Comparing Episodic and Semantic Interfaces for Task Boundary Identification

by

İzzet Safer

B.Sc., Koç University, Istanbul, Turkey, 2005

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

The Faculty of Graduate Studies

(Computer Science)

The University Of British Columbia

April, 2007

Abstract

Multi-tasking is a common activity for computer users. Many recent approaches to help support a user in multi-tasking require the user to indicate the start (and at least implicitly) end points of tasks manually. Although there has been some work aimed at inferring the boundaries of a user's tasks, it is not yet robust enough to replace the manual approach. Unfortunately with the manual approach, a user can sometimes forget to identify a task boundary, leading to erroneous information being associated with a task or appropriate information being missed. These problems degrade the effectiveness of the multi-tasking support. In this thesis, we describe two interfaces we designed to support task boundary identification. One interface stresses the use of episodic memory for recalling the boundary of a task; the other stresses the use of semantic memory. We investigate these interfaces in the context of software development. We report on an exploratory study of the use of these two interfaces by twelve programmers. We found that the programmers determined task boundaries more accurately with the episodic memory-based interface and that this interface was also strongly preferred.

Table of Contents

List of Tables

List of Figures

Acknowledgements

It is impossible to say how grateful I am to my supervisor Dr. Gail C. Murphy. I would probably not have completed this thesis on time without her wise suggestions, fast response time and calming attitude. I have learned a lot thanks to her guidance, especially how to keep things straightforward and simple. I am really fortunate to have her as a supervisor. I also appreciate the opportunity to work for a summer at IBM CAS Toronto as this was a life changing experience.

I would like to thank my second reader Dr. Joanna McGrenere for her constructive comments, fast reply and last minute help. Many thanks to Dr. Mik Kersten for creating such a great tool that inspired me on the main ideas of my thesis. Without Mylar, this thesis would not exist. I am also thankful to Brian de Alwis for his input on the user study design, and to Leah Findlater and Karyn Moffatt for their help on the quantitative analysis. I would also like to acknowledge the time and efforts of the subjects who participated in the study; thank you for your great input to my research.

My friends in the Software Practices Lab provided for engaging conversation and helped me socialize and relax: Jennifer Baldwin, Brett Cannon, Andrew Eisenberg, Thomas Fritz, Terry Hon, Chris Matthews, and Arjun Singh. I want to explicitly thank my roommates Jesse Gittens and Kenneth Rose; it was a pleasure living with you guys. I have really learned a lot from you; thanks for being there for me whenever I need help.

Thanks to my family, especially my parents; even though they were not fully aware of what I am doing, they let and supported their only son to move far away from home and study.

Last but not least, I would like to thank Kirsten Davidson for the encouragement she gave me, and her constant support whenever I encounter hard times. Thanks for being with me.

To a new life,
İZZET SAFER

The University of British Columbia
April 2007

Chapter 1

Introduction

Many computer users work on multiple tasks per day, switching between those tasks often [4, 8]. Each task typically involves more than one software application, such as a word processing document and a web browser. Each application is independent of the others. When a user switches a task, he must recreate the state of each application—loading documents, finding appropriate information in the documents—to work on the new task.

Several approaches have been proposed to address the multi-tasking nature of a user's work. In virtual workspace approaches, a user can create a workspace for a particular task that contains the appropriate applications, such as an email reading workspace with an email reader (e.g., [10]). Other approaches are more dynamic, associating information with a task either manually through queries (e.g., [5, 7]) or automatically based on how a particular application is typically used for tasks (e.g., email [1]). To support a wider variety of tasks, other approaches have focused on associating a user's interactions with applications and documents automatically with an identified task [6, 11, 14]. For instance, in the TaskTracer system, a user names and identifies the start of a task and the system then tracks and associates documents used with the active task, making it easy to recall the information for that task at a later time [6].

A difficulty with asking the user to identify when a task starts and ends is that a user can sometimes forget to perform the identification. A missed task boundary means either that information accessed may not be associated with any task or that it may be associated with the wrong task [18], depending upon the details of the user interface (UI). Although there has been some preliminary work on inferring task boundaries through the application of machine learning [18, 20], the predominate means of identification in use today is manual. This thesis investigates interfaces to help a user who forgets to explicitly indicate where a task starts and ends. In particular, this thesis investigates whether an interface based on use of episodic memory (personal experiences) or semantic memory (knowledge about facts) provides a better basis to help a user identify points (start and end points) in their previous programming work. In this thesis, these interfaces are investigated in the

context of the work of software programmers.

1.1 Mylar: A Task-Oriented Environment for Programming

An estimated 20,000+ programmers are currently manually indicating where a task starts and ends in the context of Eclipse Mylar, which supports a task-focused style of programming[1] [15]. In this thesis, we use Mylar as an example system in which to investigate task boundary identification. The users targeted in this investigation are thus programmers. We discuss the implications of this choice in Section 5.2.

Mylar reduces the information shown in the Eclipse integrated development environment (IDE)[2] to just the information relevant to the task at hand [14]. Mylar determines what is relevant for a task based on the interactions performed by a user as part of the task [15]. Mylar creates a context—a set of resources[3] and relationships relevant to the user's task—based on how the user interacts with the system's resources and with Eclipse. The task context mechanism determines a degree-of-interest (DOI) value for each resource and relationship based on the frequency and recency of interactions with the resources. The resources and relationships with the highest DOI are used to focus the UI.

Mylar supports task switching: when a user switches to a new task, the contents of Eclipse views are updated according to the task context of the newly activated task. Mylar tasks are created, activated, deactivated, marked complete, and marked incomplete manually by the users. A user must activate a task explicitly; this activation marks the start point of a task to Mylar. The end boundary of a task occurs either when the task is deactivated explicitly or when another task is activated, which implicitly deactivates any currently activated task. Users can sometimes forget to change to the appropriate task or deactivate a task once work on the task is complete.[4] Mylar users have also noted this problem with task boundary identification.[5]

[1] `http://www.eclipse.org/mylar`, verified 11/04/07; usage estimate is a personal communication with M. Kersten, 11/04/07.

[2] `http://www.eclipse.org`, verified 11/04/07.

[3] Throughout the thesis, we will refer programming elements like classes, methods, fields etc. as resources.

[4] It is possible to have no currently active task with the Mylar UI.

[5] Bug #166489 on `http://bugs.eclipse.org/bugs` and in `mylar-dev` mailing list on `http://dev.eclipse.org/mhonarc/lists/mylar-dev/msg01218.html`, verified 11/04/07.

1.2 Interfaces for Task Boundary Identification

In this thesis, we report on an investigation of two different styles of user interfaces that we have built to help a user (programmer) go back in time and identify where a task occurred. These interfaces could be used to reset the start of a currently active task or could be used to extract a task from previous work completed by a user. One style of interface provided by our Animated Snapshots tool stresses the use of *episodic* memory, using an animation to replay screen captures taken automatically as a user worked; a user can indicate during the replay which point in their work indicated the start or end of a particular task. The other style of interface provided by our Interaction History tool stresses the use of *semantic* memory, enabling a user to find particular points in their previous interaction based on the content of the information accessed.

We present the results of an exploratory study into the effectiveness and preferences of these two tools by users. In this study, we had twelve programmers use the tools to re-identify task boundaries in four programming tasks that we asked them to perform. We were interested in such questions as whether there were differences in the accuracy of the boundaries determined using the different approaches, how the subjects defined where a task boundary occurs, and the subjects' preferences for the episodic versus semantic approaches used in the tools. We found that the subjects found boundaries more accurately with the episodic-based Animated Snapshots tool. We also found that they strongly preferred the episodic-based tool.

1.3 Overview of the Thesis

This thesis begins with a description of current work in task-oriented user interfaces (Chapter 2). We then describe the two tools we built (Chapter 3) and argue how they each stress a different kind of human memory system for recalling tasks. We present a description of our study and its results (Chapter 4), before discussing our findings (Chapter 5) and concluding (Chapter 6).

Chapter 2

Related Work

As outlined in the introduction, a number of approaches have been proposed for providing better task support for computer users. We focus on task support that links relevant information to tasks. We also provide background information on episodic and semantic memory.

2.1 Task-Oriented User Interfaces

Some efforts in task-oriented user interfaces require a user to manually manage information associated with tasks. Rooms [10] is a window management system that allows users to organize their various tasks in different virtual workspaces. Each *Room* contains several windows specific to a task (e.g., a mail reader, a text editor and a programming editor). A user manually creates rooms and can navigate between them; windows can be shared or even moved across rooms and tasks. The Lifestreams [7] and Presto [5] systems manage documents and user events without an explicit organizational structure as in Rooms. Lifestreams uses chronology as the main storage attribute associated with events; Presto makes use of metadata attributes to categorize the interactions. Both let users mine streams of events or documents through queries, which in fact creates a virtual view for a task (e.g., all mail conversations with John). Bellotti et al. show that threads of email messages can often be representative of tasks [1]. Their Taskmaster email and task management system keeps the context of a task as one: threads, links, attachments and documents are grouped together. Users found this organization useful; however, tasks were limited to those exchanged in email. The GroupBar [19] and Scalable Fabric [17] systems let users do the task management manually by grouping the application windows that are used together. GroupBar augments the application desktop toolbar mechanism by providing the ability to collect running applications in different groups and lets groups to be treated as single units. Scalable Fabric holds groups of unused applications in the periphery of the screen instead of the traditional minimization.

Other systems use monitoring of a user's actions. UMEA was the first

system to monitor a user's actions to determine what information was part of a task and to present interfaces that allowed a user to see just the information related to a particular task [11]. UMEA captured a user's actions at a fine-level of granularity. TaskTracer used a similar approach but introduced an architecture to enable the collection of interaction for particular kinds of tasks, such as capturing only office document interactions rather than every window interaction [6]. Mylar refines this approach in the domain of programming by introducing a degree-of-interest model for the information accessed as part of a task [14, 15]. This degree-of-interest model is used to focus the information presented in the interface on just that subset relevant to the task at hand and is used to drive actions for the user, such as performing automatic searches on references to the resources of highest interest. WindowScape also uses an interaction history mechanism to provide a task-oriented window manager [21]. Instead of associating information with tasks, WindowScape makes it easy for a user to return to a previous arrangement of windows by selecting a snapshot of the screen captured automatically and presented in a timeline.

The three first approaches, UMEA, TaskTracer and Mylar, all have an explicit notion of project or task. Each of these approaches requires the user to explicitly name, activate and deactivate tasks. More recently, the TaskTracer project has considered implicit identification of tasks through the application of machine learning techniques to automatically recognize the active task, task switches or predict if a new task has started [18]. Shen and colleagues report that a hybrid approach of two machine learning methods, Naïve Bayes and SVM, yields the best performance and the most accurate task predictor. Their task predictor based on application windows and documents attains a maximum precision of 80% with coverage of 20%, whereas their predictor using incoming emails reaches 92% precision with 66% coverage. The coverage values indicate the proportion of time a prediction can be made. These results are promising but require higher coverage values before they can be put in production systems.

2.2 Memory Models and Interfaces

Humans have different systems for recalling and knowing information. Two common systems described in the psychology literature are episodic and semantic memory [12]. These two models vary in use of autobiographical versus cognitive characteristics and in the nature of the stored information.

Tulving describes the episodic memory as history embedded in personal

experiences of what happens to an entity [22]. These experienced events always occur at a particular place and time compared to other events [22]. Episodic memory helps answer the time (*when?*) and place (*where?*) of events. The semantic memory is the memory for facts, words and concepts. Semantic memory helps answer facts (*what?*) about events [23]. Considering the case of programming activity, a programmer uses both of these memory models when he or she thinks about previous work. The episodic memory may be more useful to recall personal experiences during programming such as, sequences of actions, the state of the screen, etc. The semantic memory may be more useful when it comes to knowing the names of resources.

Most interfaces for searching stored information stress semantic memory, such as search engines for the Web. The xMem (extended memory navigation) project [2] evaluates the use of the semantic memory analogy in creating a more efficient Web browser history mechanism. xMem helps users access already visited pages easily through keywords. The keywords are assigned automatically to a page according to its content and are chosen from a repository. The authors found that users are faster and more satisfied finding the Web pages in the navigation history. Several interfaces have investigated the use of features that instead stress the use of episodic memory. As one example, Czerwinski and Horvitz showed that users who were reminded with videos of computer work performed a month ago were better able to recall most of the events that occurred during that time [3]. As another example, Ringel et al. showed that the use of personal (e.g., calendar appointments) and public (e.g., holidays, news headlines) landmarks as cues to remember a particular event helped users find requested items significantly faster [16].

The work presented in this thesis differs from previous efforts in three ways. First, it is the only work of which we are aware that focuses on interfaces for determining task locations from previous activity. Second, in comparison to the Czerwinski and Horvitz work [3], our work considers the recall of tasks performed recently, within the last two hours. Finally, we provide a comparison of semantic and episodic memory based interfaces to address the same problem.

Chapter 3

Tools

To support an investigation into which style of interface programmers prefer for determining task boundaries, we developed two tools: the Animated Snapshots tool, which stresses usage of episodic memory and the Interaction History tool, which stresses usage of semantic memory (Figure 3.1). In this chapter, we explain these two tools, describe common elements employed by the tools, and present implementation details. Each of these tools runs within the Eclipse IDE.

The descriptions of the tools in this chapter focus on the features and configurations used in the user study reported on in this thesis. Other features are likely needed for use in practice; these are described in Chapter 5.4.

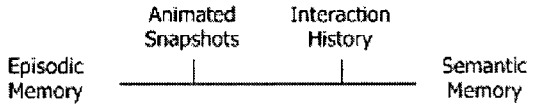

Figure 3.1: Tools to find task boundaries. The Animated Snapshots tool presents a user's personal experiences through snapshots of the user's working environment. The Interaction History tool presents resources and actions applied to the resources.

3.1 Animated Snapshots Tool

The Animated Snapshots tool silently captures snapshots of a programmer's screen as the programmer works and then provides a means of replaying the captured snapshots to locate the boundaries of tasks. We believe that by providing the overall look of the programmer's screen we are providing an indication of a personal experience that will enable the use of the programmer's episodic memory to recall where a task started or ended.

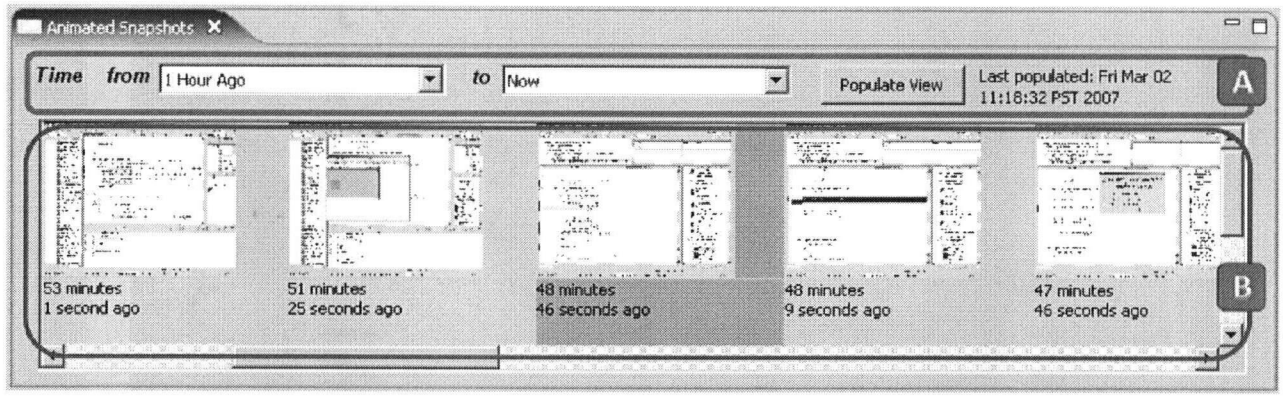

Figure 3.2: Animated Snapshots tool. Part A provides controls to specify the time and duration of interactions considered. Part B shows thumbnails of the snapshots representing a programmer's working environment.

The Animated Snapshots tool presents thumbnails of snapshots taken as a programmer worked (Figure 3.2). When a user right-clicks a thumbnail, a sequence of snapshots representing the programmer's previous work is animated (Appendix A provides a detailed example). The snapshots are shown at 90% of the full screen size and are surrounded with a border to make users aware that what they see is just an image, and not their actual workspace. The user can control the animation with keystrokes that are bound to typical controls, such as pause, moving forward or backward frame by frame, or moving to the beginning or end of the animation. The user can also use keys to indicate if a snapshot indicates the start of the task under consideration or the end of a task. The snapshots captured are of the screen anytime the IDE is active; thus, other applications, such as chats and emails are also sometimes captured. We believe the inclusion of these other windows also helps leverage a user's episodic memory since a user might remember a task starting around when a certain email arrived.

3.2 Interaction History Tool

Through semantic memory, people can infer and generalize relationships between facts [22] and use these inferences to find important landmarks during time. We built the Interaction History tool (Figure 3.3) that stresses the use of a programmer's semantic memory since it is based on a programmer's understanding of the resources that are the target of the tasks being performed.

The Interaction History tool shows the resources touched through selections or edits by the programmer in the tree view on the left side of the tool (Figure 3.3B). When a resource is selected (e.g., `setAttribute` method), the Interaction History tool shows the actions applied to that resource by the programmer in the tree view on the right hand side of the tool (Figure 3.3C), and highlights (in pink) every interaction event involving that resource in the timeline (Figure 3.3D). The actions tree is disabled until the user selects a resource. This design choice is intended to make the programmer think about resources first, rather than the actions performed on resources. If the user also selects an action (e.g., Edit in Java Editor), the timeline highlights (in blue) every interaction event in which the selected action is applied to the selected resource. Each interaction event displayed in the timeline also shows the time at which the interaction event occurred. This tool also has a gutter control at the bottom of the timeline (Figure 3.3E) to facilitate the selection of interactions concerning the same resource.

