Adinda: A Knowledgeable, Browser-Based IDE

Arie van Deursen
Delft University of Technology
Arie.vanDeursen@tudelft.nl

Ali Mesbah
Delft University of Technology
a.mesbah@tudelft.nl

Bas Cornelissen
Software Improvement Group
Amsterdam, The Netherlands
b.cornelissen@sig.nl

Andy Zaidman
Delft University of Technology
a.e.zaidman@tudelft.nl

Martin Pinzger
Delft University of Technology
m.pinzger@tudelft.nl

Anja Guzzi
Delft University of Technology
a.guzzi@tudelft.nl

ABSTRACT

In practice, many people have to work together to develop and maintain a software system. However, the programmer's key tool, the Integrated Development Environment (IDE), is a solo-tool, serving to help individual programmers understand and modify the system. Such an IDE does not leverage the knowledge other team members may have of the design and implementation of the system. We propose to resolve this problem by exploring, experimentally, new ways of inferring knowledge from past IDE-interactions, and of maximizing collaboration among developers. Our approach, called ADINDA, revolves around transforming the IDE into a set of integrated services, accessible via a web browser, and enriched with Web 2.0 technologies. Such services will not only help developers perform traditional IDE tasks, but also facilitate the required informal communication and collaboration needs of software development projects. In this paper, we report on our vision, approach and challenges for building ADINDA, and initial results.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Programming Environments

Keywords

IDE, Web 2.0, collaboration, interaction mining

1. INTRODUCTION

Software engineering is a team sport: sometimes hundreds of professionals collaborate to devise, build, evaluate, and later modify a software system [16]. But the programmer's key tool, the *Integrated Development Environment* (IDE), is a soloist tool. It primarily helps individual programmers to be more effective during the classical edit-compile-run cycle [17]. However, it does not help in answering important questions developers might have such as: Which of my team mates worked on this piece of code before? What other parts has this person changed in the past? How many of the co-developers are working on the project code at this moment? Who is modifying what part of the system? Can I get real-time feedback on the changes they are making to the code?

While answers to these developer questions may be available in the minds of certain team members, the underlying knowledge is often left implicit, and unavailable to other team members. Most of the reasoning leading to a particular piece of code gets lost, leaving only the new code itself as result of a complex program comprehension process.

This leads to a number of research hypotheses. The first is that making the knowledge leading to code modifications explicit, and sharing it among all team members, will lead to a significant increase in productivity and software quality.

Our second hypothesis is that a large part of this knowledge can be inferred automatically from data collected by developer tools. This requires that the IDE monitors developer activity, stores this information in a shared repository, and uses it to support co-developers.

Our third hypothesis is that engineers themselves can help make such knowledge available, and that light-weight knowledge-sharing techniques can minimize the time used for this activity. In the domain of the Web 2.0 [13], mechanisms such as tagging, micro-blogging, and virtual presence have gained immense popularity. These could be integrated into the development environment, which, combined with the aforementioned knowledge collecting capabilities, would make it significantly easier for developers to stay informed of the activities of other team members.

To evaluate these hypotheses, we are constructing a research prototype of a development environment, codenamed ADINDA, incorporating automated knowledge collection and sharing support.

2. THE ADINDA VISION

To facilitate knowledge collection and sharing among different users, we propose a radical restructuring of the traditional IDE, transforming it into a set of cooperating services. Thus, ADINDA comprises a (thin) client connected to a range of different services for conducting developer tasks such as compiling, editing, testing, as well as collaboration-related services, such as task management, joint editing, tagging, social networking, data mining, and knowledge sharing. The centralized services collect all sorts of information concerning the actual development activities of all team members, and use this to assist individual developers.

When combining this approach with Web 2.0 technologies such as AJAX [10], the opportunity of a fully browser-based IDE emerges,

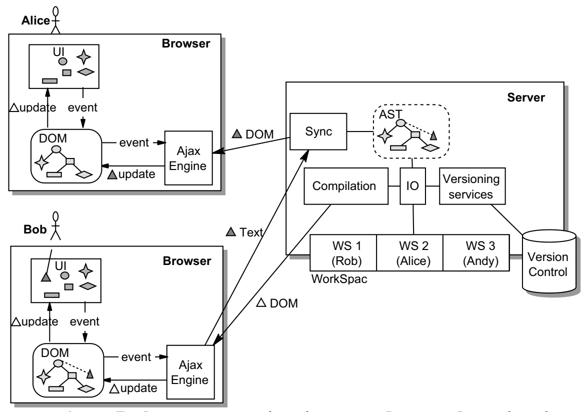


Figure 1: Delta-communication and synchronization in ADINDA.

not only offering collaboration and assistance benefits to the IDE, but also universal access and configuration sharing.

