Tool Demonstration: The Visualizations of Code Bubbles

Steven P. Reiss and Alexander Tarvo
Department of Computer Science
Brown University
Providence, RI. 02912
{spr,alexta}@cs.brown.edu

Abstract—**Code Bubbles is an integrated development environment that concentrates on the user experience. The environment is very visual and includes a number of different visualizations, both static and dynamic. We will demonstrate the environment and the various visualizations on a realistic scenario based on our current work.**

I. Code Bubbles

Code Bubbles is an integrated development environment for Java built on the notion of working sets rather than files [1,2]. Code Bubbles lets the user create relevant working sets composed on code fragments, typically methods, small classes, notes, documentation, etc. Each of these is displayed in a separate bubble or lightweight window. The user can rearrange the bubbles as needed to provide a logical context for the particular maintenance or development task they are currently working on.

Code Bubbles provides a wide range of features to support this style of development. To support interruptions and multiple tasks, it provides a large overview space in which the user can embed multiple working sets. Moreover, working sets can be saved and restored for later use. To support working set creation, the environment provides a number of navigation aids, making it easy to bring up a called method, all uses of a method or variable, or the type of an item; and easy to browse or search for elements by name or name fragment.

Code Bubbles also supports collaborative development, allowing users to share working sets and to have multiple users working simultaneously on the same code base. It supports cloud-based development by allowing the environment back end to run in the cloud while the front end runs on the user's machine [5].

Code Bubbles provides a range of debugging features. In addition to the normal notions of breakpoints and stepping, Code Bubbles lets the user see multiple debugging sessions in parallel (including previous ones), and provides bubbles for each stack level.

II. Code Bubbles Visualizations

Code Bubbles is an inherently visual environment. The basic environment provides a visualization of the user's code though code bubbles that can be organized in various ways, that are linked to each other, and that are highlighted by package and class.

Since the initial development and deployment of Code Bubbles, we have worked on a variety of additional visualizations that have been designed to enhance the programming experience. These include visualizations that provide context, visualizations that show static software structures, visualizations of file histories, and visualizations that support debugging and understanding the dynamics of the program.

III. Using Code Bubbles

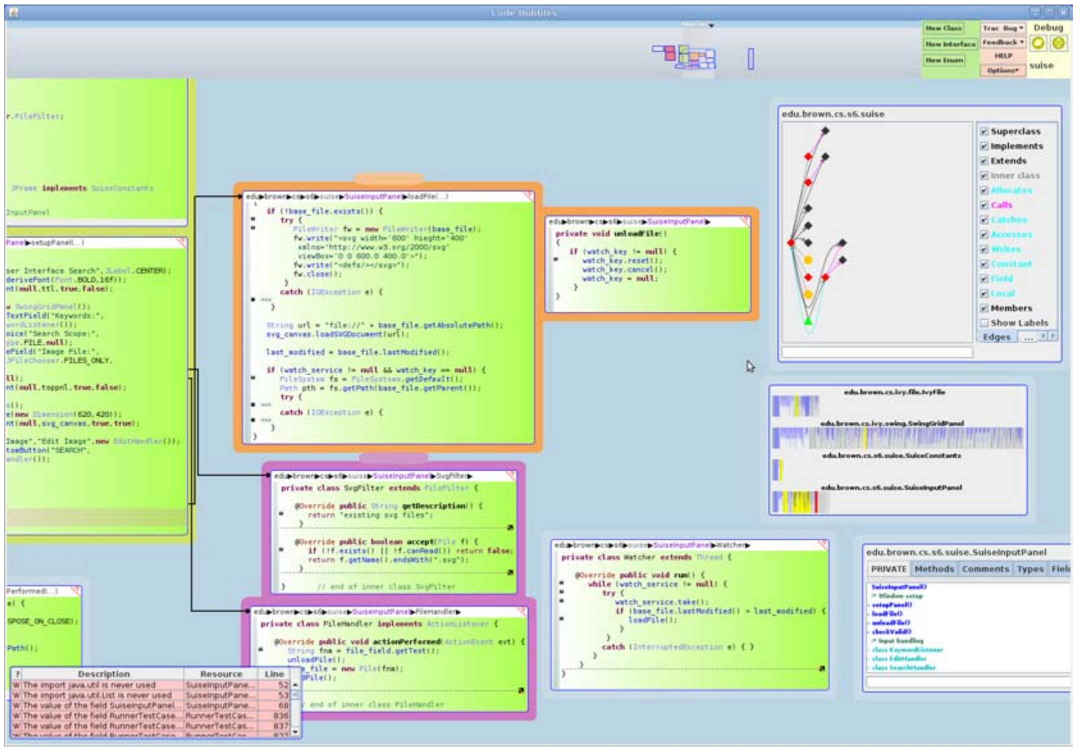
We have been using Code Bubbles for its own development as well as the development of a number of other projects. Recently, for example, we have been working on extending the S^6 search engine [4] to handle searching for user interfaces.

