Code Bubbles: A Practical Working-Set Programming Environment

Steven P. Reiss
Department of Computer Science
Brown University
Providence, RI 02912
spr@cs.brown.edu

Jared N. Bott
Department of EECS
University of Central Florida
Orlando, FL, 32816
jbott@cs.ucf.edu

Joseph J. LaViola Jr.
Department of EECS
University of Central Florida
Orlando, FL, 32816
jjl@eecs.ucf.edu

***Abstract*—Our original work on the Code Bubbles environment demonstrated that a working-set based framework for software development showed promise. We have spent the past several years extending the underlying concepts into a fully-functional system. In our demonstration, we will show the current Code Bubbles environment for Java, how it works, how it can be used, and why we prefer it over more traditional programming environments. We will also show how we have extended the framework to enhance software development tasks such as complex debugging, testing, and collaboration. This paper describes the features we will demonstrate.**

***Keywords*-integrated development environments; working sets; debugging; collaborative tools**

I. Background

The Code Bubbles environment is an attempt to transform the user interface of an integrated development environment so that it more closely conforms to the way that programmers work. It features a working-set-oriented approach where programmers can display all the information needed for their current task on a single screen.

We first demonstrated the concepts behind Code Bubbles two years ago and showed through user studies that the underlying ideas had a strong potential [1], [2]. Current development environments are file-oriented and somewhat inefficient to use, requiring a significant amount of navigation and forcing programmers to maintain the current context in their heads. Code Bubbles simplifies this by displaying smaller, logical chunks of the program that are more relevant to the task at hand, by letting the programmer organize and display a large number of such "bubbles" simultaneously, and by providing facilities for saving and restoring the resultant contexts. Code bubbles makes the programmer more efficient and provides a more intuitive approach to software development.

We have been extending these concepts into a complete and practical environment. The current environment supports all phases of large scale Java programming including development, maintenance, exploration, debugging, testing, and collaboration. Moreover, the environment is designed to be able to run in the "cloud", with most of the tools running on a remote server and the front end running locally. The environment is currently being used for real software development projects including itself, and by students in our courses. The environment is available for download and is open source. http://www.cs.brown.edu/people/spr/codebubbles/ provides more details; a video can be seen at http://www.eecs.ucf.edu/isuelab/videos/codebubbles.mp4.

Our demonstration will show how Code Bubbles can and is being used to make the programmer more efficient. This paper describes the main aspects of the system including advanced features such as debugging, testing, providing context, and collaboration.

II. Basic Functionality

Programmers normally use the Code Bubbles environment by finding the set of methods, classes, fields, documentation, notes, errors, bug reports, etc. that are required for their current task and laying them out on the screen as shown in Figure 1. Each of these elements of a working set is displayed in its own bubble. Bubbles are organized on the screen so that they do not overlap. Arrows between bubbles show creation or programmer-specified relationships. Bubbles are colored either according to their type (e.g., note bubbles are yellow), or according to their package for source bubbles. All source and note bubbles are editable, and, where bubbles are duplicated or represent overlapping sources (e.g. a class bubble and a bubble for a method of that class), changes in one are immediately reflected in the other. The colors have been chosen to be non-obtrusive while still providing the programmer with important information.

The environment provides a variety of ways of finding and creating bubbles. These include a powerful search facility, the ability to find all definitions (including inherited or virtual ones) or references to an identifier, and implicit links from items such as error messages and test cases. Bubbles are easily moved around the screen and resized as needed. Reflow and elision are used to reduce the amount of screen space required for each bubble. Bubbles with a common purpose can be grouped, with the group being shown as a colored outline, and grouped bubbles can be manipulated as a set.

The visible screen is actually a small portion of a much larger canvas the programmer can use and navigate over. An overview of the canvas is shown at the top of the window.