3. RESEARCH CHALLENGES

In order to realize the vision of such a knowledgeable development environment, the following research challenges have to be faced:

3.1 Integrated Software Development Services

The construction of the ADINDA core itself requires addressing a wide variety of challenges. The key to building a browser-based IDE lies in designing a server that can be used to create, store, manipulate, compile, test, and share programs interactively.

Client/server interaction. The client/server architecture we envision for ADINDA is illustrated in Figure 1. A sketched scenario in Figure 1 is when a developer Bob changes, for instance, a Java class in the browser. The delta changes are then propagated to the server, where the corresponding class file is updated accordingly, compiled, and the compilation message is returned to the browser. The main architectural question is concerned with what program representation the client and server should use. We are investigating whether the browser's Document Object Model (DOM) can be reused to represent programs as Abstract Syntax Trees (ASTs), to exchange only modified program text and corresponding AST subtrees between client and server.

Change propagation. Figure 1 depicts how the changes made by Bob are propagated to Alice via ADINDA server's synchronization unit. The server-based approach opens up possibilities for multiple users working together on a single resource. In the specific context of using the browser as an IDE, we are investigating real-time synchronization methods that offer the best support for collaborative editing. Interesting questions that emerge include: whether the client/server delta-communication should be based on client pull or server push to increase data coherence? How does network latency influence the synchronization process? How should conflicts be resolved? Can text-based merge algorithms be extended to tree or graph-based merge algorithms? Which update strategy is the most efficient and convenient to developers?

Versioning method. Traditional version control systems operate at the file level and involve an update-edit-commit cycle [6]. The server-based nature of ADINDA opens up possibilities for much finer version control methods. The challenge here is finding the best granularity level (e.g., abstracter levels than text or lines, or methods, classes, models, or even process steps [8]) and time frame (e.g., after each edit, each save, automatic or explicit) for committing changes into the repository. The interesting question is how such a versioning approach can support task and user awareness, providing, e.g., the possibility to follow all changes made by particular developers or replay changes made by a different developer.

3.2 Communication and Collaboration

On the Internet, new forms of informal communication have emerged in what has become known as Web 2.0 [13]. Technologies such as wikis, micro-blogging, tags, and feeds help us organize, manage, and categorize web content in an informal and collaborative way. We believe Web 2.0 collaboration concepts and techniques can be applied to support the required informal communication and collaboration needs of software development projects.

Cost-effective tagging strategies in software development. Tags are surprisingly simple, yet effective, and their use on the Internet has been subject to various studies [12, 1]. Tags are starting to become part of software development tools: in IBM's Jazz [5], for example, work items can be tagged, and its use in practice has recently been investigated by [15]. The questions that we are focusing on include: What information are developers willing to tag? Should tags be attached to entities such as work items (as in IBM Jazz), or are free format hash-tags in comments (as in Yammer or Twitter) more appropriate? Does the usefulness of tagging depend on the team size? What tagging tool support (e.g., tag completion, renaming, versioning, following, managing) is required? To what extent can tags help programmers categorize and find the information they are looking for? At what source code level should tags be applied (e.g., class files, methods, statements, ...)?

Micro-blogging for traceability purposes. A major issue in software engineering is *requirements traceability*, aimed at tracking which requirements are responsible for a particular design decision, code fragment, test case, etc. Micro-blogging, best known via Twitter, offers a light-weight mechanism to inform others of "what you are doing." Our main concern here is finding new ways in which micro-blogging can facilitate coordination and communication during software development.

In addition, software development is a highly dynamic activity, and as software evolves, features are completed, code is refactored, methods are modified or discarded, and bugs are fixed. This rises the question of: What implications does this evolution have for user-created data such as tags or micro-blog posts?

Active Participation. A method that has been applied successfully in modern Web 2.0 news websites such as Digg[1] and Reddit[2] is allowing the registered users to vote (up or down) and comment on the posted news items, rendering them more, or less prominent on the website. In addition, to encourage active participation users can earn *reputation points* depending on the quality and quantity of their activities. For example, a user of the StackOverflow[3] website is awarded 10 reputation points for receiving an 'up' vote on an answer given correctly to a question posted by someone else. Thus, users are motivated to contribute and improve their reputation. A question that arises is how such concepts of voting and reputation building can be applied to software development for encouraging developers to participate in, for instance writing good documentation (which is always a challenge) and answering other developers' questions.