Figure 1 shows Code Bubbles as we are developing code for S^6. This figure shows the basic visualization of code including code bubbles, links between the bubbles, highlighting by package, and a compact representation of a file that includes key comments to provide context for that file.

In the middle right of the figure is a visualization of the context of the current working set. This view shows a SeeSoft [3] like view that indicates what files are represented in the working set, showing the bubbles from those files in yellow and the currently focused bubble in red. Panning over the view will show the actual code.

In the upper right of the figure is a visualization of the static structure of the software system. Here we have chosen to display the structure of the particular package we are working on. The visualization is hierarchical and designed to be used with large systems. Different color nodes represent packages, classes, interfaces, enumerations, exceptions, and methods. The user can choose which types of nodes to display and can dynamically expand and collapse nodes. Arcs represent the different relationships between nodes. Again, the user can determine which types of relationships should be displayed. The graph can also be localized to the subgraph induced by a particular node. This visualization is designed to address specific questions about software structure that might not be obvious from the code.

Complex systems such as S^6 use version management systems and record their history. Code Bubbles provides access to such systems. For example, Figure 2 shows how two Code Bubbles versioning features. The file history view on the left illustrates the history of each line of the selected file. Time runs from right to left, with the current code being displayed on the left as a SeeSoft-style view with tool tips to show the actual code. Times where the file was changed are shown with a saturated color. Color represents the different authors of this file. The graph at the top



```
edu▶brown▶cs▶s6▶suise▶SuiseInputPanel▶loadFile(...)
   if (!base_file.exists()) {
      try {
         FileWriter fw = new FileWriter(base_file);
         fw.write("<svg width='600' hieght='400'
          xmlns='http://www.w3.org/2000/svg'
          viewBox='0 0 600.0 400.0'>");
         fw.write("<defs/></svg>");
         fw.close();
       }
      catch (IOException e) {
 ...
    }

   String url = "file://" + base_file.getAbsolutePath();
   svg_canvas.loadSVGDocument(url);

   last_modified = base_file.lastModified();

   if (watch_service != null && watch_key == null) {
      FileSystem fs = FileSystems.getDefault();
      Path pth = fs.getPath(base_file.getParent());
      try {
 ...
      catch (IOException e) {
 ...
    }
}
```

```
edu▶brown▶cs▶s6▶suise▶SuiseInputPanel▶
private void unloadFile()
{
   if (watch_key != null) {
      watch_key.reset();
      watch_key.cancel();
      watch_key = null;
    }
}
```

```
edu▶brown▶cs▶s6▶suise▶SuiseInputPanel▶SvgFilter▶
private class SvgFilter extends FileFilter {

   @Override public String getDescription() {
      return "existing svg files";
    }

   @Override public boolean accept(File f) {
      if (!f.exists() || !f.canRead()) return false;
      return f.getName().endsWith(".svg");
    }

}        // end of inner class SvgFilter
```

```
edu▶brown▶cs▶s6▶suise▶SuiseInputPanel▶FileHandler▶
private class FileHandler implements ActionListener {

   @Override public void actionPerformed(ActionEvent evt) {
      String fnm = file_field.getText();
      unloadFile();
      _file = new File(fnm);
     dFile();

   / end of inner class FileHandler
```

```
edu▶brown▶cs▶s6▶suise▶SuiseInputPanel▶Watcher▶
private class Watcher extends Thread {

   @Override public void run() {
      while (watch_service != null) {
         try {
            watch_service.take();
            if (base_file.lastModified() > last_modified) {
               loadFile();
             }
          }
         catch (InterruptedException e) { }
       }
    }
}
```

?	Description	Resource	Line
W	The import java.util is never used	SuiseInputPane...	52
W	The import java.util.List is never used	SuiseInputPane...	53
W	The value of the field SuiseInputPanel...	SuiseInputPane...	68
W	The value of the field RunnerTestCase...	RunnerTestCas...	836
W	The value of the field RunnerTestCase...	RunnerTestCas...	837

FIGURE 1. Code Bubbles being used for code development. The bubbles (views) on the left represent code fragments, either methods or classes. Links between the methods show relationships that were used to bring up the additional bubbles. The bubble at the bottom left lists the errors and warnings for the code and is updated as the user types. The bar at the top shows the overall context where the user can set up multiple working sets as needed. The three bubbles on the right show some of the static visualizations provided by the system. The top view shows the static structure of the package we are working on. The middle view shows the files that are visible in bubbles, highlighting the bubbled areas and the current bubble. The bottom view shows a summary of a particular file including comments.

