Research in Coding and IDEs

Jan-Peter Krämer
Media Computing Group
RWTH Aachen University

http://hci.rwth-aachen.de/cthci
Status Quo
Time in Software Development

[LaToza2006, Maintaining mental models: a study of developer work habits]
Software is complex and hard to understand.
Task context

- What is relevant information?
- What strategies are applied to find information?
Models for Developer Strategies

[Ko2006, An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks]

31 Professional Java Developers

5 Maintenance tasks
(3 Bugs, 2 Enhancements)

500 SLOC Java Paint Application
Models for Developer Strategies

[Ko2006, An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks]
Models for Developer Strategies

[Sillito2008, Asking and Answering Questions during a Programming Change Task]

9 experienced developers (pair programming)

1 of 5 maintenance tasks per session

ArgoUML
60k SLOC

16 developers from industry

Real world change task

Real world source code
Models for Developer Strategies

[Sillito2008, Asking and Answering Questions during a Programming Change Task]

- Finding focus points
- Expanding focus points
- Understanding a subgraph
- Questions over groups of subgraphs
package org.jhotdraw.contrib;

import org.jhotdraw.framework.DrawingView;

/**
 * DesktopEvent is used to alert the Desktop of a change in the Drawing in the Desktop.
 * 
 * @author C.L.Gilbert <dnobe@users.sourceforge.net>
 * @version $CURRENT_VERSIONS$
 */

public class DesktopEvent extends EventObject {
    private DrawingView myDrawingView;
    
    /**
     * Some events require the previous DrawingView (e.g. when a new DrawingView
     * is selected).
     */
    private DrawingView myPreviousDrawingView;

    public DesktopEvent(Desktop newSource, DrawingView newDrawingView) {
        super(newSource, newDrawingView, null);
        this(newSource, newDrawingView, null);
    }

    public DesktopEvent(Desktop newSource, DrawingView newDrawingView, DrawingView newPreviousDV) {
        super(newSource);
        setDrawingView(newDrawingView);
        setPreviousDrawingView(newPreviousDV);
    }

    private void setDrawingView(DrawingView newDrawingView) {
        myDrawingView = newDrawingView;
    }

    public DrawingView getDrawingView() {
        return myDrawingView;
    }

    private void setPreviousDrawingView(DrawingView newPreviousDV) {
        myPreviousDrawingView = newPreviousDV;
    }

    public DrawingView getPreviousDrawingView() {
        return myPreviousDrawingView;
    }
}
Tools Used in Eclipse

[Murphy2006, How Are Java Software Developers Using the Eclipse IDE?]

Figure 5. Use of navigation views by all 41 developers (nobody used the Declaration view).

Table 3

Top 10 commands executed across all 41 developers

<table>
<thead>
<tr>
<th>Command Identifier</th>
<th>Use (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>14.3</td>
</tr>
<tr>
<td>Save</td>
<td>11.3</td>
</tr>
<tr>
<td>Next word</td>
<td>7.3</td>
</tr>
<tr>
<td>Paste</td>
<td>6.8</td>
</tr>
<tr>
<td>Content assist</td>
<td>6.7</td>
</tr>
<tr>
<td>Previous word</td>
<td>5.9</td>
</tr>
<tr>
<td>Copy</td>
<td>4.6</td>
</tr>
<tr>
<td>Select previous word</td>
<td>3.4</td>
</tr>
<tr>
<td>Step (debug)</td>
<td>3.2</td>
</tr>
</tbody>
</table>

Table 4

Nonlocal navigation commands available in the JDT, their key bindings on the Windows platform, how many of the developers used the commands, and each command's rank (a rank of one indicates the command that the developers used most; the lowest rank is 1,142—the number of commands). This data shows that the command used most often is opening a selected element's declaration (a rank of 21); the command used by the largest number of users is the search for references in a workspace.
Easing Access to Task Context
Looking into this now, see also related bug 124224.
Recommender Tools

[Singer2005, NavTracks: supporting navigation in software maintenance]
[DeLine2005, Easing program comprehension by sharing navigation data]
[Čubranic´2005, Hipikat: recommending pertinent software development artifacts]

