Codebook: Social Networking over Code

Andrew Begel, Robert DeLine
Microsoft Research
andrew.begel@microsoft.com, rob.deline@microsoft.com

Abstract

Social networking systems help people maintain connections to their friends, enabling awareness, communication, and collaboration, especially at a distance. In many studies of coordination in software engineering, the work artifacts, e.g. code, bugs, specifications, are themselves the objects that link engineers together. In this paper, we introduce Codebook, a social networking web service in which people can be "friends" not only with other people but with the work artifacts they share with them. Providing a web interface to the graph of these connections will enable software engineers to keep track of task dependencies, discover and maintain connections to other teams, and understand the history and rationale behind the code that they work on and use.

1. Introduction

Software development is highly social. A software engineer's daily work requires coordination with many other people and tracking many shared artifacts. Today, this coordination and tracking is largely accomplished through frequent communication and monitoring of artifacts: attending meetings; chatting in the hallway; reading code revision messages; examining work item statuses; exchanging emails; and so on. These coordination activities both take time away from technical work and are subject to the usual communication breakdowns, like untimely, inaccurate, misdirected or missing information.

Recently, social networking web services, like LinkedIn, MySpace, and Facebook, have provided a new way for people to coordinate and keep track of each other's activities. A Facebook user, for example, declares other users as "friends" and (after their approval) sees a "newsfeed" of their Facebook activities, like posting messages and photos. By reading the newsfeed and browsing pages, a Facebook user can stay aware of their friends' activities and coordinate events like political rallies. In this paper, we propose a new social networking web service, called Codebook, to help software engineers track status and coordinate activities. A software engineer can use Codebook both to track colleague's activities and changes to the status of work artifacts. Codebook also serves as an information portal. An engineer can look up a person or work artifact to find out basic information about it, including its history.

2. Data Model

A typical social networking web service can be abstracted as a graph with an event model. In the graph, the nodes represent people and the edges are self-declared friends links. A typical event notification policy is that, when an event happens at a node (e.g. a user posts a photo), that event is reported to all nodes that are directly connected to the source node. Codebook enhances this abstraction in several ways. First, the graph has nodes not only for people, but also for their work artifacts. Based on the previous work of Hipikat [3] and the Bridge [12], Codebook's graph includes nodes for code at various levels of granularity (DLLs, source files, namespaces, types, members), work items, bugs, revisions, non-code documents, and email messages, as well as people. This graph is produced automatically by crawling the contents of project data silos, like code revision histories, work item repositories and email archives. The graph has different kinds of edges to represent the relationships inferred from these data silos. Figure 1, for example, depicts a Bridge graph with thirteen nodes, shown as icons: the revision node (with a check mark, in the upper-left corner) has a "commits" edge to the person who committed the change and two "changes" edges to the changed files; these files each "contain" a class definition, which in turn "contain" member definitions; the bug item node has edges to the people who took actions on that bug item; and the email node has edges to the people who sent and received the email, as well as edges to a bug item and method that were textually mentioned in the email content.

This enriched graph allows Codebook users to become friends with both people and code artifacts. If a tester closes a bug in a bug repository, for example, the Bridge crawler will introduce a new "closes" edge between the tester's node

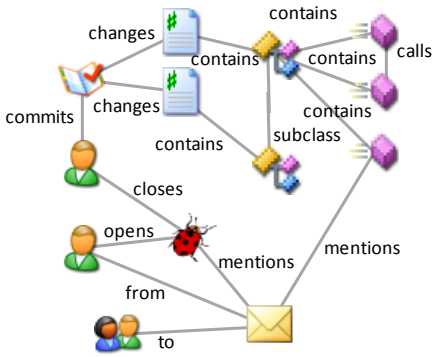

Figure 1. An example Bridge graph.

Figure 2. Mockup home page for a function.

and the bug item's node, which generates an event for both nodes. Users who are friends with the tester will see the bug closure in her newsfeed; at the same time, users who are friends with the bug item (for example, developers waiting for a bug fix) will see this event in the bug item's newsfeed. The graph also provides many kinds of edges along which to propagate events. For example, when a revision changes a method, it also changes the class containing the method, the source file containing the class, and the DLL built from the source file. How to propagate events across these indirections is an open research question which we discuss at the end of this paper.

On a typical social networking web service, each node (person) has its own web page, which serves as a "home page" with that person's information, newsfeed activities, and list of friends. Similarly, on Codebook, each node, whether representing a person or a code artifact, has a home page with that node's information, newsfeed activities, and list of friends. Figure 2 shows an example of a method's home page. An artifact's home page provides a centralized "place" for seeking information, analyzing data, having discussions, and meeting others interested in the same topic [5].

3. Usage Scenarios

We designed Codebook both to allow software engineers to track people and work artifacts and to provide a portal for information seeking. Here we describe several of the key scenarios we intend Codebook to support.

