RC24158 (W0701-108) January 24, 2007
Computer Science

IBM Research Report

Waypointing and Social Tagging to Support Program Navigation

Margaret-Anne Storey*, Li-Te Cheng, Ian Bull*, Peter Rigby*
IBM Research Division
One Rogers Street
Cambridge, MA 02142

*Department of Computer Science
University of Victoria
Canada

Waypointing and Social Tagging to Support Program Navigation

Margaret-Anne Storey
Dept. of Computer Science
University of Victoria, Canada
mstorey@uvic.ca

Ian Bull
Dept. of Computer Science
University of Victoria, Canada
irbull@uvic.ca

Li-Te Cheng
Collaborative User Experience
IBM Watson Research, Cambridge
li-te_cheng@us.ibm.com

Peter Rigby
Dept. of Computer Science
University of Victoria, Canada
pcr@uvic.ca

Abstract

As the "software space" of source code, documentation, models, and other programming artifacts continue to grow in size and complexity, programmers face the challenge of navigating this space, as well as documenting and sharing their journeys for other developers and future successors. Current navigational structures are either closely tied to the semantics of the software or are constructed in a constrained top-down fashion to match the architecture or requirements of the system. In this paper, we introduce the notion of combining waypoints from geographical positioning and social tagging from shared bookmark systems to allow programmers to create shared, tagged points in software space. We report preliminary progress on our prototype (tagSEA), and discuss our future plans.

Keywords
Software development, navigation, social tagging.

ACM Classification Keywords
H.5.3 [Group and Organization Interfaces].

Introduction

Navigation is a key activity in software development. Integrated development environments (IDEs), such as Visual Studio, NetBeans, and Eclipse, provide a wide variety of mechanisms to support navigation through a software system. These locomotional structures include tree-based resource views, model element views, tabbed views, cross referenced hyperlinks, browser support, and search facilities. These features support navigation through source code relationships and artifacts. Software documentation in the form of inline comments and Javadocs closely mirror the modular structure of the software and support navigation through searching and hypertext. Higher level design documentation is intended to provide navigational hooks into where engineering concerns and concepts from the problem domain are implemented. Such documentation styles tend to be constructed in a top-down manner and quickly become out of date as the software evolves.

Many IDEs support a variety of annotation mechanisms to help the programmer form their own locomotional structures over the software space. The programmer can bookmark and tag parts of the code to indicate locations for future navigation or to indicate tasks such as “TODO”. Our preliminary research has indicated that bookmarks and tasks tend to be underused by programmers. The bookmark metaphor emerged from the notion of bookmarking pages in a sequential one dimensional document or book. However, software is multi-dimensional and there are many concerns beyond programming issues that are relevant when tagging code. Bookmarks lack meta-data which impede how they can be grouped, filtered and searched. Moreover, bookmarks tend to quickly get out of date as they lack the notion of decay. Tasks are used more frequently but they are difficult to search and organize due to a lack of metadata. In most IDEs, bookmarks and task tags are not anchored to the source code, and hence cannot easily be shared across teams of programmers. Bookmarks are also notoriously difficult to manage [2]. The best that can be supported in web browsers is to impose a hierarchical decomposition on them. Such a view shows only one dimension of the software.

Another drawback with current documentation and annotation mechanisms in IDEs is that they poorly document lightweight concepts that crosscut multiple software artifacts. Moreover, it is very cumbersome to document repetitive software engineering tasks that require navigation to different areas of the code.

In this paper, we borrow a metaphor from the discipline of wayfinding in physical spaces called *waypoints*, and combine it with the notion of *social tagging*, to address some of the limitations of current navigation and documentation techniques. Our goal is to investigate if the concept of waypoints, when combined with social tagging, can suggest a new interface in the IDE to support efficient shared navigation and active documentation of the source code and related artifacts.

