RC24399 (W0711-019) November 5, 2007
Computer Science

IBM Research Report

How Programmers Can Turn Comments into Waypoints for Code Navigation

M.-A. Storey[1], L.-T. Cheng[2], J. Singer[3], M. Muller[2], D. Myers[1], J. Ryell[1]

[1]University of Victoria
Victoria, BC
Canada

[2]IBM Research Division
One Rogers Street
Cambridge, MA 02142 USA

[3]National Research Council
Ottawa, ON
Canada

How Programmers can Turn Comments into Waypoints for Code Navigation

M.-A. Storey[1] L.-T. Cheng[2] J. Singer[3] M. Muller[2] D. Myers[1] J. Ryall[1]
University of Victoria[1] Victoria, BC, Canada
IBM Research[2] Cambridge, MA, USA
National Research Council[3] Ottawa, ON, Canada

Abstract

We have developed a new approach for software navigation called TagSEA (Tagging of Software Engineering Activities). TagSEA combines the notion of "waypointing" with "social tagging" to support programmers in defining navigational structures over a software system. In this paper we present the results from a case study series, conducted with professional programmers, that demonstrates how this tool supports navigation and under what circumstances. We conclude with insights into user-definable navigational structures, and how they can support software maintenance more effectively.

1. Introduction

Navigation is a fundamental activity in software maintenance. As such, integrated development environments (IDEs), such as Visual Studio, NetBeans, and Eclipse, offer a wide variety of mechanisms to support navigation. These features include tree-based outline views, tabbed views, cross reference hyperlinks, and search facilities.

Many IDEs also offer a variety of user-definable navigational structures. These user-driven features are an important complement to system-defined navigational structures, allowing the user to create their own structures for moving about the software space. One example of a user-defined navigational structure is bookmarks, which support a user in annotating code locations for further inspection by allowing the addition of descriptive text. Bookmarks are stored in the user's local workspace metadata and do not alter the source code. In addition to bookmarks, most IDEs support annotations that indicate tasks, such as "TODO".

From our research with software engineers, we have observed that typical annotation mechanisms (notably bookmarks and tasks) tend to be ineffective at supporting software navigation, despite the intuition that they should be sufficient. The bookmark metaphor emerged from the notion of marking pages in a book with sequential pages. Software, on the other hand, is a complex multi-dimensional space. Moreover, bookmarks and task annotations lack metadata that are needed to group, filter, search, and manage them.

We propose that there is a need for a more elaborate user-definable navigation mechanism. We start by considering a metaphor that is richer than bookmarks, that of *waypoints*. Waypointing originates from the discipline of wayfinding in physical spaces (e.g. [5]). Wayfinding has led to user experience design concepts, such as navigation aids [3] and landmarks [12]. Waypoints are created by marking a location of interest. They have associated metadata, such as creation time and author, and they can be shared across users and applications and may be gathered into routes.

In the context of software engineering, to leverage the metadata capability of waypoints, we add the concept of tagging. Furnas showed that users prefer to use their own vocabularies for common objects and concepts [6]. Tagging tools and social bookmarking systems [9] provide a mechanism for users to create and apply their own vocabularies. TagSEA, a tool for **Tag**ging **S**oftware **E**ngineering **A**ctivities, combines the notion of waypoints with tagging. By tagging a waypoint with a set of unconstrained keywords, the user implicitly creates a simple navigational structure that can be used to locate specific targets by searching or pivoting on one or more of the attached keywords.

An early version of the TagSEA tool was introduced in [1]. Here, we report the results from a case study series that explores how professional programmers use the system, and sheds light on how other annotation mechanisms support navigation. This paper is structured as follows. In Section 2, we provide details on various IDE features and research tools that support user-definable navigational structures. In Section 3, we briefly review the version of TagSEA used in the case studies. Section 4 summarizes our key research questions. The case study series and findings are described in Sections 5 and 6. Section 7 provides additional data from anonymous early TagSEA adopters. In Section 8, we synthesize and discuss the results from the studies in light of our research questions. Section 9 summarizes the contributions and proposes future work.

2. User-defined software navigation

Although most IDEs typically offer a number of canonical navigation features such as bookmarks and task annotations, our research, and that of Murphy *et al.* [13], indicates that bookmarks and tasks are rarely used in software development environments. Using Murphy et al.'s data, which they shared with us, we examined the logs of 42 programmers. Task selection events were as low as .2% of the total user view selections, and bookmark selection events were as low as .02%. Indeed only 13 of the 42 programmers used tasks, and only four of the 42 programmers used bookmarks. We believe this low usage is due to two reasons. First, although it is easy to bookmark or to mark a region with a task, these structures lack sufficient semantic information to facilitate recovery at a later stage. Second, bookmarks and task annotations suffer from a lack of visibility and are unfortunately easily forgotten and difficult to manage and thus quickly become outdated.

A further problem is that in many IDEs, bookmarks are not stored within the source code, and therefore cannot easily be shared across teams of programmers. Since software is frequently developed by teams, this is a significant issue. Tasks are sometimes more useful as they can be added directly within the source code. Ying *et al.* analyzed task annotations (such as TODO) that are stored within the source code and categorized how such comments can "talk" [19]. Some of the categories these researchers observed supported future navigation. However, we have observed that task annotations lack sufficient structure or metadata to facilitate search. Although task annotation keywords can be customized (to keywords such as "FIXME" or "XXX"), it is cumbersome to do so and the customized tasks lack user interface (UI) management support.

