Improving Program Comprehension by Answering Questions

Brad A. Myers
Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
http://www.cs.cmu.edu/~bam
bam@cs.cmu.edu
Questions

• “To be or not to be?”
• “What is the meaning of life?”
• “Ask not what your country can do for you – ask what you can do for your country.”
• “Which outfit should I wear?”
• “What does this code do?”
• “What just happened? …”
Natural Programming Project

• Researching better tools for programmers since 1978
• Natural Programming project started in 1995
• Make programming easier and more correct by making it more natural
  – Closer to the way that people think about their problems and solving their tasks
• Methodology – human-centered approach
  – Perform studies to inform design
    • Provide new knowledge about what people do and need to know
  – Guide the designs from the data
    • Design of languages, environments and documentation
  – Iteratively evaluate and improve
• Target novice, expert and end-user programmers
End User Programming

- People whose primary job is not programming
- In 2012, in USA at work: — Scaffidi, Shaw and Myers 2005
  - 3 million professional programmers
  - 6 million scientists & engineers
  - 13 million will describe themselves as programmers
  - 55 million will use spreadsheets or databases at work (and therefore may potentially program)
  - 90 million computer users at work in US
- We should make better tools for all of these people!
Debugging

- Study commissioned by NIST USA (2002) of 14 software vendors
  - Software errors cost $\sim$60 billion annually
  - Software engineers spend 70-80% of time testing and debugging
    - Time for 1 developer to fix 1 bug was $\sim$17.4 hours
- Current debugging techniques same as for last 70 years
  - Same for end-user and professional environments
Goal: Gentle Slope Systems

Program Complexity and Sophistication

Difficulty of Use

Low Threshold

Goal

Web Development
Java
Visual Basic
Flash
Server-side
C Programming
JavaScript
ActionScript
CSS & HTML
Basic
Email
Filters
editor

© 2013 – Brad A. Myers
Human-Computer Interaction Institute
Improve Developer Experience

- Use human centered approaches to:
  - Find out what developers *need to know*
  - Understand developers’ *barriers* that cause *wasted time*
  - Make developers *more effective*
  - *Reduce errors* in their understanding and in the resulting code
  - Insure that developer tools are *useful*
Why Would Being Natural be Good?

• Programmers are People Too
  – Take the human into account

• Language should be close to user’s plan
  – “Programming is the process of transforming a mental plan into one that is compatible with the computer.”  
    — Jean-Michel Hoc

• Closeness of mapping
  – “The closer the programming world is to the problem world, the easier the problem-solving ought to be…. Conventional textual languages are a long way from that goal.”  
    — Green and Petre
Hard to understand

drawImage(img,10,20,30,40,11,21,31,41,red,obs);

- 8 ints

item = new Item("C12","S123","S123","P123","I123",","1","2","3","4","5",1.0d,10.0d);

- 11 strings

Let Shape1.FillColor = &H00FF00FF&
Study of Errors

- Study of novice errors and debugging
- Developed a model of problems and errors
  - Problems causing other problems

(EUP'03)
Study of Errors

• All of the observed debugging problems could be addressed by “Why” questions
  – 32% were “Why did”; 68% were “Why didn’t”

• Current debugging techniques require user to **guess** where bug is or where to look
  – Most of initial guesses are **wrong**, even for experts
Original Design: Whyline for Alice

- Andy Ko, PhD 2008
- Answers as an elaborate visualization of control and dataflow

*(CHI ‘04)*
Whyline for Java

- New algorithms
- New user interface design
  - Visualization primarily as navigation aide
  - Importance of search
- Not sufficient to just scrub through time

(ICSE’2008)
Whyline

- **Whyline = Workspace that Helps You Link Instructions to Numbers and Events**
- **Initial study:**
  - Whyline with novices outperformed experts with Eclipse
  - Factor of 2.5 times faster
    - $(p < .05, \text{Wilcoxon rank sums test})$
- **Formal study:**
  - Experts attempting 2 difficult tasks
  - Whyline over 3 times as successful, in $\frac{1}{2}$ of the time

