In the Natural Programming group at Carnegie Mellon, we start every project by first studying people and their problems, and using the knowledge we gain from these studies to design more helpful tools.

Today I’ll be describing an empirical user study we did to better understand programmers’ maintenance tasks, and I’ll discuss how we can use this understanding to design more helpful maintenance tools.

Everyone here recognizes this basic problem...
Many successful tools for writing code . . .

Write once

Read forever . . .

. . . Fewer successful tools for understanding and maintaining code.

• We write code once, but we end up reading and maintaining it for far longer.
• The problem is, most of the successful tools in modern IDEs only help write code.
• There are far fewer successful tools for understanding and maintaining code.

So if we want to build maintenance-oriented IDEs
Many successful tools for writing code . . .

Read forever . . .

Fewer successful tools for understanding and maintaining code.

• We write code once, but we end up reading and maintaining it for far longer.
• The problem is, most of the successful tools in modern IDEs only help write code.
• There are far fewer successful tools for understanding and maintaining code.

So if we want to build maintenance-oriented IDEs
If we were to design maintenance-oriented IDEs that tried to help programmers be more efficient and effective at maintenance tasks, how should they work?

To answer this, we have to answer two other questions:

- What information and operations do programmers need to efficiently and successfully complete maintenance tasks.
- How do modern IDEs help or hinder these tasks?

For the rest of this talk, I’ll be discussing an empirical USER study that suggests some answers to these questions and some design ideas for tools that are currently under development.

Our user study had two major goals.
If we were to build maintenance-oriented IDEs . . .

How should they work?

What information and operations do programmers need?

How do modern IDEs support these tasks?

• If we were to design maintenance-oriented IDEs that tried to help programmers be more efficient and effective at maintenance tasks, how should they work?
  
  • To answer this, we have to answer two other questions:
    • What information and operations do programmers need to efficiently and successfully complete maintenance tasks.
    • How do modern IDEs help or hinder these tasks?

For the rest of this talk, I’ll be discussing an empirical USER study that suggests some answers to these questions and some design ideas for tools that are currently under development.

Our user study had two major goals.
Goals of the User Study

Investigate the how IDEs influence programmers’ ability to acquire and manipulate information in maintenance tasks.

Focus on day-to-day maintenance tasks such as bug repairs and feature enhancements.

- First, we wanted study how IDEs influence programmers’ ability to acquire and manipulate. This was as opposed to the impact of social or organizational issues.
- We also focused on day to day maintenance tasks such as bug repairs and feature enhancements, as opposed to large reverse engineering projects or substantial rewrites.

The study itself had several components.
The User Study

- We recruited 10 programmers, who reported expertise with Java.
- We designed 1 program for programmers to maintain and 5 tasks for it. I’ll describe both shortly.
- Programmers used the Eclipse 2.0 IDE. We chose Eclipse because we wanted to identify inadequacies of even the best of Java IDEs.
- Programmers were given an internet connection and were free to use any resources they saw fit.
- Programmers were paid $10 per task completed.

The task structure...
The Task Structure

- **70 minutes**, as many tasks as possible
- Full resolution screen-capture at 12 fps

---

- Interruptions about every ~3 minutes

  [Gonzalez and Mark 2004]

  Based on evidence that software engineers were interrupted every three minutes in the workplace.

  How do programmers recover from interruptions? (see CHI 2005)

---

- Programmers worked for 70 minutes and could attempt the tasks in any order they wanted to.
- We used screen-capturing software to record all of the programmers’ work.
- Also, because of recent evidence about interruptions in software development in the workplace, we decided to interrupt programmers about every three minutes.
- Not only did this add a level of realism to the task, but it also allowed us to see how programmers recovered from interruptions. You can find more details about this aspect of the study in our CHI 2005 paper.

*The application that programmers maintained was*
The Paint Application

- 508 lines of Java Swing code
- Draw, erase, clear and undo colored strokes

The paint application, seen here, was 508 lines of Java Swing code, and allowed users to draw, erase, clear and undo colored strokes.

Three of the tasks required programmers to repair some bug and two required them to add new features to the application.

*We described the tasks as user complaints and requests. For example,*
Yellow

“Users have complained they can’t choose a yellow color...”

Error

The green slider’s value was used twice and the blue slider’s value was ignored.

• “Users have complained that they can’t choose a yellow color.”
• The actual error, which we didn’t tell the programmers, was that the green slider’s value was used twice and the blue slider’s value not at all.

