

# Using Visual Momentum to Explain Disorientation in the Eclipse IDE

Brian de Alwis and Gail C. Murphy

Dept of Computer Science, University of British Columbia

E-mail: {bsd, murphy}@cs.ubc.ca

## Abstract

*We report on a field study about how software developers experience disorientation when using the Eclipse Java integrated development environment. We analyzed the data using the theory of visual momentum, identifying three factors that may lead to disorientation: the absence of connecting navigation context during program exploration, thrashing between displays to view necessary pieces of code, and the pursuit of sometimes unrelated subtasks.*

## 1. Introduction

Several software developers have informally reported to us occasions of becoming *lost* or *disoriented* when exploring a system's source code. The disorientation occurs when developers lose the context or relevancy of their recent actions to their overall goal.

The research literature includes only passing references to disorientation during software development tasks (e.g., [1, 3, 14]), usually describing it either as a 'loss of context' or 'getting lost.' Several tools appeal to preservation of context to argue the effectiveness of their techniques [6, 9, 13], although their evidence provides only indirect substantiation of this claim. The few studies that focus on a developer's interactions with their development tools have considered relative novices working on small programs [11], analyzed developers' tool usage patterns [12], and assessed developers' high-level satisfaction with several tools [10]. We have not found any studies that investigate reported areas of dissatisfaction. Thus, despite acknowledgement of the problem, there are as yet no studies reported which investigate how and when disorientation occurs for software developers.

To help address this gap, we conducted an exploratory field study. We observed and interviewed eight developers from the Eclipse Project as they conducted their normal development work using the Eclipse Java Development Tooling, a popular state-of-the-practice open-source integrated development environment (IDE) for Java. We also interviewed two other IBM developers who used Eclipse, but who were working on a closed-source system. These expert

developers were working on their normal tasks on a large software system.

Our qualitative study sought to answer two questions: (Q1) Do expert developers become disoriented? Or is disorientation only a phenomenon suffered by novices or newcomers? (Q2) Are there factors of the development environment that exacerbate or prevent disorientation, or aid in recovery from disorientation? Half of the developers observed did become disoriented during the study, and follow-up interviews revealed that eight of the ten developers had become disoriented on previous occasions. We examined patterns of complaints about the Eclipse user interface (UI) by using *visual momentum* [16] to identify factors that contribute to the disorientation. Our analysis supports previous recommendations in the literature for designing IDEs, and suggests the usefulness of visual momentum in analyzing the effects of UIs.

Although we did not find disorientation to be a frequently occurring phenomenon, we believe it can have significant impact on a developer because of the time and effort required to recover from disorientation. Understanding the causes of disorientation during development tasks, and how it can be prevented, may improve the effectiveness of software developers.

### 1.1. Visual Momentum

Software development tasks generally require correlating information gleaned from different *displays*, referring to content displayed on a part of the computer screen at some particular time. To explain our study observations, we use a heuristic measure for evaluating the ability of a user interface to alleviate or prevent disorientation, named *visual momentum* [16]. This concept was inspired by techniques used in cinematography and synthesized from findings and guidelines identified by cognitive psychology and interface design.

Visual momentum is a qualitative measure of a user's ability to extract relevant information across displays. Interfaces with low momentum are essentially serial displays, where each display is perceptually independent of both its prior and subsequent displays, thus requiring the user to carry the mental burden of transitioning and reorienting be-

tween each display, and possibly leading to disorientation. Interfaces with high visual momentum aid in coordinating the information displayed; users are able to focus on their task and are not concerned with managing the interface. Watts-Perotti and Woods [15] suggest a number of techniques to improve visual momentum.

## 2. Study format

The field study was conducted at IBM Canada's Ottawa Software Laboratory in the two weeks prior to the M8 milestone release of Eclipse 3.0. Of the ten participants, eight were active developers of the Eclipse Platform (referred to as P1–P8), and the other two were former Eclipse Platform developers (referred to as E1,E2) now developing a closed-source system. All participants normally use Eclipse for Java development. Although the participants worked on the Eclipse Platform, none were from the teams responsible for developing the actual Eclipse Java Development Tooling. We chose this population to eliminate the possibility of disorientation occurring because of novice effects (Q1). We chose to examine Eclipse because of its popularity and because the Eclipse Project had previously undertaken an effort to address suspected factors relating to complaints of disorientation [2].

