Carnegie Mellon University
Research Showcase @ CMU

Institute for Software Research | School of Computer Science

2006

JASPER: An Eclipse Plug-In to Facilitate Software Maintenance Tasks

Michael Coblenz
Carnegie Mellon University, mcoblenz@andrew.cmu.edu

Andrew Ko
Carnegie Mellon University

Brad Myers
Carnegie Mellon University, bam@cs.cmu.edu

Follow this and additional works at: http://repository.cmu.edu/isr

Published In
.

This Conference Proceeding is brought to you for free and open access by the School of Computer Science at Research Showcase @ CMU. It has been accepted for inclusion in Institute for Software Research by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

JASPER: An Eclipse Plug-In to Facilitate Software Maintenance Tasks

Michael J. Coblenz
Computer Science Department
Carnegie Mellon University School of Computer Science
mcoblenz@andrew.cmu.edu

Andrew J. Ko and Brad A. Myers
Human-Computer Interaction Institute
Carnegie Mellon University School of Computer Science
{ajko, bam}@cs.cmu.edu

ABSTRACT

Recent research has shown that developers spend significant amounts of time navigating around code. Much of this time is spent on redundant navigations to code that the developer previously found. This is necessary today because existing development environments do not enable users to easily collect relevant information, such as web pages, textual notes, and code fragments. JASPER is a new system that allows users to collect relevant artifacts into a working set for easy reference. These artifacts are visible in a single view that represents the user's current task and allows users to easily make each artifact visible within its context. We predict that JASPER will significantly reduce time spent on redundant navigations. In addition, JASPER will facilitate multitasking, interruption management, and sharing task information with other developers.

Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: User interfaces; D.2.6 [**Programming Environments**]: Graphical environments, Integrated environments, Interactive environments

General Terms

Design, Documentation, Human Factors

Keywords

Natural programming, Concerns, Eclipse, Programming Environments, Programmer Efficiency

1. INTRODUCTION

Software developers' time is highly fragmented [2]. Interruptions take them from their work regularly [6] and information that is vital in their tasks is often unavailable, forcing them to defer their work until later. Unfortunately, there are few ways that developers can keep track of the information that was relevant to their task, other than writing it down [8], using primitive controls like tabs and scrollbars to mark relevant information [5], or simply relying on their unreliable memory. Even when developers can work uninterrupted, the lack of a mechanism for tracking relevant information means that they must constantly re-find information, causing significant navigational overhead [5].

JASPER (Figure 1) is a new Eclipse plug-in that aims to remedy this problem by providing a workspace for developers to gather

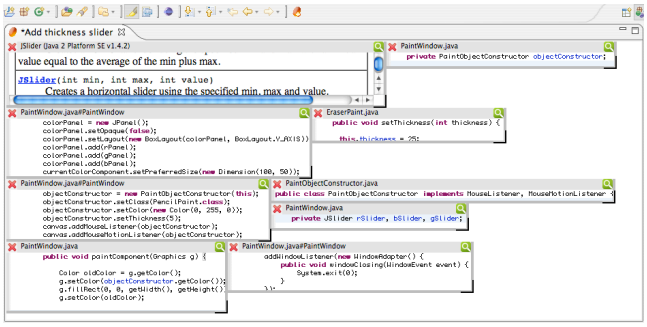

Figure 1. JASPER shows the working set for a user's task.

task-relevant information in a consistent, persistent, and straightforward manner. JASPER stands for **J**ava **A**id with **S**ets of **P**ertinent **E**lements for **R**ecognition. Developers can create a separate workspace for each of their tasks, and within each of these, add a variety of task-relevant information. Such information includes code fragments (defined at a line granularity and updated incrementally as code changes), portions of documentation or bug descriptions in HTML documents, and textual notes. Users select a relevant region and click the *add* button, and JASPER places the artifact in the current *working set*, which represents a particular task. All artifacts in a working set are visible in the working set's view for easy reference. Users can click on an artifact to easily navigate to its original context.

As software developers do their work, they frequently navigate among numerous software artifacts, such as code, documentation, and notes. Our research [5] has shown that software developers spend approximately 35% of their time performing the mechanics of these navigations in their IDE. Reducing this navigation time would be likely to significantly improve developer productivity.

