

Towards an Infrastructure for Software Visualization Research

Erik Trainer and David Redmiles

Donald Bren School of Information and Computer Science
University of California, Irvine
Irvine, CA, USA 92667
{etrainer, redmiles@ics.uci.edu}

ABSTRACT

One way to view the complexity of a software system is in terms of the dependencies between its modules. Empirical studies have shown that dependencies between source-code modules create coordination needs among developers working on those modules. As described in our previous work, we developed Ariadne, a dependency plug-in infrastructure for the Eclipse IDE. We chose the Eclipse infrastructure so we could leverage its services as well as make our own prototype general enough to work with different programming languages, CM repositories, and visualizations. In this current work, we report on the choices that drove the integration of our tool with Eclipse, some of the problems we faced prototyping visualizations, and some suggested improvements to both the infrastructure and research tools based on our experiences. These observations can serve as an initial step toward bridging new research on software visualization with data mined from collaboration infrastructures.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – user interfaces. H.5.3 [Information Interfaces and Presentation]: Group and Organization Interfaces – *collaborative computing, computer-supported cooperative work*; H.1.2 [Models and Principles]: User/Machine Systems – *human factors, human information processing*.

General Terms

Design, Experimentation, Human Factors.

Keywords

Coordination, Dependencies, Infrastructure, Awareness, Visualization, Collaborative development environments

1. INTRODUCTION

Coordination is a large part of the work that software developers must undergo throughout the life cycle of a software development project. One of the main reasons coordination proves to be a challenge is the number of dependencies that software engineers

must face, not only between activities in the software development process, but among system modules as well. For example, the "calls" relationship in software is a type of dependency between artifacts: module A calls module B. Empirical studies, including our own, have concluded that this particular dependency relationship creates dependencies between the people working on these modules as well [2, 3, 5, 6].

Managing dependencies is not an easy task, especially when people are distributed over space in time rather than collocated. To support issues surrounding collaborative software development such as awareness of colleagues' activities and finding the right people to contact about specific software modules, we developed a dependency management infrastructure called Ariadne. In our most recent work to date, we have designed and implemented a novel visualization as a plug-in to Ariadne and evaluated it with principled usability inspection methods [8]. Additionally we have conducted several informal trials with users.

Since 2005 we have maintained different versions of Ariadne in the form of Eclipse (<http://www.eclipse.org>) [9] and more recently, Jazz (<http://www.jazz.net>), plug-ins. In this paper, we report on our experiences integrating the tool with Eclipse and third-party visualization frameworks. We describe some of the benefits of leveraging these tools for researchers, but also some of the barriers as well, including tradeoffs between flexibility and learnability, and gaps between infrastructure for development and experimental visualization frameworks. Last, we conclude with some suggestions for improvements to the discussed infrastructure.

2. ARIADNE

Ariadne integrates different components according to a layered architecture style. It consists of a core dependency management plug-in and several client plug-ins: dependency generation, authorship annotation, and visualization (Figure 1).

A core plug-in manages the linking of the dependency generation and authorship annotation plug-ins to create a social dependency graph consisting of dependencies between developers through the artifacts they work on [5]. The core plug-in listens to visualizations supplied by the visualization plug-in and displays dependencies in its specified format.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Conference '04, Month 1–2, 2004, City, State, Country.

Copyright 2004 ACM 1 58113 000 0/00/0004 \$5.00

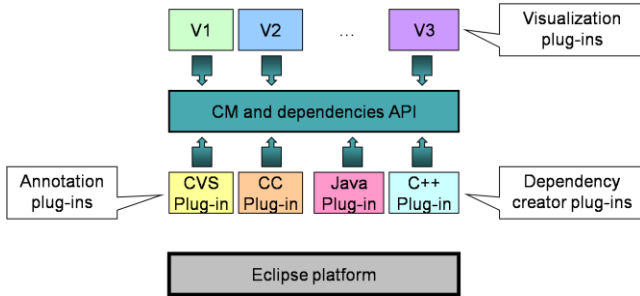


Figure 1. The “CM and dependencies API” core plug-in sits on top of the Eclipse infrastructure and manages three types of client plug-ins.