![Diagram showing comparison between Whyline and control](CHI '09)
WebCrystal

- Investigate CSS and HTML responsible for example behaviors
- Navigate around HTML hierarchy
- Ask “how-do-I” questions about look, position and behavior
- Generates code in user-selected format
- Combine code for multiple elements

(CHI ’12)
Study of Design Requirements for Maintenance-Oriented IDEs

- Studied expert use of Java Eclipse IDE in a lab setting (2004-2006)
- Focus on day-to-day maintenance tasks such as bug repairs and feature enhancements
- Lab study with detailed analysis
- Rich dataset → multiple papers

(ICSE’05)
Time Spent on Different Activities

- Reading code: 22%
- Editing code: 20%
- Navigating dependencies: 16%
- Searching for task-related words: 13%
- Testing Paint: 13%
- Reading the Java API: 13%
- Switching environments: 6%
- Reading task descriptions: 3%

© 2013 – Brad A. Myers
**Times for Bottlenecks**

- 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>(\sim 11 \text{ minutes})</td>
</tr>
<tr>
<td>Navigating to fragment in <em>different</em> file <em>(via tabs and explorer)</em></td>
<td>(\sim 7 \text{ minutes})</td>
</tr>
<tr>
<td>Recovering working set after returning to a task</td>
<td>(\sim 1 \text{ minute})</td>
</tr>
<tr>
<td><strong>Total Costs</strong></td>
<td><strong>(\sim 19 \text{ minutes})</strong></td>
</tr>
</tbody>
</table>

= 35% of uninterrupted work time!
Forming Working Sets

• How does _____ work?
  – Searched for seemingly task-relevant words
  – Only 50% of searches led to relevant code

• Why did(n’t) _____ happen?
  – Formed hypotheses about potential causes of unexpected behavior
  – 88% of hypotheses were false

Programmers had trouble relating the behavior they saw (or didn’t see) to the code responsible for it.
A Programmer’s Working Set

- A collection of task-relevant code fragments
- In modern software development, dependencies are distributed and non-local
Jasper: Working Set Tool

- **Jasper** = Java Aid with Sets of Pertinent Elements for Recall
- Allow programmers to grab arbitrary fragments of code to represent working sets
  - Allow programmers to view in one place, one screen
- Influenced Reiss *et. al’s* Code Bubbles and DeLine’s (Microsoft) Debugger Canvas in Visual Studio

(ETX’06)
7 Studies of Understanding and Exploring Code

- PhD of Thomas LaToza (2012)
- Extensive investigations of how developers understand and explore code
  - **4 Surveys**: 740 developers at Microsoft
  - **Interviews**: 11 developers at Microsoft
  - **Controlled Experiment**: 16 students and staff at CMU x 3 hours
    - 11,821 lines of navigation events & 32 code changes
  - **Field Observations**: 17 developers at Microsoft x 90 minutes
    - 386 pages of transcripts
    - Minute by minute activity

*(PLATEAU’2010)*
Many hard-to-answer questions about code

Rationale (42)
Why was it done this way? (14) [15][7]
Why wasn’t it done this other way? (15)
Was this intentional, accidental, or a hack? (9)[15]
How did this ever work? (4)

Debugging (26)
How did this runtime state occur? (12) [15]
What runtime state changed when this executed? (2)
Where was this variable last changed? (1)
How is this object different from that object? (1)
Why didn’t this happen? (3)
How do I debug this bug in this environment? (3)
In what circumstances does this bug occur? (3) [15]
Which team’s component caused this bug? (1)

Intent and Implementation (32)
What is the intent of this code? (12) [15]
What does this do (6) in this case (10)? (16) [24]
How does it implement this behavior? (4) [24]

Refactoring (25)
Is there functionality or code that could be refactored? (4)
Is the existing design a good design? (2)
Is it possible to refactor this? (9)
How can I refactor this (2) without breaking existing users? (9)
Should I refactor this? (1)
Are the benefits of this refactoring worth the time investment? (3)