978-1-4673-1067-3/12/$31.00 © 2012 IEEE

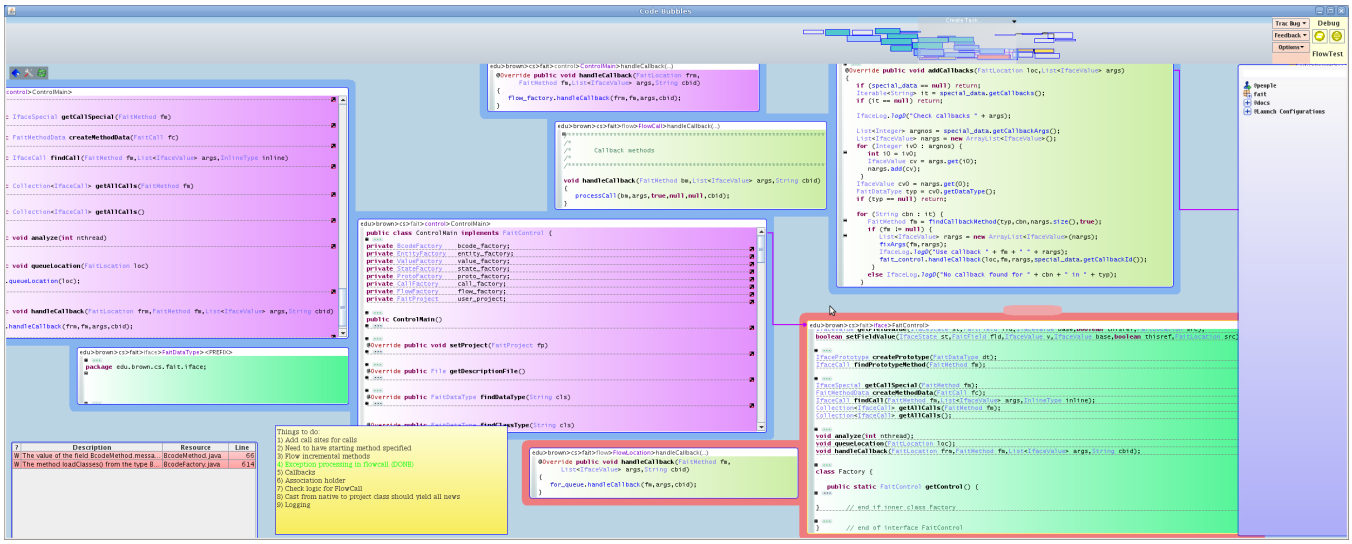

Figure 1. A view of the Code Bubbles environment showing a variety of bubbles in the current working set. These bubbles include source code bubbles, notes bubbles, documentation bubbles, and an error message bubble. The view shows eight visible code bubbles, a fraction of the thirty-six bubbles that were opened so far in the session and can be seen in the overview panel at the top of the display.

The canvas can be used to handle multiple tasks and manage interruptions, letting the programmer move to a new context as needed and then return to their original context. Bubbles can either be fixed on the screen or part of this canvas.

In developing a practical implementation of the code bubbles concept, we have had to deal with a number of issues. One was ensuring that performance of the system meets the user's expectations, especially for editing and screen management. Another was defining what UNDO meant in the bubbles environment so that it met the user's expectations that each bubble is an independent editor while maintaining consistency. A third was providing natural ways for creating new packages, classes, fields, and methods.

An issue that came up as the environment was used more was the complexity of the bubble displays created by the programmer. The example in Figure 1 shows the set of bubbles from about an hour of work; not only is the display filled with bubbles, but the overview shows the large set of bubbles that were opened and used to get to the current display. With this many bubbles on the display, it is important to ensure that the programmer retains control over the layout and organization of the bubbles. We have introduced a number of mechanisms including group-aware positioning of new bubbles, providing a short (2 second) interval for manual placement of new bubbles, and automatic cleanup of the bubble layout.

III. Debugging

Code Bubbles provides a complete debugging environment designed to handle complex multithreaded programs. Debugging is done in separate contexts, and multiple debugging contexts can be used simultaneously. An example of a debugging context is shown in Figure 2.

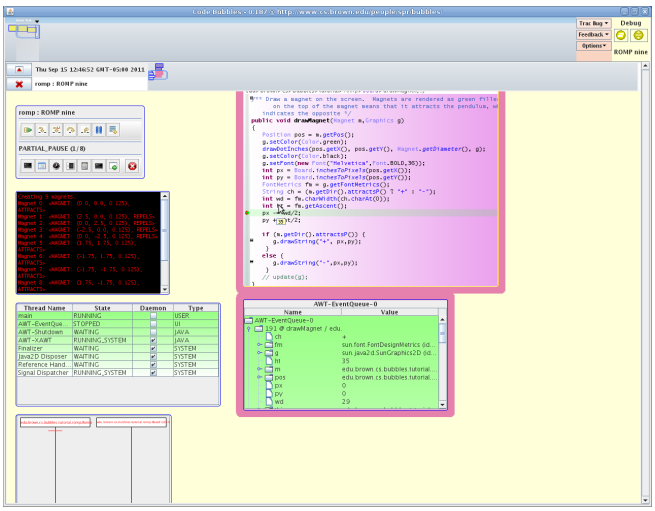

Figure 2. A view of the debugging view in the Code Bubbles environment showing control panel, a console bubble, a stack bubble, and a debugger history bubble on the left, and the current source and stack bubbles on the right.