- Calculate a Degree of Interest for source code elements based on:
  - reading history
  - editing history
  - history of other team members
  - information from version control systems

- Remaining Problems:
  - Still only text-based visualization
  - Recommendations for irrelevant code are still irrelevant
```xml
<?xml version="1.0" encoding="utf-8"?>

  <mx:Script>
    <![CDATA[
      public function loadData():void {
        URLLoader
      }
    ]]>
  </mx:Script>
</mx:Application>
```

[Brandt2010, Example-centric programming: integrating web search into the development environment]
Changing the Presentation


To navigate using the CT Scrollbar, a developer can either use the scrollbar at left in the usual way, or she can click on a location in the thumbnail to jump to the corresponding place in the code. Whenever the mouse cursor is inside the thumbnail area, labels appear showing the names of likely navigation targets, specifically the names of second-level items with no children (e.g., enums) and third-level items (fields and methods) as shown on the right side of Figure 1. In the current design, these pop-up labels occlude the code shape, which is an area for improvement.

The CT Desktop, shown in Figure 2, shows a thumbnail image of every source file in the project, arranged on a desktop surface. Each thumbnail has a label at the top, which shows the file name and serves as a handle for moving the thumbnail. A developer can arrange the thumbnails on the desktop as she sees fit. The code thumbnails are drawn exactly like those in the CT Scrollbar, except that the currently visible portion is drawn with a filled rectangle to make it more apparent. We use the same font size for all thumbnails on the desktop, which means that each thumbnail's height is proportional to its file's length. The document whose editor is active is highlighted with a thicker border than the others. Documents that are currently closed are shown with a grey background, grey title and no scroll area. As with the CT Scrollbar, moving the cursor over a thumbnail pops up target labels, and clicking on a thumbnail activates the document's editor and scrolls to the chosen part of the document. Clicking a thumbnail's title area activates the document's editor without scrolling the document. Double-clicking a grayed thumbnail opens the document and activates its editor.

When the programmer uses any of the standard search tools, the search results are highlighted in both the CT Scrollbar and CT Desktop. This makes it easy to see all search results at a glance.

Both the CT Scrollbar and Desktop are intended to allow the developer to form spatial memory of the code. The CT Scrollbar provides a stable, one-dimensional space per document, with visual landmarks to help the user distinguish different parts at a glance (namely, the code shape, the brackets and the target labels). The CT Desktop provides a stable, two-dimensional space of all the documents, again with visual landmarks (namely, the thumbnail landmarks, plus their placement). Our UI design choices were driven by our study goals. Specifically, we were interested in whether developers could form spatial memory of the code and how that would affect their navigation choices.
Bragdon2010, Code bubbles: a working set-based interface for code understanding and maintenance
Bragdon2010, Code bubbles: a working set-based interface for code understanding and maintenance
[Bragdon2010, Code bubbles: a working set-based interface for code understanding and maintenance]
Bragdon2010, Code bubbles: a working set-based interface for code understanding and maintenance
[Bragdon2010, Code bubbles: a working set-based interface for code understanding and maintenance]
Canvas Interfaces in the Wild

[DeLine2012, Debugger Canvas: Industrial experience with the code bubbles paradigm]

Figure 5: Number of unique downloads per week, after the initial release.
Utilizing the Call Graph
Reachability Questions
[LaToza2010, Developers ask reachability questions]

<table>
<thead>
<tr>
<th>Question</th>
<th>Find output of error text downstream from m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where is method ( m ) generating an error?</td>
<td></td>
</tr>
<tr>
<td>What resources are being acquired to cause this deadlock?</td>
<td>Find calls to methods acquiring the resources</td>
</tr>
<tr>
<td>What is this test doing which is different from what my app is doing?</td>
<td>Find differences between two call stacks</td>
</tr>
<tr>
<td>Is there another reason why this status is non-zero?</td>
<td>Find upstream statements that change the status</td>
</tr>
</tbody>
</table>
In practice: Feasible paths most interesting

[LaToza2010, Developers ask reachability questions]
Utilizing Call Graph Information