Users also complained that ...
Undo

“Users have complained that the undo button doesn’t always work.”

Error

The undo() method failed to call repaint() on the canvas.

- ... the undo button doesn’t always work.
- The actual bug here was that the undo procedure wasn’t repainting the canvas, and so it was only repainted when something else caused a repaint.

*Users also complained that ...*
Scroll

“Users complained that there are repainting issues when scrolling in the window.”

Error

The canvas was not resized when the window was resized.

- ...there are repainting issues when scrolling the window.
- This was a result of the canvas not being resized when the window was resized.

The two feature enhancements were to ...
“Users have requested a line tool.”
Thickess  “Users have requested control over the stroke thickness.”

We ran the ten subjects, which resulted in 10 70-minute videos.
The first thing we did was to go through all of the videos and get a sense for what programmers were doing: how did they structure their tasks, what information they were looking for.

Once we had a sense of programmer’s activities, two experts went through all of the videos and transcribed several types of events by hand. These included...

We then used this data for our analyses. The full details of this procedure are in the paper.

*The first thing we'll look at is how programmers divided their time.*
The first thing we did was to go through all of the videos and get a sense for what programmers were doing: how did they structure their tasks, what information they were looking for.

Once we had a sense of programmer’s activities, two experts went through all of the videos and transcribed several types of events by hand. These included...

We then used this data for our analyses. The full details of this procedure are in the paper.

*The first thing we'll look at is how programmers divided their time.*
Two experts transcribed several events from the videos by hand

- User interface actions
- Dependency navigations
  - Use to definition
  - Method to invocation
  - Class to superclass
  - etc.
- Reading code
- Switching environments
- Testing Paint
- etc.

- The first thing we did was to go through all of the videos and get a sense for what programmers were doing: how did they structure their tasks, what information they were looking for.
- Once we had a sense of programmer’s activities, two experts went through all of the videos and transcribed several types of events by hand. These included...
- We then used this data for our analyses The full details of this procedure are in the paper.

The first thing we’ll look at is how programmers divided their time.
On average, programmers spent the majority of their time doing five basic activities: reading code, editing code, explicitly navigating dependencies in code, testing code, and finally, searching for task-relevant words in the code.

For example, using search commands to search for any code with the word “undo” in it.

The remaining time was spent ...

*We also looked at the sequence of programmers’ tasks.*
For example, this is one programmer’s sequence of work, divided up by the categories of particular events. The programmer spent the first part of the task finding and reading, then navigating dependencies, then modifying and testing.

When we looked at the particular code that programmers were reading, navigating, and modifying, it turned out to be a very small subset of the application code. Furthermore, all of the programmer’s tasks followed this basic pattern, so we generalized these into three major activities.

The first formed a working set of task–relevant code. We’ll define this on the next slide. The second navigated this working set and the third modified code in the working set.

So what is a working set?
A Programmer’s Task

- For example, this is one programmer’s sequence of work, divided up by the categories of particular events. The programmer spent the first part of the task finding and reading, then navigating dependencies, then modifying and testing.

- When we looked at the particular code that programmers were reading, navigating, and modifying, it turned out to be a very small subset of the application code. Furthermore, all of the programmer’s tasks followed this basic pattern, so we generalized these into three major activities.

- The first formed a working set of task-relevant code. We’ll define this on the next slide. The second navigated this working set and the third modified code in the working set.

So what is a working set?
A Programmer’s Working Set

A collection of task-relevant code fragments

Modern languages, toolkits and APIs require distributed, non-local dependencies

- A working set is a collection of task-relevant code fragments.
- For example, these are all 508 lines of the paint program and the grey regions represent the code that that one programmer modified or referred to once he began making modifications for the stroke thickness task.
- It’s important to note that the highly distributed nature of these code fragments is not due to poor design; in fact, many of the programmers thought the code was particularly well designed.
- Rather, modern APIs and toolkits tend to require highly distributed dependencies. For example, to add the thickness slider, he had to add initialization code and an event handler, and update the data model to support thickness data.