We observed P1–P8 separately for two hours each as they pursued their normal development tasks, collecting video tapes, screen captures, and detailed notes. We were unable to observe E1,E2 at work due to confidentiality concerns. We also conducted semi-structured interviews: a brief interview was conducted after one hour to ask about the developer's progress and current approach, and a longer interview took place at the end of the session to obtain more detail. This study design allowed us to stress realism, an important consideration as we assumed disorientation was more likely to occur as developers work on larger systems in their actual work environment.

During the interviews participants were asked about their program navigation approach, tools used, difficulties they encountered, and specifics of their chosen change tasks. Only towards the conclusion of the final interview were the participants asked outright if they had experienced any episodes of lostness during the session, and for specific details. All interviews were taped and subsequently transcribed word for word, including features such as pauses and inflection.

Our analysis in brief involved analyzing the interview transcripts and field notes to identify difficulties described in using the Eclipse user interface, correlating them with the screen captures. The screen captures were analyzed to gather information on how developers use features of their development tools or other applications, including counting the number of transitions between Eclipse windows and other applications such as e-mail readers or web browsers, as well as the number of times where there was a total re-

placement of content—where information in a window became completely obscured. We then used visual momentum to assess the situations around the occurrences of disorientation.

## 3. Overview of study results

The developers observed chose what they believed would be relatively small to medium tasks, able to be completed in a day: none were observed to have modified more than 25 files or classes during the study period. The first three rows of Table 1 describe the tasks performed: whether they were working on code they had written, the number of files modified during the study, and whether they had been interrupted by others during the study.

During the study, four of the eight developers either reported (P2,P8) or were observed (P4,P6) becoming disoriented. The follow-up interviews revealed that eight of the ten developers had experienced disorientation on previous occasions. The fourth and fifth row of Table 1 summarize which subjects experienced disorientation. A value of *rep* in the row labelled '*Disoriented in study?*' that disorientation was self-reported during the interview; a value of *obs* indicates that evidence of disorientation was observed by the researcher. The following row, labelled '*Disoriented previously?*', refers to a developer self-reporting having experienced disorientation in their past.

We asked each developer whether he felt supported by Eclipse, and whether he customized Eclipse's appearance beyond setting predefined preferences. Only four of the developers customize Eclipse's appearance in terms of tool layouts (e.g., an Eclipse view or plug-in). Most of the developers felt that the tools generally provided just the right amount of information (P1,P2,P5,P7,P8) – or even too much (E1, who used a minimal setup so as to avoid distractions). Most developers ran Eclipse as a single window maximized to occupy the full screen. The final row describes the amount of *thrashing* observed; this is described in more detail in Section 4.2.

Developers also spent time on other related tasks during the study period, such as responding to queries from users and other developers via e-mail, instant messaging, visits by other developers, and triaging and responding to problem reports. These other tasks were often interleaved during down-time, such as occurred during builds, waiting for Eclipse to start, or when fetching and committing files to or from the version control system. Although not formally verified, many of the e-mails received seemed to pertain to e-mails generated in response to changes to problem reports (P1,P3).

## 4. Analysis of study results

Eclipse provides many features that support high visual momentum, such as overlays of related information (e.g.,

**Table 1. General summary of observations and responses.**

	P2	P8	P6	P4	P1	P3	P5	P7	E1	E2
<b>Modifying own code?</b>	75%	50%	no	yes	yes	yes	yes	yes	shared	shared
<b># Files modified</b>	5	8	25	6	1	13	2	2	–	–
<b>Interrupted?</b>	yes	yes	yes	yes	yes	no	no	no	–	–
<b>Disoriented in study?</b>	rep	rep	obs	obs	–	–	–	–	N/A	N/A
<b>Disoriented previously?</b>	yes	yes	yes	yes	yes	yes	no	no	yes	yes
<b>Feels supported by Eclipse?</b>	no	yes	yes	no	yes	yes	yes	yes	yes	yes
<b>Customizes Eclipse</b>	yes	yes	no	no	no	yes	no	no	yes	no
<b>Runs Eclipse full-screen</b>	no	yes	yes	no	yes	no	yes	no	no	–
<b>Thrashing</b>	high	high	high	med	low	med	med	med	–	–

problem markers), bookmarks, and overview bars. From our observation and comments in interviews, these non-obscuring information presentation techniques were used and appreciated by developers. However, through our analysis, we identified three factors contributing to the Eclipse UI becoming a set of serial displays, which may lead to developer disorientation.