3.3 Interaction Mining

The server-based setup of ADINDA enables the tracking of all developer activities: the server can collect knowledge concerning who opened certain files and in which order, which classes were

[1]`http://digg.com`
[2]`http://www.reddit.com`
[3]`http://stackoverflow.com`

often inspected or modified together, and so on. With this knowledge in hand, the system is capable of making recommendations that help developers be more effective. Taking advantage of the wealth of developer activity data available on the ADINDA server involves addressing the following research questions:

Tracking activities. The server-based setup permits the collection of all sorts of data on developer activity. Which user activities should be tracked and stored? What data-model should be used for this purpose? Which activities (when traced) are likely to yield meaningful information? At what level of abstraction and aggregation should the activities be recorded?

Understanding code dependencies. Understanding code dependencies is a prerequisite for many software engineering tasks, and many faults are due to inadequate understanding of such dependencies. Code dependencies can be both explicit (e.g., method calls) and implicit (being, e.g., part of a larger design pattern).

The search for hidden dependencies, particularly those that *crosscut* the primary modularization, is an active area of research. Commit-level repository mining has been applied for these purposes as well [4], but interaction mining at the fine-grained level of individual edits is a more promising direction that warrants further investigation.

Recommend programming code inspection. A significant part of programming is spent on program comprehension and code navigation. In Web 2.0 systems such as LinkedIn or Digg, users are guided through the application by means of advice such as "Viewers of this profile also viewed ...". [18] have applied data mining techniques to provide similar behavior based on commits stored in the source control system.

Is it beneficial to come up with such advice based on stored editing behavior? How useful are such recommendations? How can the IDE be extended so that it can distinguish between successful trails (actually leading to the desired insight or program change) or unsuccessful ones (paths that were explored, but did not help in the task at hand)?

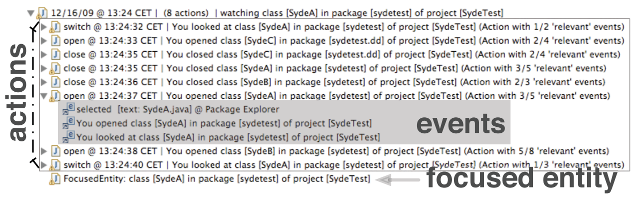
Team members' knowledge and activities. Finding the right developer to conduct a certain task in the project is an important research question for software engineers: For instance, to assist teams in spotting the best developer to fix a certain bug, [2] propose to analyze each developer's past activities in the bug tracking repositories. By tracking all user activities in ADINDA, we can go much further and analyze interaction patterns between the developers. Key questions that emerge include: Who is working on or looking at which parts of the system? What have they been doing exactly? Which of the team members has knowledge about the code a developer is working at? Who has made most of the changes in this particular file?

4. APPROACH AND INITIAL RESULTS

To address the range of challenges listed, we have conducted a number of experiments. Our main activity is to implement an initial prototype of ADINDA as discussed below. Furthermore, in order to be able to conduct early experiments, we are implementing Eclipse plugins to try out some of our ideas in an existing IDE, discussed in Section 4.2.

4.1 Implementing ADINDA

Based on an earlier prototype for a web-based IDE called WWWorkspace [14], we have implemented an initial prototype of ADINDA. This IDE establishes a connection with Eclipse on the server-side. The editors operate on the browser's DOM-tree using AJAX. In the current prototype version, it is possible to create user workspaces, Java projects, packages, and class files through the browser-based interface. There is support for syntax highlighting,



```
12/16/09 @ 13:24 CET |  (8 actions)  | watching class [SydeA] in package [sydetest] of project [SydeTest]
  switch @ 13:24:32 CET | You looked at class [SydeA] in package [sydetest] of project [SydeTest] (Action with 1/2 'relevant' events)
  open @ 13:24:33 CET | You opened class [SydeC] in package [sydetest.dd] of project [SydeTest] (Action with 2/4 'relevant' events)
  close @ 13:24:35 CET | You closed class [SydeC] in package [sydetest.dd] of project [SydeTest] (Action with 2/4 'relevant' events)
  close @ 13:24:35 CET | You closed class [SydeA] in package [sydetest] of project [SydeTest] (Action with 3/5 'relevant' events)
  close @ 13:24:36 CET | You closed class [SydeB] in package [sydetest] of project [SydeTest] (Action with 2/3 'relevant' events)
  open @ 13:24:37 CET | You opened class [SydeA] in package [sydetest] of project [SydeTest] (Action with 3/5 'relevant' events)
    selected  [text: SydeA.java] @ Package Explorer
    You opened class [SydeA] in package [sydetest] of project [SydeTest]
    You looked at class [SydeA] in package [sydetest] of project [SydeTest]
  open @ 13:24:38 CET | You opened class [SydeB] in package [sydetest] of project [SydeTest] (Action with 5/8 'relevant' events)
  switch @ 13:24:40 CET | You looked at class [SydeA] in package [sydetest] of project [SydeTest] (Action with 1/3 'relevant' events)
  FocusedEntity: class [SydeA] in package [sydetest] of project [SydeTest]
```

