Navigating Code

SWE 795, Fall 2019
Software Engineering Environments
Today

• Part 1 (Lecture) (~40 mins)
  • Navigating code

• Part 2 (Project Presentations, Part 1) (~40 mins)

• Break

• Part 3 (Project Presentations, Part 2) (~60 mins)
Code navigation: examples

- A developer wants to find method that implements $x$.
- A developer wants to find all of the methods involved in feature $x$.
- A developer wants to understand what a method does or when it is called.
- A developer wants to understand how to reuse a function by finding examples of code snippets.
- A developer wants to switch back to a method they were just editing.
Task context

• Could be
  • Set of **information necessary** to complete a task
  • Set of locations in **code** that must be **edited** to implement a change (e.g., add feature, fix bug)
• Which is it? Often used interchangeably…
• Sometimes known as a “working set”
How Effective Developers Investigate Source Code

• Unsuccessful subjects made all of their code modifications in one place even if they should have been scattered to better align with the existing design.
  • --> better support navigating across methods
• Program segments that were clearly relevant to the change task were not acknowledged when displayed accidentally.
  • --> support intentional searches
• The successful subjects created a detailed and complete plan prior to the change whereas the unsuccessful and average subjects did not.
  • --> support building a change plan
• Successful subjects did not reinvestigate methods as frequently as unsuccessful subjects.
  • --> support understanding methods
• The successful subjects performed mostly structurally guided searches (e.g., keyword and cross-reference searches), rather than searches based on intuition (browsing) or aligned with the file decomposition of the system (scrolling).
  • --> support structural relationship traversal

Structural Relationship Traversal

• Developer is currently viewing an element in code
  • e.g., class, method, statement, field reference
• Developers wishes to navigate to a related method
  • By reference, call, data dependency, …

• How do developers make navigation decisions?
Information foraging

• Mathematical model describing navigation
• Analogy: animals foraging for food
  • Can forage in different patches (locations)
  • Goal is to maximize chances of finding prey while minimizing time spent in hunt
• Information foraging: navigating through an information space (patches) in order to maximize chances of finding prey (information) in minimal time
Information environment

- Information environment represented as **topology**
- Information patches connected by traversable **links**
- For SE, usually modeled as call graphs
  - methods are nodes and function invocations are edges
Traversing links

- Links - connection between patch offered by the information environment
- Cues - information features associated with outgoing links from patch
  - E.g., text label on a hyperlink
- User must choose which, of all possible links to traverse, has best chance of reaching prey
Scent

• User interprets cues on links by likelihood they will reach prey
  • e.g., do I think that the “invoke” method is likely to implement the functionality I’m looking for?
Simplified mathematical model

• Users make choices to maximize **possibility** of reaching prey per cost of interaction

• Predators (idealized) choice = \( \text{max} \ [V / C] \)
  • \( V \) - value of information gain, \( C \) - cost of interaction

• Don’t usually know ground truth, have to estimate

• Predator’s desired choice = \( \text{max} \ [E[V] / E[C]] \)
What’s a concern?

Let me try to explain to you, what to my taste is characteristic for all intelligent thinking. It is, that one is willing to study in depth an aspect of one's subject matter in isolation for the sake of its own consistency, all the time knowing that one is occupying oneself only with one of the aspects. We know that a program must be correct and we can study it from that viewpoint only; we also know that it should be efficient and we can study its efficiency on another day, so to speak. In another mood we may ask ourselves whether, and if so: why, the program is desirable. **But nothing is gained —on the contrary!— by tackling these various aspects simultaneously.** It is what I sometimes have called "the separation of concerns", which, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts, that I know of. This is what I mean by "focusing one's attention upon some aspect": it does not mean ignoring the other aspects, it is just doing justice to the fact that from this aspect's point of view, the other is irrelevant.

Crosscutting concerns

• Ideal: one concern per module

• But, in practice modules exhibit
  • Scattering — single concern implemented in many modules
  • Tangling — single module containing many concerns
Significant time spent navigating across task context

- Each instance of an interactive bottleneck cost only a few seconds, but . . .

<table>
<thead>
<tr>
<th>Interactive Bottleneck</th>
<th>Overall Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigating to fragment in <em>same</em> file (<em>via scrolling</em>)</td>
<td>~ 11 minutes</td>
</tr>
<tr>
<td>Navigating to fragment in <em>different</em> file (<em>via tabs and explorer</em>)</td>
<td>~ 7 minutes</td>
</tr>
<tr>
<td>Recovering working set after returning to a task</td>
<td>~ 1 minute</td>
</tr>
<tr>
<td><strong>Total Costs</strong></td>
<td>~19 minutes</td>
</tr>
</tbody>
</table>

= 35% of uninterrupted work time!

