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 discuss briefly related work.

1. Introduction

One of the most important and influential principles in software engineering is the idea of information hiding proposed by Parnas [13]. According to this principle, software modules should be both “open (for extension and adaptation) and closed (to avoid modifications that affect clients)” [7]. Information hiding aims to decrease the dependency (or coupling) between two modules so that changes to one to 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 give 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 integrate these pieces in order to create the whole software. This work of building the whole software from its parts requires a lot of coordination effort [5] [3] [4]. Reconstructing the system from its pieces happens often for testing and integration purposes during both development and maintenance phases. Furthermore, 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 users; somehow it 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 components - suggest that software developers working on the implementation of these modules still need to interact regularly to make sure that their work is aligned, so that the integration of these modules flows smoothly when necessary. In other words, because software modules interact, this creates a similar need of interaction among the software developers implementing them. In other words, technical dependencies between modules create social dependencies between the software developers implementing these modules [3, 5]. Based on this observation, we argue that the source-code itself can be an important resource to identify social relationships that need to be built among software developers to facilitate the integration process, because it contains information about the technical dependencies among pieces of software. This paper then describes our approach, and associated tool – Ariadne, 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 (social dependencies) is integrated with the technical dependencies by collecting information from a configuration management repository associated with the Java program,

2. From Technical to Social Dependencies

Our approach combines source-code analysis and collection of information about the software developers in order to appropriately describe 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, let’s say that a component *A* uses the services of another 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 another 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* which other components. Figure 1 below presents an example of a call graph of a small Java program. 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.

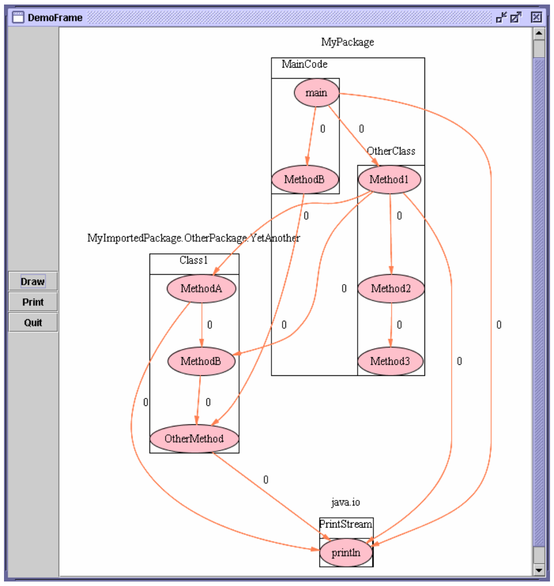


Figure 1: An example of a call graph

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 a component *A* depends on another 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*. However, 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

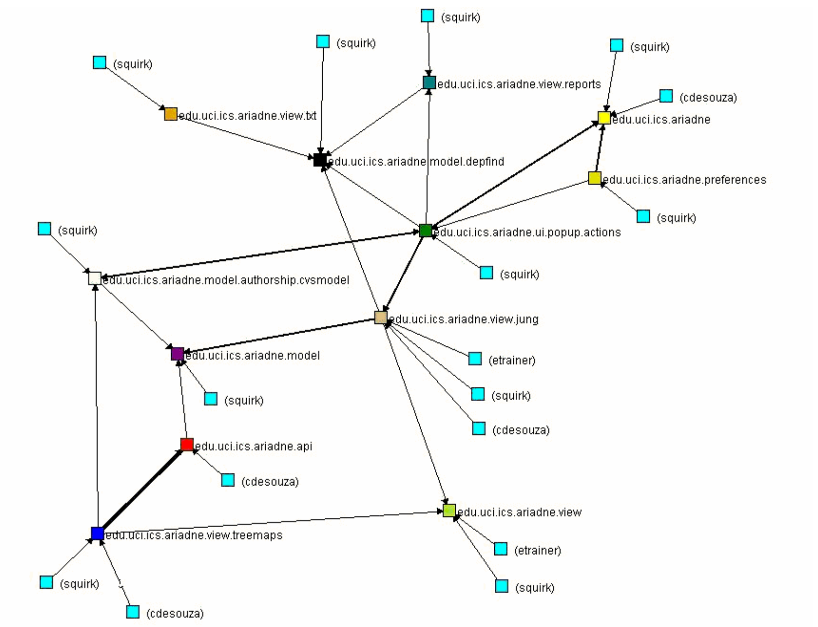
Authorship information about each node of the call-graph can be extracted from a configuration management (CM) repository, because such a tool contains revision information about each and every change made to the software system being analyzed [2]. Typical revision information for each change includes: the changes applied to the software, date and time of these changes, author of the changes, the files where the changes were applied, among others. Combining information from the call-graph with authorship information present in the CM repository can then create a "social call-graph", which describes which software developers depend on which other software developers for a given piece of code. Figure 2 below presents an example of a "social call-graph" from a small software development project being conducted at UCI. A directed edge from package A to B indicates a dependency from A to B. Directed edges between authors and packages indicated 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 [10], in another company, ownership architectures could be used since it documents the relationship between developers and source code [1]. The "social call-graph" is based on the actor-network approach proposed by Latour [8], where networks of artifacts and human (both called "actants") are represented together.

