Developing and Evaluating the Code Bubbles Metaphor

Andrew Bragdon
Brown University, Department of Computer Science
acb@cs.brown.edu

ABSTRACT

Today's integrated development environments (IDEs) are hampered by their dependence on files and file-based editing. A novel user interface that is based on collections of lightweight editable fragments, called bubbles, which when grouped together form concurrently visible working sets is proposed. An overview of this interface, as well as a summary of the results of a quantitative and a qualitative evaluation of the interface is presented.

1. INTRODUCTION

Programmers spend between 60-90% of their time reading and navigating code and other data sources [1]. Programmers form working sets of one or more fragments corresponding to places of interest [2]; with larger code bases, these fragments are scattered across multiple methods in multiple classes – forming a working set, comprising the context of an activity. Viewing these fragments concurrently is likely to be beneficial, as it has been shown that concurrent views should be used for tasks in which visual comparisons must be made between parts that have greater complexity than can be held in limited working memory [3].

Because contemporary integrated development environments (IDEs) are file-based it is difficult to create and maintain a view in which multiple fragments are visible concurrently. This requires the programmer to manually and repeatedly perform numerous interactions to place, resize, scroll, and reflow a different file window for each fragment. Instead, IDEs are optimized for switching among different views using tabs, forward/back buttons, etc. Perhaps as a result, programmers may spend on average 35% of their time in IDEs actively navigating among working set fragments [2], since they can only easily see one or two fragments at a time.

I argue in favor of a new approach, where the IDE shows multiple editable fragments concurrently, letting the user see and work with complete working sets. The result reduces navigations and supports new higher-level interactions over and within the working set. The approach is founded on the metaphor of a *bubble* – a fully editable and interactive view of a fragment such as a function, method documentation, or debugging display. Bubbles, in contrast to windows, have minimal border decoration, avoid clipping their contents by using automatic code reflow and elision, and do not overlap but instead push each other out of the way. Bubbles exist in a large virtual space where a cluster of bubbles comprises a concurrently visible working set. Code Bubbles and two accompanying user studies are fully described in [4] [5].

2. RELATED WORK

The work closest to the bubbles approach let the programmer work in terms of program fragments. These efforts let the programmer edit in terms of individual functions, or similar units. This was the approach taken in Desert [6] and also in IBM's Visual Age environments [7] and in the Sheets environment [8]. All these were loosely based on non-file based programming languages such as Xerox's Smalltalk and its successors, various versions of Lisp, and visual languages such as NI's LabView. JASPER displays small read-only views that represent the user's current task as a means for navigation [9]. A number of tools have been developed to add navigation aids to file-based environments, e.g. Mylar [10]; these tools focus on identifying working sets, whereas this work focuses on displaying working sets concurrently.

ICSE '10, May 2-8, 2010, Cape Town, South Africa

3. FORMATIVE STUDY

Early in the design process, I sought to determine whether users perceived value in a fragments-based approach to reading and editing code, and to gain qualitative feedback into using existing approaches – tiled panes in Eclipse 3.4.2. and overlapping child windows in Visual Studio 2008's multiple document interface (MDI) mode – for seeing methods side-by-side. Five professional developers in the Providence, RI area were recruited via web ads.

Overall, developers thought it was very helpful to have multiple methods side-by-side, but felt it was prohibitively difficult to achieve this result for more than 2-3 functions using tab panes or MDI. They wanted the system to help reduce or eliminate the tedious and repetitive operations needed to create such working sets. They did not like having to manually drag, scroll, rearrange, apply code formatting commands, and resize panes/windows, with one developer comparing it to a "jigsaw puzzle", and although they liked the free-form layout of MDI they did not like having to manage which window was on top (Z-order) which was necessary when using overlapping windows. We further observed in both cases that once the screen had been filled, it became difficult for developers to continue adding additional content, as this necessitated tab switching within specific panes, or changing Z-order with MDI.

4. THE BUBBLES METAPHOR

Based on the formative study, four significant problems with current interfaces that make it difficult to create side-by-side views of code were identified, that a novel interface would need to address. Steps taken to address these issues are summarized from [4]:

File-based views are often large, requiring multiple interaction steps to concisely display a single method: A popup search box opens a relevant method in a bubble in a single step.