Because different services correspond to plug-ins and the core plug-in hides concerns like linking dependencies and authorship from its clients, Ariadne is general enough to support various programming languages, CM repositories, and forms of visualization. By default, we have provided Java dependency generation, CVS, SVN, and Jazz SCM authorship extractors, social network visualizations [5], and our most recent visualization [8]. Most recently, the Jazz SCM authorship extractor was implemented as a plug-in and integrated into Ariadne with minimal effort. We feel that this experience can be attributed to the extensibility of Ariadne’s architecture.

3. INFRASTRUCTURE CHOICES

3.1 Requirements

There were several requirements for Ariadne in the early stages of design. First the tool needed to be integrated with a development environment. Second, the tool needed interfaces to CM repositories to retrieve authorship information and to the actual source-code in the development environment to calculate dependencies. Third, the tool needed a set of interfaces to graphics libraries for visualizations of dependency data. We found existing graphics libraries to be insufficient and eventually experimented with different visualization frameworks. However, this led us outside the Eclipse infrastructure, requiring us to give up a certain amount of integration with the tool. In the following subsections we report on the tradeoffs involved.

3.2 The availability of APIs

Ariadne has taken the form of an Eclipse plug-in since its first implementation [9]. This decision was driven by Eclipse’s large user base and its corpus of user-generated plug-ins at the time (some developed specifically for academic research). Eclipse also provided a medium through which we could deploy our software tools to developers, the target users of Ariadne. First and foremost, however, the combination of Eclipse’s plug-in mechanisms and wealth of available APIs lowered the barrier to developing Ariadne, because so much in the Eclipse environment could be reused.

The availability of published APIs allowed us to reuse certain mechanisms built into Eclipse’s infrastructure, such as the SearchEngine class to calculate dependencies and Team APIs to mine CM repositories for source-code authorship. Hooks into different GUI components such as perspectives, views, and dialogs made it easy to give Ariadne the same “look and feel” as the editors and other user interface components in Eclipse.

Additionally, we were able to reuse the Eclipse Resource Model to represent much of the artifact data used by Ariadne as well as import and export dependencies using in-place Eclipse storage mechanisms.

However even a good infrastructure will not always provide the APIs tool building requires. Sometimes services are not offered because their design has not been finalized. These services regularly undergo change and can thus break client code that relies on them. As such, researchers use them at a risk. In fact, in the design of one of our plug-ins, we had to use unpublished Team APIs provided by Eclipse that ended up changing later and breaking our code as a result.

Other times, certain services the researcher desires may not be available at all and must be implemented from scratch or by extending existing services. Indeed, in order to handle Eclipse resources in remote CM repositories, we had to implement our own remote resource API.

3.3 Moving outside the infrastructure

The first few prototype visualizations we implemented in Ariadne were fully integrated into the Eclipse editor (Figure 2). We used

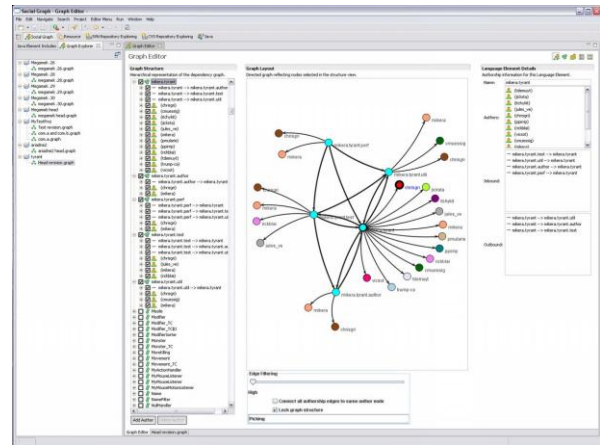


Figure 2. A prototype visualization integrated into an Eclipse editor.

social network visualizations to show dependencies in the system’s architecture annotated with developers working on those components. However, due to the variability of graph layout algorithms and the limited scalability of node-edge visualizations, we developed a second visualization (Figure 3).

