Crosscutting Concerns
Today

• Part 1 (Discussion)(60 mins)
  • Discussion of readings
• Break!

• Part 2 (Lecture)(~70 mins)
  • Crosscutting concerns

• Part 3 (Group work)(~20 mins)
  • Time to work on HW2
Crosscutting Concerns

• What’s a concern?
  • In what ways is it similar or different than task context?
  • Why does it matter if they crosscut?

• What approaches might help reduce challenges developers experience?
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
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…
- Also known as a “working set”
Problems caused by crosscutting concerns

• Identifying & understanding elements in task context

• Navigating between elements in task context
Scattered concerns are associated with higher defects

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

Form working set of task-relevant code

Navigate dependencies in working set

Modify code in working set

task started

Solutions to crosscutting concerns

• Relationship traversal
  • Find starting point, traverse relationships to find other related code locations
  • See information needs & debugging lectures

• New modularity constructs—Aspects
  • Reduce scattering & tangling by introducing new crosscutting module that can be weaved into code

• 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
New modularity constructs—Aspects

- Key idea: modularize scattered code into aspects
  - Developers work with sets of methods in an aspect just like in a class
  - Aspects *woven* back into underlying code during compilation

- Specific *join points* (e.g., field access, method call) can be intercepted, invoking method in aspect
  - *Point cut descriptor* describes which join points a specific aspect method will be invoked from

- Ideally, developer of module can be *oblivious* of code contained in aspect that is woven in

Critiques of Aspects

• Determining the behavior of a module now requires global reasoning
  • Need to weave aspects before clear how code will behave.

• Complexity of when and if code will be inserted
  • Fragile pointcut problem: point cut descriptors are fragile, often depending on textual properties of identifiers
  • Multiple aspects could apply to same join point, making it unclear what ordering applies
Recommenders

- Based on {edits, navigation} past developers did on similar tasks, predict relevant elements
LaToza

Rose

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

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
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

Mylar

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

https://www.youtube.com/watch?v=PsPX0nElJ0k

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>
<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.