[LaToza2010, Searching Across Paths]
Static Analysis in the Wild

[Clang Static Analyzer, http://clang-analyzer.llvm.org/]
Call Hierarchy
Stacksplore

[Karrer2011, Stacksplore: Call Graph Navigation Helps Increasing Code Maintenance Efficiency]
Blaze

[Krämer2012, Blaze: Supporting Two-phased Call Graph Navigation in Source Code]
Analyzing Navigation Behavior
Information Foraging Theory

Predator

Scent

Prey
Information Foraging Theory

[Lawrance2010, Reactive information foraging for evolving goals]

Predator  Scent  Prey
<table>
<thead>
<tr>
<th></th>
<th>Xcode</th>
<th>Call Hierarchy</th>
<th>Stacksplorerer</th>
<th>Blaze</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find Change Location</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Side Effects of Change</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task Success</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task Completion Time</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

33 Developers

80,000 Lines of Code

[Krämer2013, How Tools in IDEs Shape Developers' Navigation Behavior]
Task Success

\[ p = 0.015 \]

- Xcode
- Call Hierarchy
- Stacksplorer
- Blaze

# of successful participants
Task Completion Time

p=0.022

<table>
<thead>
<tr>
<th>Tool</th>
<th>Total Time Required (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xcode</td>
<td>1900</td>
</tr>
<tr>
<td>Call Hierarchy</td>
<td>1425</td>
</tr>
<tr>
<td>Stacksporer</td>
<td>950</td>
</tr>
<tr>
<td>Blaze</td>
<td>475</td>
</tr>
</tbody>
</table>
Effectiveness

- Xcode
- Call Hierarchy
- Stacksplorer
- Blaze

Efficiency

- Xcode
- Call Hierarchy
- Stacksplorer
- Blaze

Why?

UI Differences

Navigation Behavior
[Fouse2011, ChronoViz: A system for supporting navigation of time-coded data]
Comparing Navigation Behavior
$I_1 = (p_{1,1}, ..., p_{640,480})$

$I_2 = (p_{1,1}, ..., p_{1024,768})$

1. Features
2. Transformations
H=(m_1, \ldots, m_i)

[Piorkowski2011, Modeling programmer navigation: A head-to-head empirical evaluation of predictive models]
A Predictor

[Piorkowski2011, Modeling programmer navigation: A head-to-head empirical evaluation of predictive models]

\[ H = (m_1, \ldots, m_i) \]  
\[ \text{Navigation History} \]  
\[ H = (a, b, a, d) \]

\[ M_i \]  
\[ \text{All methods known to developer at time } i \]  
\[ M_4 = \{a, b, d\} \]

\[ A_i: M_i - \{m_i\} \rightarrow \mathbb{R} \]  
\[ \text{Activation value for each method in } M_i \]  
\[ A_4(a) = 3 \]
\[ A_4(b) = 2 \]

\[ R_i: M_i - \{m_i\} \rightarrow \mathbb{N} \]  
\[ \text{Rank-transformed version of } A_i \]  
\[ R_4(a) = 1 \]
\[ R_4(b) = 2 \]