Figure 2: An interaction recorded in James.

code compilation (which is done on the server, see Figure 1), and code completion in the browser.

The primary objective of our implementation effort is to be able to experiment with solutions to the various challenges listed in Section 3. Thus, we are not trying to rebuild a complete IDE: we focus on solely on IDE functionality required for our experiments. Furthermore, in our implementation we reuse existing components where possible, such as Eclipse plugins and APIs in the server or AJAX editors running in the browser.

4.2 Experimental Collaborative Program Comprehension

Driven by curiosity concerning the opportunities of merging Web 2.0 and interaction mining, we also conducted some experiments within an existing IDE (Eclipse), allowing us to obtain initial results as long as our browser-based IDE is work in progress.

In particular, we have created James, an Eclipse plugin aiming at combining status messages as in Twitter ("What's happening?") with interaction data collected from developers navigating through the code. Users are requested to explicitly tell the plugin what they are doing in the form of a short, twitter-like, message, indicating their *quest goal*. Their search and navigation is recorded by the plugin via an *interaction trace*. Furthermore, users are requested to indicate whether their journey through the code has been successful. Finally, users can add annotations to every piece of information, adding more value to this shared knowledge about the system under analysis.

The information gathered is made available to other developers. In fact, James will suggest where to look in the code, given their goal and their private navigation history. The quest goal in combination with successful interaction traces is furthermore presented in a style resembling a Frequently-Asked-Questions list.

James' implementation relies on listeners capturing single low level (Eclipse) *events*, which are then grouped in *actions* (such as "file X has been opened"). The events recorded at the moment of writing consist of events relative to UI parts in Eclipse and to the selection of pieces of code. Actions are grouped in a single *interaction* through timers, modeling the fact that people takes a few instants to decide on what to focus on. An interaction also stores the entity on which the user focused at last.

In Figure 2 we can see an example of an interaction, as currently displayed by James. All the interactions made while a quest goal message is set, will form the *interactions trace* for that particular quest goal.

Among the challenges to be addressed, one important issue is finding meaningful criteria for the identification of information relevant to the user's goal (as opposite to "noise" generated by browsing irrelevant parts of the system). Once the information is filtered, it needs to be merged with the previously recorded data (collected from many users). The merging of the data gathered from different users is another point of investigation (i.e. traces from different users working toward the same goal, can have a different importance, based on some yet to be defined metrics), as well as the scalability to the possibly huge amount of data collected. Another important research question to be tackled is how the collected data can survive code refactoring, maintaining its valuable information.

From our preliminary usage of the James tool, we are trying to answer the research question: "Does the collected data help to understand a certain piece of code or concept?". As a first experiment, we are planning to collect quest messages and relative interactions traces from a group of users while they perform given tasks on a small to medium size system. We will categorize users in groups, for example: experts and newbies of the code. This data will be analyzed and will then serve us to better understand whether and how the various users' interaction traces that led to the successful completion of a task matches with each other. A close similarity between interaction traces (or parts of them) will help answering our research question.

5. RELATED WORK

The research directions that we propose builds upon a number of rich existing research areas. Particularly relevant are the fields of collaborative software engineering [16], integrated development environments [17] and repository mining [11]. The proposed approach is furthermore relevant for the field of globally distributed software engineering — for an overview of existing tool support in this area we refer to [9].