Background

Waypoints are used by geographical positioning systems (GPS) to save locations of interest [5]. Geographical locations may include checkpoints on a route or a significant ground feature to be avoided. Waypoints can be specified by manually entering latitude and longitude (and optionally altitude) coordinates or they can be saved as the user passes close by a landmark or point of interest. Waypoints can be referenced according to distance and bearing to a previously saved waypoint, and they can be shared across users and applications. Waypoints are often gathered within *routes* (see Fig.1). A route provides a path from one point to another together with inter-mediate destinations (waypoints). When saving a route, the user can omit “wrong turns” or inefficient steps. Waypoints can be clustered into sets according to some attribute or concern e.g. favorable anchoring locations. The GPS can be used to reveal waypoints that are close by (e.g. good restaurants). These are recommendations based on proximity

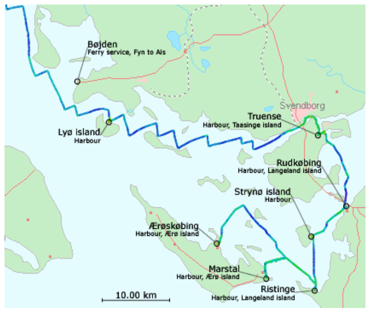


Figure 1: GPS waypoints

and customer ratings.

Social tagging, also known as social bookmarking, enables users to create shared bookmarks to online resources with additional metadata beyond the site location. Social tagging websites such as flickr.com and del.icio.us are used to "tag" images and share bookmarks respectively by a large user community. A tag is a one-word descriptor or term to describe the image or bookmark. The advantages of these social tagging systems is that the user is free to choose any descriptive terms and is not restricted by a preconceived vocabulary, taxonomy or ontology. This bottom-up approach results in semi-structured information spaces that are often referred to as "social classifications" [3]. This differs significantly from the knowledge engineering and semantic web communities that advocate having shared ontologies of well-defined terms and structures to enable machine computation. Although social tagging may seem to undermine the more formal approach to knowledge modelling, tagging is now accepted as a complementary mechanism to the highly structured top-down approach. For example, Amazon has an ontology underlying its structured information space, but is now exploring how consumers can augment this structured information space with their own user-defined tags [10].

Tagging is not a new concept to software engineering. Tags have been used for decades for annotating check-in and branching events in software version control systems. This use of tags is for identifying version control transactions rather than for tagging within the source code and documentation. There are also some online projects that use tags at the software component level. Swik (swik.net) and O'Reilly's CodeZoo (www.codezoo.com) are two website examples that house socially shared tags about software systems and components. However, the tags are not directly tied to the code or documentation.

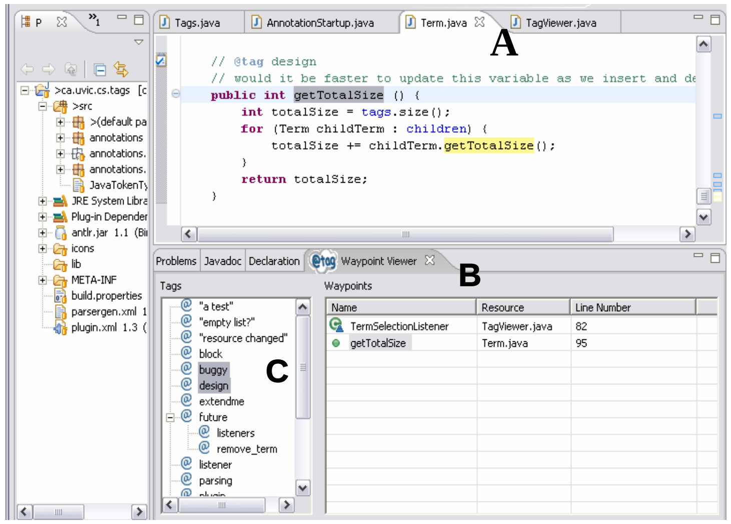
Several researchers are also investigating how *recommenders* can be used to improve navigation. Robillard's concern graphs allow the user to explicitly create navigational structures over source code [8]. He proposes using semantic analysis to automate or recommend the assignment of artifacts to concerns. The NavTracks tool provides a list of recommended files to support navigation [9]. Recommendations are determined based on previous navigation history. The Mylar tool builds a degree of interest model that is task-based to reveal or prune elements that are or are not related to the current task [4]. Early feedback is positive and indicates that the approach reduces information overload when working on large systems. However, it does require that the user declare when they are working on certain tasks. Teamtracks also uses the notion of collaborative filtering to prune/emphasize elements in the view that are of less/more interest based on team navigation actions [1]. Although not implemented for software systems, the notion of shared sequences of bookmarks with metadata has been developed for web pages and documents [7].