History (23)
When, how, by whom, and why was this code changed or inserted? (13)[7]
What else changed when this code was changed or inserted? (2)
How has it changed over time? (4)[7]
Has this code always been this way? (2)
What recent changes have been made? (1)[15][7]
Have changes in another branch been integrated into this branch? (1)

Implications (21)
What are the implications of this change for (5) API clients (5),
security (3), concurrency (3), performance (2), platforms (1), tests
(1), or obfuscation (1)? (21) [15][24]

Testing (20)
Is this code correct? (6) [15]
How can I test this code or functionality? (9)
Is this tested? (3)
Is the test or code responsible for this test failure? (1)
Is the documentation wrong, or is the code wrong? (1)

Implementing (19)
How do I implement this (8), given this constraint (2)? (10)
Which function or object should I pick? (2)
What’s the best design for implementing this? (7)

Control flow (19)
In what situations or user scenarios is this called? (3) [15][24]
What parameter values does each situation pass to this method? (1)
What parameter values could lead to this case? (1)
What are the possible actual methods called by dynamic dispatch here? (6)
How do calls flow across process boundaries? (1)
How many recursive calls happen during this operation? (1)
Is this method or code path called frequently, or is it dead? (4)
What throws this exception? (1)
What is catching this exception? (1)

Contracts (17)
What assumptions about preconditions does this code make? (5)
What assumptions about pre(3)/post(2)conditions can be made?
What exceptions or errors can this method generate? (2)
What are the constraints on or normal values of this variable? (2)
What is the correct order for calling these methods or initializing these objects? (2)
What is responsible for updating this field? (1)

Performance (16)
What is the performance of this code (3) on a large, real dataset? (3) (8)
Which part of this code takes the most time? (4)
Can this method have high stack consumption from recursion? (1)
How big is this in memory? (2)
How many of these objects get created? (1)

Teammates (16)
Who is the owner or expert for this code? (3)[7]
How do I convince my teammates to do this the “right way”? (12)
Did my teammates do this? (1)

Policies (15)
What is the policy for doing this? (10) [24]
Is this the correct policy for doing this? (2) [15]
How is the allocation lifetime of this object maintained? (3)

Type relationships (15)
What are the composition, ownership, or usage relationships of this type? (5) [24]
What is this type’s type hierarchy? (4) [24]
What implements this interface? (4) [24]
Where is this method overridden? (2)

Data flow (14)
What is the source of this data? (2) [15]
What code directly or indirectly uses this data? (5)
Where is the data referenced by this variable modified? (2)
Where can this global variable be changed? (1)
Where is this data structure used (1) for this purpose (1)? (2) [24]
What parts of this data structure are modified by this code? (1) [24]
What resources is this code using? (1)

Location (13)
Where is this functionality implemented? (5) [24]
Is this functionality already implemented? (5) [15]
Where is this defined? (3)

Building and branching (11)
Should I branch or code against the main branch? (1)
How can I move this code to this branch? (1)
What do I need to include to build this? (3)
What includes are unnecessary? (2)
How do I build this without doing a full build? (1)
Why did the build break? (2)[59]
Which preprocessor definitions were active when this was built? (1)

Architecture (11)
How does this code interact with libraries? (4)
What is the architecture of the code base? (3)
How is this functionality organized into layers? (1)
Is our API understandable and flexible? (3)

Concurrency (9)
What threads reach this code (4) or data structure (2)? (6)
Is this class or method thread-safe? (2)
What members of this class does this lock protect? (1)

Dependencies (5)
What depends on this code or design decision? (4)[7]
What does this code depend on? (1)

Method properties (2)
How big is this code? (1)
How overloaded are the parameters to this function? (1)
Many opportunities for better tools

- Of all the reported questions
  - 34% addressed by commercial tools
  - 25% addressed by research tools
  - 41% unaddressed by any tools
No single activity dominates work

**Prompt:** Percent of work time last week that I spent

![Box plot and bar chart showing the distribution of work time across various activities such as communicating, understanding, non-code activities, editing, writing, designing, overhead, unit testing, and other code-related activities. The chart includes median, 1st quartile, 3rd quartile, and IQR, with outliers marked. The right side of the image shows a bar chart illustrating the number of activities in the past week.]
Most time debugging and investigating