We looked at some of the graphic support in the Eclipse APIs but soon realized that some of the graphical frameworks such as GEF (Graphical Editing Framework) would not be as easy to use with the abstract shapes, and query support we envisioned needing. These libraries provide only a basic common set of shapes that were too general for our purposes. The overhead in learning how to use the framework also discouraged us.

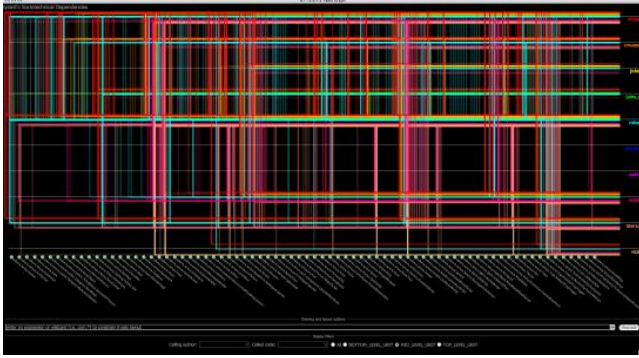


Figure 3. A new visualization launched separately from Eclipse.

With this observation in mind, we experimented with various lightweight open-source visualization frameworks such as JUNG (<http://jung.sourceforge.net>) and Prefuse (<http://www.prefuse.org>). JUNG is a graphing package that comes with automated algorithms for computing social network metrics. However, it only produces visualizations that can be represented by nodes and edges. These visualizations constrain the extent to which one can express all elements of a socio-technical relationship [8].

Next we experimented with Prefuse, another visualization toolkit. It allowed us much more freedom in coming up with abstract representations for dependency data, provided fast and powerful query mechanisms and data indexing, and required little overhead in terms of learning how to use the tool. However, integrating the visualization into Eclipse proved to be challenging, particularly because of cross-platform issues with SWT and Swing. As a result, we had to launch the visualization in a separate window, outside of the development editor (Figure 3).

4. DISCUSSION

By and large, software development is practiced in industry. Teams of individuals, detailed design artifacts, and committal to a process of development are things of an organization, not of academic researchers. We thrive on early prototyping and proof of concept to get across our ideas. The learning curve associated with an infrastructure’s API is daunting.

The fact that we reused much of the APIs and services in Eclipse illustrates its powerful flexibility. However, flexibility comes at a price. In this instance, the price was in the form of a learning curve associated with APIs and the plug-in mechanism. Many of the APIs we ended up reusing were for integrating back-end services of Ariadne like data storage, dependency generation and CM repository management. Despite this reuse, the infrastructure’s flexibility only got us so far. Once we implemented several visualizations, we lost the ability to integrate them into the Eclipse environment.

The Eclipse infrastructure is meant for software development. As such, it provides integrated support for code editors, dependency generation, source-code versioning, and basic graphical APIs for software-specific activities, such as UML class diagramming. However, Eclipse does not provide support for prototyping visualizations beyond those long-familiar to software developers; the existing set of visual representations is too simple.

On the other hand, while third party visualization toolkits can be more powerful in terms of the kinds of visualizations they can produce, they are difficult to reconcile with domain-specific infrastructures like Eclipse. JUNG ended up to be a tool that was too oriented toward social network analysis and was also limited in the types of visualizations it could produce. Prefuse worked better in that we could come up with a customized visualization with little effort but lost the ability to interact with the visualization and the infrastructure workspace at the same time.

We believe these issues are indicative of a larger problem: trying to reconcile a platform for software development and visualization at the same time. The third-party tools’ APIs are too general to support activities grounded in software development and are difficult to integrate into existing workflows. On the other hand, the graphic APIs provided by Eclipse are too limited in their expressiveness.

This disconnect poses a barrier for researchers looking to get the most out of their visualizations, especially from data provided by novel infrastructures for collaboration, such as Jazz. With the move toward these platforms [1, 7], infrastructures can now store more information about the team as a whole rather than the individual, such as who is assigned to bug reports and implementation requests, as well as which team members are sharing their work versus which are not. Although this data will soon be available to researchers looking to better understand team coordination strategies and problems [4], the research community should be mindful of the interplay between infrastructure and visualization as they build their tools. Getting over learning curves is a frustrating process, but in our experience, it is worth the benefits.