Artifacts are task-relevant pieces of data that form small, coherent units. For example, a method implementation, a few contiguous variable declarations, or a particular syntactic element such as a certain `for` loop, may all be artifacts. As developers do their work, they locate relevant artifacts. In JASPER, users select a relevant region and click the *add* button, and JASPER places the artifact in the current *working set*, which represents a particular task. All artifacts in a working set are visible in the working set's view for easy reference. Users can also easily access the context of each working set item.

In addition to saving programmers' time navigating, we believe that JASPER may aid programmers in several other ways. Users of the system can share task information with other programmers. If a user is interrupted and cannot proceed with a particular task, another user could continue, taking advantage of the information collected by the first user. Working sets can be archived as part of a version-control system. Then, when considering changes related to a particular task, users may examine the previous working set

used. This may give context to future tasks, helping to answer questions such as "why did the previous developer not notice this bug?" The answer might be of the form "because a particular line of code was not in the working set, the developer probably did not know about a dependency." It can also reveal dependencies: future users may not understand all the dependencies involved, but inspecting dependencies discovered by other users may help explain the changes. Working sets can also be thought of as each representing a particular aspect of the software under development. They may form a concise, convenient way of informally documenting aspects in the context of aspect-oriented programming. Working sets represent cross-cutting concerns, and as such, could be used by the aspect-oriented programming community as documentation.

In this paper, we will describe some of the related work in this area, and then describe JASPER's use and implementation. We will end with a brief discussion of some of the implications of JASPER's design on other software engineering tools.

2. RELATED WORK

Existing development environments do not support collecting small segments of code. Instead, they force users to choose artifacts at the file or syntax granularity rather than the code granularity. In Eclipse, users must first select a file to view and then scroll through the file, or select an artifact from a long list. Because artifacts are always shown in the context of their files, only a small number of artifacts may be displayed simultaneously. The result is that users typically must navigate away from relevant code, requiring them to subsequently re-locate artifacts that were previously visible.

Eclipse allows users to arrange multiple panes to try to have relevant artifacts be visible. But this is too cumbersome. In the study described in [5], users never arranged their Eclipse window to show all the relevant information for their tasks. Even opening a split view to show just two different regions of a file requires dragging the file's tab to create a split, dragging the file to put a copy of the file in the previous pane, and individually scrolling each pane to the relevant region. Furthermore, even if one were to do this, a large portion of the screen would be occupied with only two artifacts. But in the study in [5], one task, for example, required artifacts from about four different files, and each file typically had several relevant artifacts. It would be impractical to manually arrange the panes in Eclipse to show this information. In Microsoft's Visual Studio 2005 and Eclipse 3.2, only tabs and panes are available, not overlapping windows. Previous versions of Visual Studio included support for overlapping windows, but this feature was removed, presumably because people found it too hard to use.

Although no other systems have the goal of providing an explicit way for developers to create documents that represent their tasks, other systems have had related goals. Eclipse has, as of version 3.1, a feature already called "Working Sets," but it is very limited. It essentially acts as a filter on the existing Package Explorer, limiting the view to files or projects chosen for a particular working set. Although this may help, its functionality is extremely restricted. Mylar [4] is an Eclipse plug-in that helps users collect and view *frequently used* artifacts (as opposed to artifacts explicitly chosen by the developer). Like JASPER, Mylar displays task-relevant data to reduce the time that programmers spend on navigation. However, instead of displaying the *contents* of the artifacts, Mylar only displays the name of each. Mylar shows a list of elements of the Java model that are relevant according to a degree-of-interest model, which considers user actions such as navigations and edits to infer user interest in artifacts. However, Mylar is limited to showing these syntactic elements: it cannot represent arbitrary sequences of lines of code, such as the first three lines of the body of a certain `for` loop. These artifacts were shown in [5] to be relevant to programmers, so this limitation hinders Mylar's ability to show relevant information.

Automatic inference may be a benefit to Mylar, but it may also be a hindrance. Users will not be able to predict whether Mylar will infer the relevance of a given element, so when trying to navigate to an item, they must first check Mylar's list, and if it is not there, then use the main list in the package explorer. This unpredictability may reduce Mylar's effectiveness. JASPER is predictable because it does not add or remove items automatically.

FEAT [7] represents frequently used artifacts as *concern graphs*. A *concern* represents task-relevant data, and consists of syntactic elements. Users must learn to use FEAT's interface to navigate various kinds of dependencies among elements to locate relevant elements. The result is a graph of elements where the edges in the graph are relationships between elements. But the study in [5] showed that relevant artifacts are not limited to those kinds of items; sometimes only a line or two of a large method implementation is relevant. Furthermore, FEAT does not facilitate viewing the contents of many elements in a concern simultaneously, as JASPER does. An extension to FEAT, described in [9], allows the system to automatically infer concerns from a user's interactions with the IDE. FEAT, in addition to helping navigation, helps users find relevant artifacts. JASPER does not have this goal, since many of the dependencies expressible in FEAT are already navigable in Eclipse.