Outgoing edges are in color of activity
Circle size: % of coding activity time
Edge thickness: % of transitions observed
Frequent question: Reachability

- Programmers investigate *reachability questions*
  - How can this code *be reached*, either upstream or downstream
  - E.g., control flow from user scrolling $\rightarrow$ update status line
- Survey shows such control flow questions are difficult and important
- No easy way to discover with current tools
  - Call graphs are too general
  - Call hierarchy too deep

(\textit{ICSE'2010})
**REACHER**

- Visualize exactly the paths of interest
- **Search** along the paths
- Focused questions and answers enable effective analysis of complex codebases
- Developers with Reacher 5.6 times more **successful** than those working with Eclipse only 

*(VL/HCC'11)*

---

Image of a code editor with Reacher features highlighted.
Study of APIs

- Started as PhD work of Jeff Stylos, 2009
  - Inspired by Steven Clarke, Microsoft Visual Studio group
- Application Programming Interface
  - Libraries, frameworks, SDKs, …
- Barriers to understanding of APIs
- Measures: learnability, errors, preferences
- Expert and novice programmers
- Studied:
  - Default parameters in constructors
  - Factory pattern
  - Object design
  - SAP’s Web Services APIs
“Factory” Pattern

- Instead of “normal” creation: `Widget w = new Widget();`
- Objects must be created by another class:
  `AbstractFactory f = AbstractFactory.getDefault();
  Widget w = f.createWidget();`
- Used frequently in Java (>61) and .Net (>13) and SAP
- Results:
  - When asked to design on “blank paper”, no one designed a factory
  - Time to develop using factories took **2.1 to 5.3 times longer** compared to regular constructors (20:05 v 9:31, 7:10 v 1:20)
  - All subjects had difficulties understanding factories in APIs

(ICSE'2007)
Object Method Placement

• Where to put functions when doing object-oriented design of APIs when multiple classes work together
  – `mail_Server.send( mail_Message )`
  vs.
  `mail_Message.send( mail_Server )`

• When desired method is on the class that they start with, users were between 2.4 and 11.2 times faster \( (p < 0.05) \)
  – Initial intuition that class size correlated with difficulty was wrong

• Starting class can be predicted based on user’s tasks

![Time to Find a Method](image)

(FSE'2008)
Study of APIs for SAP

- Study APIs for Enterprise Service-Oriented Architectures ("Web Services")
- Naming problems:
  - Too long
  - Not understandable
  - Differences in middle are frequently missed

```
CustomerAddressBasicDataByNameAndAddressRequestMessage
CustomerSelectionCommonName
```

```
CustomerAddressBasicDataByNameAndAddressQueryMessage_sync
CustomerSelectionBySimpleSelectionByIDAndDescription
```
eSOA Documentation Results

- Multiple paths: unclear which one to use
- Some paths were dead ends
- Inconsistent look and feel caused immediate abandonment of paths
- Hard to find required information
- Business background helped

(IS-EUD'2009)
Our Tools to Help with APIs

- Mica
- Jadeite
- Calcite
- Euklas
- Graphite
- Apatite
Mica Tool to Help Find Examples

- Makes Interfaces Clear and Accessible
- Use Google to find relevant pages
- Match pages with Java keywords
- Also notes which pages contain example code or definitions

(VL/HCC’06)
Jadeite: Improved JavaDoc

- **Jadeite**: Java API Documentation with Extra Information Tacked-on for Emphasis
  - [http://www.cs.cmu.edu/~jadeite](http://www.cs.cmu.edu/~jadeite)
  - Fix JavaDoc to help address problems
    - Focus attention on most popular packages and classes using font size
    - “Placeholders” for methods that users want to exist
    - Automatically extracted code examples for how to create classes

---

See Also (auto-generated):
- Transport
- MimeMessage
- InternetAddress

---

(VL/HCC’09)
Calcite: Eclipse Plugin for Java

- **Calcite: Construction And Language Completion Integrated Throughout**
  - [http://www.cs.cmu.edu/~calcite](http://www.cs.cmu.edu/~calcite)
- Code completion in Eclipse augmented with Jadeite’s information
  - How to create objects of specific classes
    - `SSLSSocket s = ???`