#### 4.1. Absence of connecting navigation context

The first factor identified pertains to a lack of connecting context when switching between files. This problem occurred most often during program exploration, as developers follow program relationships to understand or assess parts of the source code.

Eclipse supports many sophisticated program navigation traversals such as finding all callers of a method, all references to a field, or the declaration of a class. Many such functions are tied to hot-keys, meaning that they can be invoked with little effort, enabling rapid descents through a call chain. All developers that we observed made use of these program navigation operations.

These rapid descents led to a flurry of different files being examined, causing many interface changes as the different views recontextualize themselves to the changing files. There is rarely any visible connection as to how the developer came to the current file, and the developer must instead remember the connections or rebuild them, which can be difficult. Thus each file switch is *perceptually independent* from the previous: this is what is meant by low visual momentum. Eclipse’s editor tabs are inadequate for re-orienting as they are not maintained in most recently used order.

These issues explain the behaviour of P5 and P6 observed during a long period of source exploration. Both would periodically stop to close unimportant files, ensuring their editor tabs contained only those files relevant to their task.

#### 4.2. Thrashing to view necessary context

The second factor identified pertains to an inability to simultaneously view all the information necessary for a task.

Developers complained about not being able to see more of the surrounding context of the source code being viewed, and P2,P3,E1 customized their Eclipse window layouts to increase available screen space for source code. But given the sizes of these large systems, developers are effectively limited examining the system as a whole through a limited *keyhole* view [15]. As a result, developers must frequently switch between displays, maintaining relevant information in their working memory. P2,P4,P8 were observed to repeatedly scroll and jump both between and within files to correlate content from several windows or files. This frequent switching indicates low visual momentum.

We use the term *thrashing* [7] to refer to frequent and recurring switching. We assigned a subjective rating of the thrashing exhibited by each participant (Table 1), with *low* indicating little to no thrashing, *med* for some thrashing, and *high* for wild thrashing.<sup>1</sup> Note that developers who exhibited high degrees of thrashing also reported or were observed being disoriented. The switching generally involved switching between files in Eclipse, or between Eclipse and either their problem reporting system or their e-mail application to record information. The thrashing generally occurred when they appeared to forget information to be copied across, and had to return to the original source.

One potential way to avoid thrashing is to use more than one window. P4 was the only developer observed to use more than one Eclipse window, but his two windows were nearly completely overlapping and were used instead to

<sup>1</sup>This subjective measure was assessed after counting the number of transitions between Eclipse editors and other applications, as well as the number of times where there was a total replacement of content. We did not incorporate a measure of time nor did we take into account the context to eliminate false positives. For example, some developers would process e-mail while waiting for a regression test pass to complete; although this would technically result in total or near-total replacement, it was not a genuine switch as the old information was not being carried forward.

manage different aspects of his work, rather than provide complementary information. This developer still exhibited a medium degree of thrashing as a result.

### 4.3. Pursuit of digressions

The third factor we identified involved the absence of support for managing developers' switches in work. These switches occur at a lower-level than those between working spheres as noted by [5].

We found developers often managing several tasks at any particular moment. Some tasks are suspended to pursue subtasks, called an *embedded digression* [4]. These digressions are usually related to the task: for example, attending a problem report may require some exploration to view the implications of a change. Some digressions may become more substantial or spawn further digressions: we observed P1 and P8 fixing or diagnosing bugs, and what were expected to be five minute jobs took more than an hour and a half.

Developers were observed to pursue embedded digressions without recording the task they had suspended. Because Eclipse had no knowledge of this digression, the developers had to remember to resume their suspended task, which is known to be unreliable [8]. If the environment had high visual momentum, the displays associated with a common task would be perceptually tied and displays related to other tasks would appear perceptually independent, such that a developer will be able to discern whether a particular display pertains to their current subtask or original task.