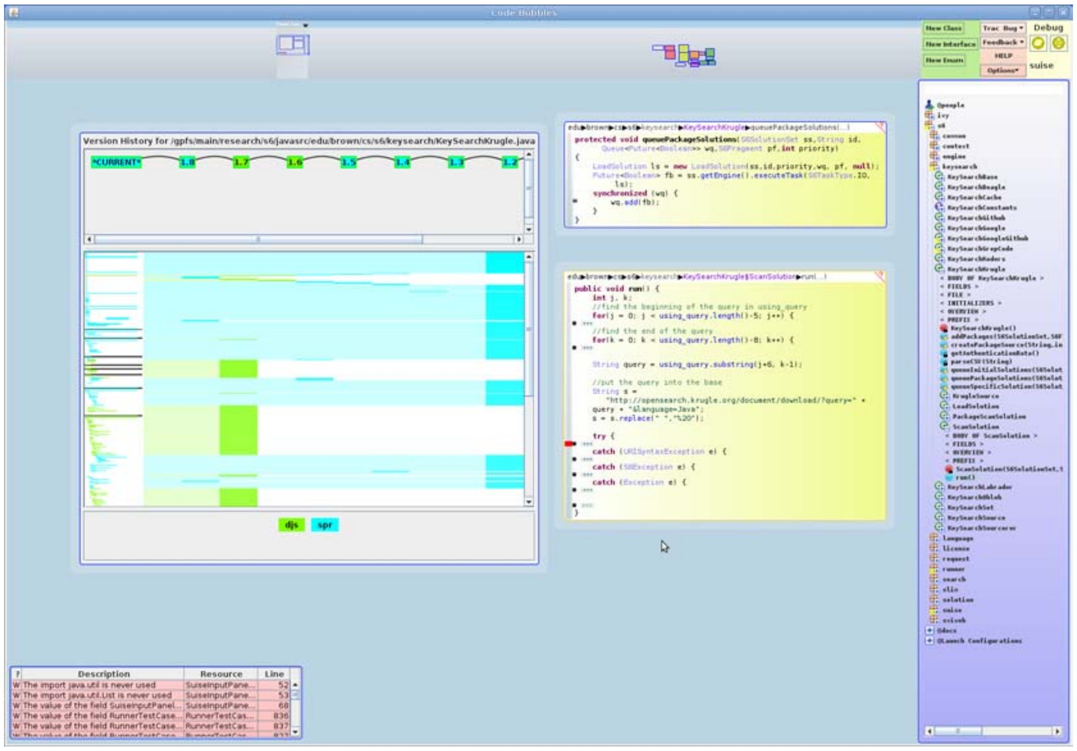
illustrates the complete version history of the file, while the view at the bottom show the different authors. Another feature illustrated in this figures is that the system can determine what other programmers are working on in terms of the project and can highlight any changed lines with appropriate annotations.

Program development requires support both for static code development and for debugging. Hence Code Bubbles provides an appropriate visual environment for debugging. An example of debugging part of S^6 can be seen in Figure 3. This view illustrates how Code Bubbles lays out the current context using a combination of stack and code bubbles. It also shows a visualization of the history of the debugging session based on UML sequence diagrams in the lower left. This view is interactive in that the user can click on it to see the code and values at that point in the execution history. The figure also contains a performance visualization, here represented as a sortable table, that is updated dynamically as the program runs.

Code Bubbles provides additional debugging visualizations as well. On the left side of Figure 4 is the Java/Swing interactor. This tool can show the call stack for the instantiation of each widget in the user's application and can show how each pixel of the output was drawn.