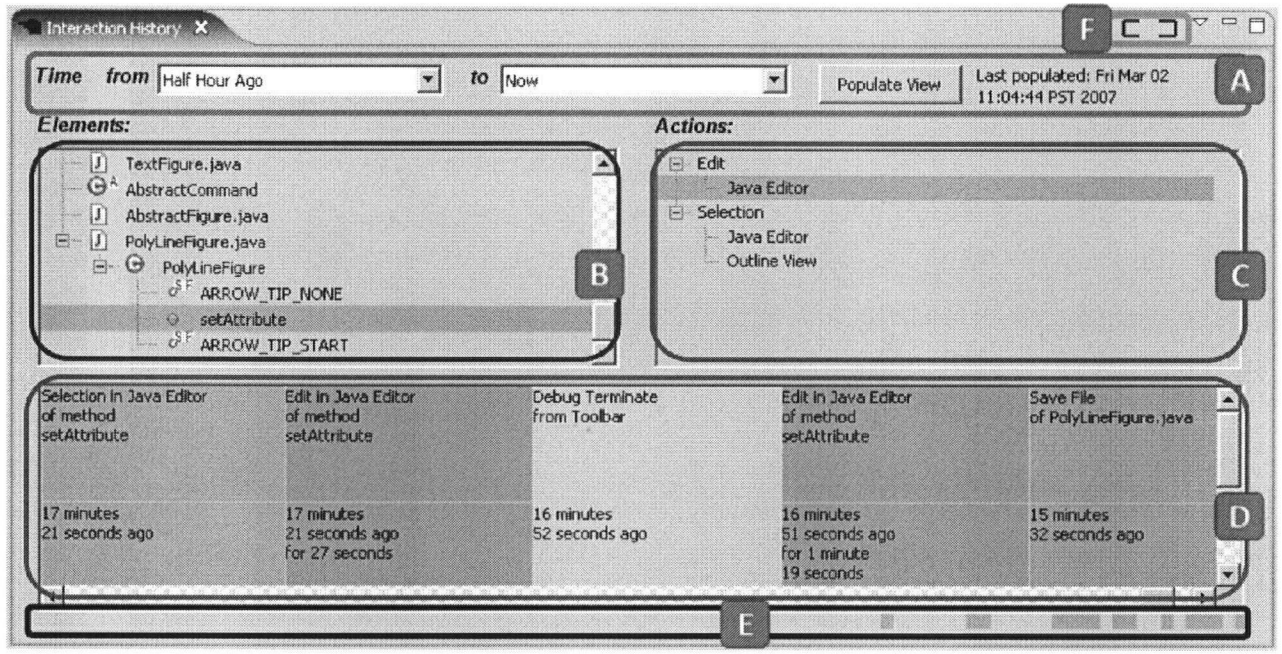

Figure 3.3: Interaction History tool. Part A provides controls to specify the time and duration of interactions considered. Part B shows resources touched given the time frame. Part C displays the actions applied to the resource selected in Part B. Part D shows the interactions and highlights the interactions corresponding to the selections in Part B and C. Using the gutter in Part E, a programmer can jump to different occurrences of the interactions in the timeline concerning the resource selected in Part B. Buttons in Part F set the selected interaction as the start or end point of a task.

Since the timeline can only display five interaction events at a time, the gutter can help a user jump to different locations in time. To set the start or end of a task, the user finds the appropriate interaction event in the timeline, selects the event (causing the event to be highlighted in a green colour) and then clicks on the opening or closing bracket (depending on whether the start or end of a task is being set) at the top right of the window (Figure 3.3F).

3.3 Common Elements

The two tools share a number of common user interface controls and interact with a view to display information about selected boundaries.

3.3.1 Time Range Control

To enable a user to scope the amount of interaction considered when selecting the boundaries of a task, each tool supports a time range control (Figure 3.2A and Figure 3.3A). This control consists of two dropdown lists that show predefined temporal milestones (e.g., *Now*, *10 Minutes Ago*, *1 Hour Ago*, *Today*, *1 Week Ago* etc.) and a configurable calendar to specify a time. For the study reported on in this thesis, the default setting of this control was from *Half Hour Ago* until *Now*. The *Now* time unit always denotes the actual time of the inquiry. When a user clicks on the populate button, the tool shows the appropriate selection of the user's interaction history. The control also displays the time when the populate button is clicked.

3.3.2 Timeline Control

The Timeline Control is shared by the tools to display the interaction history occurred within the time range selected by the user through the Time Range Control. Depending on the tool, the timeline displays either thumbnail pictures (i.e., Figure 3.2B) or a series of textual interactions (i.e., Figure 3.3D). In both cases, the control shows when each event happened. The time associated with the interactions is relative to the time when the data is populated. The timeline control enables users to select an interaction event (causing the event to be highlighted in a green colour), so that they can be set as the start or the end point of a task.

3.3.3 Task Boundaries View

When a user identifies a start or end point of a task using one of the tools, the Task Boundaries view (Figure 3.4) displays information about the selected point.

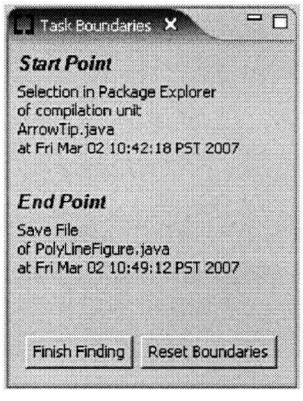

Figure 3.4: Task Boundaries view. This view shows the information about the interaction events set as task boundaries using either one of the tools. The *Finish Finding* button denotes the end of the boundary exploration, whereas the *Reset Boundaries* button clears the selection.

For the purposes of the study, a button labelled *Finish Finding*, was provided on this view to indicate when the subject had completed finding the boundaries of a task. Clicking this button simply records the selected boundaries to a log. In the actual tool, the Task Boundaries view provides support to extract the information associated with a task. We describe this feature in Section 5.4.5.

3.4 A Use Case

To provide a sense of how these pieces come together to help a user mark start and end points of a previous task, we describe an example of the use of the Interaction History tool. (An example of the use of the Animated Snapshots tool is provided in Appendix A.)

Consider the case of a programmer who is working on fixing a blocking bug. Halfway through the implementation, the programmer remembers that he or she has to explain the patch to his or her manager in a scheduled meeting. By tracking the change with Eclipse Mylar, the programmer can use

the collected task context to focus the IDE on the patch to show just the related resources while describing the fix. Unfortunately, the programmer had not created and activated a Mylar task at the start of the implementation. To capture the task context from this point on, the programmer creates a suitable task in Mylar. He then uses the Interaction History tool to reset the start of the task to an earlier point. He opens the Interaction History tool and populates the tool with the interactions that occurred in the last half hour. He finds and selects the first resource he edited from the elements tree, selects the first edit of that resource from the timeline and sets that interaction as the start point of the bug fix task. Resetting boundaries of a task can be done with a button on the Task Boundaries view, suitable for use for this action. For extracting the programming information into a fresh task, see Section 5.4.5.

3.5 Implementation

Our tools are implemented as a plug-in for the Eclipse IDE. The tools use the facilities of the Eclipse Mylar Monitor [15] to track user interaction with Eclipse. Eclipse Mylar uses the Monitor component as the basis from which to construct task contexts. The Monitor collects *interaction events* and the context model in Mylar relies on a collection of interaction events. A *direct* interaction event is a single unit of work done with the IDE (i.e., selection, edit, and executed command) whereas an *indirect* event (i.e., propagation and prediction) denotes possible relations between the interactions and resources. Each interaction event records the time at which it occurred, the kind of event i.e., selection, edit etc., an id for the source control that originates the interaction (e.g., Outline View) and an id for the target element. The Monitor framework in Mylar keeps track of these interaction events that have occurred and stores them in an XML formatted log file.

In addition to the history recorded by Mylar Monitor, our plug-in associates each interaction event with a full screen image capture of the users' screen.[6] We capture the screen in a separate thread using the Graphics Context (GC) class in the graphics package of SWT,[7] and compress the raw image data on the fly using another thread. This approach allows us to save high quality images in a small amount of disk space.

[6] We associate nearly all of the interaction events with snapshots except the events that would yield the same snapshot as the previous event.

[7] `http://www.eclipse.org/swt`, verified 11/04/07; SWT: The Standard Widget Toolkit is a replacement for AWT and Swing to create native operating system looking graphical user interface elements such as menus, buttons and labels.

The programmers in our study generated an average of 955 snapshots in 51 minutes of programming time. These snapshots had an average size of 62KB (minimum 20KB, maximum 80KB).[8] Our implementation can capture these images efficiently; none of the subjects in the study we describe in this thesis complained about machine slowness or interruptions.

Our tools use only two data sources to display the interaction information: the Interaction History tool uses only the log file generated by Mylar Monitor; the Animated Snapshots tool uses the same log file and the snapshots generated by the listener module in our plug-in.

We placed our tools under a new Eclipse perspective, the Task Boundaries perspective. Figure 3.5 presents the state of Eclipse IDE when Task Boundaries perspective is active (Figure 3.5A), showing our Interaction History tool (Figure 3.5B), Animated Snapshots tool (Figure 3.5C), and Task Boundaries view (Figure 3.5D) as well as Package Explorer (Figure 3.5E) and open Java editors (Figure 3.5F). We also designed our system so that no interaction event would be captured (logging or snapshots) when the Task Boundaries perspective is active. This choice eliminates the recursive cases that can happen when a user is populating the tools with recent interactions and the tools end up populate themselves with interactions involving themselves.

[8] P4 3Ghz 1GB RAM, display having 1280 by 1024 pixels of resolution and 16 bit colour quality.

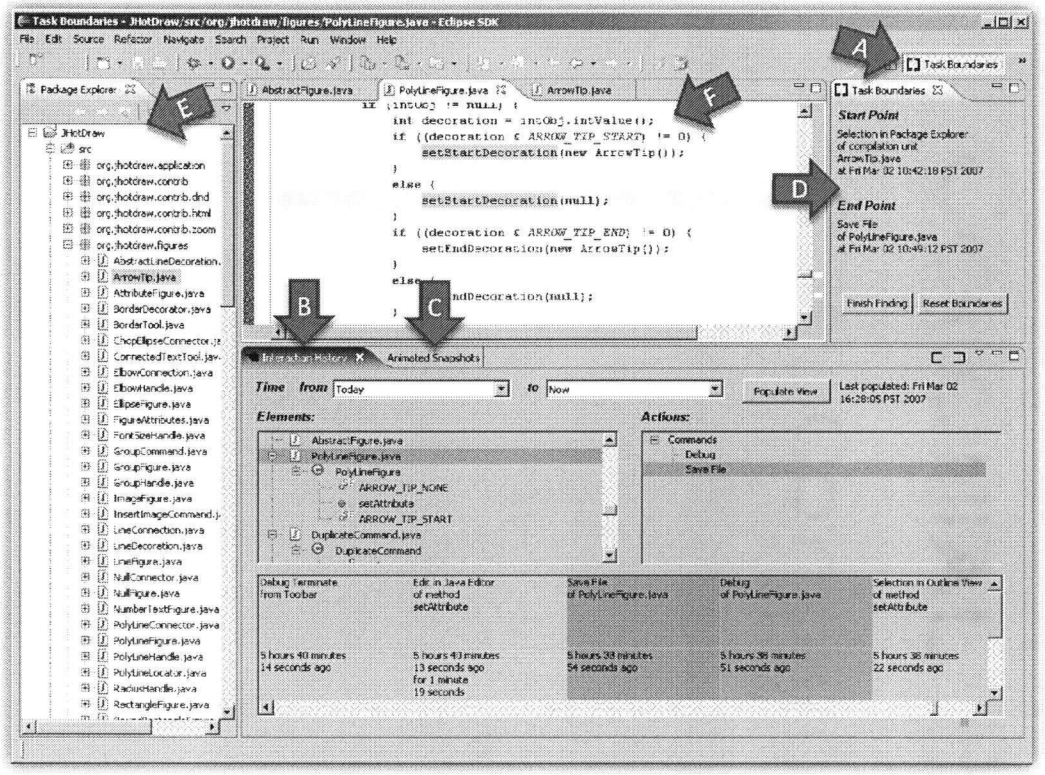

Figure 3.5: Task Boundaries perspective containing our tools. The perspective (Part A) shows the Interaction History tool (Part B), the Animated Snapshots tool (Part C, hidden), and the Task Boundaries view (Part D). This perspective also contains the Package Explorer (Part E) and the open Java Editors (Part F).

Chapter 4

Study

Our goal in conducting the study described in this thesis is to perform an exploratory investigation of how one kind of multi-tasking user—programmers—identify the boundaries of recently performed tasks. We consider a variety of questions in our comparison of the two interface approaches. How do the programmers perform with each tool: are they more accurate or faster with one than the other? Which style of interface do the programmers prefer? Do the programmers interpret how their actions corresponded to a task consistently? Are there any patterns in how they use the tools? In this chapter, we describe the study and present an analysis of the results.

4.1 Subjects

We gathered twelve subjects from the University of British Columbia (UBC) by sending an invitation to participate to suitable mailing lists and newsgroups of the Computer Science Department and through an announcement made in an undergraduate software engineering class. To be a participant, an individual was required to have at least a third or fourth year undergraduate standing, be more than 19 years of age and have experience in the Java programming language and Eclipse IDE over two courses, or over four months of industrial experience.[9] We offered an electronic gift certificate of $20 as compensation for participation.

The twelve subjects had, on average, 5 (std. dev. 2.9) years of experience with Java, and 2.8 (std. dev. 1.3) years with Eclipse. The average age was 27 (std. dev. 6). Ten of the twelve subjects had at least Master's level education and five of the subjects were female.

[9] We decided not to add a criterion on Eclipse Mylar plug-in knowledge because it would not matter from our research perspective if the participants used Mylar or not.

Table 4.1: Assignment of tools to tasks and subjects. AST and IHT refer to Animated Snapshots tool and Interaction History tool respectively.

		Subject Number					
		1 & 2	3 & 4	5 & 6	7 & 8	9 & 10	11 & 12
Task	1-Nodes	AST	AST	IHT	IHT	AST	IHT
	2-Arrows	AST	IHT	IHT	AST	IHT	AST
	3-Duplicate	IHT	IHT	AST	IHT	AST	AST
	4-Size	IHT	AST	AST	AST	IHT	IHT

4.2 Method

To evaluate our tools, we needed each subject to have a substantial amount of programming interaction with the Eclipse IDE to produce a work history in which multiple task boundaries could be investigated. To produce such a work history, we required subjects to perform four programming tasks for approximately one hour. As we were more interested in the effects of our tools rather than the effects or the success in programming tasks, we kept the task order consistent, and randomized the tools over the tasks. Table 4.1 shows which subjects used which tools in which order. This distribution considers all possible permutations of two tools to four programming tasks.

A session in our study spanned about two hours and involved a subject:

1. completing a tutorial about the software system used as the target of tasks to be performed (10 minutes),
2. performing four assigned programming tasks on the target software system (60 minutes),
3. applying our tools in an assigned order to find the boundaries of the just completed programming tasks (30 minutes), and,
4. completing an interview and a questionnaire (20 minutes).

One investigator (the author of the thesis) observed each subject performing each step and made notes on their behaviour performing the tasks and using the tools. He also made notes during the interview. In addition, we instrumented the Eclipse IDE used by each subject to gather information about the interaction of the subject as they used the tools.

Based on the investigator's discretion, in a few cases noted in the results (Section 4.3.5), help was provided to the subjects when obvious errors in

use of interfaces were encountered. Subjects were not helped during the programming part of the study.

4.2.1 Target Software System

The target software system for the tasks used in our study was the open source JHotDraw framework (v6.0b1).[10] JHotDraw supports the development of graphical editors. For the purposes of our study, we simplified the code base by removing some functionality (i.e., the debugging menu and applet support) and code (i.e., test packages and three sample applications). We provided each subject a short tutorial on JHotDraw, covering basic components of the architecture, packages, and sample applications (see Appendix C for the tutorial). The intent of this tutorial was to provide each subject some basic knowledge of the source code on which we were asking them to complete programming tasks.

4.2.2 Programming Tasks

We asked each subject to perform four short programming tasks on the JHotDraw framework using Eclipse (see Appendix D for task descriptions). Each task required browsing the code to find the correct place to modify and adding or changing code within one Java class; usually copying and pasting from nearby code and editing a part of the newly pasted code. We designed the programming tasks to be largely independent of each other to provide as fair a comparison as possible for the tools used to assign task boundaries.

Each subject was asked to perform the tasks consecutively, in the same specified order; task switching or going back to a previous task was not allowed. We advised each subject to allocate a maximum of fifteen minutes for each task. If a subject did not complete the task in the assigned time, we asked them to switch to the next task. On the other hand, the time limit was purposefully not kept strict considering a subject's success on the programming task. We wanted subjects to finish programming under and close to one hour; if a subject was fast in successfully implementing the first tasks, we let that subject run overtime for couple of minutes in the later tasks, especially when that subject was close to a correct implementation.

[10] `http://www.jhotdraw.org/`, verified 03/31/07.

Table 4.2: Programming tasks. The last four columns categorize the solution of the subjects for each task.