Another simple mechanism for programmers to mark locations of interest for future examination, involves inline commenting in the source code. A familiar example is the prevalence of informal expressions and memorable keywords, such as "HACK" or "fix me" to highlight suspect code (e.g. [16]). These annotations assert information on the code or design and may also indicate locations for future work. Such comments may be scattered throughout the program if the concerns or tasks being documented crosscut the established software structure. A drawback with the use of distributed in-line comments is that the programmer either needs to remember the locations of the code to be revisited or the terms used so that they can be searched. To help provide support for navigation via in-line comments, various programming languages have special syntax. For example, the Javadoc documentation facility in Java has the "@see" and "@link" tags, which are accompanied by notation referring to parts of code (e.g. packages, classes, methods) or URLs [18]. Modern Java IDEs automatically turn these tags into clickable hyperlinks. Java annotations can provide similar affordances, but can also affect how programs are compiled and run.

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, as well as for documenting bugs in bug tracking systems. Brothers' ICICLE was an early exploration of a limited, controlled-vocabulary of tag-like structures during code inspection [2]. Code Snippets (bigbold.com/snippets) and ByteMycode (bytemycode.com) support social tagging of source code, but require the user to post code fragments on public servers where tagging is then applied to the fragments. However, tags have not been adequately explored as a mechanism for categorizing and retrieving lightweight annotations in source code.

There are related research tools that support user-definable navigational structures. ConcernMapper [15] supports programmers navigating cross-cutting concerns by allowing them to automatically find and group together related pieces of code within a concern. The programmer can then browse the code by clicking on the elements identified in the concern. These annotations reside in the user's workspace. They are easy to create and access; and since they are linked to elements in the program, they are automatically updated as the program evolves. However, since the documented concerns are not stored in the source code, they are not easy to share across workspaces and programmers. They also rely on a top-down approach to creation, whereby the programmer specifies the concern up front and then identifies code that belongs to it. Another research tool, JTourBus [14] also proposes the notion of "tours" through the source code as a form of documentation. The tours concept is similar to the notion of routes of waypoints. We have also explored the concept of "tours" as presentations for programmers [20].

Another class of research tools, Mylar [10], NavTracks [17], and TeamTracks [4], monitor the user's interactions and then prune the available navigation targets. These tools attempt to automatically create personalized structures to facilitate navigation. Mylar is the most advanced of these tools, but it relies on the programmer opening and closing a task when they switch context.

The tools described support the user in either explicitly or implicitly defining navigational structures. What is missing is a lightweight, bottom-up mechanism for explicitly tagging code that provides convenient UI mechanisms for searching, grouping, managing and filtering related code. The tool we have prototyped, called TagSEA, aims to fill this gap.

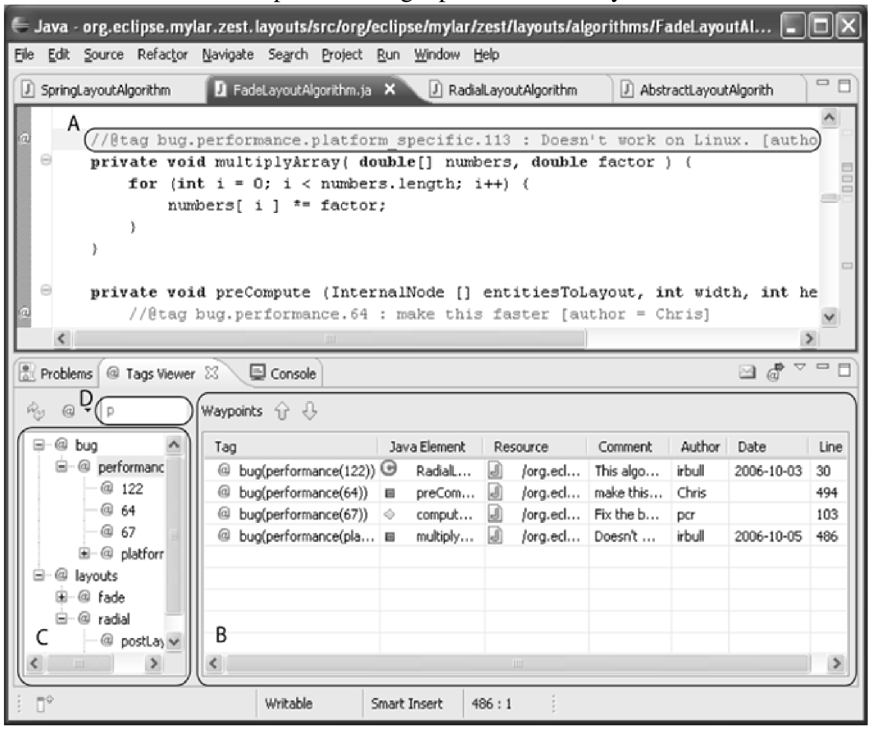
3. Tagging Software Engineering Activities

TagSEA has been implemented as a plug-in for the Eclipse IDE (www.eclipse.org). In TagSEA, the waypoint analogy corresponds to marking locations in the software such as Java source code elements (e.g. class or method), or a specific line in a source or documentation file. The tagging element comes in (a) because waypoints are described by a set of tags supplied by the programmers, and (b) because each user's tags are visible to other users. In addition to the tags, metadata may be automatically associated with each waypoint, such as the version of the file, creation date, author, related bugs, etc.