The right side of Figure 4 shows another debugging visualization. This view, which is generated completely automatically and is updated dynamically as the program runs, shows what each thread is working on over time. It automatically determines the relevant transactions and tasks in the program through a combination of static and dynamic analysis. In the visualization, time runs along the X-axis, and each thread is represented as a row. The transaction and task the thread is working on are shown as a pipe within that row, color coded by the task and transaction type. For example, the visualization in the figure shows an example of S6 in action. The red on the left indicates 32 threads obtaining results from the underlying search engine. The green that follows is the transformation phase, run in 6 threads. Finally, the orange on the right is the testing phase, again run in 6 threads. One can see immediately from the diagram that testing is the most expensive part of the process.

Availability. Code Bubbles is available both as open source through SourceForge, and as a runnable and automatically updated binary. For more information see http://www.cs.brown.edu/people/spr/codebubbles.

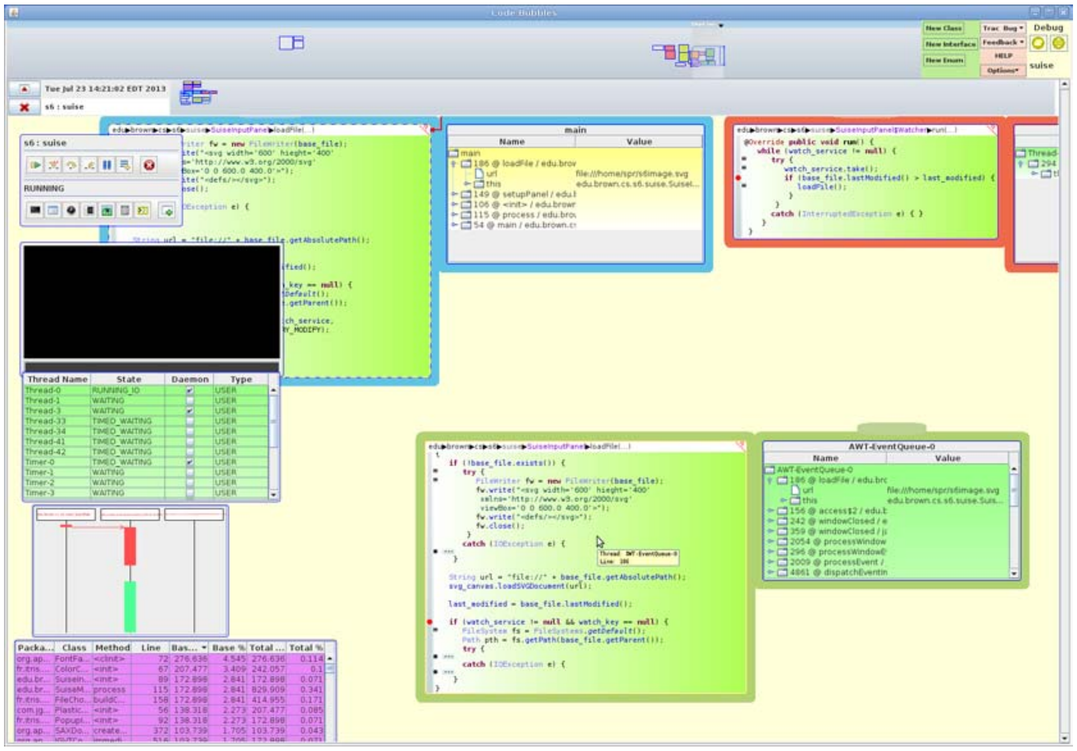


```
edu▶brown▶cs▶s6▶keysearch▶KeySearchKrugle▶queuePackageSolutions(...)
protected void queuePackageSolutions(S6SolutionSet ss,String id,
      Queue<Future<Boolean>> wq,S6Fragment pf,int priority)
{
   LoadSolution ls = new LoadSolution(ss,id,priority,wq, pf, null);
   Future<Boolean> fb = ss.getEngine().executeTask(S6TaskType.IO,
         ls);
   synchronized (wq) {
      wq.add(fb);
   }
}
```

```
edu▶brown▶cs▶s6▶keysearch▶KeySearchKrugle$ScanSolution▶run(...)
public void run() {
   int j, k;
   //find the beginning of the query in using_query
   for(j = 0; j < using_query.length()-5; j++) {
 ...
   //find the end of the query
   for(k = 0; k < using_query.length()-8; k++) {
 ...

   String query = using_query.substring(j+6, k-1);

   //put the query into the base
   String s =
      "http://opensearch.krugle.org/document/download/?query=" +
   query + "&language=Java";
   s = s.replace(" ","%20");

   try {
 ...
   catch (URISyntaxException e) {
 ...
   catch (S6Exception e) {
 ...
   catch (Exception e) {
 ...

 ...
}
```