JQuery [3] is a query language for Java code to help users visualize the structure of a search through software. However, JQuery does not help manage tasks or working sets. Instead, it helps users record the history of their searches through the source code. The resulting representation does not directly represent the working set; it instead represents the search.

3. WORKING SETS IN JASPER

JASPER is an Eclipse plug-in that maintains a list of working sets on which a user is working. Each working set consists of several working set items. A working set is intended to correspond to a particular task, or goal, that the user intends to accomplish. For example, a user might be working on a small drawing application, and have several tasks: fix the "undo" feature; add a tool for drawing lines; and permit users to change the thickness of drawn lines. Another example might be a refactoring unsupported by automated tools, which requires a careful inspection of several interrelated parts of a system that are distributed amongst several source files.

In Figure 2, a user of JASPER has created a new working set by clicking the New button, and named it according to the task: add a thickness slider to the Paint program. Double-clicking a working set icon opens a pane that displays the contents of the working set. Users may also load a saved working set from a file by clicking the "Open…" button and save an existing working set by choosing the "Save" item in the File menu.

After creating a working set, the user continues to work as usual. When the user finds a relevant artifact, it can be easily added to the working set. There are many ways to create a new item:

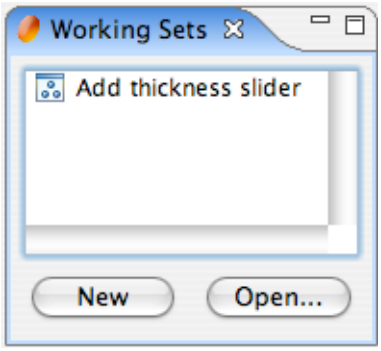

Figure 2. A list of working sets, containing one item.

- Dragging a URL into the working set creates a new URL item for any web content. Dragging text in creates a new text item whose contents are initialized to that text.
- When a text editor or Java editor has the focus, an "add" button, , appears in the toolbar (see the top of Figure 1). When a user clicks this button, JASPER adds the selected text or Java code to the working set. The user can later retrieve the context of the item by double-clicking on the working set item or clicking a magnifying glass icon in its title bar. JASPER then opens a standard Eclipse Java editor for the file and scrolls it so that the item is visible. If an editor for that file is already open, JASPER re-uses it and just scrolls to the correct place.
- When a working set view has the focus, a "new note" button, , appears in the toolbar (see top of Figure 3). When a user clicks the button, JASPER creates a new, empty text item.

Figure 3 shows a working set that contains three items: a web page with documentation for JSlider, a text note, and some Java code. The Eclipse toolbar is visible at the top of Figure 3, including the "new note" button and the "auto-layout" button, described below.

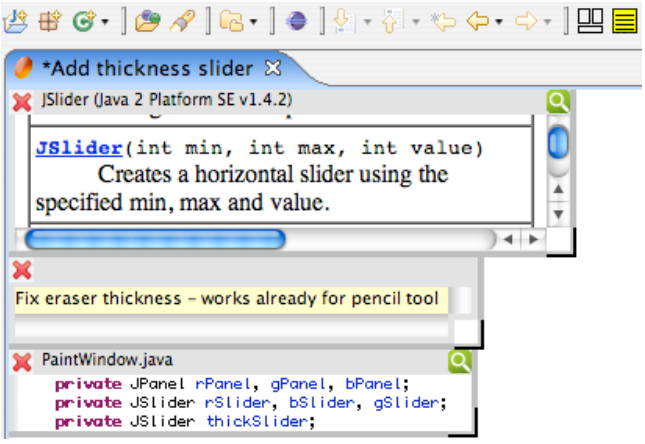

Figure 3. A working set that includes three items.

The interaction with existing working set items is modeled after the standard GUI windowing paradigm. In JASPER, working set items behave similarly to windows: they can be dragged or resized as desired, and closed when no longer needed. This offers an interface that is familiar to the user and consistent with traditional interaction techniques. The working set view provides a workspace in which users may manipulate working set items; items may not leave the view. All items have a title bar, which shows information about Java items and URL items.