Figure 1 shows a view of TagSEA. Programmers create waypoints by associating tags with parts of the source code using a Javadoc-style keyword. Specifically, a waypoint is created by the programmer typing "@tag" in a comment block, followed by the tag keywords. Descriptive text can be added. Individual tags are delimited by spaces (see Fig. 1A). A programmer can also associate **hierarchical tags** to a waypoint using dot-separated or bracket notation as follows: "@tag bug.performance" or "@tag bug(performance)" This indicates that there is a waypoint with the tag "bug" whose subtype is "performance". The Javadoc-style syntax allows for easier adoption of TagSEA by Java programmers, who are already familiar with similar conventions, such as "@author" and "@version". The resulting waypoints are automatically associated with the closest Java element (e.g. a method). Once waypoints are created they can be used by programmers to navigate and understand the code or to share information with others by uploading "waypointed" code into a source control system.

Waypoints indicated by the "@tag" notation are highlighted in the editor's text, left-margin, and scrollbar so that they stand out as clear landmarks of interest. Using the **Waypoints Viewer** (see Fig. 1B), programmers can jump immediately to a particular waypoint, or they can view all waypoints with specific tags associated with them, or search for waypoints with certain characteristics. For instance, a programmer could view all waypoints tagged with a specific bug id or task, or all waypoints created by a particular programmer. Selecting one or more tags in the **Tag Tree** (see Fig. 1C) reveals the associated waypointed Java elements in the middle pane. Then, clicking on the waypoint entries in the Waypoints pane (Figure 1B) opens the associated file editor, positions the editor at the appropriate location, and highlights the waypointed Java element. Thus, programmers can quickly navigate to places of interest.

Managing a growing sea of tags is a concern for social tagging systems [9] and this may be a problem for large software systems. To address this concern, TagSEA provides some initial support for dynamic **filtering** and **searching** of waypoints (see Fig. 1D). Every keystroke in the filtering text box immediately updates the list of tags that partially match the entered query, allowing a user to condense and explore tag spaces through partial text entry. Users can also sort



```
    //@tag bug.performance.platform_specific.113 : Doesn't work on Linux. [autho
    private void multiplyArray( double[] numbers, double factor ) {
        for (int i = 0; i < numbers.length; i++) {
            numbers[ i ] *= factor;
        }
    }

    private void preCompute (InternalNode [] entitiesToLayout, int width, int he
        //@tag bug.performance.64 : make this faster [author = Chris]
```

Tag	Java Element	Resource	Comment	Author	Date	Line
bug(performance(122))	RadialL...	/org.ecl...	This algo...	irbull	2006-10-03	30
bug(performance(64))	preCom...	/org.ecl...	make this...	Chris		494
bug(performance(67))	comput...	/org.ecl...	Fix the b...	pcr		103
bug(performance(pla...	multiply...	/org.ecl...	Doesn't ...	irbull	2006-10-05	486

Figure 1. TagSEA plugin for Eclipse

and access waypoints via metadata. We have also added support for **refactoring** of waypoint tags so that they can be easily renamed, reorganized or deleted. Reducing the number of unique tags created can be addressed by using a consistent set of tags over time [9]. TagSEA provides an **automatic tag completion** feature to suggest the use of existing tags based on a partially typed tag.

4. Evaluating TagSEA: Research Design

We formulated four **research questions** regarding the additional tool support provided by TagSEA and its role in aiding how programmers navigate code:

Q1. What kinds of waypoints and tags will professional programmers create, and how will they evolve over time?

Q2. How do programmers make use of waypoint comments compared to their usage of Eclipse task annotations and other source code comments*?*

Q3. If programmers choose not to use TagSEA, why not?

Q4. If TagSEA is seen as a useful tool, how can it be further improved?

We conducted a case study series with professional programmers using TagSEA for their everyday programming tasks. This qualitative research approach is congruent with the research questions we pose. An alternative approach would have been to conduct a controlled study in the lab or in the field. However, a formal study would not show us how professional programmers would use a tool such as TagSEA for supporting real-world navigation tasks, nor would it reveal why some choose not to use the tool.

We made the tool available to eight professional programmers at two industrial sites. We used a *purposeful sampling technique* [11]. Specifically, we selected users because they had the relevant experience for participation and were accessible to us as researchers. Our in-depth study of these individual programmers provided insights on all of the research questions posed above (see Table 1).

In addition to the case study series, we solicited a small number of anonymous users to participate in our study through the TagSEA website. From these participants, we collected usage statistics to help us further study the kinds of tags and waypoints TagSEA users create. Since we did not have direct access to these users we could not collect any additional background information about them. Hence, this supplemental data was used to bring insights only to the first research question. The supplemental data is presented in Section 7.