*Given the idea of a working set, we’ll now take a more detailed look at how programmers formed, represented, and navigated their working sets, and how the IDE supported them in these processes.*
A Programmer’s Working Set

A collection of task-relevant code fragments

Modern languages, toolkits and APIs require distributed, non-local dependencies

- A working set is a collection of task relevant code fragments.
- For example, these are all 508 lines of the paint program and the grey regions represent the code that that one programmer modified or referred to once he began making modifications for the stroke thickness task.
- It’s important to note that the highly distributed nature of these code fragments is not due to poor design; in fact, many of the programmers thought the code was particularly well designed.
- Rather, modern APIs and toolkits tend to require highly distributed dependencies. For example, to add the thickness slider, he had to add initialization code and an event handler, and update the data model to support thickness data.

Given the idea of a working set, we’ll now take a more detailed look at how programmers formed, represented, and navigated their working sets, and how the IDE supported them in these processes.
Forming Working Sets

How does ____ work?  

Searched for seemingly task-relevant words

Why did(n’t) ____ happen?

Formed hypotheses about potential causes of unexpected behavior

Only 50% of searches led to relevant code

88% of hypotheses were false

Programmers couldn’t relate the behavior they saw (or didn’t see) to the code responsible for it.

- In general, programmers formed working sets by starting with a question.
- For enhancement tasks, asked how some similar existing behavior is implemented. They answered this question by searching for task-relevant words. The problem was that only about half of searches...
- For the debugging tasks, asking why did or why didn’t something happen. Answered it by generating hypotheses about the potential causes of the behavior. The problem was that the majority of hypotheses were wrong.
- In general, answering these questions was inefficient and error-prone because programmers couldn’t relate the behavior they saw or didn’t see when testing the program to the code that was responsible for it.

As they formed working sets, they had to represent them somehow.
Representing Working Sets

- Represented by explorer and file tabs
- Files are the wrong granularity
- "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
- When changing tasks, working sets were lost as tabs and nodes changed

To do so, programmers used Eclipse’s tabs and package explorer to approximate the set of code fragments in their working sets. They left the tabs and explorer nodes that contained task-relevant fragments open and closed irrelevant ones.

- Obviously, files aren’t the most efficient way of representing fragments. As a result, programmers frequently had navigate within a file to find a particular fragment.
- Including code made finding other code more difficult. For example, opening a single file tab makes all of the other file names tabs more inscrutable, hindering navigation.
- Also, when programmers temporarily left a task, they had to recover it when they returned to the task.

As programmers formed and represented their working sets, they spent considerable time navigating dependencies in their working set.
Navigating within a Working Set

Navigated ~65 dependencies over 70 minutes

58% direct
- declaration of...
- use of...
- called by...
- definition of...

42% indirect
- "The method that computes the value that is passed to this method and used in this expression"

Many (but not all) supported by Eclipse commands

Supported only by scroll bars, package explorer, tabs, and find

- On average, they made 65 navigations over the 70 minutes.
- We identified two major types of dependency navigations.
- Direct, for example, many of these were supported directly by Eclipse, and they were used when available, especially Eclipse’s open declaration command.
- Indirect, for example, these navigations had to be performed manually, and because most indirect relationships were between files, incurred significant interactive overhead.

When we look at each instance of an interactive bottleneck...
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 sets</td>
<td>~1 minute</td>
</tr>
</tbody>
</table>

**Total Costs**

~19 minutes

35% of uninterrupted work time (~55 minutes)

- Each individual instance of interactive overhead cost anywhere from 2–10 seconds.
- However, when we added up all of this time, it turned out to be a significant proportion of programmers’ work time.
- For example, navigating to a fragment in the same file via scrolling cost 11 minutes overall. This doesn’t include the time to decide where to navigate to.
- Within a file...
- Recovering working set
If we we’re to build maintenance-oriented IDEs . . .

How should they work?

What information and operations do programmers need?

How do modern IDEs support these tasks?

• Now that we have a greater understanding of programmers maintenance tasks and how modern IDEs support them, how should maintenance-oriented IDEs work?
If we were to build maintenance-oriented IDEs . . .

How should they work?

✓ What information and operations do programmers need?

✓ How do modern IDEs support these tasks?

• Now that we have a greater understanding of programmers maintenance tasks and how modern IDEs support them, how should maintenance-oriented IDEs work?
Files

- Allow programmers to grab arbitrary fragments of code to represent working sets
- Fragments would be automatically updated as they change over time
- Including a fragment could automatically include dependent fragments

The first obvious requirement is that, in addition to files, programmers need to be able to collect and manipulate fragments of code, independent of files

Another important point is that ...
• Allow programmers to grab arbitrary fragments of code to represent working sets
• Fragments would be automatically updated as they change over time
• Including a fragment could automatically include dependent fragments

The first obvious requirement is that, in addition to files, programmers need to be able to collect and manipulate fragments of code, independent of files.