When users create Java items, JASPER automatically chooses a size that fits the contents of the item. However, JASPER does not have enough information to choose a correct size for empty text items (it cannot be known *a priori* how much text the user will type) and for web pages (the page is likely to be large, with only a small relevant portion).

Even though JASPER can size Java items correctly initially, further edits may change the appropriate size. Since the user may have manually positioned items, it would be inappropriate for the system to automatically resize items after changes. Even if resizing were acceptable, it would be likely to obscure an adjacent item, requiring items to move automatically. But this would interfere with the user's spatial memory of the locations of items. Therefore, each item has a resize widget in the lower right corner.

Java items consist of a contiguous sequence of lines of source code from a file. If the code corresponding to the working set item is edited, the text shown in the working set item is updated immediately. If the original file is edited above the working set item, JASPER ensures that the working set item is updated so that it displays the same code. The code is formatted exactly as it was in the original view, including syntax coloring and indentation so it will look familiar to the user. However, when necessary to fit all the items in the view, the code is shown using a smaller font so that more code will fit in the working set view at the same time.

Java items are not directly editable in the item view. This is because, in informal observations of developers [5], they almost always preferred to see a significant amount of context when editing. Therefore, developers will be unlikely to make any significant edits directly in the tiny working set view, so the space that would be required for all the controls necessary to control the view, such as scroll bars, is better used to display other items. When clicked, then, instead of displaying an insertion point, Java items can be dragged. This makes the drag region significantly bigger, which is useful since it is expected that dragging will be a relatively common operation.

URL items uniquely identify a particular web page. The URL item displays the web page with scroll bars so that the user can choose which portion of the page to view. URLs are commonly used for referring to bugs (e.g. Bugzilla bug reports) and documentation. Javadoc is typically viewed in an external web browser, which would normally require frequent switching between the IDE and the browser. But JASPER allows users to keep frequently used items, such as documentation, visible in the IDE. URL items behave as the platform-default web browser does. For example, links are clickable, and images are properly rendered. However, to reduce space requirements, there are no additional browser controls on the URL items, such as back and forward buttons. When a user clicks the magnifying glass icon in the title bar, JASPER opens the page in the platform-default external web browser.

Text items allow users to record notes and information about the task. These items are editable within JASPER, since they are intended to be used for taking notes and recording information. Text items may be copied from text files or entered directly by the user using JASPER. Because text items are part of the content of the working set document, and not pointers to content like code and URL items, there is no magnifying glass icon. Since text items are editable, they include scroll bars. This enables maximum flexibility: users can have many or only a few text items, and they can choose to have only a portion of certain items visible — some

portions may be irrelevant. The scroll bars then serve as a visual indicator that some information is not currently visible.

4. AUTOMATIC SIZING AND LAYOUT

We designed JASPER to represent code and other content literally, as in Figure 1, rather than as a list of summaries or some more succinct and automatically laid-out view. One consequence of this decision is that manually positioning each item could be a time-intensive operation, especially since upon adding an item users might need to rearrange several other items to make room. Instead, JASPER automatically chooses a size and location for each new item. The size is chosen to be just large enough in each dimension such that the text of the item fits inside it. The position is chosen so that the new item does not overlap with any other item being displayed. Because choosing the position optimally is NP-hard, JASPER uses an approximation of an optimal solution.

JASPER automatically shows the working set so that the items are as large as possible while still fitting in the view. The scale of the items is selected automatically when items are added, moved, or deleted, and is constrained so that the text never gets too large. If an added item does not fit, then the scale of the entire working set view is decreased so that there will be enough space to fit all of the items. The font size is proportionally decreased, in addition to making the items smaller, so that all the text of the items remains visible. If an item is removed, JASPER checks to see if the entire view should be rescaled so it is larger.

Figure 4 shows how JASPER displays the working set from Figure 1 in a smaller pane. Each item is automatically scaled down so that the entire working set is visible.

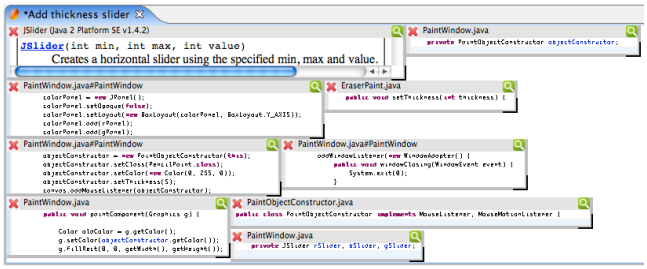

Figure 4. Same as Figure 1, but using a smaller pane.