Switching tasks incurs startup cost rebuilding task context

- Represented by explorer and file tabs
- When changing tasks, working sets were lost as tabs and nodes changed
- “Including” code in the working set by opening a file or expanding a node made it more difficult to navigate to other code in the working set

DeLine’s study of developers

- Confirmed Ko’s observation that:
  - Navigating and “re-finding” areas of the code that had already been visited was frequent, difficult and distracting
  - Textual searching and returning
  - Tabs got problematic when many opened
- All subjects wanted better inline comments and overview documentation.
- Wanted code annotations
- All subjects agreed that finding the entry point and understanding the control flow was the most difficult task

Field study of developers at IBM

- 8 IBM developers doing their own tasks using Eclipse for Java
- Interviews and 2-hour observations of actual use
- Experts do become disoriented
  - Did use Eclipse’s advanced navigation tools, like find-all-callers
  - No trace of how got to the current file, or how to get back
  - Thrashing to view necessary context
- No support for switching tasks

Working with task context

Code navigation tools

• Structural relationship traversal
  • Find starting point, traverse relationships to find other related code locations

• Recommenders
  • Based on {edits, navigation} past developers did on similar tasks, predict relevant elements

• Working set navigation
  • Make it easier to navigate back and forth between task context elements
  • Make it easier to resume tasks by redisplaying working set
Structural relationship traversal

Call hierarchy view
JQuery

```java
public void displayBox(Rectangle r) {
    // Checks whether the given figure is correct.
    public boolean includes(Figure figure) {
        // Decomposes a figure into its parts. A part is a part of itself.
    }
    public FigureEnumeration decompose() {
        // Sets the Figure's container and registers it as a figure change listener. A figure may have any kind of FigureChangeListener. A container may have a single container.
    }
    public void addToContainer(FigureChangeListener listener) {
        // Removes a figure from the given container.
    }
    public void removeFromContainer(FigureChangeListener listener) {
        // Gets the Figure's listeners.
    }
    public FigureChangeListener listener() {
        // Adds a listener for this figure.
    }
    public void addFigureChangeListener(FigureChangeListener listener) {
        // Removes a listener for this figure.
    }
    public void removeFigureChangeListener(FigureChangeListener listener) {
        // Releases a figure's resources. Release a figure is removed from a drawing.
    }
```

LaToza
Recommenders

• Based on {edits, navigation} past developers did on similar tasks, predict relevant elements
TeamTracks

- Shows source code navigation patterns of team
- Related Items – most frequently visited either just before or after the selected item
- Favorite Classes – hide less frequently used
- Deployed for real use – 5 developers for 3 weeks
- Successful, but usability issues, seemed most useful for newcomers

![TeamTracks Diagram]

Figure 1. Screen shot with (A) the typical class view, (B) our Class View Favorites, and (C) our Related Items.

Task context navigation

• Make it easier to navigate back and forth between task context elements
• Make it easier to resume tasks by redisplaying task context
Fig. 7. The 50 lines of code and other information that developer B indicated as relevant, portrayed in a mockup of a workspace that help developers collect relevant information for a task in one place, independent of the structure of a program.

Concern Graphs

- Abstract (formal) model that describe which parts of the source code are relevant to different concerns
- FEAT tool builds concern graphs “semi-automatically”
- Shows only code relevant to the selected concern
- User-specified or detected using intra-concern analysis
- User can make queries

1 – task list
3 – package explorer filters to show what relevant to this task
   Most relevant are bold
4 – active search shows what might be relevant
5 – switch to different task

Code Bubbles

Figure 1 (a) user opens a bubble via the pop-up search box, (b) resulting bubble, (c) user opens definition of two more bubbles side-by-side (automatically grouped); (d) a large working set of bubbles, including a (f) bubble stack of references; (e) an overview is shown in the panning bar; (g) hover preview

https://www.youtube.com/watch?v=PsPX0nEIJ0k

Debugger Canvas

Figure 1. The user stepping through a parallel ray tracing program using Debugger Canvas. All four threads have their own colored borders. The currently executing method has a prominent yellow border. Each code bubble has its own Locals pop-up (gray title bar), allowing state comparison.