Another important point is that ...
• Allow programmers to view all code fragments in one place, one screen
• No navigation to compare fragments
• No navigation to view indirect dependencies

• All of the fragments should be visible in one place, preferably in a single screen.
• This is important in avoiding the costly navigation: if everything is in one place, then programmers don’t have to navigate to compare fragments or follow dependencies.

Another idea is to explicitly show these dependencies...
• Explicitly visualize direct dependencies between code fragments

• Every time a fragment is added, dependences could automatically be shown.

Every time a new fragment was added, dependencies with that fragment could be shown. This way, programmers don’t have to manually determine these dependencies.

*It's also important that programmers be able to maintain multiple, independent working sets of code fragments to represent different maintenance tasks.*
• Explicitly visualize direct dependencies between code fragments

• Every time a fragment is added, dependences could automatically be shown.

It's also important that programmers be able to maintain multiple, independent working sets of code fragments to represent different maintenance tasks.
• Allow programmers to manage **multiple** working sets of code fragments for different maintenance tasks

• Would enable programmers to switch tasks without the cost of recovering working sets

• This would enable programmers to switch maintenance tasks without the cost of recovering working sets.

*Finally, to help programmers form working sets...*
• Allow programmers to manage **multiple** working sets of code fragments for different maintenance tasks

• Would enable programmers to switch tasks without the cost of recovering working sets

Finally, to help programmers form working sets...
• Allow programmers to ask about their program’s behavior

• Help form working set by automatically via static and dynamic analyses

• Similar to our Whyline debugging tool

IDEs need to allow programmers to ask about their program’s output and behavior.

For example, programmers could ask, “What happens when I press the undo button?” and the IDE could determine all of the code fragments executed as a result of the button being pressed.

And we can due this without natural language; we have been working on a debugging tool called the Whyline, which we presented at CHI last year, which allows programmers to ask questions through direct manipulation.

*Putting all of these ideas together, here is one possible interface for representing a working set*
• Allow programmers to ask about their program’s behavior

• Help form working set by *automatically* via static and dynamic analyses

• Similar to our Whyline debugging tool

• IDEs need to allow programmers to ask about their program’s output and behavior.

• For example, programmers could ask, “What happens when I press the undo button?” and the IDE could determine all of the code fragments executed as a result of the button being pressed.

• And we can due this without natural language; we have been working on a debugging tool called the Whyline, which we presented at CHI last year, which allows programmers to ask questions through direct manipulation.

*Putting all of these ideas together, here is one possible interface for representing a working set*
• Allow programmers to ask about their program’s behavior

• Help form working set by *automatically* via static and dynamic analyses

• Similar to our Whyline debugging tool

IDEs need to allow programmers to ask about their program’s output and behavior.

For example, programmers could ask, “What happens when I press the undo button?” and the IDE could determine all of the code fragments executed as a result of the button being pressed.

And we can due this without natural language; we have been working on a debugging tool called the Whyline, which we presented at CHI last year, which allows programmers to ask questions through direct manipulation.

*Putting all of these ideas together, here is one possible interface for representing a working set*
One Possible Concept

Task-relevant code fragments

Multiple working sets

• Don’t worry about reading the details.
• Notice how all of the relevant code fragments are on one screen, instead of distributed amongst multiple files.
• Relationships between code fragments are shown with arrows.
• Other working sets are listed on the right.
• More details and design ideas are given in the paper.

To briefly summarize,
Contributions

- Programmers organized maintenance tasks in terms of working sets of code fragments
- Programmers spent 35% of their time simply navigating within a small set of fragments
- New user interfaces for representing working sets could eliminate this overhead, dramatically increasing productivity

This work was funded in part by the National Science Foundation, under NSF grant IIS-0329090, and as part of the EUSES consortium (End Users Shaping Effective Software) under NSF grant ITR CCR-0324770. The first author is also supported under a National Defense Science and Engineering Graduate Fellowship.

http://www.cs.cmu.edu/~natprog

- We are currently working on implementing and evaluating these ideas and we hope to present them soon. Maybe next year in Shanghai?
- Thank you.

(I also want to note that this work was done as part of the EUSES consortium)