The bubbles environment provides standard debugging capabilities such as breakpoints, stepping, update and continue, and value displays. Bubbles are used to show the current location and the current stack. Separate bubbles are used for controlling the debugging session, the console, and showing the state of threads. When the user steps into a new method, new bubbles for the source and values are created to the

right of the original bubbles. Where separate threads are involved, the corresponding bubbles are displayed vertically. A heuristic layout manager determines where new source and values bubbles should be placed, when bubbles on the screen should be reused, and when bubbles can be removed.

Value bubbles can show the whole stack, a single stack frame, a single variable, or a component of a variable such as a field. These bubbles can be live, showing the current value and updating as the values change, or frozen, recording the current value so the programmer can refer to it later in the debugging session.

Debugging in the environment includes a feature we call "active debugging" where part of the debugger executes in the user's application as a separate thread. This is used to provide the programmer with additional information including performance analysis, instant deadlock detection, detailed thread state monitoring, Java Swing hierarchy information, and a history of the last few seconds of execution just before a breakpoint was reached.

Microsoft has developed an alternative implementation of the Code Bubbles paradigm for debugging in the Debugger Canvas extension to Visual Studio [3].

IV. Providing Context

Programmers are used to working with files and they serve a useful purpose in programming. Files provide context and organization of the code. They are used by a variety of tools and as the basis for reading code. The organization of the system in terms of files and their contents is important to the programmer and the system. Our implementation of Code Bubbles, while providing a method-centric approach, attempts to provide the appropriate context in a variety of ways. These are shown in Figure 3.

One approach we use is to provide higher-level bubbles with appropriate elision facilities. This includes bubbles for a whole class and bubbles for a whole file. A class bubble is shown in the lower right of the figure. A second approach, shown in the lower left, provides a file overview compactly summarizing the content of the file. This overview can be used to create bubbles, browse the file, or even reorganize the file. The bubble in the upper left of the figure shows the file locations of bubbles in the current context. Each bubble is highlighted to show what file it comes from and where. The bubble with the current focus is highlighted in red. Finally, the package viewer, shown in the upper right of the figure, provides a flexible view of the static structure of the system at the package, class, or method level. The user can specify what types of items to display, and what relationships are relevant (ranging from class hierarchy to calls and field references).

V. Collaboration

Code Bubbles provides support for collaborative development including both general facilities for collaboration and

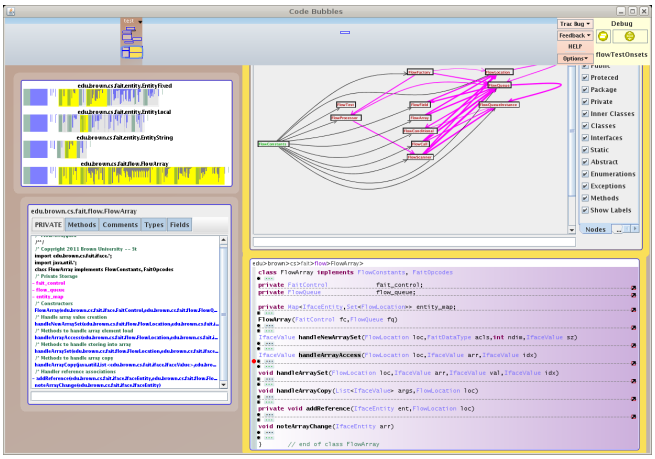

Figure 3. Context display bubbles. A class bubble is shown in the lower right while a file overview is shown in the lower left. In the top left, the file context provides a pictoral view showing where the current codebubbles are located in the file structure; the top right contains a graphical package viewer.

facilities for sharing source and running the environment in the cloud.

Code Bubbles is implemented as a front end that talks to Eclipse through a socket-based message bus. Eclipse is used to provide debugging, parsing, source access, project management, and similar facilities. The message bus is also used as a means for coordinated editing. Multiple Code Bubbles front ends can talk to the same instance of Eclipse. This lets multiple programmers work on the same source simultaneously, with edits showing up in any common bubbles on all displays as they are made.

In addition, the environment provides support for programmer communication. This includes chat bubbles that support most of the common chat protocols and chat history bubbles that retain a permanent record of such conversations. It supports sharing the set of bubbles in the current working set through either email or as part of a chat conversation. In addition, for pair programming, it supports shared working sets with immediate update as bubbles are moved, created and resized.