Apart from bringing these fields together, ADINDA aims at leveraging the Web 2.0 for software development. This is an emerging area, with initial results by, e.g., [3, 7]. Furthermore, existing tools such as IBM Jazz and Microsoft Team Foundation Server are integrating web technologies such as virtual presence into the software development tool suites.

In the open source domain, Bespin[4] is a Mozilla initiative leveraging the HTML5 canvas tag to experiment with a program editor that runs in the browser. Furthermore Eclipse E4 plans[5] include making Eclipse APIs available as services, and making a user's workspace accessible via the web.

6. CONCLUDING REMARKS

In today's software development environments, most of the knowledge developers collect about the software project, for instance during maintenance tasks, is lost eventually. In this paper we propose ADINDA, a browser-based software development environment that collects information while developers interact with the system. ADINDA leverages Web 2.0 techniques, such as tagging and micro-blogging, to enrich the collected information. In this paper, we have identified a series of research challenges that need to be resolved to realize ADINDA, and reported on our approach and initial results.

7. REFERENCES

[1] M. Ames and M. Naaman. Why we tag: motivations for annotation in mobile and online media. In *CHI '07: Proc. SIGCHI conference on Human factors in computing systems*, pages 971–980. ACM, 2007.

[2] John Anvik, Lyndon Hiew, and Gail C. Murphy. Who should fix this bug? In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 361–370, New York, NY, USA, 2006. ACM.

[3] A. Begel and R. DeLine. Codebook: Social networking over code. In *31st International Conference on Software Engineering, ICSE Companion Volume*, pages 263–266. IEEE Computer Society, 2009.

[4] S. Breu and T. Zimmermann. Mining aspects from version history. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE 2006)*, pages 221–230. IEEE Computer Society, 2006.

[5] L.-T. Cheng, C. R.B. de Souza, S. Hupfer, J. Patterson, and S. Ross. Building collaboration into IDEs. *Queue*, 1(9):40–50, 2004.

[6] Reidar Conradi and Bernhard Westfechtel. Version models for software configuration management. *ACM Comput. Surv.*, 30(2):232–282, 1998.

[7] Robert DeLine. Del.icio.us development tools. In *CHASE '08: Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering*, pages 33–36, New York, NY, USA, 2008. ACM.

[8] D. Dig, K. Manzoor, R. Johnson, and T. N. Nguyen. Refactoring-aware configuration management for object-oriented programs. In *ICSE '07: Proceedings of the 29th international conference on Software Engineering*, pages 427–436, Washington, DC, USA, 2007. IEEE Computer Society.

[9] K. Dullemond, B. van Gameren, and R. van Solingen. How technological support can enable advantages of agile software development in a GSE setting. In *Fourth IEEE International Conference on Global Software Engineering (ICGSE)*, pages 143–152. IEEE, 2009.

[10] Jesse Garrett. Ajax: A new approach to web applications. Adaptive path, February 2005. `http://www.adaptivepath.com/publications/essays/archives/000385.php`.

[11] Michael W. Godfrey, Ahmed E. Hassan, James D. Herbsleb, Gail C. Murphy, Martin P. Robillard, Premkumar T. Devanbu, Audris Mockus, Dewayne E. Perry, and David Notkin. Future of mining software archives: A roundtable. *IEEE Software*, 26(1):67–70, 2009.

[12] S. A. Golder and B. A. Huberman. Usage patterns of collaborative tagging systems. *J. Inf. Sci.*, 32(2):198–208, 2006.

[13] Tim O'Reilly. What is Web 2.0: Design patterns and business models for the next generation of software. Oreillynet, 2005. `http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html`.

[14] W. Ryan. Web-based Java integrated development environment. BEng Thesis, University of Edinburgh, Division of Informatics, 2007. `http://www.willryan.co.uk`.

[15] C. Treude and M.-A. Storey. How tagging helps bridge the gap between social and technical aspects in software development. In *Proceedings of the 31st International Conference on Software Engineering (ICSE'09)*. IEEE Computer Society, 2009.

[16] Jim Whitehead. Collaboration in software engineering: A roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 214–225. IEEE Computer Society, 2007.

[17] Andreas Zeller. The future of programming environments: Integration, synergy, and assistance. In *FOSE '07: 2007 Future of Software Engineering*, pages 316–325, Washington, DC, USA, 2007. IEEE Computer Society.

[18] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. *IEEE Trans. Software Eng.*, 31(6):429–445, 2005.

[4]`https://bespin.mozilla.com/`

[5]`http://www.eclipse.org/e4/`