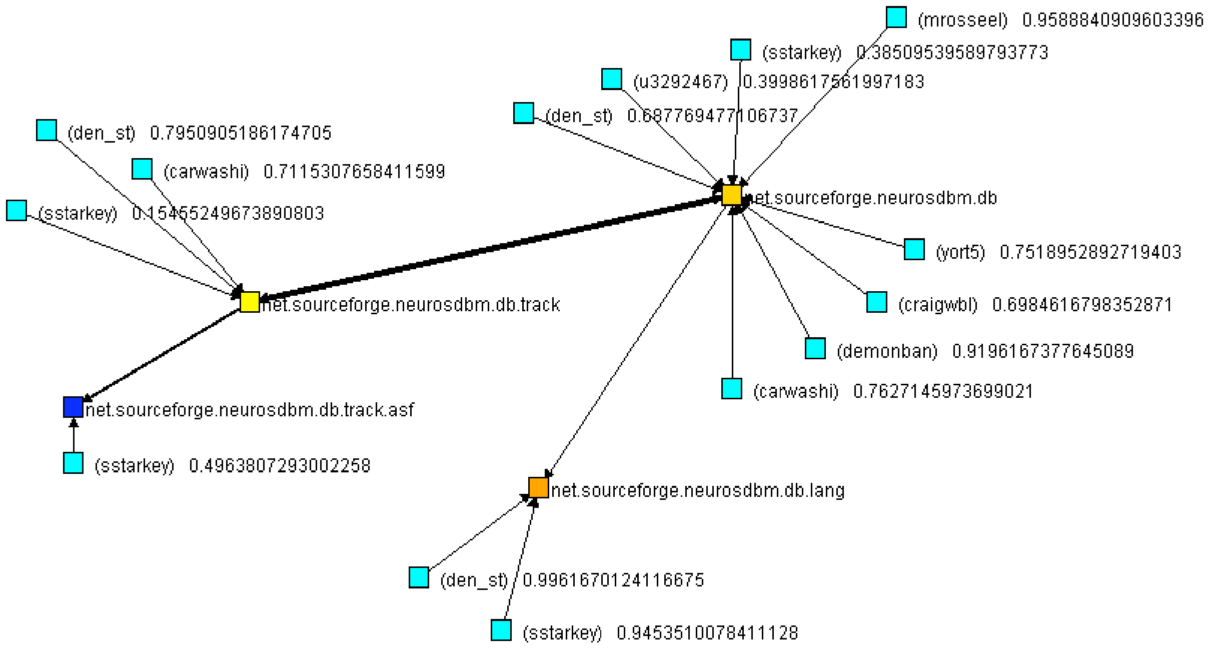
(mrosseel) 0.9588840909603396
(sstarkey) 0.38509539589793773
(u3292467) 0.3998617561997183
(den_st) 0.6877694771067373
(den_st) 0.7950905186174705
(carwashi) 0.7115307658411599
(sstarkey) 0.15455249673890803
net.sourceforge.neurosdbm.db
(yort5) 0.7518952892719403
net.sourceforge.neurosdbm.db.track
(craigwbl) 0.6984616798352871
(demonban) 0.9196167377645089
(carwashi) 0.7627145973699021
net.sourceforge.neurosdbm.db.track.asf
(sstarkey) 0.4963807293002258
net.sourceforge.neurosdbm.db.lang
(den_st) 0.9961670124116675
(sstarkey) 0.9453510078411128

Figure 2: An example of a "social call graph"

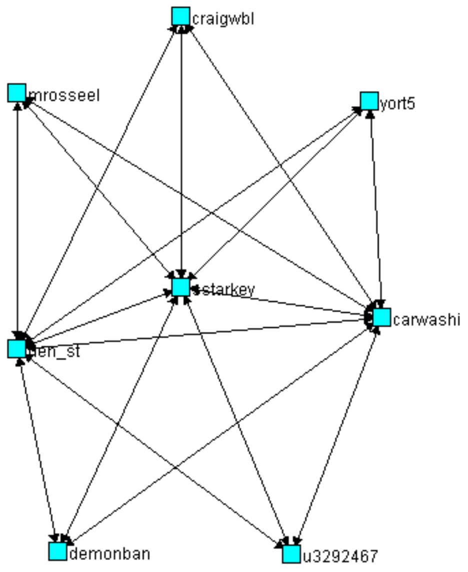

Figure 3: An example of a social network graph describing dependency relationships among software developers

3. Ariadne

Ariadne is a Java-based tool that creates call-graphs, "social-call graphs", and sociograms. Ariadne is currently implemented as a plug-in to the Eclipse IDE (Integrated Development Environment). Eclipse is an open-source project that aims to develop a powerful IDE with an extensible architecture based on plug-ins. Therefore, one can create plug-ins to extend the IDE and still have access to all resources provided by this IDE. And that is exactly what Ariadne does. It analyses Java projects being used in Eclipse and automatically connects to the configuration management repository associated with these projects to retrieve authorship information about the project. Currently, we are supporting only CVS repositories. In order to construct call-graphs Ariadne uses another open-source project called DependencyFinder, which creates the call-graph for any compiled Java project. Our current implementation can present call graphs at three different levels of abstraction: methods, files, and packages. Basically, information from the methods is aggregated to generate information about files, and similarly information about files is aggregated to generate information about packages. Any of these call-graphs can be populated with authorship information generating the equivalent "social call-graphs". From those, sociograms are generated and displayed using JUNG (Java Universal Network/Graph Framework). Note that call-graphs and social call-graphs are directed graphs, while sociograms are undirected graphs. Ariadne is also able to export all the information that it collects as Excel files for later analysis using other social-network programs.