Using waypoints and social tagging to navigate in software space

We are investigating if combining waypointing and social tagging is a useful metaphor to support navigation in the software space of source code and related artifacts. In software space, waypoints may be locations of software model elements (e.g. class, method, package, file, function), or they may be a location that corresponds to a file name and line number for any type of file or for any version of a file. Waypoints are indexed through a set of tags supplied by the programmers. In addition to the tag terms, the meta-data captured or explicitly entered may include the version number of the software, creation date of the waypoint, author, related bugs etc. Also a "decay date" can be added to the metadata to indicate it should decay after a fixed time period or when some event occurs e.g. a bug report is closed.

In order to explore some of the ideas discussed thus far, we have implemented a simple plug-in for the Eclipse Java IDE (www.eclipse.org). This plug-in, called "tagSEA" (Tags for Software Engineering Activities), allows the user to create implicit waypoints by tagging model elements in the source code. Figure 2 shows a view of this early prototype. Users can create tagged waypoints in their code either through entering Javadoc tags or inline comments (see Fig. 2 A). Tags are identified by preceding them with "@tag". Individual tags are delimited by spaces, or they may consist of multiple words by placing the string in quotes. tagSEA provides auto-completion support for entering tags based on previously defined tags.

tagSEA also includes the Waypoints Viewer (see Fig. 2 B). Selecting one or more tags listed on the left column of this viewer reveals the software model elements, their locations, and annotations, that have been waypointed in the right side of the viewer. Clicking on the waypoint entries on the right side of the viewer opens the associated file editor, positions the editor at the appropriate location, and highlights the waypointed software model element. Thus, programmers can use the Waypoints Viewer to navigate to places of interest in the software. Tags can also be added through the Bookmark feature. Bookmark tags are not saved with the source code and are stored only in the metadata of the workspace thus facilitating private tags.



```
    // @tag design
    // would it be faster to update this variable as we insert and de
    public int getTotalSize () {
        int totalSize = tags.size();
        for (Term childTerm : children) {
            totalSize += childTerm.getTotalSize();
        }
        return totalSize;
    }
```

Name	Resource	Line Number
TermSelectionListener	TagViewer.java	82
getTotalSize	Term.java	95

Figure 2: TagSEA prototype integrated in an IDE

Tag spaces are often criticized for producing flat structures [3]. However, there are reports of users using their own conventions to encode hierarchical relationships across tags. We believe that programmers may be more comfortable adopting a hierarchical syntax given their experience with formal languages and abstractions. The user can specify the following hierarchical tag: "@tag bug(390)" in the Javadoc comments. This indicates that there is a "bug" tag, with bug subtypes specified by the parameter in brackets. The hierarchy of tags is displayed using a tree at the top of the

Waypoint Viewer (Fig. 2 C). The hierarchical feature can also be used to indicate groupings in addition to “is-a” relationships. The user can further specify or alter hierarchical relationships by relocating tags within the tag hierarchy tree. We are investigating ways to maintain changes to tags and documentation. When tags are renamed, the associated instances throughout the software will be updated.

Our prototype is preliminary. In particular, we need to add facilities for capturing and entering more metadata. We are also interested in providing flexible mechanisms for filtering and searching for waypoints, as well as support for adding routes and sets of waypoints. We plan to explore how Mylar, a degree of interest model for Eclipse, can be used for filtering waypoints for large systems [4]. A degree of interest model will provide the notion of distance which is crucial to the concept of waypoints. There are many notions of distance which can capture the user’s interest model in addition to the semantics of the software model.

In GPS applications, waypoints are neither dependent on nor tied to any one application. They can be shared across users and applications. Similarly we assume the same benefits could be realized for waypoints in software spaces when the semantics of the referenced locations are shared across applications. Moreover, by using tagging to specify waypoints, we can also investigate how social tagging can be used to share and exploit tagging vocabularies and taxonomies, while also increasing awareness of development activities beyond the confines of the IDE. To facilitate this line of inquiry, we plan to integrate tagSEA with Dogear, an enterprise social bookmarking tool [6].

The tagSEA prototype only provides a list and tree based interface to manage tag and waypoint information. We are planning to experiment with visualizations such as the “tag clouds” popularized by social tagging sites such as flickr and del.icio.us to provide alternative user interfaces.