Table 1. Data collected and implications on research questions posed during this work.

	Research questions			
Data collected:	**Q1**	**Q2**	**Q3**	**Q4**
Selected users from industrial sites:				
Pre-study questionnaire	X		X	
Waypoint and tag analysis	X			
Comment and task analysis		X		
Focus Group	X	X	X	X
Post-study interviews, questionnaires	X	X	X	X
Anonymous early adopters of TagSEA:				
Waypoint and tag analysis	X			

5. Case study series

5.1 Developers studied

We recruited six programmers, whose primary task is software development, from an industrial research lab in Cambridge, Massachusetts (MA) and two from a development lab in Victoria, British Columbia (BC), Canada. We did not insist they use TagSEA, although we did ask them to download it. TagSEA was made available to these programmers over an eight-week period. All of the recruited programmers worked on separate coding projects. We refer to the programmers from Cambridge as C1-C6, and the users from BC as B1 and B2.

5.2 Data collection

We used five data collection methods. First, following research ethics approval, we administered pre-questionnaires to the recruited users from the industrial sites. The pre-questionnaire asked for details on programming experience, project type and size, experience with advanced features in Eclipse, and experience with social tagging tools. Second, we designed simple scripts to extract comments from the source code from programmers who gave permission, and ran the scripts on multiple versions of their source code as they used the TagSEA tool. This data allowed us to examine usage of other mechanisms, such as task annotations. Unfortunately, not all of the developers could submit their source code due to privacy concerns. Third, programmers were asked to submit a file containing their tags and waypoints so that we could analyze them. This data was generated using a script. Tag and waypoint uploads were solicited from the programmers at the Cambridge lab three times, i.e. at roughly three week intervals in an eight week period, and once from the BC users. Fourth, a focus group was conducted with the programmers in Cambridge to help us validate the initial analysis of the tag and interaction data. It was particularly effective as we had both adopter and non-adopter issues to explore. The focus group was held three weeks after the initial download so that we could learn about the adopters' early experiences and also find out why the non-adopters did not use the tool. Finally, exit interviews and post study questionnaires were conducted, asking programmers at both sites for their insights on the tool.

5.3 Data analysis

We used a qualitative analysis approach to construct a descriptive story about each user, detailing how they used or did not use the tool given their particular experience and programming context. For each user, we gathered the available data and performed a preliminary exploratory analysis [11]. This was followed by a coding process that involved segmenting and labeling text from the interview, open-ended questionnaires and focus groups. Where possible we looked for converging sources of evidence to support our claims. For example, the conclusions we drew from the comments, annotations, and the tag data were verified through interviews or questionnaires.

We manually classified each of the comments, annotations and TagSEA waypoints using a set of codes. The codes used for the classification were derived through an iterative analysis of the source code comments, task annotations and tagged waypoints. Three coders, all with qualitative research experience, independently derived codes for all waypoint tags for each user and then through consensus merged their codes into a single set of codes. These codes were also used to code the in-line comments and the task annotations. Where possible, we verified with the programmers that we were interpreting the intent behind commenting and tagging correctly.

Two major categories of intent for commenting and tagging were identified: INFO and TODO. INFO refers to the general category of annotations that assert information. In some cases, INFO could be refined to two subcategories, INFO-FEATURE, and INFO-AUTHOR, indicating information about a particular feature and author respectively. TODO is a general category for annotations that document a task. It has four subcategories, each referring to a specific action to take: testing code (TODO-TEST), fixing a bug (TODO-BUG), changing or completing some aspect of the code (TODO-CHANGE), or checking something (TODO-CHECK). This coding is similar to Golder and Huberman's breakdown on the types of tags users use in social bookmarking systems [7], except we refine the task type into four TODO categories that are specific to software maintenance.

In addition to categorizing each of the user's waypoints, we count the number of waypoints and tags at each time slice. As a reminder to the reader, a tag is a user supplied keyword (e.g. "performance") to index a waypoint (a specific line of code or Java element), and a waypoint may have one or more tags associated with it. The tag density provides a measure of the extent of tag reuse. Tag density = (#unique tags)/(#tag occurrences). A lower density indicates more reuse of tags. The upper bound value of 1 would occur if each tag was used only once (i.e. no reuse).

6. Findings

6.1 Adopters

Two programmers from Cambridge, C2 and C3, and one user from the BC lab, B2, adopted the tool. We use information from the questionnaires, focus group and the data submitted to us in forming these user stories. The two Cambridge users also submitted snapshots of their source code comments for analysis. Using the code categories described in Section 5.3, we were able to code C2 and C3's use of descriptive in-line comments and task annotations and then compare these to the TagSEA waypoints. Table 2 shows a summary of the number and category of waypoints created per user, and how they changed over the study period. It also provides a count of the number of waypoints, unique tags and density of tags.