Task	Description	Average Time	Success	Partial	Buggy	Lost
1-Nodes	Modifying the list of possible connection places to make two nodes connected from their corners	8.4 (std. dev. 3.8)	10	0	1	1
2-Arrows	Fixing an introduced bug to properly display the arrow tips	14 (std. dev. 2.6)	5	0	2	5
3-Duplicate	Adding a key handler to duplicate a selected figure when the space key is pressed	12 (std. dev. 4.0)	8	1	0	3
4-Size	Displaying the size of the active drawing window	15 (std. dev. 0.3)	0	3	7	2

Each subject was told to make his or her best effort to perform the tasks; successful completion of a task was not a measure for our research questions. However, to provide an overview of the subjects' work on the tasks, we did analyze the produced solutions and categorized each as *success*, *partial*, *buggy* or *lost*. If the solution met the requirements exactly, we categorized that solution as *success*. We categorized an implementation as *partial* if it satisfied only a part of the specified requirements. For example, for the last programming task, if the user managed to display the size of the active drawing window each time a window was created, but was not able to update the information when a window is resized or changed focus, we counted this implementation as *partial*. We considered an implementation as *buggy* if the subject just edited unrelated portions of the code, or found the correct place but did not complete a working solution. Lastly, we named a solution as *lost* if the subject failed to find the correct place to edit, did not do any changes to the code at all, or spent all of the time browsing the code base. Table 4.2 summarizes this analysis, showing the average number of minutes subjects spent working on each task and the number of subjects who implemented a successful or failing solution. Programmers worked 51 minutes and generated 988 interaction events on average.[11]

4.2.3 Boundary Determination

Next, we asked each subject to determine boundaries for the completed programming tasks using the assigned tools. We motivated the problem of task boundary detection by describing the support possible for a programmer if his or her work is associated with different tasks. For instance, a programmer can see just the parts of the code base that he or she worked on for a particular task, making it easier to explain a solution to a colleague. As another example, it becomes possible to track the time needed to complete a task, which may help improve a programmer's work.[12]

We introduced the tool assigned to a subject for the first boundary determination task according to the subject's number (Table 4.1). We showed a short demo describing the use of the first assigned tool. At the end of the demo, we asked the subject to find the boundaries of the first task he or she had recently performed. Similarly, we presented the second tool just before it was to be used according to the assignment of tools to tasks.

[11] About 80% of the 988 interactions were selections, edits and executed commands, the rest were other kinds of events used internally by Mylar.

[12] Most of our subjects heard of Mylar before and knew that Mylar provides these kinds of support. One subject [S10] was actively using Mylar in his work.

4.2.4 Interview and Questionnaire

We interviewed each subject after he or she completed finding the boundaries of the tasks. Table 4.3 lists the questions used in the interviews. Each interview took between fifteen minutes to half hour.

After the interview, we asked each subject to complete a questionnaire to measure the workload caused by our tools (see Appendix B for the questionnaire). The questionnaire was a variant of NASA-TLX (Task Load Index) [9]. We asked each subject to complete one form per tool, and to assign scores between one and twenty[13] to the workload categories *Mental Demand*, *Physical Demand*, *Temporal Demand*, *Performance*, *Effort* and *Frustration Level* caused by that tool. Then we asked each subject to pairwise compare (six categories, fifteen pairwise comparisons) the importance of each category for each tool compared to attempting to find task boundaries independent of a tool. For example, a subject who believed accurately finding a boundary was more important than the physical effort required to investigate previous activity (i.e., by clicking and scrolling in an interface) would rate *Performance* more important than *Physical Demand*. We used the comparison of the categories to weight the scores the subjects assigned. We counted the number of times a category is selected as more important compared to another category, and multiplied that number with the percent score assigned for that category, for a particular tool. We then added all of the weighted scores for each category to find the overall workload for a tool.

Table 4.3: Interview questions.

Q1	Which tool would you use to figure out boundaries of very recent tasks? Which one would you use for the tasks you have worked on several days, weeks, months ago?
Q2	Do you think the tools were accurate enough to show you the exact point you were looking for?
Q3	What is a task boundary to you? What do you think of as the start? What do you think of as the end?
Q4	What is the reason you did not use the feature X in the tool Y?
Q5	Do you have any comments or suggestion to improve the tools?

[13] These scores are normalized afterwards to percents.

The original TLX questionnaire asks subjects to assign scores and to compare the workload categories according to a base task and not to their general preferences. Since there are no existing alternatives to the interfaces we investigated for determining task boundaries, we had no base tool to which our new interfaces could be compared.

Lastly, we asked subjects to provide information regarding their previous experience and to state which tool they preferred.

4.3 Results

All of the subjects were successful in using our tools to find the start and end boundaries of the programming tasks. In this section, we present the analysis of our findings in terms of accuracy of the selected boundaries, time overhead of boundary identification, which tool the subjects prefer to use, what do start and end task boundaries mean for the subjects, how do they detect boundaries, various tool usage patterns, and suggested improvements.

4.3.1 Accuracy of Selected Boundaries

An interface for task boundary identification should make it easy for a user to determine accurately where in his past activity the start or end of a task occurred. To learn how well each tool supported a subject in this determination, we defined three sets of boundaries per task per subject. First, we noted and logged the *actual* points in a subject's activity where he or she started or ended a new task. The actual start point was considered the first interaction concerning that task and the actual end point was considered the last interaction concerning the same task. Second, according to how a subject defined a task when he or she was interviewed, we set the *declared* points. For example, for a subject who stated that the start of a task is when the first correct change occurs, we used the first recorded *edit* event initiating the correct implementation of a particular task as the declared start point. In cases where a subject did not define the meaning of a start or end point of a task or when a subject was indifferent or inconsistent about his or her decision, we considered the actual points as the declared points. Lastly, we defined the *selected* points according to the boundaries defined using the Task Boundaries view (Figure 3.3.3).

We then compared the selected points to declared and actual points to analyze the accuracy of the boundary determinations. Our comparison uses time difference as a measure for closeness for a boundary point. We considered a start or end selection as accurate when it matched the start

or end declared point respectively (time difference equals zero). When we detected inconsistency about what a subject stated and what a subject did, we acted carefully on the comparison, trying to favour the subject's selection. For example, for a subject who defined the declared points different from the actual points (a subject who considers the start point of a task somewhere after the very first interaction and the end point somewhere before the very last interaction concerning that task) we required his or her selected point to be the same as the declared point to count that determination as accurate. When we noticed that a subject's selected points matched the actual points even though he or she had different declared points, we also considered that determination as accurate. When a subject's selection did not match neither the declared nor the actual points, we recorded the time difference between the selected point and the declared point as the error margin. In cases where a subject selected a boundary point more than once, we only considered the most correct selection to be as fair as possible to the subjects.[14]

Figure 4.1 shows the accuracy (error margins) based on time of each of the 96 boundary explorations performed as part of the study (12 subjects x 4 tasks x 2 boundaries per task) plotted against the time spent in determining the boundary. Data points plotted as negative values along the vertical time accuracy axis indicate a subject selecting a boundary that occurred before the actual or declared boundary point; positive values are times that occurred after the actual or declared boundary point. Values along the zero indicate accurately determined boundary points. The results suggest that subjects were able to detect boundaries more accurately with the Animated Snapshots tool compared to the Interaction History tool.

To determine whether this result is statistically significant, for each subject, we averaged the time differences obtained using each tool and applied a Wilcoxon T-Test.[15] We found that the subjects were significantly more accurate with the Animated Snapshots tool (two-tailed $p = 0.0098$). We also calculated a three-way analysis of variance (ANOVA) with repeated measures (two tools, within-subjects; six tool orderings, between-subjects;

[14] We always chose the most accurate selection instead of the last one because we were interested in the best performance of the subjects with the tools. It turns out that the most accurate selections were always the last selections. See Section 4.3.5 for a detailed explanation.

[15] The reason we did not employ a regular paired T-Test was because of the nature of the study; no subject used both tools to find the boundaries of the same programming task and our data points did not obey the Gaussian distribution. The T-Test is calculated using GraphPad Prism v5.0 for Windows. `http://www.graphpad.com/prism/Prism.htm`, verified 11/04/07.

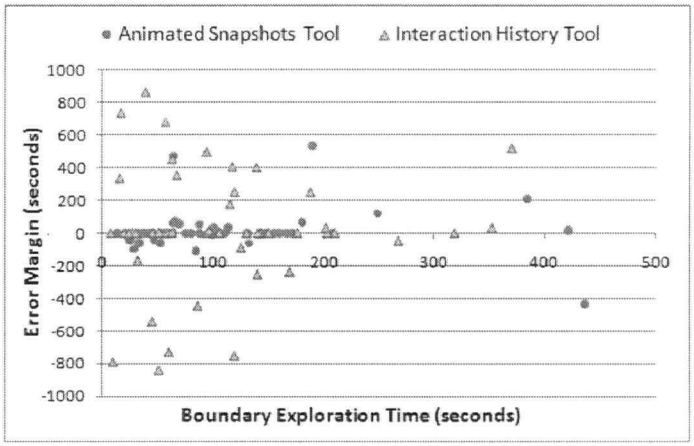

Figure 4.1: Boundary exploration time versus accuracy in time. The x axis shows the time spent finding a boundary, and the y axis shows how accurate the finding is in terms of time (seconds). A point on the x axis represents that the finding is accurate. A point above the x axis shows that the marked boundary is not accurate and happened after the right boundary; conversely, a point below the x axis shows that the marked boundary is not accurate and happened before the right boundary.

two trials of each tool, within-subjects).[16] The results show that subjects found boundaries significantly more accurately with the Animated Snapshots tool than the Interaction History tool ($F(1,6) = 26.513, p = 0.002$). The ANOVA showed a significant effect of tool ordering, meaning that the order that the tools were presented to the subjects made a difference to their performance ($F(5,6) = 5.295, p = 0.033$). When we reran the repeated measures on trial 1 (accuracy results of the first use of the Animated Snapshots tool versus the first use of the Interaction History tool) and trial 2 (results between the second uses) separately, we found that the order of the tools made a difference only in the first trial. Given the small number of subjects involved in the study, further experimentation is necessary to understand the interaction of these effects to the overall result.

To understand overall how the uses of the two tools compared, we also processed the data to allow for some tolerance in the accuracy. Specifi-

[16]The ANOVA is calculated using SPSS v13.0 for Windows. `http://www.spss.com/spss`, verified 11/04/07.

cally, we marked a subject's boundary selection as accurate if there was a maximum of 30 seconds or five interactions events (whichever was less) of difference to the compared boundary. Marking a subject's selection as accurate means considering the error margin as zero. Figure 4.2 shows the level of accuracy after we employed the tolerance on the data. Of the 24 uses of the tool to find 48 boundaries, the subjects were able to detect a boundary using the Interaction History tool accurately 50% of the time. Subjects were more successful with the Animated Snapshots tool, with which 87.5% of boundaries were determined successfully.

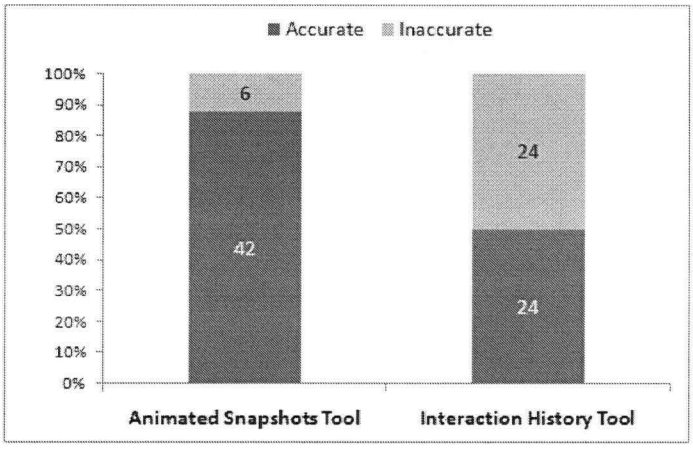

Figure 4.2: Boundary exploration success. The results displayed are after employing a level of tolerance to the accuracy data.

For an interface for task boundary identification to be acceptable to users, a user must also consider the identification to be accurate. Because a user can easily create a large number of interaction events, we do filter very common events to decrease the volume to a consumable level. In the Interaction History tool we disregarded common interactions, such as copy and paste, and keyboard events like go to end of the line. In the Animated Snapshots tool we skipped interactions that would yield the same snapshot as the previous one. We asked the subjects about their impressions of the accuracy of the two tools in terms of presenting complete and correct information. Seven subjects [S5, S6, S7, S9, S10, S11, S12] (58%) thought both tools were accurate and showed the points for which they were searching. Subject S12 thought the Interaction History tool was too precise. The remaining five subjects were mixed in their views: three [S3, S4, S8] (25%)

believed that only Animated Snapshots tool was accurate while the other two [S1, S2] (16%) stated the opposite.

There was some disagreement amongst the subjects about what accuracy meant. For instance, since the Animated Snapshots tool tool captures snapshots only when the IDE is active, a user's recollection of their experience sometimes differed with the recorded snapshots. This situation occurred when a user would switch focus from the IDE, such as when running a sample application to test the code being written. This focus switch would stop the recording of snapshots. The playback would then show only the portion of the executed sample application when the IDE was active. Three subjects [S2, S6, S11] noted this discrepancy, although only one of the three [S6] thought that the Animated Snapshots tool was inaccurate as a result.

Another possible factor in a user's perception of boundary identification accuracy is his or her success in the task being performed. Our own analysis of the interaction history logs recorded during sessions suggests that success in the programming task does not play a crucial role in successful boundary detection. Rather, the first and the last actions of a subject during a task have more impact. We asked the four subjects [S2, S4, S6, S12] (33%) who were successful in at most one programming task whether they believed that their performance on the tasks not completed successfully had a negative effect in the boundary exploration step. Two subjects, S4 and S12, did not believe that there was a negative effect on the boundary exploration step. On the other hand, contrary to our findings, S2 and S6 each believed that there was an effect. Each commented that on a successful task completion he or she would close the editors, but on unsuccessful tasks, he or she could not use this action as a cue for determining a task's boundary.

4.3.2 Time Overhead of Boundary Identification

Since boundary detection is a means to an end, such as providing better support for task switching, a tool should support finding the correct boundaries quickly. To compare the tools on the time the subjects required to determine boundaries, we calculated an overall boundary detection time for each task. For task start points, the boundary detection time was calculated as the time that passed from the subject first populating a tool with a portion of interaction history information until the start boundary was identified by the subject. For task end points, the boundary detection time was calculated as the time when the task start point was identified until the task end point was identified. The overall boundary detection time was the sum of these two values. In cases where a subject identified start or end

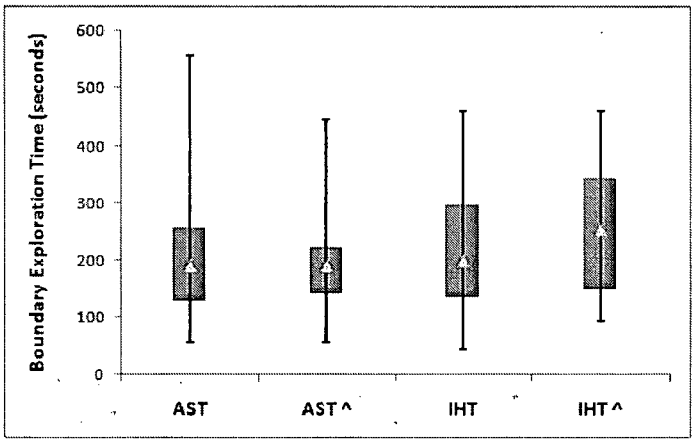

Figure 4.3: Time to find both boundaries of a task. AST and IHT refer to Animated Snapshots tool and Interaction History tool respectively. The bars show the range of values, the triangles indicate the median values and the boxes span between the first and the third quartiles. AST^ and IHT^ represent the cases where a boundary is detected accurately.

points multiple times, we calculated only the times spent to find the most accurate boundaries.[17] For example, if a subject determined both boundaries, realized that the start boundary is wrong, and then found a more accurate start boundary, we added the time spent from the first start point was identified until the task end point was identified and from that very point until the second more accurate start point was identified.

The bars labelled AST and IHT in Figure 4.3 show the overall boundary detection times for each tool based on 24 data points per tool (12 subjects x 2 tasks used with a particular tool x 1 set of boundaries per task). These times appear roughly the same on average. The bars labelled AST^ and IHT^ in Figure 4.3 show the detection times for just those boundaries (both start and end points) that were determined accurately: 19 instances out of 24 for Animated Snapshots tool and 9 instances out of 24 for Interaction History tool. These values suggest that it takes more time to find correct boundaries using the Interaction History tool than the Animated Snapshots tool. Given the small sample of accurately determined boundaries, we did

[17] We always chose the most accurate selection instead of the last one to be as fair as possible to the subjects and because we were interested in the best performance of the subjects with the tools in terms of accuracy.

not attempt to test this data for statistical significance.

4.3.3 Subject Preference of Tools

We gathered data about each subject's preference for a tool in two ways: we asked each subject during the interview and we had each complete a questionnaire about the workload associated with using each tool.

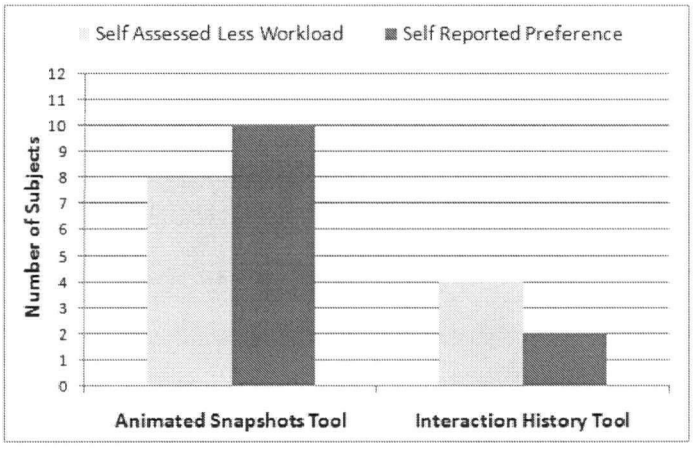

Figure 4.4: Preference of the tools. The self assessed less workload is calculated using the TLX questionnaire variant (Appendix B). The self reported preference is stated by the subjects in the interview and through the preference ranking in the questionnaire.