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 have available, "social call-graphs" could easily generate social network graphs describing the dependency relationship among software developers *without* depicting dependencies among software components. Figure 3 below presents an example of such situation. This example is based on our own fieldwork [3] through interviews and non-participant observation. Each point represents a software developer member of a team. Members of the client team are represented by *cN*, where N is an integer from 1 to 8. Similarly, members of the server team are represented by *sN*, and finally, members of test team are presented by *tN*. The other letters (*n*, *d* and *a*) indicate other teams in the organization. Arrows indicate dependency relationships from the source to the target of the arrow, for example, developer *c2* depends on developer *s1*.

Currently we are investigating the usage 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 be able 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.



(squirk)
edu.uci.ics.ariadne.view.txt
edu.uci.ics.ariadne.model.depfind
edu.uci.ics.ariadne.view.reports
(cdesouza)
edu.uci.ics.ariadne
edu.uci.ics.ariadne.preferences
edu.uci.ics.ariadne.ui.popup.actions
edu.uci.ics.ariadne.model.authorship.cvsmodel
edu.uci.ics.ariadne.view.jung
edu.uci.ics.ariadne.model
(etrainer)
edu.uci.ics.ariadne.api
edu.uci.ics.ariadne.view
edu.uci.ics.ariadne.view.treemaps

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

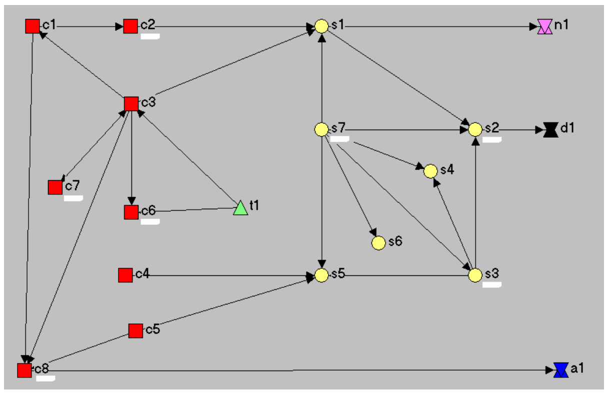


Figure 3: An example of a social network graph describing dependency relationships among software developers

3. Ariadne

Ariadne is a Java-based tool that creates technical, social, and social-technical graphs, that is, call-graphs, social network graphs and "social-call graphs". Ariadne is currently implemented as a plugin 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 plugins. Therefore, one can create plugins 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 get 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. This call graph is then populated with authorship information. Our current implementation can present social-call graphs at three different levels of abstraction: methods, files, and packages. Basically, information from the methods is aggregated to generate information about the files, and similarly information about the files is aggregated to generate information about the packages. Furthermore, the social network diagrams are being generated using JUNG and Ariadne is also able to export the information that it collects as Excel files.

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, as in the situation described in [3] where developers who shared a dependency where performing duplicate work because they were not aware of each other. In this case, our approach is similar to the one adopted in the ExpertiseBrowser system [12] described in the next section.
- Project managers or researchers interested understanding the interplay between the changes in the architecture of the software and it social impact. This approach is similar to the one proposed by (Ducheneaut, Mahendran, and Sack, 2002). For example, by analyzing the density of a social network or by identifying bridges in this network we can understand the key role played by some software developers 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 [11]: workflow, process, scale, and social interdependencies)) or as interdependence between artifacts (for example, program dependencies [14], building mechanisms in configuration management tools [2], 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. Additional examples can be found in the literature. For instance, McDonald and Ackerman [10] 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 [9] and Expertise Browser [12] 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. As of right now, our tool analyzes only one project in Eclipse each time. Currently, we are working on extending our tool so that if the project being analyzed is dependent on another project also in Eclipse, 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 source code and the specific developers who act as "bridges" between different projects.

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.

7. References

[1] Bowman, I. T. and Holt, R., "Reconstructing Ownership Architectures To Help Understand Software Systems," International Workshop on Program Comprehension, pp. 28-37, Pittsburgh, PA, USA, 1999.

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

[3] de Souza, C. R. B., Redmiles, D., et al., "Sometimes You Need to See Through Walls - A Field Study of Application Programming Interfaces (to appear)," Conference on Computer-Supported Cooperative Work (CSCW '04), Chicago, IL, USA, 2004.

[4] Grinter, R., Herbsleb, J., et al., "The Geography of Coordination: Dealing with Distance in R&D Work," ACM Conference on Supporting Group Work (GROUP '99), Phoenix, AZ, 1999.

[5] Grinter, R. E., "Recomposition: Putting It All Back Together Again," Conference on Computer Supported Cooperative Work, pp. 393-402, Seattle, WA, USA, 1998.

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

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

[8] Latour, B., "Where are the missing masses? The sociology of a few mundane artifacts.," in *Shaping Technology / Building Society: Studies in Sociotechnical Change*, W. Bijker and J. Law, Eds. Cambridge, MA: MIT Press, 1994, pp. 225-258.

[9] McDonald, D. W. and Ackerman, M. S., "Expertise Recommender: A Flexible Recommendation System and Architecture," Conference on Computer Supported Cooperative WORK (CSCW '00), pp. 231-240, Philadelphia, PA, 2000.

[10] McDonald, D. W. and Ackerman, M. S., "Just Talk to Me: A Field Study of Expertise Location," Conference on Computer Supported Cooperative Work (CSCW '98), pp. 315-324, Seattle, Washington, 1998.

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

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

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

[14] Podgurski, A. and Clarke, L. A., "The Implications of Program Dependencies for Software Testing, Debugging, and Maintenance," Symposium on Software Testing, Analysis, and Verification, pp. 168-178, 1989.