C2 (female) used the tool over the course of the eight week study and continued to use the tool after the study. She joined the company shortly before this pilot and was assigned to extend existing code in a team project. The majority of her waypoints were created to support future tasks (see Table 2). In the interview with C2, we were able to verify that she mostly created waypoints for navigation. She said they were a reminder of the places that needed more examination. From an analysis of her source code, we were able to verify that she uses neither tasks nor bookmarks and that very few of her source comments could be classified as supporting future navigation. This user only used one instance of a hierarchical tag, but she did use multiple tags (at most two) per waypoint. She did not add additional comments to the waypoints. There were instances of the same tag being used on more than six waypoints

Table 2. Shows # of waypoints for C2, C3 and B2 for the first data collection period (T1). The change in number of waypoints is indicated at the second (T2) and third (T3) data collection periods. Note that waypoints were both added and deleted (indicated with a '+' or '-' sign). Waypoint count, #tags, #tag instances and a tag density metric are also given.

#Waypoints of type: User:	Info	Info-Author	Info-Feature	ToDo	ToDo-Bug	ToDo-Change	ToDo-Check	ToDo-Test	Total Way points	Unique Tags/ Instances	Tag Den-sity
C2 T1	1	5	11	1	1		4		23	15/23	.65
Δ T2		-1	+2/-3				+5/-1		25	14/25	.56
Δ T3							+1		26	14/26	.54
C3 T1			2		1	7	3	2	15	10/24	.42
Δ T2					+4	-1			18	11/28	.34
Δ T3					+1/-1	+5	+1		24	19/45	.42
B2 T1			6			12	7		25	14/55	.25

across three files of code.

One interesting tag we observed was called "home". In the interview, C2 indicated it was a special tag to support navigation. She also created one tag type to match her name and it was used to indicate several places she had made changes in the code. She agreed that the Javadoc @author feature could have been used, but she said she preferred the lighter-weight approach of TagSEA for indicating her changes. In the exit interview, C2 indicated she would continue to use TagSEA and that it was preferred over bookmarks andEclipse tasks. C2 mentions that "she liked TagSEA for being fast and lightweight to type in things".

C3 (male) joined the company six months before the study. He also used the tool over the course of the eight week study and continued to use the tool after the study. C3 worked individually on code that provided infrastructure support for a larger project. Table 2 shows a summary of the tagged waypoints created by C3, the type of tags used and how they changed. For this user, there were changes made consistently throughout the study time to the tags and waypoints, indicating that this user was able to manage and update the tagged waypoints effectively. When asked in the post-questionnaire if his use of the tool changed over time, he replied: "yes -- I started with tags that described the function of the code, but I moved to tags that were task-based (bugs, todos, API changes, etc.)". This user did not create any hierarchical tags, but used multiple tags frequently and in quite sophisticated ways. He also added comments for all the waypoints. One interesting tag this user added to a preexisting waypoint was: "badbadbad". The existing tags on this waypoint were used to indicate what he needed to do. The additional "badbadbad" tag was used as a prominent reminder.

There are bursts of commenting and tagging activity that occur just before a major release, change or refactoring. C3, in the post-questionnaire, said the most useful instance for using waypoints was "tagging particular changes in an API for later integration". C3 used TagSEA as a "todo/remember-this-stuff-for-later" tool. He tended to use TagSEA as a status indicator of his TODOs rather than to support navigation. The difference in use between C2 and C3 is reflected in Table 2, where C2's use of waypoints is more focused on asserting information, while C3's use of waypoints is more focused on TODOs. C3 said he would favor using TagSEA over tasks and bookmarks. He mentions that TagSEA is more flexible because the user can define their own tags directly in code and they show up in the tags view and "you can pivot around them".

B2. From the two programmers that responded to our email from the BC lab, only B2 downloaded the tool and completed the pre-questionnaire. B2 (male) was only able to use the tool for two weeks due to other deadlines at work. This programmer had over 10 years of programming experience. He worked on a demanding project with two other team members. He was unable to submit his source code to us for analysis. However, Table 2 shows the data from one version of his submitted tagged data. We also asked him to submit interaction data (logged by TagSEA) on how he used the tool. This data showed us that he used the waypoints to support his navigation activities, and that he also used the refactoring facility for managing his tags. In the interview, he mentioned that the refactoring facility was very powerful. He created hierarchical tags and added additional comments.

B2's intent behind tagging is summarized in Table 2. In the questionnaire he commented that he used waypoints for "exploring complex code paths" and "TODOs". He did not use them for asserting information. A post study interview confirmed that he used the tagged waypoints to support things that needed further examination. He also made several suggestions for improvements, including that TagSEA should "use icons to represent type of waypoint (TODO, documentation, code review, code execution path, etc)" and that "the context sensitive help system needs to reflect the tag hierarchy". We were not able to analyze his code, so we could not determine how he used ordinary comments or task annotations for navigation. However, in the post-study questionnaire, he indicated that he used neither bookmark nor task annotations. He felt that the Eclipse task annotations were worthless because they could be obscured by many general TODO task annotations generated by the IDE. Although he was aware that they could be customized, he indicated in the interview that it was too tedious. When asked about bookmarks in the questionnaire, he replied: "I never use bookmarking. TagSEA augments the IDE tools and should become a standard part of Eclipse".

6.2 Non-adopters

Two of the six users from Cambridge did not use the tool for technical incompatibilities and two used it for only a short time. Their reasons for not using the tool were determined through the focus group and interviews. One programmer, **C1**, did not adopt the tool because his work required Java and non-Java development, and TagSEA did not support the non-Java work. **C6** used an IDE configuration that could not run TagSEA.