During the interview, most subjects (ten out of twelve or 83%) stated a preference for the Animated Snapshots tool (Figure 4.4 - self reported preference). These subjects stated a need for visual cues; their actions were familiar even when they could not remember the exact resources with which they interacted. Two subjects [S3, S5] stated that using Animated Snapshots tool is trivial and intuitive. Five subjects [S1, S5, S6, S7, S9] stated that even though they believe the Interaction History tool is accurate and shows the appropriate information, it is too much work to find the point of interest.

This preference for the Animated Snapshots tool was also evident in the answers to the questionnaire (Figure 4.4 - self assessed less workload). Figure 4.5 shows in detail the workload scores assigned to the tools by the subjects. These scores indicate that eight subjects perceived Animated Snap-

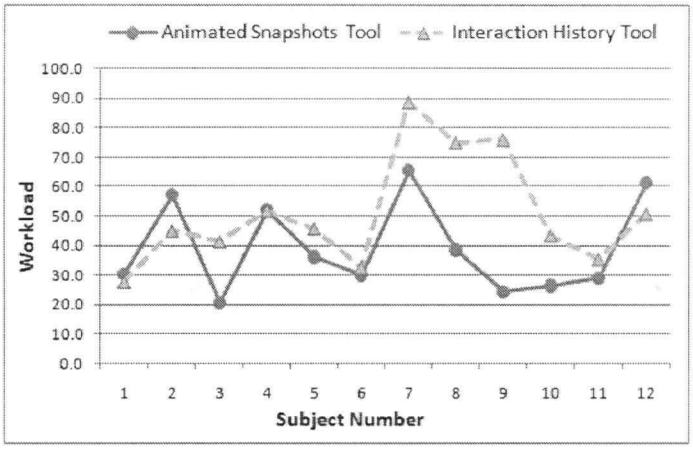

Figure 4.5: Workload caused by the tools. The workload is calculated using the TLX questionnaire variant (Appendix B).

shots tool as having less workload; the remaining four denoted Interaction History tool as having less workload. When giving tools scores, we asked each subject to explain why he or she gave a lower or higher score concerning a workload category. The subjects stated that the Animated Snapshots tool was less mentally and physically demanding because they found it easier to remember resources through visual cues and the tool required less manual work. On the other hand, subjects found the Interaction History tool as less temporally demanding because the tool just displayed still information. Most of the subjects felt more successful using the Animated Snapshots tool because they were more confident when they selected a point. Subjects denoted that they spent roughly an equal amount of effort while using each tool. According to the comparison of workload categories, subjects found the *Frustration Level* most important; the Animated Snapshots tool had the best (lowest) scores in terms of the frustration caused compared to the other categories.

Two of the ten subjects [S4, S12] who stated a preference for the Animated Snapshots tool during the interview had answers in the questionnaire suggesting Animated Snapshots tool involves a higher workload. For one subject (S4), the workload assessment differed between the tools within a percentage point; we attribute the likely cause of this difference between the interview and the questionnaire as imprecision in our questionnaire scores. We do not know the reason why the preference of the other subject (S12)

differed.

We also asked the subjects whether the recency of the tasks in the experimental situation might affect their preference. Six subjects [S3, S5, S6, S7, S8, S9] (50%) responded they would use Animated Snapshots tool for very recent (10 to thirty minute ago) tasks, while three subjects [S2, S10, S12] (25%) preferred the Interaction History tool. Subject S11 was indifferent; he said "either just works fine". For older (several days, weeks, months ago) tasks, eight subjects [S2, S3, S5, S7, S9, S10, S11, S12] (66%) preferred the Animated Snapshots tool. On the other hand, two subjects [S6, S8] (16%) stated they would use the other tool.

Subject S1 was neutral about the recency, she stated:

> Recency does not matter, it depends on what I am looking for or what I remember. If I do not remember the file, I would play the replay to find the task (use Animated Snapshots tool). If I remembered, I would use Interaction History tool.

On the other hand, subject S4 was not motivated or impressed by the whole process at all; she stated:

> I would not use either, I did not see that the pain of finding boundaries would be worth it. If you force me to use the tools, I would choose the Interaction History tool for the old tasks because I might remember the class name. I would use the same tool for the recent tasks too although I do not have a strong preference. I might have preferred Animated Snapshots tool if the thumbnails were big enough.

4.3.4 Meaning of a Task Boundary

In this study, the tasks were assigned. However, the definition of which work was part of the task was left to individual subjects. This choice allowed us to investigate whether or not subjects would define task boundaries equivalently in the interview segment of the study.

Before the interview, seven subjects [S1, S5, S8, S9, S10, S11, S12] (58%) asked us what we meant by the start and end points of a task when each first began the boundary detection part of the study. We simply answered that whatever point they considered the boundary to be was appropriate so as not to constrain subjects to a particular set of points.

Not surprisingly, there is great variation in where tasks start and stop to different subjects. Two subjects [S6, S10] thought a task starts when

each first began browsing the code for the assigned task. Subject S10, having experience with Mylar, stated in a witty way that the boundaries are when he activated and deactivated the Mylar task. Two subjects believed the first change attempt starts a task. These two subjects differed in which change attempt mattered: S1 believed it was a change only to a correct place whereas S3 believed it was any change. Subject S8 stated that he sometimes preferred the place where he started browsing, while other times, he preferred the place where he did the first correct implementation. Three programmers [S5, S11, S12] were more sure about their decision; each stated that the start point is where they saw the first clue about the right implementation attempt (e.g., opening the *Open Type* view[18] which would end up opening the right file containing the solution). Two subjects [S7, S9] approached the start point notion in a more conceptual way; each stated that a task starts when they read the description and have a global understanding of that task, before starting programming.[19]

For the end point of a task, we had three different answers. Two subjects [S1, S6] (16%) stated a task ends at the last point they finished editing the code. Five subjects [S3, S5, S8, S9, S10] (41%) denoted a series of actions for the end point: finishing editing, saving the files, running the application for testing and closing the editors. The remaining five subjects [S2, S4, S7, S11, S12] stated the end as the point just before the next tasks begins. S2 stated that whenever he closes the editors before starting a new task denotes the end point of that task and the next action is the start point for the next task. S4 said that whenever we prompted her to move to the next task marked the end of the current task and the beginning of the next task.

Given that our study involved a particular kind of task, a software evolution task, we did observe some common processes across the subjects. Almost all of the subjects ran a sample application after reading the task description and then ran that application again at the end of the task, if the task was completed successfully. It is possible such process marks might help define task switches for particular kinds of tasks.

[18] This view provides a means to search through the names of declared classes.

[19] Even though these preferences denote the subjects' own belief of a start point, while looking for task boundaries, subjects S1 and S7 selected as the start point where they began to browse, stating that they said and acted differently for the purposes of the study thinking we would define a start boundary as such.

4.3.5 Process of Boundary Detection

To investigate how programmers in our study detected boundaries, we considered the process each used, such as whether they used the boundaries determined for other tasks as cues or whether they changed a boundary once it was set.

Nine of the subjects [S1, S2, S4, S5, S6, S7, S9, S10, S11] (75%) selected and set the very first interaction event as the start point of the first task. Similarly, eight subjects [S1, S5, S6, S7, S8, S9, S10, S11] (66%) immediately set the very last interaction event of the study as the end point of the last task. When finding the boundaries of the tasks in between, five subjects (41%) took into consideration beginning and end points of the other tasks as cues. This approach sometimes caused a number of accuracy problems when the base point was erroneous. For example, three subjects [S1, S2, S6] (25%) were incorrect in their determination of the end point of a task when using the Interaction History tool. When we asked them to use the same tool for the next task, each immediately selected the next interaction point as the start of that task. Unfortunately, this caused inaccuracy for the start point of that task as well.

When finding boundaries of the last task, subject S2 stated that he did not explicitly check the very end of the timeline to mark the end boundary because he thought the end of the timeline would contain interactions about him using the tools finding the boundaries of the first three tasks, rather than information about programming tasks. This shows that he might not have interpreted the information being presented by the tools correctly.

Other than the aforementioned semantic cues, five subjects [S2, S6, S8, S10, S12] (41%) used temporal cues just to have a rough idea about the location of the boundaries. Some subjects (e.g., S10), after determining the starting point of the task, navigated approximately fifteen minutes further in time—the time allocated per task in the study—to look for the end point. While looking for the end boundary for the third task, S8 noticed a two minute time jump and said the jump probably occurred when he read the fourth task description and thus it would be an appropriate end point for the third task.

We observed that four subjects [S3, S6, S8, S10] (33%) each changed the start boundaries already set for one task. Each realized that the boundary might be inaccurate from the Task Boundary view because each noticed that the time difference between the start and end points selected was only couple of minutes. Each of these subjects made more accurate selections on a second try.

Subject S7 was a special case in many aspects. After he found the boundaries of the first task using the Interaction History tool, we warned him that there may be an error in his selection. He realized the error and fixed the start boundary. When he switched to Animated Snapshots tool for the second task, he inadvertently looked for the boundaries of the first task. After we warned him at the end, he changed the boundaries one more time. Lastly, when he was using this tool for the last task, and when he found the correct place for the end point, he pressed the S key instead of E by mistake and overrode his old start selection. This again caused him to look for the start point one more time.

Other than the boundary corrections mentioned above, surprisingly, only one subject [S8] selected the end boundary before selecting the start boundary for one task. All other subjects intentionally looked for the start point first. S8 chose the end point first because he selected a familiar method name in the elements tree and clicked the gutter to navigate to that method's last occurrence.

4.3.6 Patterns of Tool Use

We observed some interesting usage patterns about the tools.

We observed that all of the subjects changed the time range intelligently; when they realized that the timeline showed interactions out of the range of interest, they modified the time range either half hour or one hour back or forward.

Four subjects [S1, S4, S7, S12] got confused by the reality of the Animated Snapshots tool and wanted to click the images to control the Eclipse IDE. The subject S4 thought a message window about switching perspectives in the Animated Snapshots tool was real and tried to click the OK button to make it disappear. The other subjects clicked on the X at the upper-right corner of the displayed snapshot to end the replay, instead of pressing the Q key.

Only two subjects [S3, S7] watched the replay in the Animated Snapshots tool in full screen.[20] While the subject S3 said "this is better", seven subjects [S1, S4, S5, S6, S8, S10, S11] found the default size of the replay useful and legible enough, and did not need to toggle to full screen. Three subjects [S2, S9, S12] did not remember about the full screen feature, but once reminded at the end, each described that they would use it. The subject S7 stated that he initially thought pressing F while the replay would take him back

[20] The subject S4 pressed F by mistake while she was meaning to press E to set an end boundary.

to the thumbnails in Animated Snapshots tool. Six subjects [S3, S4, S5, S6, S11, S12] paused the replay immediately and preferred to navigate using the arrow keys, while S11 preferred to peek at the end by pressing Z to understand if the point he was looking for was in that chunk. The subject S12 said the key controls in the replay were standard and intuitive. Most of the subjects quit the replay immediately once they realized they were not in the right place.

When using the Interaction History tool, five subjects [S4, S7, S8, S9, S11] double clicked an interaction event in the timeline control, expecting the resource mentioned in the interaction would open in the Java editor. Two subjects [S4, S10] minimized the information displayed in the elements tree; they clicked on the + sign near the unrelated resources to hide their branches in the elements tree. S6 did not use any of the trees; she only navigated through the timeline control because she was not able to remember any resources or actions. Two subjects [S1, S5] were not clear on the use of the actions tree and did not see the connection with the elements tree and the timeline. Three subjects [S1, S7, S11] did not find the actions tree useful because each believed that the resources were more important and that the actions applied to them could be found later by scrolling the timeline.

The subjects had various feedback about the gutter control. S1 did not use the gutter because she was not familiar with the concept. Two subjects [S2, S3] said it took some time to understand but once they understood each found the control useful. S4 said she was confused because the highlighting in the gutter was not always properly aligned with the scroll bar in the timeline. Two subjects [S6, S9] did not find the control useful because they were not able to remember any resources or actions, thus jumping to different instances of the resources would not be helpful. Three subjects [S5, S8, S10] used and liked the gutter; S10 stated that it is useful to know where else interactions on the same resource happened. Two subjects [S7, S11] did not click on the gutter, but used it as an indicator, as a "look-at thing" as S7 said.

We did not encounter any usage errors with our tools. All of the subjects chose time ranges that would always yield in some data to be displayed. None of the subjects selected illogical start and end boundaries (i.e., the selected start boundary is after the selected end boundary).

4.3.7 Suggested Improvements

The most common suggestion for the Animated Snapshots tool was making the tool look more like a media player with navigation and playback controls.

Given such a feature, it would be easier to navigate seamlessly between the thumbnails (as S7 and S12 suggested) or fast forward or slow down the replay (as S1, S5, S7 and S11 suggested). Six subjects who stopped the replay instantly also suggested that the replay should begin at a stopped state and only play when requested by the user. Furthermore, some subjects wanted the tool to display bigger thumbnails [S4, S7] and wanted a facility to see the first and the last snapshots of the group represented by the thumbnail [S4, S7, S11] so that it would be easier to pinpoint if the interaction of interest is in that group.

One subject [S1] suggested that the Animated Snapshots tool could better imitate the duration between the interactions. For example, the act of a programmer opening a file and reading the contents without touching any code for one minute and switching to another file creates two interaction events (selecting the first file and then the second file) and thus two snapshots. The replay mechanism displays these images one after the other in a constant speed, without stressing that some period of time was spend during the first interaction with the IDE, where the user reads the code.

The most common suggestion for the Interaction History tool was to make it display more concise information. Subject S8 suggested that the information in the timeline can be clustered as sets of actions, such as showing one entry for a resource and listing the sequence of actions applied on that resource instead of displaying each interaction event. Two subjects [S7, S12] stated that they wanted to remove some information that is not useful to the task by removing certain interactions. Some subjects [S1, S7] suggested filtering the timeline for specific kinds of events, such as edits of a specific method. Furthermore, some subjects wanted the trees and the timeline to show only important interactions using decorations instead of showing them all in a plain fashion. The subject S5 stated it was hard to understand which edit was more important. Subject S1 believed that the *run* and *save* actions bloated the timeline (because she does them very often). On the other hand, she thought that even though *run* and *save* information would not make any sense and value compared to the edit information when looking for the old tasks, these actions might be useful when looking for boundaries of recent tasks.

One common suggestion on Interaction History tool was opening a resource in the Java editor when the corresponding interaction event is double clicked from the timeline. Five subjects [S4, S7, S8, S9, S11] intuitively expected this to happen and then suggested this feature. Two subjects [S1, S8] stated the Interaction History tool can employ different information visualization techniques; S8 suggested a fisheye view for the timeline.

S7 said that he would prefer vertical navigation in the timeline instead of scrolling horizontally. Two subjects [S4, S12] stated a need for place markers for Interaction History tool, as they felt lost in the timeline when they started scrolling and they wanted to go back to the initial position where they started scrolling. S4 also marked down on the paper some possible interaction events while she was navigating in the timeline; it would help to have a bookmarking mechanism for the elements in the timeline.

Subject S10 commented about the nesting of the resources in the elements tree when he saw Java methods without having a parent node for the class encapsulating those methods. This occurs when a user selects a method without touching the enclosing class (e.g., by clicking it from the Search View); in this case, we just show the method since we just include the touched resources in this tree. S10 asserted that because this nesting problem it was hard to find the resources he was looking for.

Five subjects [S2, S3, S5, S6, S8] (41%) stated an interest in an interface that would combine features of the existing tools. Subject S3 and S8 stated that a new tool could have a timeline similar to the Interaction History tool, but when an interaction is selected, the related screenshots could be replayed. S5 said that more textual information could be added to the Animated Snapshots tool thumbnails. S6 proposed that she would use the Interaction History tool to find the proximity of a place of interest, and then would like to fine-tune it with the Animated Snapshots tool. S2 preferred the contrary, he wanted to use a tool like the Animated Snapshots tool to find an approximate range of the interactions and then fine-tune with a tool like the Interaction History tool.

Some subjects expressed that they have to have a good sense of time to better use the tools. The subject S8 said he did not quite remember how much time ago he started the task. The subject S12 stated

> ...You have to have a temporal idea for both tools, like when the task was done during the day.

Chapter 5

Discussion

Overall, our study showed a preference for an episodic-based tool for finding task boundaries. In this section, we summarize the results of the study, discuss threats to the validity of the results, and describe limitations noted to the tools used in the study. We also describe other tools and controls we considered that provide a blend of episodic and semantic approaches.

5.1 Summary of the Results

Our study provides evidence that a visual replay of actions performed is preferred by programmers to find boundaries in previous tasks over a tool that relies on the semantics of the information on which the actions were performed. We believe this preference is due to personal experiences playing an important role in recalling previous task information, as Ringel and colleagues have also noted [16]. S7 showed his interest in this kind of approach by stating the following about the Animated Snapshots tool, "It is going to be fun, just like being me!". The results also indicate that a tool stressing the replay of actions, and possibly thus episodic memory, may provide a faster and more accurate way to find the task boundaries. Subject S9 stated that she was more confident about the points she selected with Animated Snapshots tool compared to Interaction History tool. She thought she was more successful determining task boundaries with Animated Snapshots tool, and it was fun to use the tool.

We also found substantial variation in individual views on when and where a task starts and ends. This finding suggests that designers for task-oriented user interfaces need to provide substantial flexibility in not only the definition of the content of a task, but also the duration and location of a task within the actions of a user.