We envision two types of users for our tool:

- Software developers who would use it to identify colleagues with whom they need to interact, that need to be informed about changes that are going to impact them, or with similar interests. For instance, de Souza *et al.* [4] describes an example where developers who shared a dependency were performing duplicate work because they were not aware of each other. In this case, we plan to use the concept of equivalence.

- Project managers or researchers interested in understanding the interplay between the changes in the architecture of the software and its social impact. For example, by analyzing the density of a social network or by identifying bridges in this network one can understand the key role played by some software developers in the coordination process or better understand their coordination and communication needs.

4. Related Work

Despite this relationship between the technical and the associated social dependencies, traditionally, interdependence relationships have been looked at from two perspectives, either as interdependence between people (work tasks (for example Mintzberg [10]: workflow, process, scale, and social interdependencies)) or as interdependence between artifacts (for example, program dependencies [11], building mechanisms in configuration management tools [7], traceability tools). In separating people and artifacts, these perspectives provide only relatively narrow and clear-cut views on what could be assumed to be a wide variety of forms and appearances. Furthermore, as the examples below recognize, software developers in their daily work recognize the integration of those approaches and make use of them to get their work done. For instance, McDonald and Ackerman [8] describe a field study where software developers use information from their configuration management tool to identify experts in the source code that they are changing. Furthermore, research prototypes have been recently created to explore this relationship: Expertise Recommender [12] and Expertise Browser [13] aim to facilitate the process of identifying, and recommending experts in parts of the software being engineered.

5. Conclusions and Future Work

This paper described Ariadne a software tool that addresses the link between technical and social dependencies. The ties between these two aspects are based on the observation that software developers who depend on each other often need to coordinate their work.

Currently, Ariadne analyzes an Eclipse project linked to a configuration management repository. If this project is dependent on another project, Ariadne will also analyze this other project, therefore creating a larger graph of dependencies among different projects and developers. By doing that, we expect to identify the pieces of source code *and* the specific developers who act as "bridges" between different projects. Indeed, we are currently analyzing different open-source projects to understand how developers slowly move from a peripheral participation to a more central one.

6. Acknowledgements

This research was supported by the U.S. National Science Foundation under grant numbers 0133749, 0205724 and 0326105, the Intel Corporation, and by the Brazilian Government under CAPES grant BEX 1312/99-5. We also very much appreciate the comments from Steve Abrams and Rogerio DePaula on this paper.

7. References

[1] D. L. Parnas, On the Criteria to be Used in Decomposing Systems into Modules, *CACM*, vol. 15, pp. 1053-1058, 1972.

[2] G. Larman, "Protected Variation: The Importance of Being Closed," *IEEE Software*, vol. 18, pp. 89-91, 2001.

[3] R. E. Grinter, "Recomposition: Putting It All Back Together Again," Conference on Computer Supported Cooperative Work, Seattle, WA, USA, 1998.

[4] C. R. B. de Souza, D. Redmiles, L.-T. Cheng, D. Millen, and J. Patterson, "Sometimes You Need to See Through Walls - A Field Study of Application Programming Interfaces (to appear)," presented at Conference on Computer-Supported Cooperative Work (CSCW '04), Chicago, IL, USA, 2004.

[5] R. Grinter, J. Herbsleb, and D. Perry, "The Geography of Coordination: Dealing with Distance in R&D Work," presented at ACM Conference on Supporting Group Work , 1999.

[6] G. Kiczales, "Beyond the Black Box: Open Implementation," *IEEE Software*, vol. 13, pp. 8-11, 1996.

[7] R. Conradi and B. Westfechtel, "Version Models for Software Configuration Management," *ACM Computing Surveys*, vol. 30, pp. 232-282, 1998.

[8] D. W. McDonald and M. S. Ackerman, "Just Talk to Me: A Field Study of Expertise Location," presented at Conference on Computer Supported Cooperative Work, Seattle, Washington, 1998.

[9] I. T. Bowman and R. Holt, "Reconstructing Ownership Architectures To Help Understand Software Systems," presented at International Workshop on Program Comprehension, Pittsburgh, PA, USA, 1999.

[10] H. Mintzberg, *The Structuring of Organizations: A synthesis of the research*. Englewood Cliffs, NJ: Prentice-Hall, 1979.

[11] A. Podgurski and L. A. Clarke, "The Implications of Program Dependencies for Software Testing, Debugging, and Maintenance," Symposium on Software Testing, Analysis, and Verification, 1989.

[12] D. W. McDonald and M. S. Ackerman, "Expertise Recommender: A Flexible Recommendation System and Architecture," Conference on Computer Supported Cooperative Wrok, Philadelphia, PA, 2000.

[13] A. Mockus and J. D. Herbsleb, "Expertise Browser: A Quantitative Approach to Identifying Expertise," International Conference on Software Engineering, Orlando, FL, USA, 2002.