C5, (male) a senior software engineer and expert with Eclipse, tried the tool on the first day, but later dropped it. His work involved creating and maintaining his own source code in a large project and he was intimately aware of how to navigate around his code. Thus he could not see any advantage to using TagSEA over Eclipse's navigational shortcuts (e.g. accelerator keys to jump to classes) and advanced features (e.g. browsing source control history to study changes). In the interview, he mentioned that he does not comment his code ("code speaks for itself"), but enters check-in comments into the version control system, which he can then review in the version history. He heavily uses a bug tracking system and version history browser to manage his tasks. We confirmed the lack of commenting through examination of his source code. We found only one task annotation within four months of revisions, and less than 10 descriptive comments that we coded as navigation markers.

C4 (male) is a software engineer on temporary assignment from another product division. C4 also largely worked with his own code that provided infrastructure support for a larger project. C4 used the tool for only a very brief time. Prior to using TagSEA he had created custom task annotation types via Eclipse's preference settings that equated to some, but not all, of the support that TagSEA offered. Unfortunately we could not access C4's source code for confidentiality reasons so we could not determine what kinds of comments he made. In the exit interview, C4 mentioned that he found typing tags to be tedious.

B1 (male) from the BC lab did not use the tool, despite downloading it. He indicated he did not use it because TagSEA was not compatible with his environment.

7. Anonymous users and TagSEA

In addition to the case study series where we had full access to the programmers and their source code, we collected, with consent, usage statistics from five early adopters of the TagSEA tool (A1-A5). Our goal was to understand the intent for creating waypoints (using the INFO and TODO categories). Data was automatically uploaded to our server every time the TagSEA tool was re-initialized. Tag names as well as the resource names and line numbers of waypoints were collected. Two of the experimenters independently examined the logged data and the number of the waypoints for each tag. Since we neither collected comments associated with the waypoints nor source code, we only provide a summary of their intent for tagging as in many cases we were not able to determine the exact user's intent. Table 3 provides a summary of the quantitative data, including the numbers of waypoints and unique tags, and the density of tags (see Sec. 5.3) from the last set of logged data. The quantitative data is discussed further in Sec. 8.

Table 3. Anonymous log data collected

User	#days logged	maximum depth of hierarchy	#way-points	#unique tags/ instances	tag density
A1	81	3	59	43/115	.37
A2	59	5	250	140/799	.18
A3	25	2	15	12/21	.57
A4	58	3	3	3/4	.75
A5	25	3	19	11/37	.30

7.1 Anonymous user #1 (A1)

During our analysis of the logged data, we noticed that three sets of logged files were overlapping for three different user IDs. The overlapping data comprises of 105 log files submitted over 81 logged days. There are two possible explanations. Either three users working on the same project agreed to participate in our study, or the same user enrolled in the study from three different workstations. From our data it is impossible to tell. We take a conservative stance and assume it is only one user.

This user made extensive use of the tag hierarchy to create a set of well-organized tag names. The leaves of the tag hierarchy are mostly composed of cryptic short-hand words. However, the more descriptive higher level nodes in the hierarchy can be used to determine the purposes of the leaf tags. From this, we were able to determine that the user used tags to indicate functionality in the code; both already implemented, and to be completed. This user's tags were categorized as INFO-FEATURE and TODO-CHANGE.

7.2 Anonymous user #2 (A2)

A2 submitted files covering 59 days. Most of the user's waypoints indicate locations where generic features are implemented. The predominant category is INFO-FEATURE. Hierarchical tags are used to organize tags according to projects or products (e.g. Project1.UI_Contribution, Project2.Layout). We replaced the project names with Project1, etc to protect the anonymity of the users' submitted data. There was one instance of a group of tags indicating one TODO-CHANGE event.

7.3 Anonymous user #3 (A3)

This user submitted 25 days of logged data. It was hard to classify some of this user's tags because they were written using characters outside of the Roman alphabet. However, the tags that we could read indicate an extensive use of the various TODO categories of tags. TODO-CHECK, TODO-CHANGE, and TODO-TEST are all used. Similar to A3, this user organized the tag hierarchy according to project names. (e.g. Project3.todo, Project3.test, Project3.review, Project3.documentme). This user also added some unusual tags to emphasize important areas in code, for example through the use of capitalization and the use of exclamation marks (e.g. Project3.!!!). These tags are indicative of the TODO-CHECK category. Finally, there is a small indication that this user used tags to indicate features of the software (INFO-FEATURE), as well as to document that he or she was the author of the code (INFO-AUTHOR). Overall, A3 made quite extensive use of tags for varied purposes.

7.4 Anonymous user #4 (A4)

A4 submitted 58 days of log data. Most of the tags appear to belong to the INFO-FEATURE category (e.g. copy.import.vector). There were a few scattered tags that indicate TODO events: TODO-CHANGE (e.g. "changerefname"); general TODO (e.g. "**.todo"); and possibly TODO-CHECK (e.g. "**.question"). We replaced two initials with "**" to preserve anonymity.