We found that the subjects were less accurate using the Interaction History tool. When we tolerated some inaccuracy in the results (as described in Section 4.3.1), the subjects were able to detect a boundary accurately only 50% of the time with the Interaction History tool. Interestingly, neither of the subjects who confidently praised the Interaction History tool (S1 and

S2) were able to find the task boundaries correctly in either of their tries. We believe these inaccuracies are due to the difficulty of distinguishing the right occurrence of an interaction amongst many similar interactions. For example, the timeline in the Interaction History tool may display a series of selections and edits of the method `createConnectors()`, the programmer may find it difficult to distinguish which of these interactions indicates a particular edit he or she desires to use to indicate a task start. We saw that programmers were more successful with the Animated Snapshots tool with which 87.5% of boundaries were determined successfully.

5.2 Threats to Validity

The primary threat to the validity of our results and any generalization from the results is the number of subjects. Furthermore, even though we acquired subjects with various experience and view points, twelve subjects is not sufficient to represent the programming community as a whole, nor are programmers necessarily representative of all users of task-oriented environments. As Kersten also found value in applying Mylar to non-programming artifacts [13], we believe our findings may have some value beyond programmers, but further study will be necessary.

We decided to introduce the tools by showing a demo video instead of letting the subjects try the features by themselves. Some subjects stated that the demo video was not very helpful to understand how the images were grouped in the Animated Snapshots tool or the actions tree and the gutter control in the Interaction History tool. We explicitly did not have a practice session on dummy tasks because the subjects might become confused between the interactions for the real and dummy tasks and we did not want to load their memory with yet more tasks. We observed that most subjects spent some time discovering the tools when using them for the first time, which may have affected their boundary determination times. We believe the success rate of finding boundaries is likely to improve with more tool training. Furthermore, since the subjects spent some time understanding and exploring the tools, the time spent using a tool might be more than would actually be the case in long term use.

Another threat to determining the time spent finding boundaries is the difficulty of knowing exactly how much time was spent using the tools. Although we logged every interaction of the subjects with our tools, most of the subjects spent time referring to the hard copy task descriptions. Furthermore, they asked questions to the investigator during boundary exploration.

These intermissions were not accounted for accurately in the tool usage time, consequently our usage data may not represent the time accurately.

We are aware that our programming tasks did not fully represent the real world. In a real working environment, programmers may have intermissions during and between tasks, furthermore, they can switch tasks. S4 stated that transition from task A to task B may not be always this clear. We designed our study to use tasks with clear cut transitions touching largely independent resources and based on a time limit per task. We disallowed task switching to simplify as much as possible the analysis of the results. In actual use, programmers may work on the same resources as part of multiple tasks, making some interfaces more difficult to use. We also asked subjects to determine the boundaries of the tasks when the tasks were completely over; in real life, a user can set the start boundary of a task anytime throughout the work, whenever he or she realizes that he or she forgot to create a task.

A session in our study concluded with an interview followed by a questionnaire. This procedure may have affected how a subject answered the questionnaire.

5.3 Tool Limitations

We received common feedback on two deficiencies about the way Interaction History tool displays interaction information. These deficiencies are caused by the limitations of the infrastructure on which the tools were built.

5.3.1 False Interactions

When a user selects a method body or a field while programming, or when the Java editor gains focus automatically (after switching a perspective or switching to Eclipse from another application), an interaction event of kind edit for the touched resource is generated after the expected selection interaction event. Semantically, this does not always correspond to the action of editing, which is adding or modifying some code. Because of this reason, some subjects (e.g., S2) stated their frustration; they did not understand why this edit information was in the interaction history when they did not edit that resource in the code.

5.3.2 Missing Context Information

Many subjects stated that the Interaction History tool needed to display more content information; for example, the subjects saw a series of *find and*

replace or *search* commands in the timeline but the tool did not show the search text used. Similarly, edit actions showed the resources that were edited, but not the changed content. Unfortunately, the detail about what was searched for or found is not tracked by the Mylar Monitor. The Animated Snapshots tool does not have this problem since the screenshot captured right before the *search* window is closed shows which information was the subject of the search.

5.4 Other Designs

In developing the tools studied in this thesis, we considered other designs. We initially designed four tools. Figure 5.1 shows the tools and how their characteristics fall on a scale of favouring more episodic or more semantic memory usage. The Animated Snapshots tool, described earlier in Section 3.1, stresses the use of episodic memory. Interaction History by Actions tool also stresses episodic more than semantic memory while the Interaction History by Elements tool stresses semantic more than episodic memory. The Interaction History tool, which was described in Section 3.2, was designed for the study to be a combination of the two Interaction History-based tools shown in Figure 5.1. Finally, the Poker tool stresses more heavily the use of the semantic memory. Elements of the other tool designs considered where also mentioned by subjects during interviews.

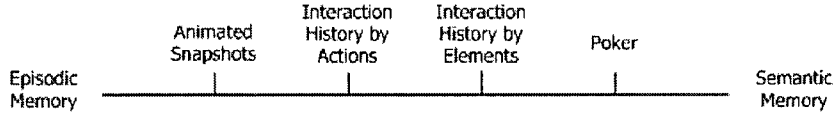

Figure 5.1: Initial set of tools to find task boundaries. The Animated Snapshots tool presents a user's personal experiences through snapshots of the user's working environment. The Interaction History by Actions tool displays the actions applied to the resources. The Interaction History by Elements tool shows the resources. The Poker tool lets a user find the resources using the features of the IDE and then guides him or her in finding the right interaction.

5.4.1 Interaction History by Actions Tool

Two subjects [S7, S10] suggested an actions-first approach in the Interaction History tool could have been useful as they believed they could better re-

member the resources through the actions. For example, for a subject who thinks that the first resource touched after the first run of an application is important, it may be useful to go through the *Run* action to find the resource. The Interaction History by Actions tool (Figure 5.2) we developed stresses the use of episodic memory by directing a user to first select an action that had been applied to the resources. This tool is similar to the Interaction History tool, except that it includes only the actions tree. A user can select an action from the tree and the corresponding interactions are highlighted in the timeline. We chose not to include this tool in the study so as to focus the study on more pure episodic and semantic-based interfaces. This tool did influence the design of the Interaction History tool included in the study.

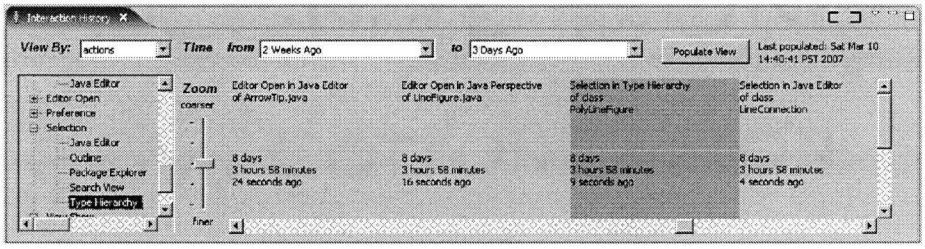

Figure 5.2: Interaction History by Actions tool. This tool is similar to the Interaction History tool, except that it includes only the actions tree.

5.4.2 Interaction History by Elements Tool

The Interaction History by Elements tool (Figure 5.3) is a primitive version of Interaction History tool (Section 3.2). This tool is similar to the Interaction History tool, except that it includes only the elements tree. This tool advocates the use of semantic memory because a user is directed first on the resources, rather than on the actions performed on the resources. When the data is populated in the tool, the tree on the left hand side displays the resources that have been subject to actions. When a user selects a resource from the tree, the corresponding interactions are highlighted in the timeline. Similar to the Interaction History by Actions tool, a user selects the boundaries of interest and set them with the brackets at the top of the view. We also chose not to include this tool in the study, focusing on the Interaction History tool as it simplified the highlighting of particular events in the timeline.

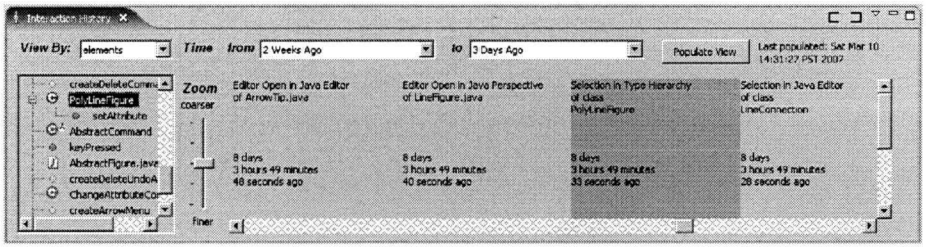

Figure 5.3: Interaction History by Elements tool. This tool is similar to the Interaction History tool, except that it includes only the elements tree.

5.4.3 Poker Tool

When using the Interaction History tool, two subjects [S3, S8] wanted to browse the code or open the *Open Type* view to remember the name of the resource of interest. Subjects S4 and S8 also mentioned that a search box for the elements tree would be useful to locate a resource in the tree. We had earlier designed the Poker tool to help a user who would like to use the features of the IDE to help find the resources. A user can browse code within any view in the IDE and once located can use the resource for boundary detection. The user can drag the resource (e.g., found in the Package Explorer, Outline View, etc.) and drop it on the Poker tool. Then, the user can select an action performed to that resource from the list of most commonly used actions: selection (anywhere in the IDE), edit, and save[21] and specify if he or she is interested in the first or in the last occurrence of that action to that resource. The interaction event corresponding to that query is selected and can be used as a start or end point for a task.

The key feature of this tool is that well known and remembered resources can be used directly to find an appropriate interaction to set as a boundary. This emphasis on the resources first means the Poker tool stresses semantic memory.

Despite its advantages, this tool had several drawbacks. When a user drops a resource on the tool, only the first or the last selection, edit or save can be selected. We made this choice to limit the queries for interactions to ensure the interface was simple and easy to use. However, the Poker tool is not complete because of this decision; interactions concerning other than selection, edit or save, or other occurrences of an interaction than the first or the last time are discarded by the tool. While designing this tool, we

[21] Save action for a Java method would yield the save action for the compilation unit containing that method.

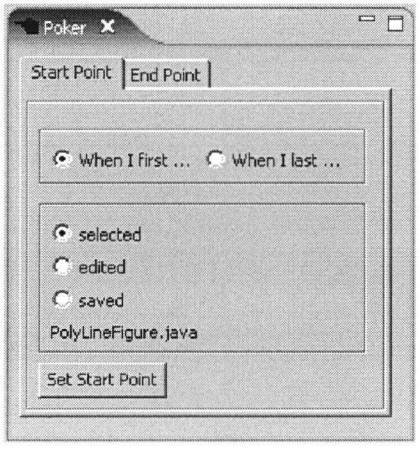

Figure 5.4: Poker tool. A user can drag and drop a resource on top of this tool and select either the first or last occurrence of an action applied to that resource.

faced a problem on finding the interactions concerning selection action. By definition, selecting a resource from a navigator view and dropping it on the Poker tool would constitute the last selection of that resource. This caused the tool to be not so useful, if a user was looking for the last occurrence of selection. On the other hand, we solved this problem by collecting our tools in the Task Boundaries Perspective and disabling capturing interaction events when this perspective is active. Therefore whenever a user selects a resource from a navigator view while finding boundaries, only work done in development environments (e.g., in the Java Perspective, Debug Perspective etc.) would be displayed.

5.4.4 Zoom Control and Break Indication

Given a broad range of time and massive amount of interaction events, the number of interactions to be displayed in the timeline can be large. In fact, some of the subjects complained there was too much information displayed in the Interaction History tool. One way to overcome this problem is to group interactions in the timeline. The Zoom Control (Figure 5.5A) is intended to provide such grouping and provides five levels of zoom from *coarser* to *finer*. Depending on the zoom level, the interaction history is chunked to

groups containing nearly equal amount of interactions[22], and the timeline shows the first interaction in each chunk. The chunking is based on the temporal sequence of the interaction events rather than their meanings, like being part of same or different tasks. This chunking algorithm is simple yet useful in our own experience; it helps roughly figure out the location of the interactions of interest.

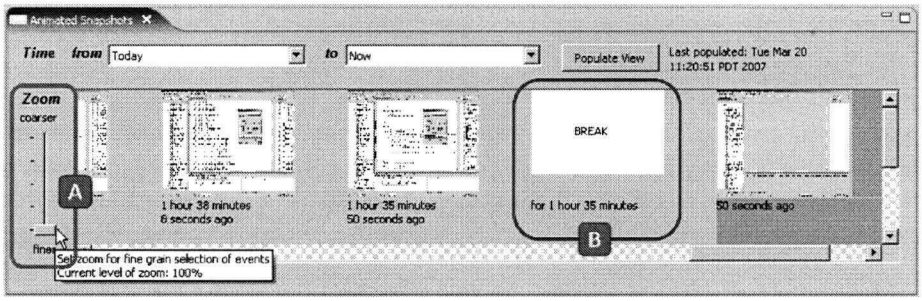

Figure 5.5: Zoom control and break indication. The zoom control (Part A) provides a means to group the interactions displayed in the timeline to reduce the information overload. At the finer level of zoom, breaks between the interactions are shown (Part B) to provide a user a temporal cue.

At the finer level of zoom, we also provide a facility to explicitly represent the breaks between the interactions, as pauses might be useful for finding boundaries. S8, for instance, mentioned such functionality would be useful. The break detection threshold is set to five minutes by default and whenever two successive interactions have more than five minutes of time difference, the timeline explicitly shows the break with the duration of the pause (Figure 5.5B).[23] It is important to note that since the breaks are inserted purely according to the temporal activity, it does not always denote a switch between two successive tasks.

We removed the zoom control from the tools we studied to simplify the user interface and make them easy to use. We set the default zoom level to *finer* to show maximum detail. Since the interaction history of a user study had at most one hour worth of data, displaying all the events did not cause any performance problems. We were aware that none of the subjects would see the break indication because the threshold was too high for condensed

[22]For example, when used with the Animated Snapshots tool, the timeline shows approximately 25, 19, 13, 6, and 3 thumbnails when the zoom level is set to *finer*, *less-finer*, *center*, *less-coarser*, and *coarser* respectively.

[23]For Interaction History tool, the text "Break" is displayed in the timeline.

programming work; instead, we wanted to observe whether the subjects would notice the time difference between two successive interactions or not. Only one subject [S8] noticed two interactions with two minutes of time difference in between and used this cue to mark the end point of the third task.

5.4.5 Extracting Task Information

Since one of the use cases of our tools is being able to extract a task from a programmer's previous work, we originally developed a feature to create a Mylar task out of the programming context between two selected boundaries. The initial implementation of Task Boundaries View (Section 3.3.3) contained a button labelled *Extract Task* instead of the *Finish Finding* button, before we simplified this tool for the purposes of the user study. Selection of the *Extract Task* button causes a new Mylar task to be created and the interesting (touched) resources between the time frame of the two boundaries are used to form a context for this task as if the programmer performed these interactions while a Mylar task is active. The resulting Mylar task can be used to focus the Eclipse UI on that task to improve programmer productivity. As an example, Figure 5.6 shows extracted task information out of a three minute implementation sequence between the selection of `NodeFigure.java` in the Java Editor, until saving the same compilation unit. The newly extracted Mylar task is activated, and instead of showing the whole code base, the Package Explorer on the left is focused only on the touched (black) and inferred (greyed out) resources.

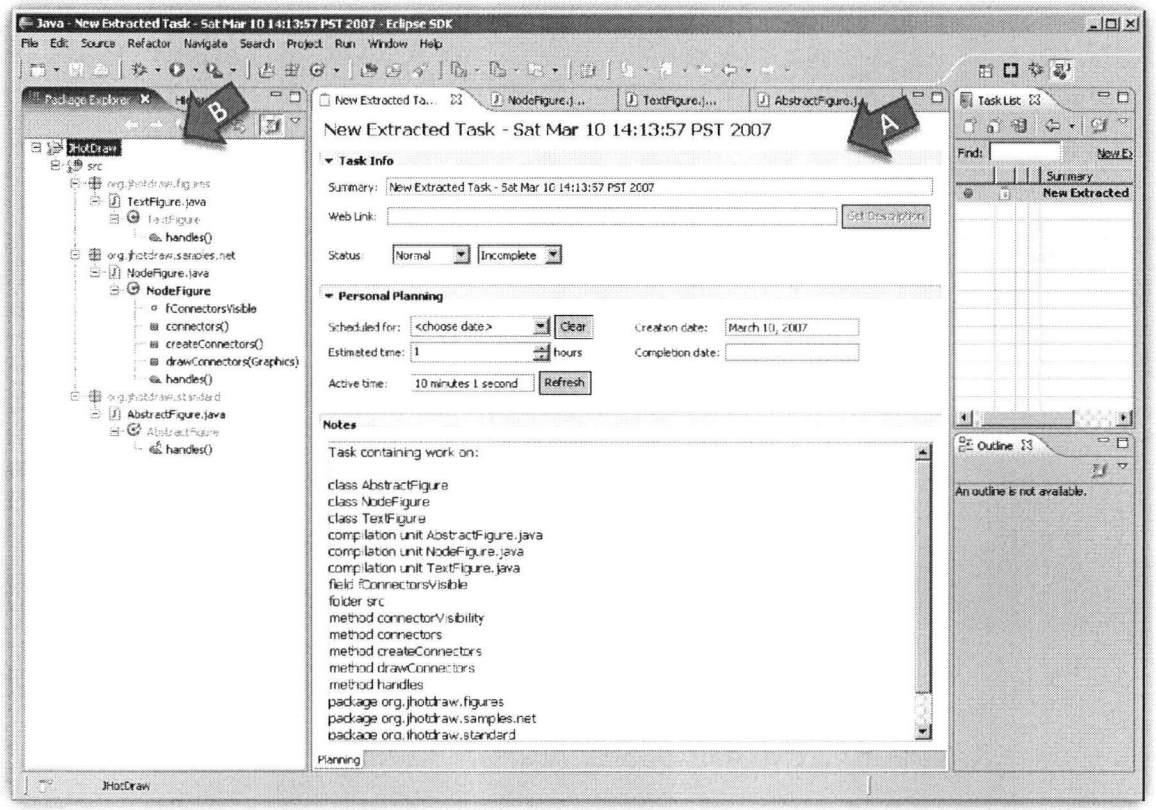

Figure 5.6: Focusing Eclipse UI on an extracted task. The extracted task (Part A) can be used to focus the Package Explorer to just show the resources relevant to that task (Part B).