?	Description	Resource	Line
W	The import java.util is never used	SuiseInputPane...	52
W	The import java.util.List is never used	SuiseInputPane...	53
W	The value of the field SuiseInputPanel...	SuiseInputPane...	68
W	The value of the field RunnerTestCase...	RunnerTestCas...	836
W	The value of the field RunnerTestCase...	RunnerTestCas...	837

FIGURE 2. The Code Bubble visualization showing file history. The bubble on the left shows the history of each line of a source file, colored by author.



main	
Name	Value
main	
186 @ loadFile / edu.brov	
url	file:///home/spr/s6image.svg
this	edu.brown.cs.s6.suise.Suisel...
149 @ setupPanel / edu.b	
106 @ <init> / edu.brow	
115 @ process / edu.brov	
54 @ main / edu.brown.c	

```
edu▶brown▶cs▶s6▶suise▶SuiseInputPanel$Watcher▶run(...)
@Override public void run() {
   while (watch_service != null) {
      try {
         watch_service.take();
         if (base_file.lastModified() > last_modified) {
            loadFile();
         }
      }
      catch (InterruptedException e) { }
   }
}
```

Thread Name	State	Daemon	Type
Thread-0	RUNNING_IO	✓	USER
Thread-1	WAITING		USER
Thread-3	WAITING	✓	USER
Thread-33	TIMED_WAITING		USER
Thread-34	TIMED_WAITING		USER
Thread-41	TIMED_WAITING		USER
Thread-42	TIMED_WAITING		USER
Timer-0	TIMED_WAITING	✓	USER
Timer-1	WAITING		USER
Timer-2	WAITING		USER
Timer-3	WAITING		USER

```
edu▶brown▶cs▶s6▶suise▶SuiseInputPanel▶loadFile(...)
{
   if (!base_file.exists()) {
      try {
         FileWriter fw = new FileWriter(base_file);
         fw.write("<svg width='600' hieght='400'
          xmlns='http://www.w3.org/2000/svg'
          viewBox='0 0 600.0 400.0'>");
         fw.write("<defs/></svg>");
         fw.close();
      }
      catch (IOException e) {
 ...
    }

   String url = "file://" + base_file.getAbsolutePath();
   svg_canvas.loadSVGDocument(url);

   last_modified = base_file.lastModified();

   if (watch_service != null && watch_key == null) {
      FileSystem fs = FileSystems.getDefault();
      Path pth = fs.getPath(base_file.getParent());
      try {
 ...
      catch (IOException e) {
 ...
    }
}
```

AWT-EventQueue-0	
Name	Value
AWT-EventQueue-0	
186 @ loadFile / edu.brc	
url	file:///home/spr/s6image.svg
this	edu.brown.cs.s6.suise.Suis...
156 @ access$2 / edu.t	
242 @ windowClosed / e	
359 @ windowClosed / j	
2054 @ processWindow	
296 @ processWindow	
2009 @ processEvent /	
4861 @ dispatchEventIn	

Packa...	Class	Method	Line	Bas...	Base %	Total ...	Total %
org.ap...	FontFa...	<clinit>	72	276.636	4.545	276.636	0.114
fr.itns....	ColorC...	<init>	67	207.477	3.409	242.057	0.1
edu.br...	Suisein...	<init>	89	172.898	2.841	172.898	0.071
edu.br...	SuiseM...	process	115	172.898	2.841	829.909	0.341
fr.itns....	FileCho...	buildC...	158	172.898	2.841	414.955	0.171
com.jg...	Plastic...	<init>	56	138.318	2.273	207.477	0.085
fr.itns....	Popupl...	<init>	92	138.318	2.273	172.898	0.071
org.ap...	SAXDo...	create...	372	103.739	1.705	103.739	0.043

FIGURE 3. Code Bubbles debugging view include different code-stack views, a debugging history view and a performance visualization.