When scaling working set items, the title bar text remains the same size—only the item content is scaled. This allows users to still distinguish items using the text in the title bar. If the title bar text were scaled, then items could become completely indistinguishable except for location and shape. By not scaling the title bar text, JASPER ensures a minimum height for all items. However, if the user continues to add items, then the titles continue to become narrower to accommodate new items, and would eventually become too small to read. However, we believe this would be unusual in practice since this would only be necessary when there are several dozen items, which is significantly larger than the working sets seen in [5]. Future work will be necessary to determine how large real working sets become. JASPER does not scale URL item content because of implementation difficulties: the SWT Browser widget does not support scaling.

If the user explicitly repositions an item or changes its size, that item is not moved when items are added or removed. This enables users to develop a spatial memory of items.

The incremental layout can be improved on: if the system could arrange all of the items at the same time, the heuristic is likely to produce a better arrangement. However, doing this automatically would result in an unexpected, counterintuitive change, since users are likely to develop a spatial memory for the positions of the items. For this reason, JASPER only moves all items on user command. JASPER has an "auto-layout" button, which rearranges all of the working set items according to its heuristic. The button, , is visible in Figure 3 at the top.

Details about the implementation of automatic sizing and layout are described in [1].

5. REFERENCING CODE

Implementing Java working set items presents a challenge, since the source code files containing the code fragments in a set may change as other developers on a team submit changes. For example, if changes are imported from a version control system to a file that the working set refers to, the location of the item may change. In fact, the item itself may change: lines may be inserted or removed, and the item may disappear entirely. JASPER must detect these changes in a robust fashion when the file is loaded and, where possible, repair references to working set items.

To facilitate reference repair, references to Java code consist of the project name, a path within the project to the file containing the code, and the line numbers of the beginning and end of the item. In addition, JASPER stores as a string the text of the item as it is currently known each time the item is archived.

JASPER must be robust to changes, but alert the user if it was unable to find the referenced item. There must be some tolerance for imperfect matches so that the item will be maintained even if it has been edited. Therefore, JASPER performs a search in the new file for the code in the item as it was last saved. Every line in the new file is scored: +1 if it matches some line of the item, and -1 otherwise. Then, JASPER finds the region that defines the *maximum contiguous subsequence* sum of scores. Ties are broken by biasing the results toward the location of the original item. This allows for lines that do not match as long as enough of the surrounding lines do. If more than half of the found region was not in the original item, the user is alerted by displaying the found region with a bright red background. In Figure 5, the lines directly above the shown line were part of the working set, but the entire working set item contents were deleted from the file outside Eclipse.

Figure 5. The working set item could not be found, so nearby text is shown with a red background.

If changes to the referenced files are made using Eclipse, JASPER detects these changes and updates the working set item's reference to the changed code. This enables different treatment of edits made using Eclipse than edits made outside. JASPER adds itself as a listener to the document that represents the file containing the item. Then, when a change is made to the file anywhere in Eclipse, JASPER is notified. The reference to the working set item is updated so that it still points to the same text. However, if the edit modifies the item itself, the reference is updated to include the union of the old item text and the new item text. This ensures that any lines the user marked as relevant are kept, and also that their replacement code is kept as part of the working set item. If the stricter approach of including only lines that existed before the change were taken, then as the user made changes, the working set items would slowly grow smaller and smaller, and might eventually not include the relevant code.

6. FUTURE WORK

One useful extension to JASPER would be to add other ways to add items to working sets. Mouse gestures could be used to select items; for example, circling a block of text could add the circled text to the working set. There should be a keyboard shortcut for creating working set items. A choice should be added to the contextual menu for code, text, and web pages to add the selected item to the working set. URLs can already be dragged and dropped in the working set view, but it should be possible to do so with text and Java code as well.

In addition to these manual methods of adding items to working sets, there may be automatic approaches that are not excessively intrusive. A view could always display several automatically-derived items not in the current working set, and a button would let users easily add the inferred items if they are relevant. The system could also take advantage of navigations and other actions that users already perform — for example, automatically adding any items that are edited.

JASPER should be integrated with other research systems that complement it. FEAT [7] could allow much more versatile and robust working set items. JASPER could allow working set items to contain arbitrary FEAT concerns. The references to code could also be improved by enabling references to be FEAT concerns. In addition, JASPER could be integrated with an artifact recommender, such as Mylar or the recommender for FEAT. A special kind of working set item could always show the system's current recommendation of relevant code, and a single click could add the suggested item to the working set.