*(VL/HCC’10)*
Euklas: Eclipse Plugin for JavaScript

- **Euklas**: Eclipse Users’ Keystrokes Lessened by Attaching from Samples
  - [http://www.cs.cmu.edu/~euklas](http://www.cs.cmu.edu/~euklas)
- Brings Java-like analysis to JavaScript
- Auto-correct uses copy source context for errors due to copy & paste
Graphite: Eclipse Plugin for Literals

• **Graphite**: GRAphical Palettes Help Instantiate Types in the Editor.

• Pop up a custom palette for specialized constants (literals) in Eclipse
  – Color palettes
  – Regular expression strings

• Customizable

(ICS’2012)
Apatite Documentation Tool

- **Apatite: Associative Perusing of APIs That Identifies Targets Easily**
  
  http://www.cs.cmu.edu/~apatite

- Start with verbs (actions) and properties and find what classes implement them

- Find associated items
  - E.g., classes that are often used together
  - Classes that implement or are used by a method

(VL/HCC'10)
Exploratory Programming and Understanding

- PhD work of YoungSeok Yoon (in progress)
- Explorations
  - When trying different approaches
  - When trying to understand an API
  - When trying out different fixes
  - ...

© 2013 – Brad A. Myers
Fluorite Logger

- **Fluorite**: Full of Low-level User Operations Recorded In The Editor [http://www.cs.cmu.edu/~fluorite](http://www.cs.cmu.edu/~fluorite)
- Logger for all keystrokes & events in Eclipse
- Analyzes frequencies and patterns
- Deleting is a high percent of all the keystrokes
- Also surveyed >100 developers

<table>
<thead>
<tr>
<th>Commands</th>
<th>Keystrokes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type char.</td>
<td>17092 (31.8%)</td>
</tr>
<tr>
<td>Line down</td>
<td>5795 (10.8%)</td>
</tr>
<tr>
<td>Delete prev.</td>
<td>5692 (10.6%)</td>
</tr>
<tr>
<td>Move caret</td>
<td>4686 (8.7%)</td>
</tr>
<tr>
<td>Line up</td>
<td>4491 (8.4%)</td>
</tr>
<tr>
<td>Col. next</td>
<td>3544 (6.6%)</td>
</tr>
<tr>
<td>Col. prev.</td>
<td>2715 (5.1%)</td>
</tr>
<tr>
<td>Select text</td>
<td>1975 (3.7%)</td>
</tr>
<tr>
<td>Sel. col. next</td>
<td>1035 (1.9%)</td>
</tr>
<tr>
<td>File open</td>
<td>907 (1.7%)</td>
</tr>
<tr>
<td>Sel. col. prev</td>
<td>857 (1.6%)</td>
</tr>
<tr>
<td>Save</td>
<td>852 (1.6%)</td>
</tr>
<tr>
<td>Delete</td>
<td>576 (1.1%)</td>
</tr>
<tr>
<td>Paste</td>
<td>459 (0.9%)</td>
</tr>
<tr>
<td>Assist(auto)</td>
<td>456 (0.8%)</td>
</tr>
<tr>
<td>Run</td>
<td>391 (0.7%)</td>
</tr>
<tr>
<td>Copy</td>
<td>314 (0.6%)</td>
</tr>
<tr>
<td>Undo</td>
<td>294 (0.5%)</td>
</tr>
<tr>
<td>Assist(manual)</td>
<td>213 (0.4%)</td>
</tr>
<tr>
<td>Sel. line down</td>
<td>212 (0.4%)</td>
</tr>
<tr>
<td>Others</td>
<td>1113 (2.1%)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>53669</strong></td>
</tr>
<tr>
<td><strong>Down arrow</strong></td>
<td><strong>5797 (13.7%)</strong></td>
</tr>
<tr>
<td>Backspace</td>
<td><strong>5693 (13.5%)</strong></td>
</tr>
<tr>
<td>Up arrow</td>
<td>4495 (10.6%)</td>
</tr>
<tr>
<td>Right arrow</td>
<td>3586 (8.5%)</td>
</tr>
<tr>
<td>Left arrow</td>
<td>2751 (6.5%)</td>
</tr>
<tr>
<td>Shift</td>
<td>1645 (3.9%)</td>
</tr>
<tr>
<td>Enter</td>
<td>1641 (3.9%)</td>
</tr>
<tr>
<td>T</td>
<td>1289 (3.1%)</td>
</tr>
<tr>
<td>E</td>
<td>1250 (3.0%)</td>
</tr>
<tr>
<td>S</td>
<td>1021 (2.4%)</td>
</tr>
<tr>
<td>N</td>
<td>1003 (2.4%)</td>
</tr>
<tr>
<td>I</td>
<td>881 (2.1%)</td>
</tr>
<tr>
<td>Space</td>
<td>859 (2.0%)</td>
</tr>
<tr>
<td>A</td>
<td>790 (1.9%)</td>
</tr>
<tr>
<td>O</td>
<td>750 (1.8%)</td>
</tr>
<tr>
<td>L</td>
<td>610 (1.4%)</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td><strong>576 (1.4%)</strong></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>42220</strong></td>
</tr>
</tbody>
</table>