3.1. Anxious for Awareness

When teams collaborate as part of a large project, a member of one team will often assign a work item to a member of another team. Tracking the status of work items assigned across teams is frustrating because the teams' independent work is not transparent to each other. The work item can be delayed due to poor communication or differing priorities or forgotten altogether because no one advocates for it [8]. Codebook can help by increasing transparency between teams. A designated member of the team assigning the work item can befriend the work item and monitor its newsfeed. Once the work item has been assigned to a member of the other team, the designated person can befriend the assignee to watch his progress on the work item and to see what other responsibilities are competing with the work item. Browsing the assignee's team's newsfeed could also provide context about the team's changing deadlines and priorities.

3.2. Who Is Using Our Code?

In a company that produces both applications and frameworks, a framework team is often unaware of all the other individuals and teams who are using the framework in their products. This makes it difficult, for example, to assign priorities to bugs and to make informed decisions about breaking changes. Assuming that the Bridge is used "universally" (perhaps company-wide, perhaps across an entire code repository like SourceForge), Codebook can make code clients visible to code providers. As sketched in Figure 2, a method's home page lists incoming method calls, each with a code owner and the time the calling method was last updated, plus a link for contacting the owners of all calling method. These relationships are aggregated at the type and DLL levels.

This problem of invisible clients is notoriously acute when the client has cloned the code it uses to avoid taking a direct dependency. Although the Bridge does not currently use a clone detector, this technology is mature enough that the Bridge could add edges (with varying confidence values) between a definition and its likely clones. This would allow a Codebook page for a method to list clones as well as callers, as shown in Figure 2, with similar support for contacting clone owners. Owners of the cloned code could monitor the clones' evolution to consider accommodating

the changes the clones need. On the other side, the clone owners could track changes to the cloned code to pick up bug fixes and to look for opportunities to take a direct dependency rather than cloning.

3.3. Brothers at Arms

In this scenario, a team decides to begin working on a new technology and wants to know if any other team at the company is working on something similar. If there is another team, they would certainly like to get in contact, in order to learn about the other team's technology, their development plans, and their schedule. This will help the team decide whether to abort their own effort and reuse the other team's technology, forge ahead on their own, or collaborate with the other team to build the technology in a way that is compatible with both teams' visions.

Analyses of similarities between specifications and between bodies of code can produce affinities between code bases owned by separate teams. Similar to how social networking web sites "suggest" friends that you might already know, Codebook could "suggest" projects from other teams that are similar to projects that you or your team owns. In addition to discovering that other projects exist, an engineer can befriend these projects to keep up-to-date with a newsfeed about project developments. Finally, an engineer could examine the other project's friends to see if yet more projects at the company are trying to accomplish similar tasks and goals.

3.4. Code Investigation

Developers often find themselves or their code implicated in a bug report when they have not made any recent changes. As they debug through the test case, they find that the bug originates in a library call that they use which now returns an unexpected result. Their first suspicion is that the last change to that code broke it. So, they search through source code repositories to find a diff that shows what the code looked like before the change. But that merely tells what happened, but not why. To understand the code rationale, the developer reads the revision comment which may point to a work item in the bug database that was "fixed" by the revision. Reading through the bug database at the work item and any other related bugs, they can identify whatever rationale was written down electronically and discover who the responsible testers and developers were who worked on the work item. At this point, the problem moves from technological to social. There is a bug in the code, at an interaction point between one developer and another. Who is going to fix it? This involves communication and negotiation between the developers to resolve the quandary.

The Bridge links code to any related bugs, people or emails where the code was mentioned. On Codebook, the code's homepage contains a newsfeed where each of these related information sources are merged into a single chronological event stream that can be viewed by a developer to understand the history and rationale behind the code. When the developer needs to contact the owners of the code or bug, they can find them on the code or bug homepage and email, IM or phone them directly just by clicking on their names. This scenario emphasizes the importance of the newsfeed even for far past events.

4. Prototype

Codebook is implemented on the Bridge [12], which is in turn implemented on a SQL Server relational database. The Bridge has been in stable use for two years and is scalable enough to index Windows Vista's revision history and bug database. In the Bridge's architecture, both nodes and edges have time ranges, which the crawlers populates. For example, a "changes" edge between a node and a code revision has a time range set to the time of the commit; a "mentions" edge between a node and an email has a time range set to the email's sent date. Hence, the Bridge can compute a node's history by sorting the outgoing edges by their time ranges.

The first step toward Codebook was an implicit query system over the Bridge, called Deep Intellisense [6], implemented as an extension of Visual Studio. Whenever a user clicks on a definition in the code editor, Deep Intellisense queries the Bridge and shows links to all related people, and a timeline of all of the revisions, work item events, and emails related to that definition – that is, the newsfeed for that definition.

We are currently implementing Codebook on top of the Bridge, which requires two changes. First, we add a new type of edge representing "friends" links. Second, we add an event system that is invoked whenever a Bridge node or edge is created or when its time range changes.[1] When an event occurs on a node, we propagate it along the node's outgoing "friends" edges to the target node's new aggregator. The aggregator collects news from all of the neighbors like a structured RSS feed. As described below, there is an open research questions about which other edges (such as "contains" edges) should be involved in event propagation.