The environment also supports a programmer's log. This log provides an annotated and searchable history of the development of the system. The log is designed to be as easy as possible to use by the programmer. Programmers are prompted to select or define a task when they start a new working set. They can hit the F2 key at any time to add notes, annotations, screen shots, or other documents to the log. Logs can be shared among programmers on a team. In addition, the system automatically records significant events such as which methods are viewed and which methods are edited as part of a task. The log is viewed through a query interface where the programmer can specify what methods,

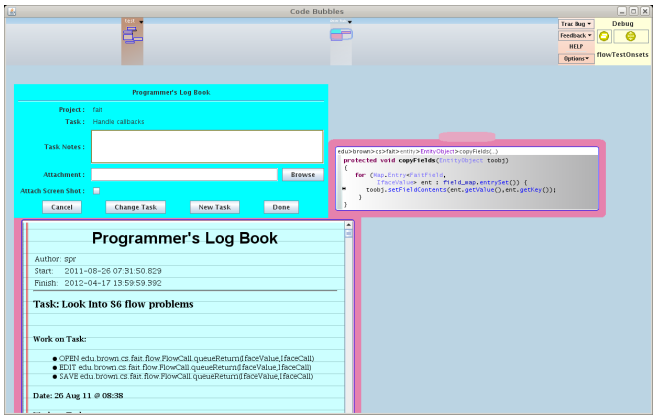

Figure 4. Programmer's Log interaction and display bubbles. This log provides an annotated and searchable history of the development of the system. The interaction bubble at the top left lets the programmer add annotations, attachments, and screen shots to the log.

classes, users, or tasks they are interested in and what types of information they want to see. An example of a log and input window are shown in Figure 4.

The programmer's log is similar to existing frameworks such as Mylyn [4], Syde [5], or SpyWare [6]. Ours has the potential to provide more useful information since we know more details such as the particular functions that are looked at, since we have a built in notion of a working set based on location on the overall canvas that lets us disambiguate among multiple tasks, and since we can provide additional information such as a screenshot showing the relationships between the relevant working set components.

Code Bubbles can be run in client-server mode, with a server instance of the environment controlling Eclipse and handling common operations such as file access, and a client instance handling the actual programmer interaction. In this mode, Eclipse and the server can be run on one machine, say in the cloud, and the client can be run on the user's desktop. This mode is further supported by an extension of the message bus that provides encrypted communication through an Internet proxy.

VI. Testing

Code Bubbles also provides facilities for JUnit-based testing. It runs a separate process (in the cloud if appropriate) that automatically finds all the tests in the current project. Tests are then run with a harness that records block, branch and call coverage information.

This testing process uses the message bus to monitor when functions are edited and when they are successfully compiled. This lets it determine when tests are out of date and when they need to be rerun. It can be configured to either run tests automatically as in [7] or on demand. A test management bubble shows the status of all tests and lets the user get test information, view the test source, or create a debugging session for a given test.

VII. Conclusion

Our demonstration of Code Bubbles will show how the environment can improve programmer efficiency while performing a variety of programming tasks. We will run through a real-life scenario on a non-trivial system, and will illustrate both the basic operation of the environment and many of the features described above. We encourage readers to try the environment for themselves by downloading it from our web site.

Acknowledgment

This work has been supported by the National Science Foundation grant CCF1130822. Additional support has come from Microsoft and Google.

References

[1] A. Bragdon, R. Zeleznik, S. P. Reiss, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. LaViola, Jr., "Code bubbles: a working set-based interface for code understanding and maintenance," in *Proceedings of the 28th international conference on Human factors in computing systems*, 2010, pp. 2503–2512.

[2] A. Bragdon, S. P. Reiss, R. Zeleznik, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. LaViola, Jr., "Code bubbles: rethinking the user interface paradigm of integrated development environments," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, 2010, pp. 455–464.

[3] R. DeLine, A. Bragdon, K. Rowan, J. Jacobsen, and S. P. Reiss, "Debugger canvas: industrial experience with the code bubbles paradigm," in *Proceedings of the 34nd ACM/IEEE International Conference on Software Engineering*. New York, NY, USA: ACM, 2012.

[4] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, 2006, pp. 1–11.

[5] L. Hattori and M. Lanza, "An environment for synchronous software development." in *ICSE Companion*, 2009, pp. 223–226.

[6] R. Robbes and M. Lanza, "Spyware: a change-aware development toolset," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 847–850.

[7] D. Saff and M. D. Ernst, "An experimental evaluation of continuous testing during development," in *Proceedings of ISSTA 2004*, 2004, pp. 76–85.