We propose a visualization editor in Eclipse that can be used by researchers to rapidly prototype visualizations over workspace data, such as dependencies in the architecture and activity in project repositories. One possible way might be to provide a visualization abstraction layer between back-end Eclipse services and third party visualization frameworks. This way, researchers using the visualization editor would have access to all the same types of services provided by Eclipse as well as a way to experiment with visualizations they know can be cleanly integrated with the workspace. While this integration would still come with the price of learning to use infrastructure APIs, it would be a first step toward bridging the gap between development environment and visualization environment.

5. CONCLUSIONS AND FUTURE WORK

In our work to date, we have built Ariadne, a dependency infrastructure for the Eclipse development environment. The plug-in paradigm used by the Eclipse infrastructure allowed us to hook in to many key services provided by the environment via APIs, but came with a learning curve.

Infrastructure has its limits, particularly with respect to visualization. Our experiences led us to move beyond the infrastructure, but we soon ran into limitations with the compatibility of third party visualization toolkit APIs with Eclipse.

Reflecting on these limitations led us to observe a disconnect between the scope of the graphical support APIs of a development platform’s infrastructure and abstract visualization toolkits. We

suggested that a bridging mechanism such as an abstraction layer might serve to better integrate collaborative infrastructures and visualization research.

In our future work we hope to evaluate the usability of Ariadne for visualizing data stored by collaborative development environments such as Jazz, as we expect that many researchers will be interested in visualizing some of the same data. As such, our work serves as a starting point for researchers wishing to integrate their own visualizations with the state-of-the-art. Moreover, they can improve upon the suggestions and tradeoffs with respect to visualizations for development environments discussed here.

6. ACKNOWLEDGMENTS

This research is supported by the U.S. National Science Foundation under grants 0534775 and 0205724, and by an IBM Eclipse Technology Exchange grant. The authors gratefully acknowledge past and present collaboration with Cleidson de Souza and feedback from research group member Roberto Silveira Silva Filho

7. REFERENCES

- [1] Booch, G. and Brown, A.W., "Collaborative Development Environments", in *Advances in Computers*, vol. 59, Academic Press, 2003.
- [2] Cataldo, M., Wagstrom, P.A., Herbsleb, J.D., and Carley, K., "Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools", In *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, ACM, New York, NY, 2006, pp. 353-362.
- [3] Curtis, B., Krasner, H., and Iscoe, N., "A Field Study of the Software Design Process for Large Systems", *Communications of the ACM*, ACM, New York, NY, 1988, 31 (11), pp. 1268-1287.
- [4] Damian, D., Marczak, S., and Kwan, I., "Collaboration Patterns and the Impact of Distance on Awareness in Requirements-Centered Social Networks", in *Proceedings of the IEEE Requirements Engineering Conference*, IEEE Computer Society, Los Alamitos, CA, 2007, pp. 59-68.
- [5] de Souza, C.R.B., Quirk, S., Trainer, E., and Redmiles, D.F., "Supporting Collaborative Software Development through the Visualization of Socio-Technical Dependencies", In *Proceedings of the 2007 International ACM Conference on Supporting Group Work*, ACM, New York, NY, 2007, pp. 147-156.
- [6] Herbsleb, J.D. and Grinter, R.E., "Architectures, Coordination, and Distance: Conway's Law and Beyond", *IEEE Software*, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 63-70.
- [7] Hupfer, S., Cheng, L., Ross, S., and Patterson, J., "Introducing Collaboration into an Application Development Environment", In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, ACM, New York, NY, 2004, pp. 21-24.
- [8] Trainer, E., Quirk, S., de Souza, C.R.B., and Redmiles, D.F., "Analyzing a Socio-Technical Visualization Tool Using Usability Inspection Methods", In *Proceedings of the IEEE Symposium on Visual Languages and Human Centric Computing*, IEEE Computer Society, Washington, DC, 2008, in press.
- [9] Trainer, E., Quirk, S., de Souza, C. R. B., and Redmiles, D.F., "Bridging the Gap between Technical and Social Dependencies with Ariadne," In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology Exchange*, ACM, New York, pp. 26-30.