5. Related Work

Socio-technical congruence – that is, the relationship between networks of people and networks of the artifacts they create – is receiving increased attention. For example, a

[1] Bridge nodes and edges are never deleted. Instead, when an artifact is deleted in a data silo, the corresponding Bridge node has its end date set to the time of deletion. Hence there are no events for deletions.

recent study of software engineering coordination showed that developers can be connected to one another through their work artifacts [2]. We believe that this is the first paper to propose a social networking web service based on software engineers' work artifacts.

Recent research tools promote developer awareness with respect to source code, such as Ariadne [11], FastDash [1], Elvin [4], Jazz [7], and Palantir [9]. Many of these tools focus on avoiding conflicts by promoting awareness of activities that are happening at the same time as a developer's work. Codebook does not focus on "live" awareness (the Bridge's crawlers are not up-to-the-minute), but on capturing a comprehensive history of people and artifacts and their connections through a wide variety of crawlers.

5.1. Open Research Questions

We envision that the frontend to Codebook could take many forms: RSS feeds, social networking web sites, IDE plugins similar to Deep Intellisense [6], or automatically created blog postings. With such a wealth of data, however, understanding how to render the information presents a challenge due to inadvertent social engineering consequences. These web pages present an incentive system that can be gamed. For example, a manager might decide to display the number of open bugs for each developer on his team, sorted by the number of bugs. Thus, the "worst" developer is the one with the most open bugs. This creates an incentive for closing bugs, independent of fixing them, for example by closing them as "won't fix." Every rendering of the data represents a value system implicit to the team's goals for its members. By choosing a particular rendering, the team offers an incentive system where each team member can look his or her best, as well as make the team look good to others. We are conducting studies to discover how particular data renderings affect team member impressions of the value system, similar to what was done in a study by Stecher and Counts [10].

A related question is how to deliver newsfeed events in a way that is relevant and not overwhelming. In particular, how far along the graph should an event propagate and how should the presentation of the event change as that distance increases? When a developer closes a framework bug, there can be many interested parties. If the developer's manager or the person who filed the bug is looking at the report, he might care about the details of what work was done. If a developer from another team is looking at the news, he might care to know that a bug was worked on, but might not need to know about the content of the bug. If the product's chief manager looks at the news, he might care only about the number of remaining open bugs. We are engaging developers, testers and program managers with news about project events, presented in several styles to see what they perceive about the person behind the news. In addition, we will ask what the news says about them if they were the source of the news. By looking at the incongruity of these two perceptions, we can hone in on renderings that provide accurate perceptions of the data to both source and readers.

References

[1] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson. Fastdash: a visual dashboard for fostering awareness in software teams. In *Proceedings of CHI*, pages 1313–1322, San Jose, CA, 2007. ACM Press.

[2] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 353–362, New York, NY, USA, 2006. ACM.

[3] D. Cubranic, J. Singer, and K. S. Booth. Hipikat: A project memory for software development. *IEEE Trans. Softw. Eng.*, 31(6):446–465, 2005. Member-Gail C. Murphy.

[4] G. Fitzpatrick, P. Marshall, and A. Phillips. Cvs integration with notification and chat: lightweight software team collaboration. In *Proceedings of CSCW*, pages 49–58, Banff, Alberta, Canada, 2006. ACM Press.

[5] S. Harrison and P. Dourish. Re-place-ing space: the roles of place and space in collaborative systems. In *CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 67–76, New York, NY, USA, 1996. ACM.

[6] R. Holmes and A. Begel. Deep intellisense: a tool for rehydrating evaporated information. In *MSR '08: Proceedings of the 2008 international working conference on Mining software repositories*, pages 23–26, New York, NY, USA, 2008. ACM.

[7] S. Hupfer, L.-T. Cheng, S. Ross, and J. Patterson. Introducing collaboration into an application development environment. In *Proceedings of CSCW*, pages 21–24, Chicago, IL, 2004. ACM Press.

[8] C. Poile, A. Begel, N. Nagappan, and L. Layman. Coordination in large-scale software development: Helpful and unhelpful behaviors. In submission.

[9] A. Sarma, Z. Noroozi, and A. van der Hoek. Palantír: raising awareness among configuration management workspaces. In *Proceedings of ICSE*, pages 444–454, Portland, Oregon, 2003. IEEE Computer Society.

[10] K. Stecher and S. Counts. Thin slices of online profile attributes. In *Proceedings of International Conference on Weblogs and Social Media*, Seattle, WA, March 2008.

[11] E. Trainer, S. Quirk, C. de Souza, and D. Redmiles. Bridging the gap between technical and social dependencies with ariadne. In *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, pages 26–30, San Diego, California, 2005. ACM Press.

[12] G. Venolia. Textual alusions to artifacts in software-related repositories. In *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*, pages 151–154, New York, NY, USA, 2006. ACM.