(CHASE’12)
Backtracking Results

- All developers *backtrack* for many reasons
  - Explorations, investigations, iterative design
- Undo not used for exploration, just typo fixing
- People use comments to remove code, so they can restore it if necessary
  - But difficult to comment & uncomment correctly
  - Often non-local changes
- Current work: new tool to help developers backtrack
Azurite: Eclipse Plugin for Selective Undo

- PhD work of YoungSeok Yoon (in progress)
- **Azurite**: Adding Zest to Undoing and Restoring Improves Textual Exploration  
  [http://www.cs.cmu.edu/~azurite](http://www.cs.cmu.edu/~azurite)
- Work out semantics of selective undo for code
  - Conflicting edits of same code must be shown to user
- Time-line visualization of all past operations
- Side-by-side view of current and past code
- **Search** through history (time) to find appropriate points
Summary of Insights

- Field and lab studies can reveal developer’s real questions
  - Answering these questions creates tools that are actually useful
- Researcher’s intuitions about what might be useful are often wrong
- Our experience highlights:
  - Developers often have specific questions in mind, which can be exploited in tools
  - Code views are central
  - Visualizations are often useful as navigation aides for code
  - Ability to search is key
    - Not just through code, but also through dynamic and static call-graphs, through time, etc.
There are lots of Gemstones!

- **And acronyms are fun!**
Thanks to:

- **Funding:**
  - NSF under IIS-1116724, IIS-0329090, CCF-0811610, IIS-0757511 (Creative-IT), NSF ITR CCR-0324770 as part of the EUSES Consortium
  - SAP
  - Adobe
  - IBM
  - Microsoft Research RISE

- >30 students:
  - Htet Htet Aung
  - Jack Beaton
  - Ruben Carbonell
  - John R. Chang
  - Kerry S. Chang
  - Polo Chau
  - Luis J. Cota
  - Michael Coblenz
  - Dan Eisenberg
  - Brian Ellis
  - Andrew Faulring
  - Aristiwidya B. (Ika) Hardjanto
  - Erik Harpstead
  - Sae Young (Sophie) Jeong
  - Andy Ko
  - Sebon Koo
  - Thomas LaToza
  - Joonhwan Lee
  - Leah Miller
  - Mathew Mooty
  - Gregory Mueller
  - Yoko Nakano
  - Stephen Oney
  - John Pane
  - Sunyoung Park
  - Chotirat (Ann) Ratanamahatana
  - Christopher Scaffidi
  - Jeff Stylos
  - David A. Weitzman
  - Yingyu (Clare) Xie
  - Zizhuang (Zizzy) Yang
  - YoungSeok Yoon
Improving Program Comprehension by Answering Questions

Brad A. Myers
Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
http://www.cs.cmu.edu/~bam
bam@cs.cmu.edu