https://www.youtube.com/watch?v=3p9XUwIlhJg

Use in practice

• Debugger Canvas offered as extension to Visual Studio
  

• Mylar —> Mylyn, part of default Eclipse

• Mylyn —> commercial

Instant access to documents, web pages and code

With Tasktop, you can indicate when you start working on a task with a single-click. Tasktop then automatically tracks the web pages and desktop documents you work with and builds a model of how relevant each is for that task. Tasktop uses the model to show you just those pages and documents that are needed for a task. Switching to another task is as simple as clicking on that task. Tasktop Dev will show you just the information needed for that task. Think of the time you will save only working with the information you need! If you are using Tasktop Dev within your IDE, Tasktop Dev will focus the views in your IDE to show you just the code, web pages and documents that are needed for a task.

Keep on top of your bugs without leaving your IDE

Tasktop Dev inherits Mylyn’s capabilities for rich bug editing in Eclipse, making the tasks from your issue tracker available in the IDE. With your task and its associated social comment thread in the IDE, you no longer have to spend time switching to a web browser, finding the appropriate task, switching back to the IDE, and so on. All of those clicks and applications switches disappear, saving time and keeping you close to the code. When changes occur to tasks on which you work, notifications appear right in your task list in Eclipse, saving you the need to check other applications and allowing you to respond seamlessly and as a part of your workflow. This capability is only available in Tasktop Dev for Eclipse and Visual Studio.

https://www.tasktop.com/tasktop-dev
Results from Debugger Canvas deployment

Figure 5: Number of unique downloads per week, after the initial release on 13 June 2011.

Figure 6. Users per day who step into a code bubble at least once, as a percentage of usage on the first day. (The gap is due to missing data.)

Perceptions of debugger canvas

Table 2: Reasons why 19 surveyed users stopped using Debugger Canvas. Each respondent may report multiple reasons.

<table>
<thead>
<tr>
<th>Reason to abandon</th>
<th>#Respondents</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editing not discovered</td>
<td>4</td>
<td>Usability</td>
</tr>
<tr>
<td>Bugs</td>
<td>4</td>
<td>Bugs</td>
</tr>
<tr>
<td>Performance</td>
<td>4</td>
<td>Bugs</td>
</tr>
<tr>
<td>Doesn't support my platform</td>
<td>3</td>
<td>Other</td>
</tr>
<tr>
<td>Screen too small</td>
<td>2</td>
<td>Useful</td>
</tr>
<tr>
<td>Data tips bug</td>
<td>2</td>
<td>Bugs</td>
</tr>
<tr>
<td>Wants features</td>
<td>2</td>
<td>Utility</td>
</tr>
<tr>
<td>Concept didn’t work for me</td>
<td>2</td>
<td>Utility</td>
</tr>
<tr>
<td>Want to resize bubbles</td>
<td>1</td>
<td>Utility</td>
</tr>
<tr>
<td>Instruction pointer update bugs</td>
<td>1</td>
<td>Bugs</td>
</tr>
<tr>
<td>Navigation not discovered</td>
<td>1</td>
<td>Usability</td>
</tr>
<tr>
<td>On demand not discovered</td>
<td>1</td>
<td>Usability</td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td><strong>27</strong></td>
<td></td>
</tr>
</tbody>
</table>

Useful when

“I often have to debug several layers on our side from the UI, via middle tier to the data layer. It often gets confusing to go into the deeper layer. This is where the canvas helps, you hit a breakpoint here and can see the stack trace as you step through the layers. This helps us debug things much faster.”

“I was working on a large project for only a week. There was a huge ramp up, of course, and Debugger Canvas was invaluable for stepping into the code to see what was going on.”

“With a really large code base that you are not familiar with it is really handy. It helps wrap your head around other people's code. That kind of visualization really helps to follow code as it crosses different classes and projects. Go-to-definition and using Reflector is just too cumbersome to navigate through all that code.”

Not useful when

For a "normal" project it isn't worth the hassle with performance.

I don't always want to get into the canvas. When I’m debugging something small: for example - Did the parameter get here? Then it doesn’t warrant opening up the canvas.

Sometimes the fix that I need to do involves code that is not in the bubbles, but is in the same files, so I'd like to be able to get to the rest of the file easily.

I stop using it when I need to see definition of classes. I'm aware of the Go-to-definition feature, but I use ReShaper and lots of tools to navigate, so I find it easier to go back to the file in those cases.

I hit a breakpoint check the value of a private field. That's when seeing the rest of the file comes in handy.