7.5 Anonymous user #5 (A5)

A5 submitted 25 days of logged data. This user has a very small tag hierarchy. It is used to mark-up code that implements an abstract model. There are tags that indicate future work. Given the small hierarchy and the lack of context, the tags are quite difficult to classify, but they generally indicate the classes of TODO-CHANGE and INFO-FEATURE.

8. Discussion

We discuss the insights we gained on the research questions posed as well as the study limitations.

8.1 Research Questions

Q1: For adopters, what kinds of waypoints and tags are created, and how do they evolve? Users from both groups used TagSEA for task management as well as to assert information about the program code and design. However, the majority of the use cases were for managing or reminding about tasks. One interesting result was that the two professional programmers from Cambridge did not, for the most part, use hierarchical tags. We speculate this may have been because these two programmers were experienced with social bookmarking tools and there are reports of experienced tagging users creating their own conventions to encode hierarchical relationships across tags [8]. The programmer from BC and the anonymous users all used hierarchical tags. We hypothesize that these users find hierarchies to be a useful way of organizing their tags.

Two of the three professional programmers that used TagSEA added comments to the waypoints. We cannot tell if the anonymous users added comments as we did not collect this data to maintain anonymity. We also see evidence of the users creating memorable tags to help them remember important code for future inspection. Memorable tags included "home", "badbadbad" and "Project-Name3.!!!".

The density of tags indicates that the tags are being used to document delocalized concerns in the code (see the density numbers in Tables 2 and 3). Most users used multiple tags (e.g. performance, todo) per waypoint. Tables 2 and 3 show us that two of the anonymous users created considerably more tags and waypoints than the other users and that they made extensive reuse of existing tags. The reason for this may have been due to more extensive programming activities during the study period, but more studies are needed to understand this difference.

We saw some indications that the tool support is effective at supporting the deletion, maintenance and creation of tags and waypoints. The logged data also shows evidence that refactoring can be used to achieve consistent tags. In summary, we noticed that the studied users co-opted TagSEA's features to support their task and documentation needs in very different ways. It will be interesting to explore in a future study how the tool's usage may also vary over a longer period of use.

Q2: For adopters, how does their use of waypoints compare to the use of tasks and comments? From the analysis of the source code from two of the Cambridge programmers, we were able to explore how they made use of source code comments, task annotations and waypoints. For asserting information, C2 relied most heavily on source code comments, but had a few waypoints for this category (perhaps to support perceived future navigation to these waypoints). C3 on the other hand used only source code comments for asserting information. For indicating tasks, C2 and C3 used waypoints for approximately half of these annotations. For the other half, C2 used mostly source code comments and a few Eclipse tasks, whereas C3 used Eclipse tasks. However, in the exit interviews both C2 and C3 mentioned a preference for waypoints over task annotations in the future. Unfortunately, we could not access the source code for the professional adopter in BC, but he indicated he did not use tasks.

Q3: Why do some users not use TagSEA? Since **t**hree of the recruited programmers did not use TagSEA for technical incompatibilities, the two users that used TagSEA only for a short time are the most interesting cases for us to consider.

C5 reported that the tool was not useful for him. We suspect that C5 replicated most of the benefits of TagSEA through disciplined management of external tools or advanced UI features. Likewise, C4 had already created custom task annotations and was frustrated by the non-working automatic tag completion facility. These results are interesting as they demonstrate that users are going to some effort to customize their environments to meet their navigation needs.

Finally, another theme to emerge from the focus group is that some users are reluctant to add shared comments in the source code and thus requested a way to privately waypoint outside the code.

Q4: How can TagSEA be improved? This is perhaps the most important question for us to answer, as an improved tool should enhance our ability to collect further insights on the previous three research questions. During the focus group discussion at the Cambridge lab there was significant discussion on whether waypoints should be personal or shared artifacts. The non-adopters in particular were concerned about this. One of the programmers used the phrase "graffiti" to describe the annotations added to the code. Others liked that the tagged comments were in the source code – e.g. from C3's questionnaire: "No -- public tags work for me. At worst, they're uninteresting to other people. At best, they help document the code." But there was a need expressed to tag code that has read-only access. Our conclusion from this is that both private and public waypoints are needed, but they should be presented to the user in an integrated way, rather than as separate tools with different user interface affordances. Moreover, the tool should have mechanisms so that annotations, if present in the code itself, could be filtered easily when formalizing a version of the code.

Based on the feedback we have received, the latest version of TagSEA (tagsea.sourceforge.net) has support for waypointing any resource (to support non-Java work and read-only code), as well as waypointing breakpoints (to help during debugging) and URLs (as programmers often refer to web pages for documentation). Importing and exporting tags and waypoints are now supported, thus facilitating sharing.

8.2 Limitations

While our qualitative approach allowed us to explore non-adopter issues, as well as the underlying factors that contributed to TagSEA and comment use, it is not possible at this point to generalize from these findings. A longer observation period would reveal if the tool would see long-term adoption and how the usage of tags may shift over time. There are also some limitations with the tool version used in the case study series. This immaturity could have affected, not just its usability, but also its usefulness. In particular, the version deployed in Cambridge did not have full support for refactoring and the automatic tag completion feature did not work as predicted – this may have had an impact on the programmers' use of hierarchical tags as well as influenced one programmer's adoption of the tool. Further studies are needed to explore the reason for the varied use of hierarchical tags and impediments to adoption.