Chapter 6

Conclusion

Current support for helping computer users multi-task by remembering and recalling information associated with a task require a user to manually indicate the start and end of tasks (e.g., TaskTracer [6], Eclipse Mylar [14]). This manual identification can cause problems with associating the wrong information with a task [6] or failing to associate the right information with a task (e.g., Eclipse Mylar bug #157933, #166489). These errors decrease the usefulness of the multi-tasking support. These errors could be reduced if a user was able to indicate where a task should have started (for an active task) or where a task did start and end in previous information monitored about the user's activity.

In this thesis, we have demonstrated that one form of computer user, programmers, were able to more accurately determine task boundaries using an interface that stressed programmers' episodic memory. This interface visually replays snapshots recorded silently as programmers worked. This interface was strongly preferred by programmers to identify task boundaries over an interface based on semantic memory, in which programmers recalled what they worked on as a cue to determining the desired boundary. We also found that programmers' meaning of a task boundary varied substantially. This preliminary evidence can be used as a basis to design a more robust interface for task boundary determination that can be subjected to more rigorous testing on a wider selection of users. These interfaces can be used to help improve multi-tasking support for computer users.

6.1 Future Work

Tulving suggests that many activities performed by humans contain features concerning both episodic and semantic memory models [22]. Five of the subjects in our study also proposed some integration of the features provided by the Animated Snapshots tool and the Interaction History tool. We thus see value in developing a new tool with characteristics from both interfaces. Specifically, the tool could chunk the most commonly used and edited resources and associated actions. When a group of actions is selected, the tool

could present both visual and textual cues: the actions could be replayed to the user with additional resource information highlighted on the snapshots.

Various techniques to analyze the interaction history could also be used to pre-process the interaction history prior to showing it to a user who is attempting to find task boundaries. For instance, we could introduce mechanisms to group related chunks of work and to detect more intelligently the breaks instead of temporal thresholds. A tool could then direct the programmer to those points where a task likely started or ended.

Constantly taking screenshots and watching the working environment of a programmer could impact a user's privacy. This issue could be addressed by encrypting the image files so that they cannot be opened and viewed by others without permission. Our current implementation stores images in raw byte format and compresses the images; these images cannot be displayed directly by any software in the market.

Bibliography

[1] V. Bellotti, N. Ducheneaut, M. Howard, and I. Smith. Taking email to task: the design and evaluation of a task management centered email tool. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 345–352, New York, NY, USA, 2003. ACM Press.

[2] S. Ceri, F. Daniel, M. Matera, and F. Rizzo. Extended memory (xmem) of web interactions. In *ICWE '06: Proceedings of the 6th international conference on Web engineering*, pages 177–184, New York, NY, USA, 2006. ACM Press.

[3] M. Czerwinski and E. Horvitz. An investigation of memory for daily computing events. In X. Faulkner, J. Finlay, and F. Detienne, editors, *People and Computers XVI, Proceedings of HCI 2002*, pages 230–245. Springer-Verlag, 2002.

[4] M. Czerwinski, E. Horvitz, and S. Wilhite. A diary study of task switching and interruptions. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 175–182, New York, NY, USA, 2004. ACM Press.

[5] P. Dourish, W. K. Edwards, A. LaMarca, and M. Salisbury. Presto: an experimental architecture for fluid interactive document spaces. volume 6, pages 133–161, New York, NY, USA, 1999. ACM Press.

[6] A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker. Tasktracer: a desktop environment to support multi-tasking knowledge workers. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 75–82, New York, NY, USA, 2005. ACM Press.

[7] E. Freeman and D. Gelernter. Lifestreams: a storage model for personal data. volume 25, pages 80–86, New York, NY, USA, 1996. ACM Press.

[8] V. M. González and G. Mark. "constant, constant, multi-tasking craziness": managing multiple working spheres. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 113–120, New York, NY, USA, 2004. ACM Press.

[9] S. G. Hart and L. E. Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In P. Hancock and N. Meshkati, editors, *Human Mental Workload*, pages 139–183, Amsterdam, The Netherlands, 1988. Elsevier Science Publishers.

[10] D. A. Henderson Jr. and S. Card. Rooms: the use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. volume 5, pages 211–243, New York, NY, USA, 1986. ACM Press.

[11] V. Kaptelinin. Umea: translating interaction histories into project contexts. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 353–360, New York, NY, USA, 2003. ACM Press.

[12] R.T. Kellogg. In *Cognitive psychology, 2nd Ed.*, Thousand Oaks, CA, 2003. Sage Publications.

[13] M. Kersten. Focusing knowledge work with task context. PhD thesis, Department of Computer Science, University of British Columbia, Vancouver, BC, Canada, January 2007.

[14] M. Kersten and G. C. Murphy. Mylar: a degree-of-interest model for ides. In *AOSD '05: Proceedings of the 4th international conference on Aspect-oriented software development*, pages 159–168, New York, NY, USA, 2005. ACM Press.

[15] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 1–11, New York, NY, USA, 2006. ACM Press.

[16] M. Ringel, E. Cutrell, S. Dumais, and E. Horvitz. Milestones in time: The value of landmarks in retrieving information from personal stores. pages 184–191, Amsterdam, The Netherlands, 2003. IOS Press.

[17] G. Robertson, E. Horvitz, M. Czerwinski, P. Baudisch, D. R. Hutchings, B. Meyers, D. Robbins, and G. Smith. Scalable fabric: flexible task

management. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 85–89, New York, NY, USA, 2004. ACM Press.

[18] J. Shen, L. Li, T. G. Dietterich, and J. L. Herlocker. A hybrid learning system for recognizing user tasks from desktop activities and email messages. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, pages 86–92, New York, NY, USA, 2006. ACM Press.

[19] G. Smith, P. Baudisch, G. Robertson, M. Czerwinski, B. Meyers, D. Robbins, and D. Andrews. Groupbar: The taskbar evolved. In *OZCHI '03: Proceedings of the 17th conference of the computer-human interaction special interest group (CHISIG) of Australia on Computer-human interaction*, Narrabundah, Australia, Australia, 2003. Computer-Human Interaction Special Interest Group (CHISIG) of Australia.

[20] S. Stumpf, X. Bao, A. N. Dragunov, T. G. Dietterich, J. Herlocker, K. Johnsrude, L. Li, and J. Shen. Predicting user tasks: I know what you're doing! In *Workshop on Human Comprehensible Machine Learning in 20th National Conference on Artificial Intelligence (AAAI-05)*. AAAI Press, 2005.

[21] C. Tashman. Windowscape: a task oriented window manager. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 77–80, New York, NY, USA, 2006. ACM Press.

[22] E. Tulving. Episodic and semantic memory. In E. Tulving and W. Donaldson, editors, *Organization of memory*, pages 381–403, New York, 1972. Academic Press.

[23] E. Tulving. In *Elements of Episodic Memory*, New York, 1983. Oxford University Press.

Appendix A

Animated Snapshots Tool Replay Example

To convey the look and feel of the Animated Snapshots tool, Figures A.1 through A.6 present images as the tool runs. After right-clicking a thumbnail in the timeline displayed by the Animated Snapshots tool, the animation starts (Figures A.1-Label 1 and A.1-Label 2). The user sees familiar code—selection of a block that will be copied and pasted—and presses the *space* key to pause the replay (Figure A.2-Label 3). When paused, the navigation controls are overlaid on the snapshot. Users can then control the replay mode using the keys: S and E sets an image as start and end point respectively, F toggles the image from its default size to full screen, *left* and *right* navigates back and forward, A and Z takes the user to the first and last image of the group, *space* pauses or resumes the replay, and finally Q quits the replay mode. The user then advances manually to the next frames by pressing the *right* key to find the exact point where the task ended. In Figures A.2-Label 4, A.3-Label 5, and A.3-Label 6, the user watches himself or herself editing the newly pasted code, saving it (Figure A.4-Label 7) and running for testing the changes (Figure A.4-Label 8). Figure A.5-Label 9 and A.5-Label 10 show the last state of the tested sample application and the user going back to Eclipse IDE right after closing the sample application. The user notices that the state of the screen shown in Figure A.5-Label 10 is where he or she ended the task. Out of curiosity, the user keeps looking at the following snapshots until reaching the point where all of the editors are closed (Figure A.6-Label 11).[24] The user then goes back to the point after closing the sample application and marks it as the end boundary of that task (Figure A.6-Label 12).

[24]Snapshots in which the user closes the editors are omitted.

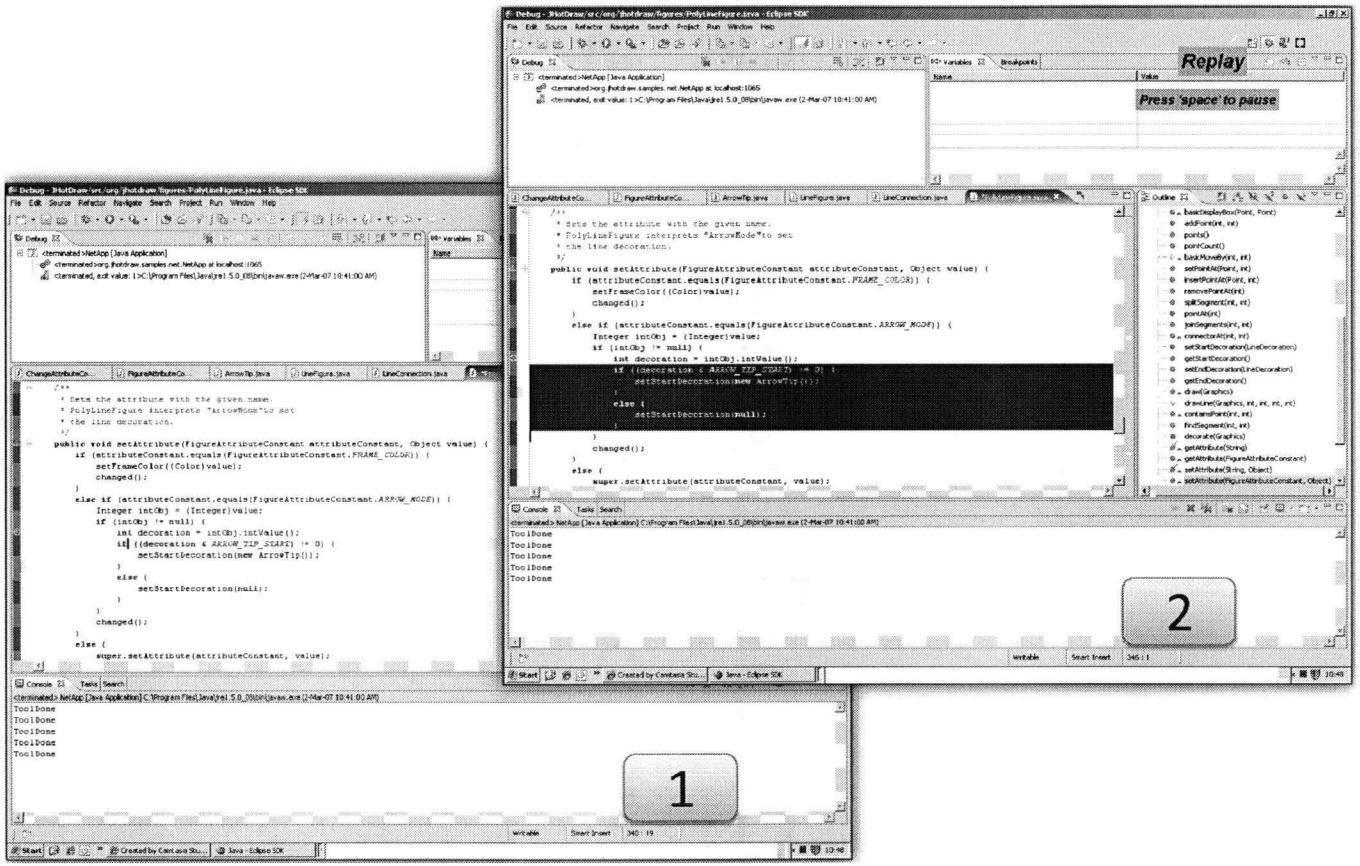

Figure A.1: Animated Snapshots tool replay example (part one)

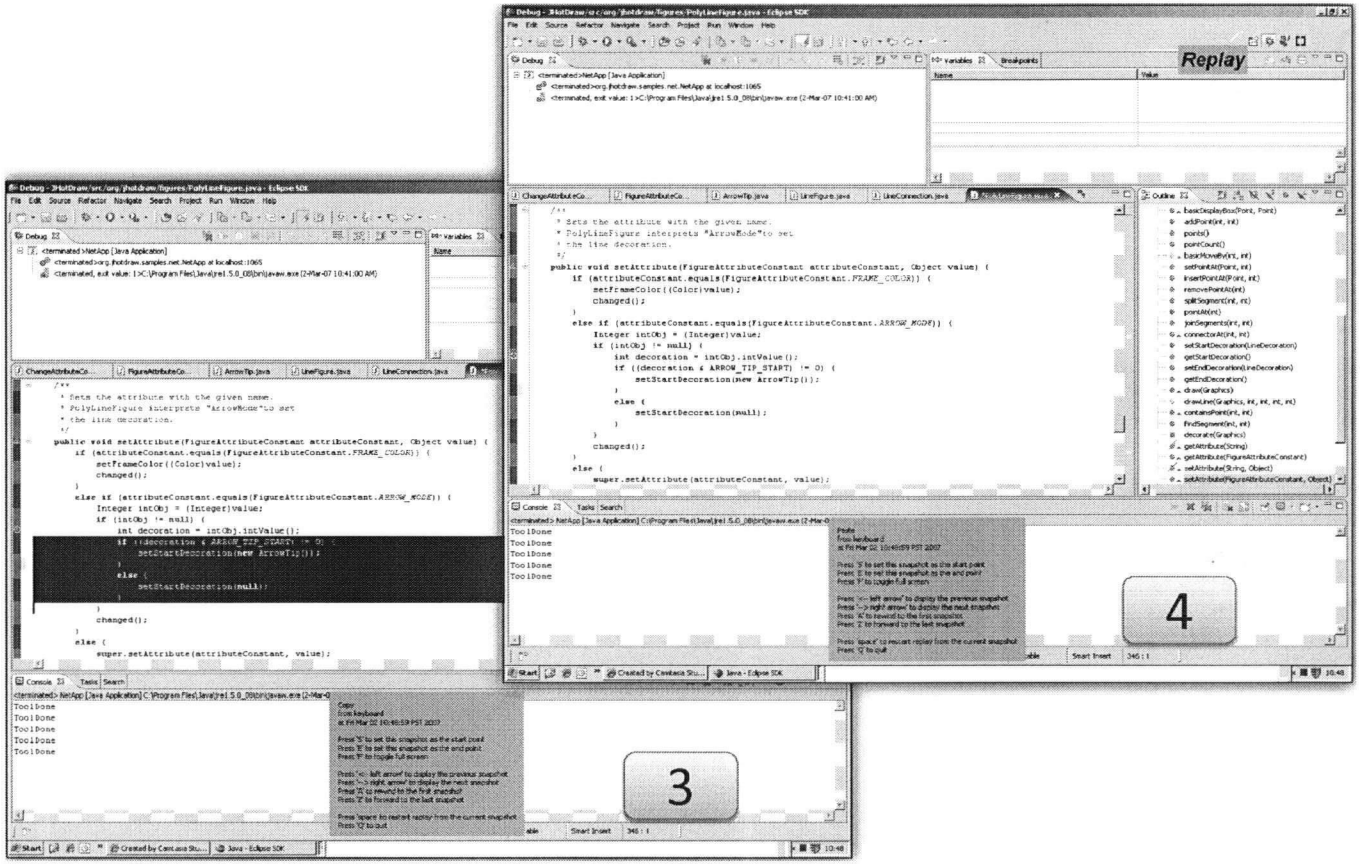

Figure A.2: Animated Snapshots tool replay example (part two)