Result: \( N \) top-ranked methods
Prediction Accuracy

Xcode

Stacksplorer

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

Recency Frequency Working Set Bug Report Similarity Within File Distance

Fwd Call Depth Undirected Call Depth
Prediction Accuracy

Xcode

Stacksplorer

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

Fwd Call Depth

Undirected Call Depth

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

Undirected Call Depth

Xcode  Call Hierarchy  Stacksplorer  Blaze

p=0.002  p=0.001

p=0.003  p=0.001

p=0.003  p=0.001
Away from static analysis only
/ Introducing Codelets...
```
// tree

function drawTree () {
var blossomPoints = [];
    resetRandom();
    drawBranches(0, -Math.PI/2, canvasWidth/2, canvasHeight, 30);
    resetRandom();
    drawBlossoms(blossomPoints);
}

function drawBranches (i,angle,x,y,width,blossomPoints) {
    ctx.save();
    var length = tween(1, 1.6, 12, 3) + random(0.7, 1.3);
    if (i == 0) { length = 97; }
    ctx.translate(x,y);
    ctx.rotate(angle);
    ctx.fillStyle="#000000";
    ctx.fillRect(0,-width/2, length, width);
    ctx.restore();
    var tipX = x + (length - width/2) + Math.cos(angle);
    var tipY = y + (length - width/2) + Math.sin(angle);
    if (i > 4) { blossomPoints.push([x,y,tipX,tipY]); }
    else if (i < 6) {
        drawBranches(i + 1, angle + random(-0.15, -.05) + Math.PI/2, canvasWidth/2, canvasHeight, 30);
        drawBranches(i + 1, angle + random(0.15, .05) + Math.PI/2, canvasWidth/2, canvasHeight, 30);
    }
    else if (i < 12) {
        drawBranches(i + 1, angle + random(-0.15, -.05) + Math.PI/2, canvasWidth/2, canvasHeight, 30);
        drawBranches(i + 1, angle + random(0.15, .05) + Math.PI/2, canvasWidth/2, canvasHeight, 30);
    }
}
```
```javascript
/*jslint plusplus: true, node: true, white: true*/

var arr = [6, 2, 8, 4, 3];

var i;
for (i = 1; i < arr.length; i++) {
    var j = i;

    while ((j > 0) && (arr[j-1] > arr[j])) {
        var tmp = arr[j-1];
        arr[j-1] = arr[j];
        arr[j] = tmp;
    }
    j = j-1;
}

console.log(arr);
```

[6, 2, 8, 4, 3]

`undefined`

`1` truthy(true)

[6, 2, 8, 4, 3]

`1` truthy(true)

[2, 3, 4, 6, 8]
Live Coding Affects Coding Behavior

[Krämer2014, How Live Coding Affects Developers’ Coding Behavior]
more files should relate to success. A single file, so viewing fewer files should relate to success; with the nature of the two tasks: task 1 involved changes to group (t=2.2, df=18, p<.05). This discrepancy is consistent viewed significantly (t=22.6, df=18, p<0.0001), but both groups viewed similar condition. For task 1, Whyline participants viewed 1, for example, lists statistics about the number of code participants were viewing and for how long. Table both the keyboard and mouse, allowing us to see what lines capture data about source file views and navigations with participants explored. The tools were instrumented to It is also informative to consider the information that was probably too small to detect such differences).

speed. There was no relationship between industry groups experienced ceiling effects, causing no difference in from the bug). Because the task was more difficult, both participants also completed task 1 twice as fast (t=4.5, p<0.05). As seen in Table 1, this was usually achieved using significant difference (χ²=10.6, p<.05). Whyline 1 or 2 "why did" questions, almost exclusively about the creation of the list of objects (see Table 1), usually starting on the "MyClass" label, participants were more successful (χ²=15.76, p<0.0001, compared to only 3 control participants, a statistically significant difference (χ²=10.6, p<.05). Whyline 10 Whyline users succeeding, compared to 0 in the control. The speed and success results for task 1 are summarized in Table 1. Whyline participants relied mostly on questions, avoiding the UIs used to debug. As seen at the bottom of Table 1, more than the control group (t=4.6,df=18,p<.0002). For task 2, whyline participants asked to be notified of the tool's availability. The enthusiasm of participants was clearly evident and all asked for the tools to be usable for future work. The control group was less enthusiastic and had no problem with the tools. No participants had usability problems with the breakpoint features, likely due to our extensive 3-month period of user testing prior to the study.

Another telling difference in participants' performance were reliance more on text searches (which is to be expected [9]). The control group, despite using breakpoints, relied more on text searches for relevant content [9]. The control group, despite using breakpoints, and were far less successful. No participants had usability issues while using Whyline tools.

To assess the key function for each task. For task 1, Whyline and the low degree of success). For task 2, whyline participants asked to be notified of the tool's availability. The enthusiasm of participants was clearly evident and all asked for the tools to be usable for future work. The control group was less enthusiastic and had no problem with the tools. No participants had usability issues while using Whyline tools.

For task 1, the number of successful participants was probably too small to detect such differences). For task 2, the number of successful participants was probably too small to detect such differences).
Lieber2014, Addressing misconceptions about code with always-on programming visualizations.
Summary