References to code may need to be adjusted to reflect more closely the external changes that happen most frequently when JASPER is in use. JASPER scores lines; it could instead work on a per-character or whitespace-delimited token basis, or it could use a minimum edit distance model like UNIX *diff*.

We are currently planning a public release of JASPER. A version of JASPER that recorded anonymous usage statistics would be released. These statistics, with the user's explicit permission, would be sent to the experimenters for analysis. Using this data, we could evaluate whether users find JASPER to be a useful tool. This would also be a valuable way of gathering direct feedback from users, and is especially important for evaluating how well JASPER scales when used on real problems.

7. EXTENSIBILITY OF ECLIPSE

Eclipse is an immensely extensible environment for software development. Extension points allow plug-ins to add functionality to many different features of Eclipse. This permitted development of JASPER, which would not be possible in most other existing IDEs. However, we were greatly hindered by the poor documentation of Eclipse and SWT. The class used to display Java code (`CompilationUnitEditor`) is an internal Eclipse class, not for public use, but we needed to use it in order to make Java items look the same as the original code. Documentation is frequently available in the form of JavaDoc for individual classes and some tutorials, but information about the architecture of the system is difficult to find. This makes answering questions like "which class should I use for X?" and "I have an instance of class X; how do I get an instance of class Y?" very difficult.

Plug-ins indicate to Eclipse how they integrate with the platform via an XML file. Unfortunately, the tags used are poorly documented. The documentation for some tags have a "not yet implemented" warning, but no reference to the appropriate tag to use for that functionality. Errors in this XML file are nearly impossible to debug because Eclipse gives no feedback when errors occur. For example, adding a button to the toolbar was a difficult task because it is necessary to fill in several attributes of an XML tag, and when the wrong values are given, the only feedback is that the button does not appear.

The SWT drawing system lacks basic features like transparency, so some approaches to showing working set items that we considered were impossible. A two-week-long attempt to port the system to Draw2D and GEF (the Graphical Editor Framework) ended in failure because of poor documentation, complex model requirements and complex interactions among classes, and great difficulty in customizing default functionality of GEF. We hope that in the future, developers of Eclipse will focus more on creating high-quality documentation.

8. CONCLUSIONS

The design of JASPER was driven by the results in [5]. Using data about how real programmers use Eclipse has led to inspirations for new tools that otherwise might not have been designed. Observing how users refer to many different artifacts suggested that a tool like JASPER would be useful. However, because the code that users modified in [5] was only 508 lines long, future feedback from users will be needed to see how JASPER scales with larger amounts of code. We are confident that our user-centered approach to tool design will continue to provide insights for tool design and refinement.

9. REFERENCES

[1] Coblenz, M. J. JASPER: Facilitating Software Maintenance Activities With Explicit Task Representations. M.S. Thesis. Technical Report CMU-CS-06-150, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. 2006.

[2] Gonzalez, V. M. and Mark, G. Constant, Constant, Multi-Tasking Craziness": Managing Multiple Working Spheres, CHI 2004, Vienna, Austria, 113-120.

[3] Janzen, D. and De Volder, K. Navigating and querying code without getting lost. In Proceedings of Aspect Oriented Software Development, Boston, 2003, 178-187.

[4] Kersten, M. and Murphy, G. C. 2005. Mylar: a degree-of-interest model for IDEs. In Proceedings of the 4th international Conference on Aspect-Oriented Software Development (Chicago, Illinois, March 14 - 18, 2005). AOSD '05. ACM Press, New York, NY, 159-168.

[5] Ko, A. J., Myers, B. A., Coblenz, M. J., Aung, H. H. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. IEEE Transactions on Software Engineering, to appear, 2006.

[6] Perlow, L. The Time Famine: Toward a Sociology of Work Time, Administrative Science Quarterly, 44, 57-81, 1999.

[7] Robillard, M. P. Representing Concerns in Source Code. Ph.D. Thesis. Department of Computer Science, University of British Columbia. November 2003

[8] Robillard, M. P., Coelho, W., and Murphy, G. C. How Effective Developers Investigate Source Code: An Exploratory Study. IEEE Transactions on Software Engineering, 30(12):889-903, December 2004.

[9] Robillard, M. P. and Murphy, G. C. Concern Graphs: Finding and Describing Concerns Using Structural Program Dependencies. ICSE 2002.