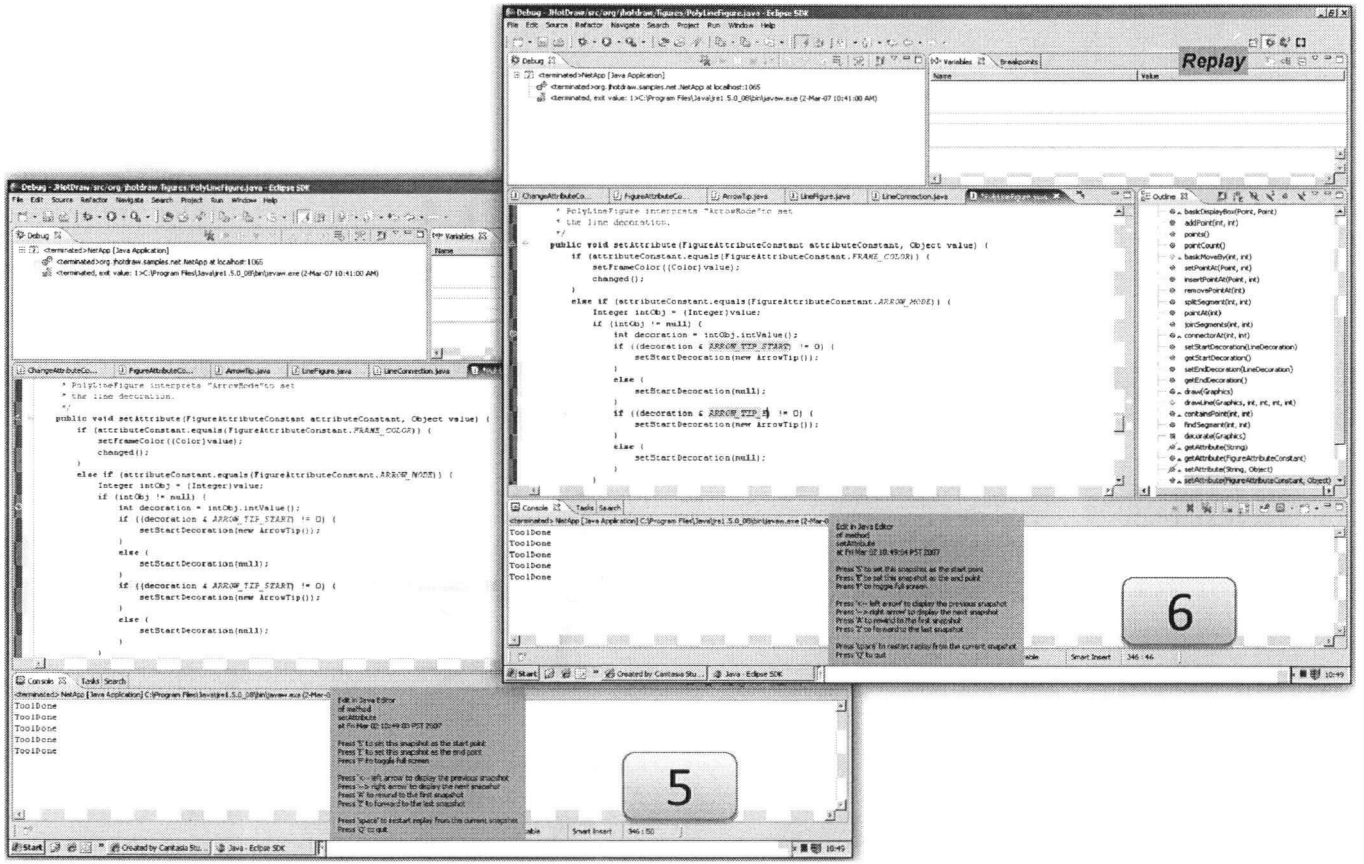

Figure A.3: Animated Snapshots tool replay example (part three)

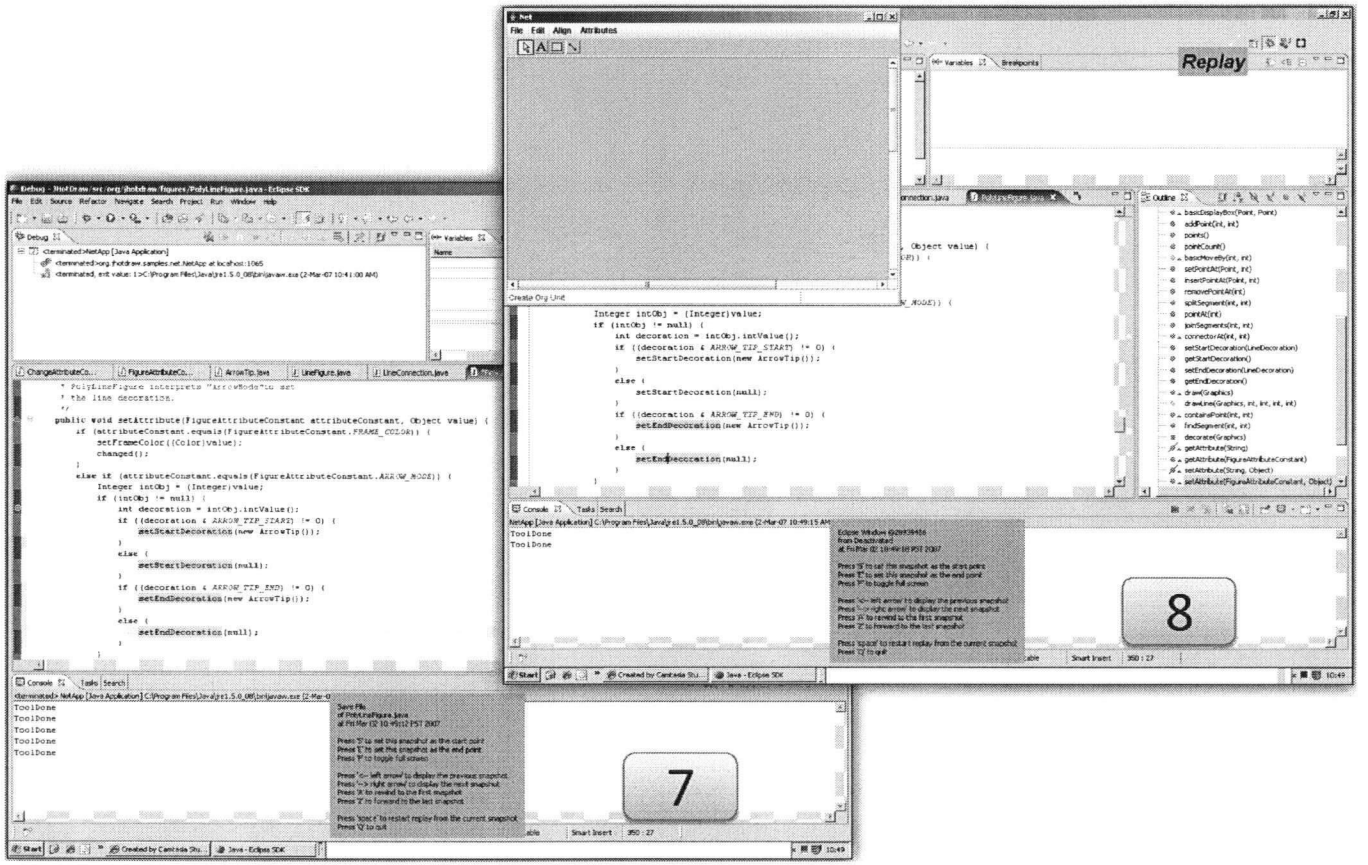

Figure A.4: Animated Snapshots tool replay example (part four)

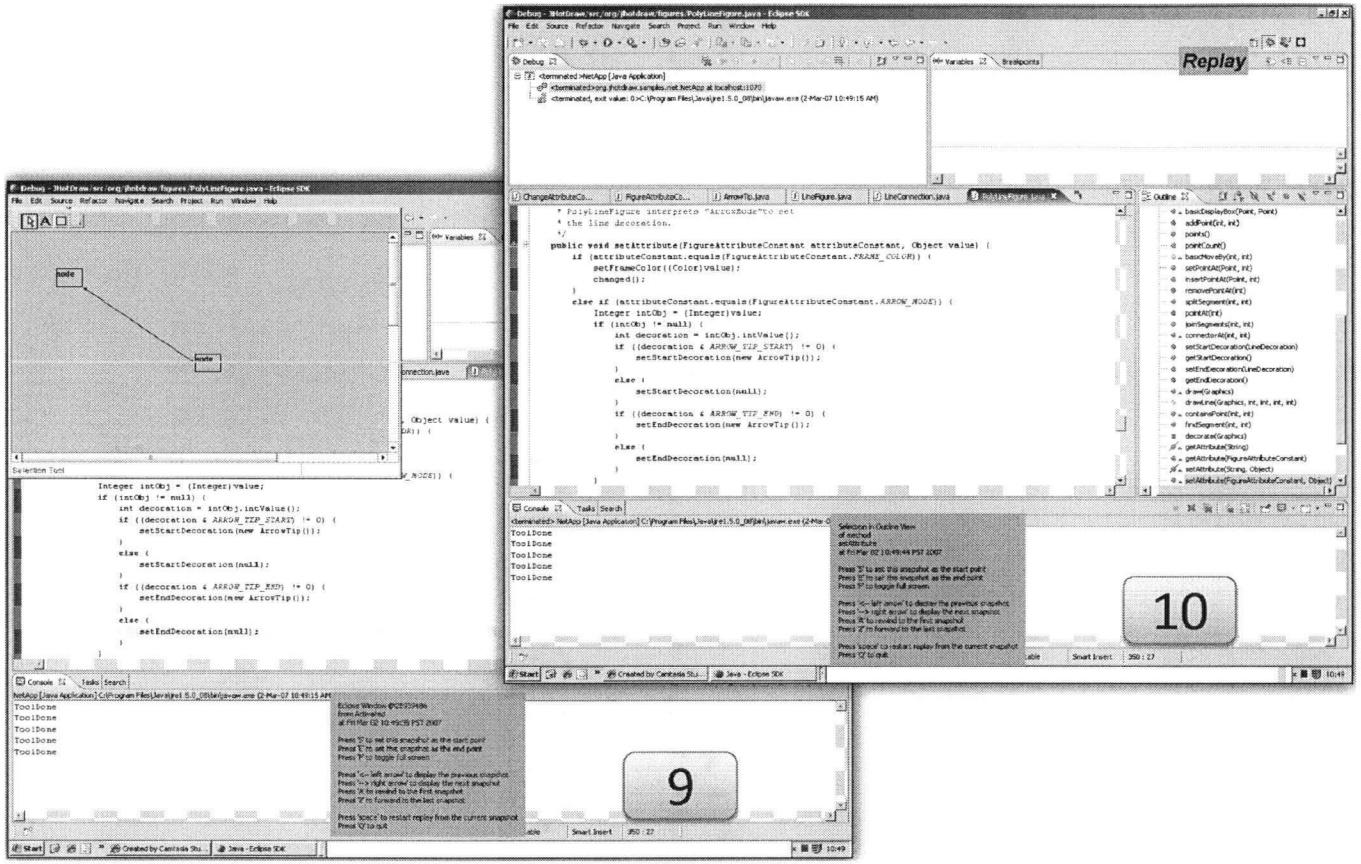

Figure A.5: Animated Snapshots tool replay example (part five)

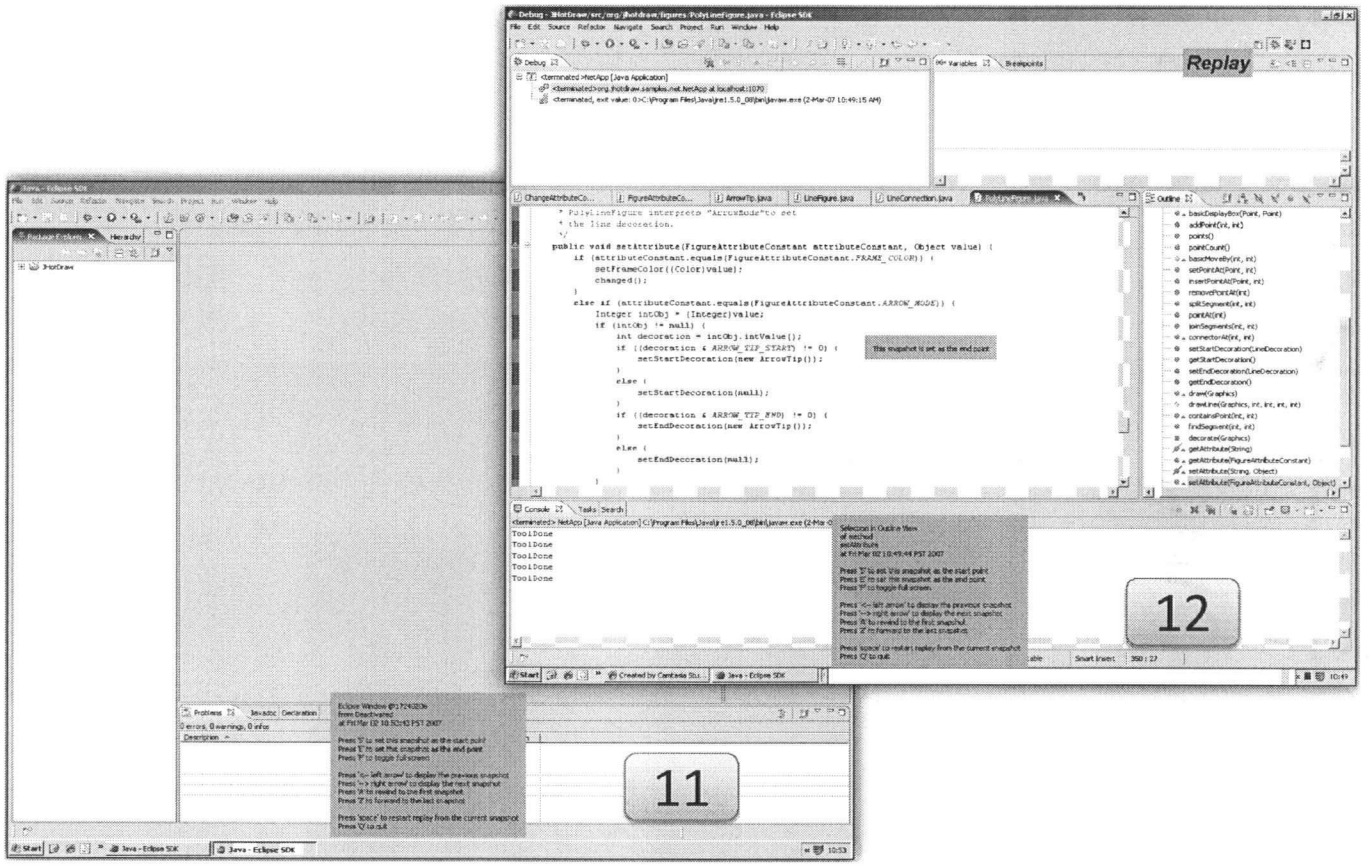

Figure A.6: Animated Snapshots tool replay example (part six)

Appendix B

Questionnaire

We used a variant of NASA-TLX (Task Load Index) [9] questionnaire to measure the workload caused by our tools. We also requested demographic information.

Workload Questionnaire (Animated Snapshots Tool)

Participant ID: _____ **Tool used in tasks #: _____**

We'd like to ask you about how difficult you perceived the *process of finding the task boundaries by means of the tool you just used*. Please answer the questions below by marking an **X** in the appropriate box in the provided rating scales.

Mental Demand: How much mental and perceptual activity was required (e.g., thinking, deciding, calculating, remembering, looking, searching, etc.)? Was the process easy or demanding, simple or complex, exacting or forgiving?

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Low **High**

Physical Demand: How much physical activity was required (e.g., clicking, scrolling, pushing, pulling, turning, controlling, activating, etc.)? Was the process easy or demanding, slow or brisk, slack or strenuous, restful or laborious?

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Low **High**

Temporal Demand: How much time pressure did you feel due to the rate or pace at which the process or elements in the process occurred? Was the pace slow and leisurely or rapid and frantic?

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Low **High**

Performance: How successful do you think you were in accomplishing the goals of the process set by the experimenter (or yourself)? How satisfied were you with your performance in accomplishing these goals?

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Good **Poor**

Effort: How hard did you have to work (mentally and physically) to accomplish your level of performance?

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Low **High**

Frustration Level: How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, relaxed and complacent did you feel during using the process?

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Low **High**

Workload Questionnaire (Interaction History Tool)

Participant ID: _____ **Tool used in tasks #: _____**

We'd like to ask you about how difficult you perceived the *process of finding the task boundaries by means of the tool you just used*. Please answer the questions below by marking an **X** in the appropriate box in the provided rating scales.

Mental Demand: How much mental and perceptual activity was required (e.g., thinking, deciding, calculating, remembering, looking, searching, etc.)? Was the process easy or demanding, simple or complex, exacting or forgiving?

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Low **High**

Physical Demand: How much physical activity was required (e.g., clicking, scrolling, pushing, pulling, turning, controlling, activating, etc.)? Was the process easy or demanding, slow or brisk, slack or strenuous, restful or laborious?

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Low **High**

Temporal Demand: How much time pressure did you feel due to the rate or pace at which the process or elements in the process occurred? Was the pace slow and leisurely or rapid and frantic?

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Low **High**

Performance: How successful do you think you were in accomplishing the goals of the process set by the experimenter (or yourself)? How satisfied were you with your performance in accomplishing these goals?

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Good **Poor**

Effort: How hard did you have to work (mentally and physically) to accomplish your level of performance?

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Low **High**

Frustration Level: How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, relaxed and complacent did you feel during using the process?

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Low **High**

Importance of Different Workload Categories

Participant ID: _____

Please select the member of each pair that had the more significant effect on the overall *workload* for all the processes (finding task boundaries using two different tools) performed in this study. Descriptions of the categories are found at the bottom of the page.

Physical Demand	☐ or ☐	Mental Demand
Temporal Demand	☐ or ☐	Mental Demand
Performance	☐ or ☐	Mental Demand
Effort	☐ or ☐	Mental Demand
Frustration Level	☐ or ☐	Mental Demand
Temporal Demand	☐ or ☐	Physical Demand
Performance	☐ or ☐	Physical Demand
Effort	☐ or ☐	Physical Demand
Frustration Level	☐ or ☐	Physical Demand
Temporal Demand	☐ or ☐	Performance
Temporal Demand	☐ or ☐	Effort
Temporal Demand	☐ or ☐	Frustration Level
Performance	☐ or ☐	Effort
Performance	☐ or ☐	Frustration Level
Frustration Level	☐ or ☐	Effort

Category definitions

Mental Demand: How much mental and perceptual activity was required (e.g., thinking, deciding, calculating, remembering, looking, searching, etc.)? Was the process easy or demanding, simple or complex, exacting or forgiving?

Physical Demand: How much physical activity was required (e.g., clicking, scrolling, pushing, pulling, turning, controlling, activating, etc.)? Was the process easy or demanding, slow or brisk, slack or strenuous, restful or laborious?