## 5. Conclusions

This paper provides an initial examination of the phenomenon of disorientation in software development, and confirms that disorientation affects expert developers (Q1). Using visual momentum, we have identified three factors in the Eclipse UI that we believe lead to the environment becoming a series of perceptually independent displays that contribute to inducing disorientation (Q2):

1. the absence of connecting navigation context during program exploration,
2. thrashing between displays to view necessary pieces of code, and
3. the pursuit of sometimes unrelated subtasks.

Although the occurrences described in this paper are specific to Eclipse, we believe an understanding of how disorientation occurs can lead to better ways to manage or prevent the disorientation across a variety of development tools.

**Acknowledgments.** Thanks are owed to our study participants, and to Elisa Baniassad for helpful comments on an earlier version of this paper. This research has been supported by NSERC and IBM.

## References

- [1] R. K. Bellamy and J. M. Carroll. Restructuring the programmer's task. *Int. J. Man-Mach. St.*, 37:503–527, 1992.
- [2] T. Creasy. Request for comments: Loss of context. Online document: <http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/platform-ui-home/loss-of-context/Proposal.html>, Nov. 2001.
- [3] R. DeLine, M. Czerwinski, and G. Robertson. Easing program comprehension by sharing navigation data. In *Proc. IEEE Symp. on Visual Lang. and Human-Centric Comput. (VLHCC)*, pages 241–248, Sept. 2005.
- [4] C. L. Foss. Tools for reading and browsing hypertext. *Inform. Process. and Manag.*, 25(4):407–418, 1989.
- [5] V. M. González and G. Mark. “Constant, constant, multi-tasking craziness”: managing multiple working spheres. In *Proc. Conf. on Human Factors in Computing Systems (CHI)*, pages 113–120, Apr. 2004.
- [6] W. G. Griswold, J. J. Yuan, and Y. Kato. Exploiting the map metaphor in a tool for software evolution. In *Proc. Int. Conf. Softw. Eng. (ICSE)*, pages 265–274, Mar. 2001.
- [7] D. Henderson Jr., Austin and S. K. Card. Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Trans. Gr.*, 5(3):211–241, 1986.
- [8] D. Herrmann, B. Brubaker, C. Yoder, V. Sheets, and A. Tio. Devices that remind. In F. T. Durso et al., editors, *Handbook of Applied Cognition*, pages 377–407. Wiley, 1999.
- [9] D. Janzen and K. De Volder. Navigating and querying code without getting lost. In *Proc. Conf. Aspect-Oriented Softw. Dev. (AOSD)*, pages 178–187, 2003.
- [10] R. B. Kline and A. Seffah. Evaluation of integrated software development environments: Challenges and results from three empirical studies. *Int. J. Hum.-Comput. St.*, 63(6):607–627, Dec. 2005.
- [11] A. J. Ko, H. H. Aung, and B. A. Myers. Eliciting design requirements for maintenance-oriented IDEs: A detailed study of corrective and perfective maintenance tasks. In *Proc. Int. Conf. Softw. Eng. (ICSE)*, pages 126–135, 2005.
- [12] G. C. Murphy, M. Kersten, and L. Findlater. How are Java software developers using the Eclipse IDE? *IEEE Software*, (to appear), July/Aug. 2006.
- [13] M.-A. D. Storey, F. D. Fracchia, and H. A. Müller. Cognitive design elements to support the construction of a mental model during software visualization. *J. Softw. & Systems*, 44(3):171–185, 1999.
- [14] O. Turetken, D. Schuff, R. Sharda, and T. T. Ow. Supporting systems analysis and design through fish-eye views. *Commun. ACM*, 47(9):72–77, Sept. 2004.
- [15] J. Watts-Perotti and D. D. Woods. How experienced users avoid getting lost in large display networks. *Int. J. Hum.-Comput. Int.*, 11(4):269–299, 1999.
- [16] D. D. Woods. Visual momentum: A concept to improve the cognitive coupling of person and computer. *Int. J. Man-Mach. St.*, 21:229–244, 1984.