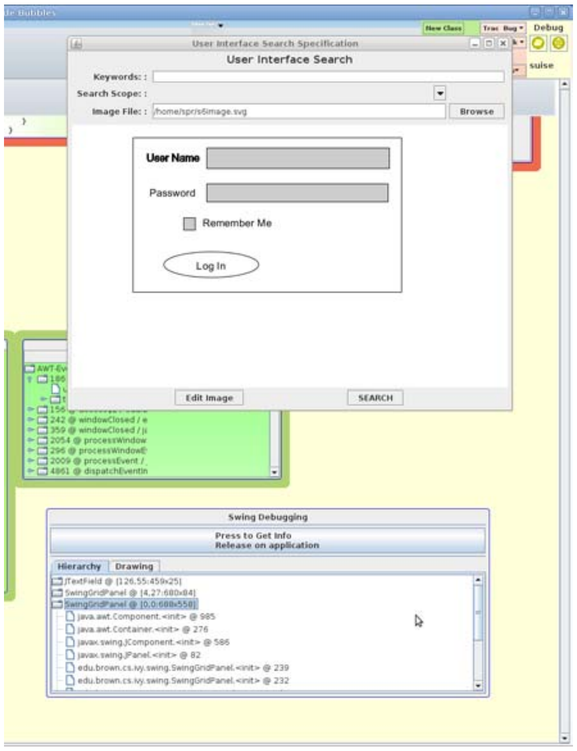


Image File: : /home/spr/s6image.svg

```
AWT-Ev
242 @ windowClosed / e
359 @ windowClosed / j
2054 @ processWindow
296 @ processWindowE
2009 @ processEvent /
4861 @ dispatchEventIn
```

```
JTextField @ [126,55:459x25]
SwingGridPanel @ [4,27:680x84]
SwingGridPanel @ [0,0:688x558]
    java.awt.Component.<init> @ 985
    java.awt.Container.<init> @ 276
    javax.swing.JComponent.<init> @ 586
    javax.swing.JPanel.<init> @ 82
    edu.brown.cs.ivy.swing.SwingGridPanel.<init> @ 239
    edu.brown.cs.ivy.swing.SwingGridPanel.<init> @ 232
```

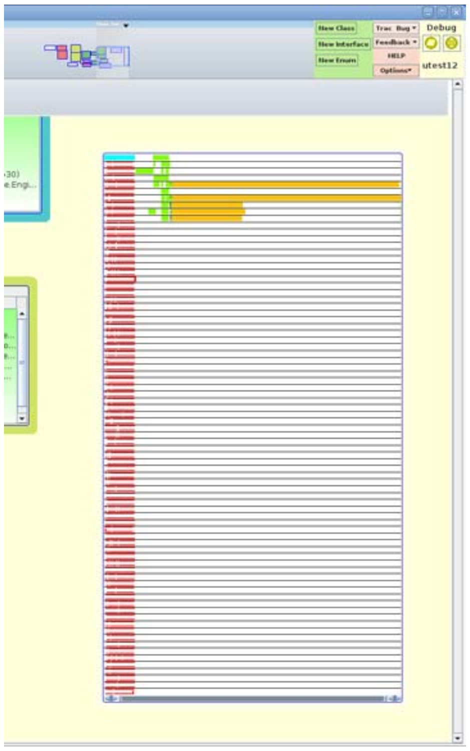


FIGURE 4. The Code Bubbles Swing interactor, shown on the right with an application interface on top, can display the dynamic call stack for the creation of each widget and the drawing operations for any particular pixel of the program's output. The Code Bubbles Thread-Transaction-Task visualization of S^6 running on a simple example is shown on the right. This view shows the different phases of the program and how they are divided among the 32 threads. Color indicates the different tasks at each time.

Acknowledgements. This work was done with support from the National Science Foundation through grants CCR-1012056 and support from Microsoft and Google.

IV. References

1. Andrew Bragdon, Steven P. Reiss, Robert Zeleznik, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J. LaViola, Jr., "Code bubbles: rethinking the user interface paradigm of integrated development environments," *ACM/IEEE International Conference on Software Engineering 2010*, pp. 455-464 (2010).

2. Andrew Bragdon, Steven P. Reiss, Robert Zeleznik, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J. LaViola, Jr., "Code bubbles: a working set-based interface for code understanding and maintenance," *Proceedings SIGCHI Conference on Human Factors in Computing Systems*, pp. 2503-2512 (2010).

3. Stephen G. Eick, Joseph L. Steffen, and Eric E. Sumner, Jr., "Seesoft - a tool for visualizing software," AT&T Bell Laboratories (1991).

4. Steven P. Reiss, "Semantics-based code search," *International Conference on Software Engineering 2009*, pp. 243-253 (May 2009).

5. Steven P. Reiss, "Plugging in and into Code Bubbles," *Proceedings Workshop on Developing Tools as Plug-ins 2012*, pp. 55-60 (June 2012).