Temporal Demand: How much time pressure did you feel due to the rate or pace at which the process or elements in the process occurred? Was the pace slow and leisurely or rapid and frantic?

Performance: How successful do you think you were in accomplishing the goals of the process set by the experimenter (or yourself)? How satisfied were you with your performance in accomplishing these goals?

Effort: How hard did you have to work (mentally and physically) to accomplish your level of performance?

Frustration Level: How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, relaxed and complacent did you feel during using the process?

Questionnaire

Participant ID: _____

Age: _____

Academic institution: _____

Academic standing: _____

Years of Java programming experience: _____

Years of Eclipse IDE experience: _____

Preference ranking of tools:

Please rank the tools to find the task boundaries according to your preference. Please assign 1 or 2 to each tool; 1 being your best preference, and 2 being your least preference.

☐ Animated Snapshots tool

Please give a brief explanation why you think *Animated Snapshots Tool* deserves this rank.

_____
_____
_____

☐ Interaction History Tool

Please give a brief explanation why you think *Interaction History Tool* deserves this rank.

_____
_____
_____

Appendix C

JHotDraw Tutorial

We prepared the following tutorial on JHotDraw as a guide for the subjects in the programming tasks. This tutorial focuses only on the parts of JHotDraw that are related to the programming tasks.

JHotDraw Tutorial

What is JHotDraw?

- A framework which targets applications for drawing technical and structured graphics.
- JHotDraw defines a basic skeleton for a GUI-based editor with tools in a tool palette, different views, user-defined graphical figures, and support for saving, loading, and printing drawings.
- An application can be created from the framework, and can be customized using inheritance and combining components.

Basic Components of the Architecture

- Any application that uses JHotDraw has a window dedicated for drawing. This `DrawWindow` is the editor window. It contains one or more internal frames, each associated with a drawing view.
- The `DrawingView`, is an area that can display a `Drawing` and accepts user input. Changes in the `Drawing` are propagated to the `DrawingView` that is responsible for updating any graphics.

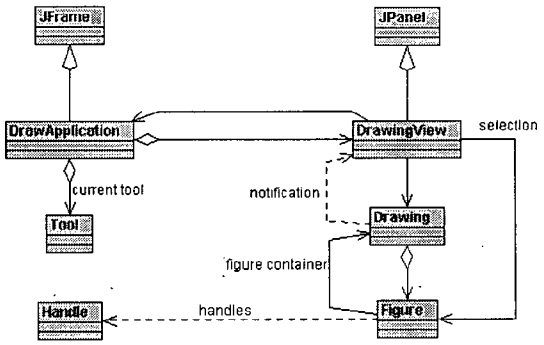

Figure 1 - Basic Components of the Architecture

- The `Drawing` consists of `Figures`, which in turn can be containers for other `Figures`.
- Each `Figure` has `Handles`, which define access points and determine how to interact with the `Figure` (for example, how to connect the `Figure` with another `Figure`).
- In a `DrawingView`, you can select several figures and manipulate them. The `DrawWindow` itself usually has one active `Tool` from the tool palette, which operates on the `Drawing` associated with the current `DrawingView`.

Uses of JHotDraw

- Two sample applications included within the package are:
  - JavaDraw
  - Net

JavaDraw Sample Application

- This sample illustrates various standard tools and figures provided with JHotDraw.
- It also provides animation support for the figures and supports attaching URLs to Figures.

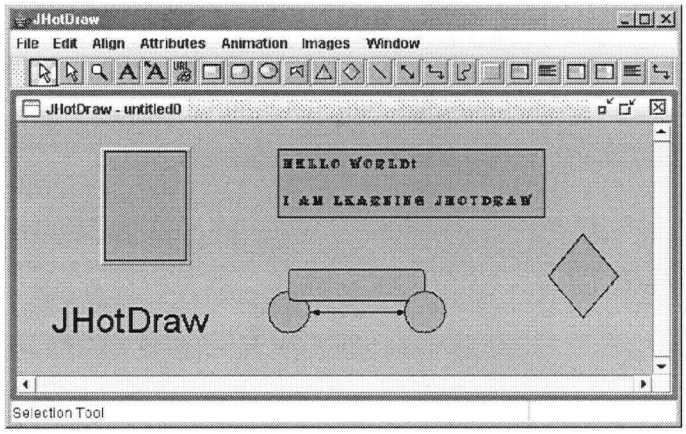

Figure 2 - JavaDraw Sample Application

- You can start the application by right clicking to:

  `org.jhotdraw.samples.javadraw`

  `JavaDrawApp.java`

  and selecting: Run As -> Java Application

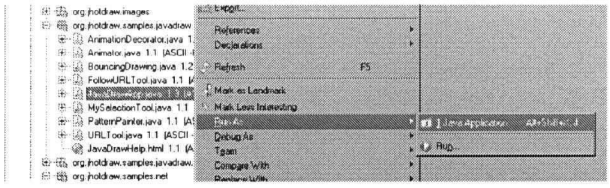

Figure 3 - Running JavaDraw

Net Sample Application

- A simple network editor. It enables the creation of nodes. When selected, a node displays a set of handles that can be used to connect it with other nodes.
- Nodes are connected at specific semantic locations. The potential locations are highlighted once the mouse moves inside a node.

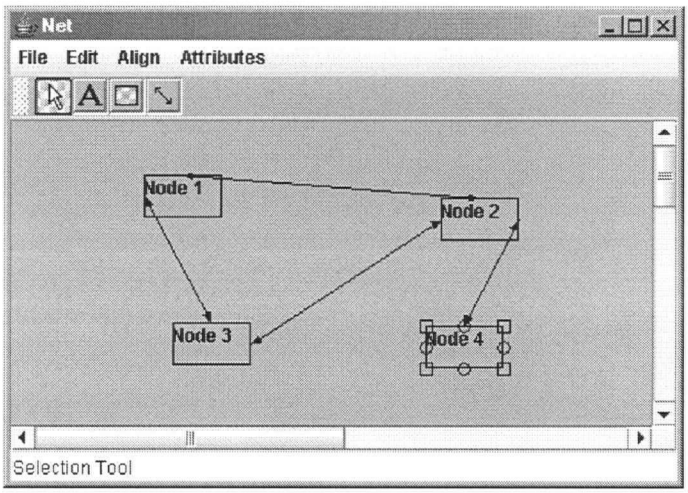

Figure 4 - Net Sample Application

- You can start the application by right clicking to:

```
org.jhotdraw.samples.net
  NetApp.java
```

and selecting: Run As -> Java Application

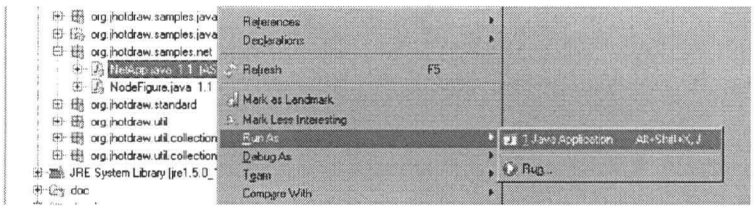

Figure 5 - Running Net

JHotDraw Package Organization

- `org.jhotdraw.application`
  - Defines a default user interface for standalone JHotDraw applications.
- `org.jhotdraw.contrib`
  - Includes the classes that were contributed by others.
- `org.jhotdraw.figures`
  - A kit of figures together with their associated support classes (tools, handles).
- `org.jhotdraw.framework`
  - Includes the classes and interfaces that define the JHotDraw framework. It doesn't provide any concrete implementation classes.
- `org.jhotdraw.samples.javadraw`
  - The package for the Javadraw sample application.
- `org.jhotdraw.samples.net`
  - The package for the Net sample application.
- `org.jhotdraw.standard`
  - Provides standard implementations of the classes defined in the framework package. It provides abstract classes that implement a framework interface and provide default implementation. The convention is to prefix such classes with `Abstract`, e.g., `AbstractFigure`, `AbstractHandle`. Classes that implement a framework interface and can be used as is start with Standard, e.g., `StandardDrawing`, `StandardDrawingView`.
- `org.jhotdraw.util`
  - This package provides generally useful utilities that can be used independent of JHotDraw.

References:

- http://www.jhotdraw.org/
- http://www.javaworld.com/javaworld/jw-02-2001/jw-0216-jhotdraw.html
- http://www.riehle.org/blogs/research/2007/2007-01-03.html
- http://softarch.cis.strath.ac.uk/PLJHD/Patterns/JHDDomainOverview.html

Appendix D

Programming Tasks

To evaluate our tools we required a history of programming activity. We designed four programming tasks to be performed by the user study participants to create programming activity history for approximately one hour. The first and third programming tasks can be directly implemented on the original JHotDraw v6.0b1 source code. For the second and fourth tasks, we modified the JHotDraw v6.0b1 code base.

For the second programming task, we removed a piece of code from the `PolyLineFigure` class that connects two figures and that displays arrow tip decorations in the connecting line. The programming task asks subjects to fix the problem of correctly displaying arrow tips by reimplementing the missing part. The solution consists of copying and pasting from a nearby code, and editing slightly the newly pasted piece.

For the last programming task, we added a label field next to the status bar to display the size information (width by height in pixels) of the active drawing windows. This enhancement is made to the `DrawApplication` class in the application package of JHotDraw framework so that any application using the framework shows this addition. As the programming task, we asked subjects to call the method that would display the information from appropriate places so that the size of the active drawing window is always displayed.

Instructions:

You will have 1 hour to work on four programming tasks. Each programming task has been allocated up to 15 minutes (the investigator will advise you to move to the next task when the time is up).

You must work on the tasks in the given order and you are not allowed to switch tasks or go back to previous tasks. Once you start a task, make your best effort to find which code should be modified or added to complete a task and attempt to make those changes.

YOU WILL NOT BE ASSESSED ACCORDING TO YOUR SUCCESS WHILE DOING THESE TASKS.

Task 1: Connecting Nodes

Description: When connecting two nodes, you can draw lines from one of the left-hand, right-hand, top and bottom side of the first node to one of the left-hand, right-hand, top and bottom side of the second node. Please modify the source code so that the connection can be done from and to the corners of a node as well.

Applies to: The Net sample application

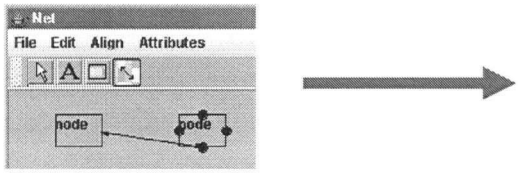

Figure 1 (a) - Initial Implementation

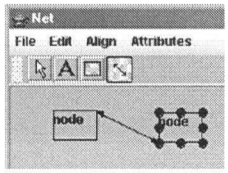

Figure 1 (b) - Desired Implementation

Task 2: Arrow Tips

Description: When connecting two shapes or nodes, by default, the connecting line shows arrows pointing in both directions (both shapes or nodes). Unfortunately, this arrow implementation does not work properly. Even though the arrows can be set from the *Attributes -> Arrow* menu as 'none' or 'at Start' to show no arrows, or just the arrow at the start point respectively, the arrow at the end point is always visible no matter what is selected from the menu. Please fix this error so that all the attributes in the *Attributes -> Arrow* menu work correctly.

Applies to: The JHotDraw framework (Hint: Test it with the *Net* sample application)

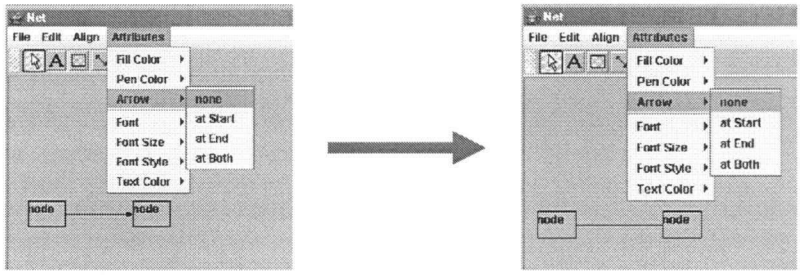

Figure 2 (a) - Initial Erroneous Implementation

Figure 2 (b) - Desired Correct Implementation

Task 3: Duplicate Shapes

Description: The JHotDraw framework already contains a way to duplicate shapes using the menu command *Edit -> Duplicate*. JHotDraw also allows execution of some commands using the keys (e.g. moving a shape using arrow keys, or deleting a shape by pressing the "delete" key). Please implement a keyboard shortcut for the *duplicate* command so that when you select a shape and press the space key a copy of that shape is created.

Applies to: The JHotDraw framework (Hint: Test it with the *Java Draw* sample application)

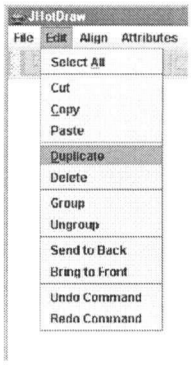

Figure 3 - Duplicate menu item

Task 4: View Size

Description: Please improve the JHotDraw framework by adding a feature to display the size of the active drawing canvas (view). A placeholder with the text "Active View Size: " is created as a convenience for you (See Figure 4-a). Please modify the source code to update this placeholder text so that it gets updated according to the size of the active view.

Applies to: The JHotDraw framework (Hint: Test it with the *Java Draw* sample application)

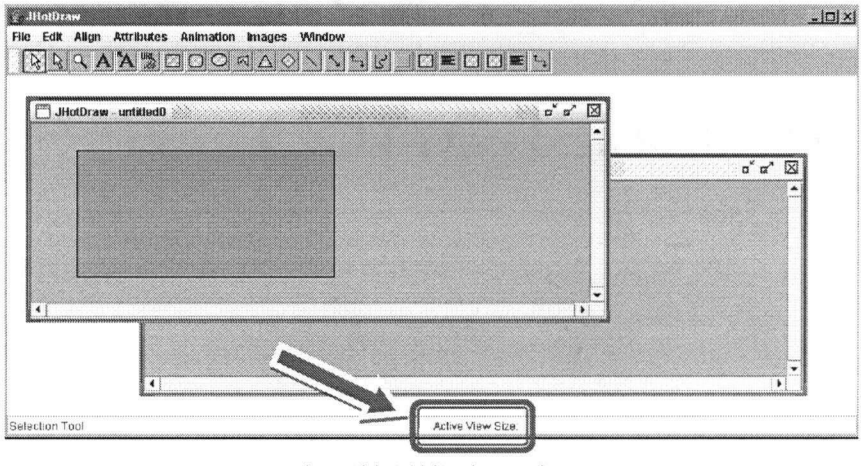

Figure 4 (a) - Initial Implementation

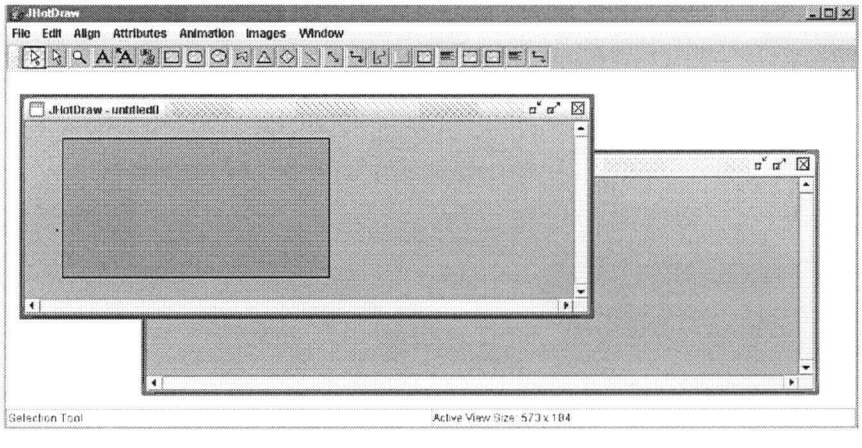

Figure 4 (b) - Desired Implementation

Appendix E

UBC Research Ethics Board Certificate

The study described in this thesis was approved by the University of British Columbia Behavioural Research Ethics Board.

The University of British Columbia
Office of Research Services
Behavioural Research Ethics Board
Suite 102, 6190 Agronomy Road, Vancouver, B.C. V6T 1Z3

CERTIFICATE OF APPROVAL - MINIMAL RISK

PRINCIPAL INVESTIGATOR:	INSTITUTION / DEPARTMENT:	UBC BREB NUMBER:
Gail C. Murphy	UBC/Science/Computer Science	H06-04060

INSTITUTION(S) WHERE RESEARCH WILL BE CARRIED OUT:

Institution	Site
UBC	Point Grey Site

Other locations where the research will be conducted:
UBC Computer Science Department, Software Practices Lab. Subjects living in the Vancouver area will be invited to the UBC campus, subjects outside of the Vancouver area will be visited in public areas at their universities by one of the investigators.

CO-INVESTIGATOR(S):
Izzet Safer

SPONSORING AGENCIES:
Natural Science Engineering Research Council

PROJECT TITLE:
Evaluating user interfaces for identifying task boundaries

CERTIFICATE EXPIRY DATE: March 2, 2008

DOCUMENTS INCLUDED IN THIS APPROVAL:		DATE APPROVED: March 2, 2007
Document Name	**Version**	**Date**
Consent Forms:		
main study consent	4	February 26, 2007
Advertisements:		
call for study	4	February 26, 2007
Questionnaire, Questionnaire Cover Letter, Tests:		
questionnaire	3	February 7, 2007

The application for ethical review and the document(s) listed above have been reviewed and the procedures were found to be acceptable on ethical grounds for research involving human subjects.

Approval is issued on behalf of the Behavioural Research Ethics Board and signed electronically by one of the following:

Dr. Peter Suedfeld, Chair
Dr. Jim Rupert, Associate Chair
Dr. Arminee Kazanjian, Associate Chair
Dr. M. Judith Lynam, Associate Chair