Code contains significant white space and does not readily fit in a compact view: To ensure that code can be easily read and edited regardless of the dimensions of its bubble, bubbles never clip text horizontally, but instead automatically reflow long lines, similar to the way a programmer would manually wrap long lines. To handle long functions, vertical elision is used to collapse blocks.

To further conserve space and reduce distraction, bubbles have minimal chrome/border decoration; instead, programmers interact using the right, middle and left buttons respectively to move, close or edit text within bubbles. A breadcrumb bar indicates the parent package and class, and can also be used to access peer methods.

Modifying a layout of panes or windows takes multiple interaction steps: Bubbles do not overlap but instead push each other out of the way, making groups of bubbles easier to read since no Z-order management is needed. When one bubble is moved on top of another, a bubble spacer automatically moves the overlapped bubbles out of the way using a simple, recursive, heuristic algorithm that minimizes the total movement of bubbles.

Window layouts are generally limited by the size of the screen: Bubbles exist in a large, continuously pannable 2-D virtual space.

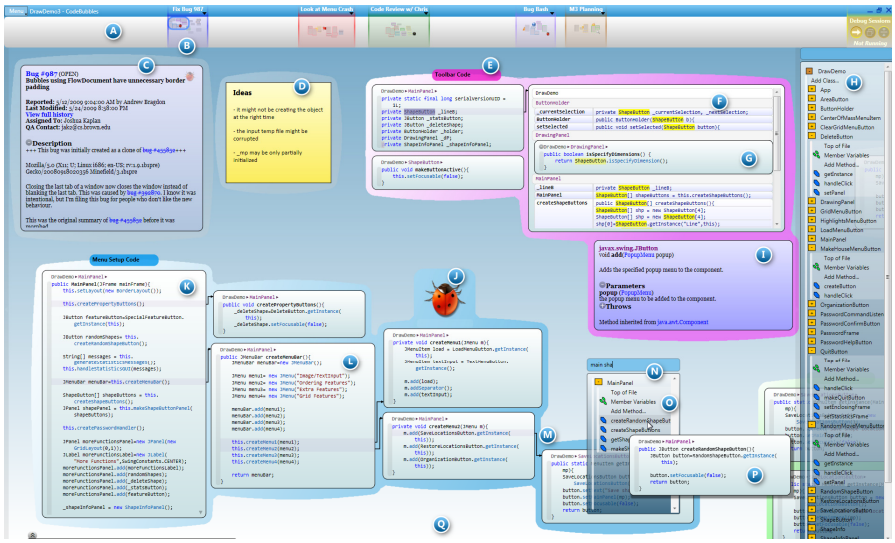

Figure 1.

The Code Bubbles IDE. Resolution: 1920x1200 (space reserved for taskbar).

(A) The workspace bar used for navigating the workspace, (B) a user-defined task section in the workspace, (C) a bug bubble, (D) a note bubble, (E) a named group of bubbles, (F) a bubble stack showing the results of a Find All References search with (G) a result expanded as a bubble, (H) the package explorer, (I) a Javadoc bubble, (J) a visual flag bubble, (K) a code bubble showing a function, with long lines reflowed, (L) opening the definition of a function call, (M) a bubble connection, (N) the popup search box with (O) a result highlighted and (P) hover-preview of that function, (Q) the 2-D workspace.

Building on this core design, many extensions were made to build the Code Bubbles IDE interface around the concept of working sets [4] [5] (see Figure 1), including: supporting additional tasks such as editing, performing reference searches, building heterogeneous working sets with different bubble types, lightweight persistent bubble groups, supporting interruption recovery and multitasking through a workspace bar, a breakpoint debugger based on bubbles, a channels interface for working with multiple debugging sessions simultaneously, and tools for annotating and sharing working sets.