Discussion

We believe that waypoints can be used as a sharable set of points related to some goal, task or software concern. By using other metadata captured with the waypoints, more views of the body of software can be generated for specific tasks (e.g. highlighting areas tagged with contentious bugs).

Waypoints can be stored in a sequence or route to indicate an order relationship between them – a useful facility for documenting a series of steps in a software development workflow. This can be useful for a software inspection or code review, as well as a step-by-step guide for newcomers to demonstrate how a feature is to be implemented using a software library or framework. Since the waypoints and tags are shared, the user need not follow a single path prescribed by one authority. Instead, the user can build a personalized route by combining paths from different experts based on the tag metadata. For example, a student programmer might initially follow the first few waypoints in a route constructed by an instructor, and then diverge to follow the steps taken by a peer.

A key issue is whether programmers will adopt waypointing as part of their regular coding work. And if waypointing is adopted, there is the issue of managing a growing and aging body of waypoints. Also the assumption behind social tagging systems is that there is a community of users that will mutually benefit from tagging. In the case of a software development team, it is not clear if there is enough of a sense of community to provide the benefits of social tagging.

Another issue that always arises when a new style of documentation is proposed is how to deal with legacy code. Since this is a lightweight approach there is not an explicit requirement to tag legacy software. However, it may be advantageous to develop some tool support to help the user semi-automatically tag legacy code. We have experimented with this idea briefly by analyzing existing comments and extracting the most popular keywords as potential candidates for tags. The extracted tags are then presented to the user as candidate tags within a Tag cloud view.

We have already begun to experiment with tagSEA within a small group of programmers. Preliminary feedback is very encouraging. One developer reported tagging as he was integrating two unfamiliar systems. The lightweight mechanism allowed him to temporarily tag areas of the code he was changing as he was experimenting with the unfamiliar code. He also reported using tagging to indicate areas that he updated to fix bugs that were in another project. The tagging feature was a useful mechanism to document these changes for future navigation. The alternative solution would have been to submit the code to a version control system, run a difference tool to list the results and navigate to each one at a time by searching through the workspace. Our goal now is to deploy it to a wider group of developers. To develop a further understanding of how waypoints will be used and shared, we will instrument the plug-in to gather usage data. We will also investigate how tags are used when compared with top-down approaches. Finally, we are interested in exploring if the metaphor of waypoints has application beyond software development.

Citations

[1] Deline, R., Czerwinski, M. & Robertson, G.G. (2005). Easing Program Comprehension by Sharing Navigation Data. In Proceedings of Visual Languages and Human-Centered Computing, VL/HCC 2005.

[2] Jones, W., Bruce, H., and Dumais, S. 2001. Keeping found things found on the web. In Proceedings of the Tenth international Conference on information and Knowledge Management, Atlanta, USA, 2001.

[3] Hammond, T., T. Hannay, B. Lund, and J. Scott, “Social Bookmarking Tools: A General Review”, D-Lib Magazine, Volume 11 Number 4, April 2005.

[4] Kersten, M. and G. Murphy, "Mylar: A degree-of-interest model for IDEs," Proceedings of Aspect Oriented Software Development, Chicago, IL, 2005.

[5] Larkin, F.J., Basic Coastal Navigation: An Introduction to Piloting, 1999. ISBN 1-57409-052-6

[6] Millen, D., J. Feinberg, and B. Kerr, “Social Bookmarking in the Enterprise”, ACM Queue, vol 3, no. 9 - November 2005.

[7] Paul Moody, WebPath: Sharable Personalized Guided Web Tours, IBM Research (Cambridge), Technical Report 98-09 (1998)

[8] Robillard, M.P. and G. Murphy, "Automatically Inferring Concern Code from Program Investigation Activities," Proceedings of 18th International Conference on Automated Software Engineering, 2003.

[9] Singer, J., R. Elves, and M.-A. Storey, “NavTracks: supporting navigation in software maintenance,” Int. Conf. on Software Maintenance, Budapest, 2005.

[10] Terdiman, D., “Amazon tries its hand at tagging”, http://news.com.com/2061-10802_3-5953622.html?part=rss&tag=5953622&subj=news, news.com, Nov 15th, 2005.