9. Conclusions and future work

In this paper, we categorized a style of navigation mechanisms whereby the programmer defines personalized navigational structures over the software space. We further discussed how the user may appropriate other mechanisms to meet this need, such as adding temporary source code comments.

TagSEA provides affordances for the specific tasks of creating, accessing and managing user-definable navigational structures. Previous tools, such as Eclipse tasks and bookmarks, lack facilities to make these user-defined navigational structures both accessible and maintainable. ConcernMapper and JTourBus are similar as they provide support for saving sets of code locations that share a common theme. TagSEA can be used in a similar manner, but it is more flexible as multiple tags and hierarchical tags can be associated with each location thus providing more flexibility for viewing, filtering and accessing the tagged waypoints.

We are encouraged that some programmers in our study wish to continue using TagSEA, as it indicates the tool fills a perceived need. We believe part of TagSEA's appeal is the ease and flexibility of creating custom tag vocabularies directly in the source code, the expressiveness offered by defining info/todo tag names meaningful to them (which corresponds to users' preference for using their own vocabularies for objects and concepts [6]). While regular source code comments can be just as expressive (e.g. using expletives in [16]), TagSEA's custom vocabularies become first class entities in the Tag Tree UI for supporting navigation.

We suggest that whatever form user-definable structures take (i.e. bookmarks, concerns, bug reports), there are several key design principles that should be followed. The structures should be easy to create (i.e. lightweight), they should be clearly visible and easy to access, and there should be management support for creating, deleting, and refactoring the structures. The structures should offer facilities for collaboration and private/public management of information. Finally, the structures should allow for the additional of metadata to help users make navigation decisions.

Our studies so far have focused largely on individual TagSEA users in isolated projects. Thus, this work may inform, and be informed from, the social bookmarking space. We need to explore how waypointing, or even how simple bookmarks and source code comments, are used in a team that engages in a high degree of collaborative development work.

10. References

[1] M.-A. Storey, L.-T. Cheng, I. Bull, and P. Rigby, Shared "Waypoints and Social Tagging to Support Collaboration in Software Development." Proc. CSCW 06, Banff, 2006.

[2] L. Brothers, V. Sembugamoorthy and M. Muller. "ICICLE: Groupware for Code Inspection." Proc. CSCW 90, Los Angeles, CA, USA, 1990, pp. 169-181.

[3] D.R. Benyon, "Beyond navigation as metaphor." Proc 2nd EuroDL Conference. Crete, 1998.

[4] R. Deline, M. Czerwinski and G.G. Robertson. "Easing Program Comprehension by Sharing Navigation Data." In Proceedings of VL/HCC 2005.

[5] A. Dieberger and A.U. Frank, "A city metaphor for supporting navigation in complex information spaces." *J. of Visual Languages and Computing* 9, 1998, pp. 597-622.

[6] G.W. Furnas, T.K. Landauer, L.M. Gomez, & S.T. Dumas, "The Vocabulary Problem in Human System Communication: An Analysis and Solution," Comm. ACM 30(11), 1987, pp. 964-971.

[7] S. Golder and B.A. Huberman, "Usage Patterns of Collaborative Tagging Systems." *Journal of Information Science*, 32(2). 2006, pp. 198-208.

[8] M. Guy and E. Tonkin, "Folksonomies: Tidying up Tags?" D-Lib Magazine, Vol. 12, No. 1, January 2006.

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

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

[11] J.W. Creswell, "Research Design: Qualitative, Quantitative, and Mixed Methods Approaches", ISBN: 0761924426, 2002.

[12] M. Muller, J., Kuchinskaya, O., Minassian, S. O., Tang, J. C., Danis, C., Zhao, C., Harrison, B., and Moran, T. P. 2005. "Shared landmarks in complex coordination environments." In *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, 2005, pp. 1681-1684.

[13] G. Murphy, M. Kersten, and L. Findlater. "How are Java software developers using the Eclipse IDE?", IEEE Software, July/August. 2006, pp. 76-83.

[14] C. Oezbek and L. Prechelt. "JTourBus: Simplifying Program Understanding by Documentation that Provides Tours Through the Source Code." Working Paper TR-B-07-08, Freie Universität Berlin, Germany, April 2007.

[15] M.P. Robillard and G. Murphy, "Automatically Inferring Concern Code from Program Investigation Activities," Proceedings of ICSE, 2003.

[16] Selznak, "We Are Morons: A Quick Look at the Win2k Source," www.kuro5hin.org/story/2004/2/15/71552/7795.

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

[18] Javadoc Home Page, http://java.sun.com/j2se/javadoc

[19] A. Ying, J. Wright and S. Abrams. "Source code that talks: an exploration of Eclipse task comments and their implication to repository mining", Workshop on Mining Software Repositories (MSR '05), St. Louis, 2005, pp. 1-5.

[20] L.-T. Cheng, M. Desmond, M.-A. Storey, "Presentations by Programmers for Programmers," 29th International Conference on Software Engineering, 2007, pp. 788-792.