5. QUANTITATIVE EVALUATION

To evaluate the performance of the Code Bubbles user interface for reading code, we compared it with the Eclipse IDE [4]. 20 3rd and 4th year undergraduate as well as graduate students were recruited from Brown University's computer science program (which uses Eclipse and Java in the majority of classes) to participate. Participants were randomly assigned to Eclipse or Code Bubbles conditions, between-subjects. After an introduction to the respective system, participants were given a warm-up task (not counted), and two code understanding tasks. Participants were given up to 45 minutes to complete each task, and for each task were asked to fix a bug by reading and understanding the code (they were not permitted to use a debugger or trace statements).

Code Bubbles users saw a significantly lower overall task completion time than Eclipse users (33.2%), and successfully completed significantly more tasks. In addition, Code Bubbles users performed significantly fewer navigations per minute (46.6%), significantly fewer repeated navigations (50.5%), and spent significantly less time navigating (68.6%). Surprisingly, the reduction in navigation time, Δt_{nav} accounted for only one third of the total reduction in task completion time Δt, experienced by Code Bubbles users; I hypothesize that the remaining time, Δt_{cog}, which accounted for the bulk of the improved performance, can be attributed to the limited nature of human working memory, and further that Code Bubbles users were able to offload their limited working memory onto working sets in Code Bubbles. This may have helped them perform several important activities more easily, including: remembering context, comparing and referring back to methods, and re-finding methods when needed – above and beyond reducing navigations.

6. QUALITATIVE EVALUATION

A qualitative study of the overall system was also conducted with 23 professional developers, recruited via Facebook.com ads, with an average of approximately 10 years of industry experience [5]. Developers were introduced to the system and asked to think aloud as they completed six development tasks.

On the whole, developers expressed a high level of interest, excitement and a range of ideas on how they might use the system. This was a surprising result, given the limitations of the prototype, the radical change from what developers are used to, and the level of experience of the participants. We believe this indicates that developers perceive significant value in a working set-based user interface paradigm for IDEs. The use of a working set-based user interface paradigm in Code Bubbles appears to have changed the cost structure of using working sets to aid in completing tasks; developers did not have to explicitly create a working set from scratch to use one, rather they "get it for free" as part of their normal workflow. As a result, annotation tools such as groups, flags, notes, connections, etc. are always available, making them something developers could count on regardless of task. This change in cost structure appeared to allow working sets to be employed more often, which has the potential to benefit a wide range of development tasks.

7. ACKNOWLEDGMENTS

The author wishes to thank Andries van Dam, Steven P. Reiss and Ken Hinckley for their advice and insight. This material is based upon work supported under a NSF Graduate Research Fellowship.

8. REFERENCES

[1] Erlikh, L. Leveraging Legacy System Dollars for E-Business. *IT Pro*, May/June (2000), 17-23.

[2] Ko, A. J., Myers, B. A. et al. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. *IEEE TSE*, 32, 12 (December 2006), 971-987.

[3] Plumlee, M. D. and Ware, C. Zooming versus multiple window interfaces: Cognitive costs of visual comparisons. *ACM Transactions on Computer-Human Interaction*, 13, 2 (June 2006), 179-209.

[4] Bragdon, A., Zeleznik, R., Reiss, S. P. et al. Code Bubbles: A Working Set-based Interface for Code Understanding and Maintenance. In *Proc. of CHI 2010*.

[5] Bragdon, A., Reiss, S. P., Zeleznik, R. et al. Code Bubbles: Rethinking the User Interface Paradigm of Integrated Development Environments. In *Proceedings of ICSE 2010*.

[6] Reiss, Steven P. The Desert environment. *ToSEM*, 8, 4 (1999), 297-342.

[7] Nackman, L. R. An overview of Montana. *IBM Research* (1996).

[8] Stockton, R. and Kramer, N. *The Sheets hypercode editor*. 1993.

[9] Coblenz, M., Ko, A., and Myers, B. JASPER: an Eclipse plug-in to facilitate software maintenance tasks. In *OOPSLA Workshop on Eclipse Technology* (2006), 65-69.

[10] Kersten, M. and Murphy, G. C. Mylar: a degree-of-interest model for IDEs. In *AOSD '05* (2005), 1590168.