Improving Software Development through Human-Centered Approaches

Brad A. Myers
Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
http://www.cs.cmu.edu/~bam
bam@cs.cmu.edu
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 algorithms and solving their tasks (not “Natural UIs”)
- Methodology – human-centered approach
  - Perform studies to inform design
    - Provide new knowledge about what people do and think, & barriers
  - Guide the designs from the data
    - Design of programming languages and environments
  - Iteratively evaluate and improve the tools
- 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 \(~\$60\) billion annually
  – Software engineers spend 70-80% of time testing and debugging
  – Time for 1 developer to fix 1 bug was \(~17.4\) hours

• Current debugging techniques *same as for last 70 years*
  – Same for end-user and professional environments
Goal: Gentle Slope Systems

Web Development
Java
Visual Basic
Flash
Server-side

Low Threshold

High Ceiling

Program Complexity and Sophistication

Difficulty of Use

C or C# Programming
JavaScript
ActionScript
CSS & HTML
Basic
Email Filters
Editors

Goal
Improve Developer Experience

• Use human centered approaches to:
  ➢ Make developers more effective
  ➢ Reduce errors in resulting code
  ➢ Insure that developer tools are useful
  ➢ Understand developers’ barriers that cause wasted time
  ➢ Direct efforts at most important issues
  ➢ Address: programming languages, APIs, tools, documentation & resources
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

• Depends on target population
  – Need studies
Not so Natural!

```java
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

- 3 kinds of parentheses and 9 special words!
- Compared to click and type: "Hello World!"

Let `Shape1.FillColor = &H00FF00FF&`
First Natural Programming Studies

• John Pane, PhD 2002

• Studies:
  – How people *naturally* express programming concepts and algorithms
    1) Nine scenes from PacMan
    2) Transforming and calculating data in a spreadsheet
  – Specific issue of language design
    3) Selecting specific objects from a group (“and”, “or”, “not”)
  – Lots of interesting results
Examples of Results

• Rule-based style
  “If PacMan loses all his lives, its game over.”

• “And”, “Or”, “Not” don’t match computer interpretation
  – … men and women, …(not an apple) or pear

• Operations suggest data as lists, not arrays
  – People don’t make space before inserting

• Objects normally moving
  “If PacMan hits a wall, he stops.”
  – so objects remember their own state
New Language and System: HANDS

- John Pane, PhD 2002
- Properties:
  - Metaphor of agent (Handy the dog) operating on cards
  - All operations can operate on single items or sets of items
  - Integrated queries with language
  - Sets can be dynamically constructed and used
    - “Set the speed of all bees to 0”

See the video: http://web.cs.cmu.edu/~pane/HANDS/HANDS.MPG
Supporting “Natural” Data Types

- Chris Scaffidi, PhD 2009
- Ask users about types of data, say “Person name”, “age”, “date”, “Project code”, …
- User-centered type system called “topes”
  - Structured
  - Constraints on the values and parts
    - May be “always” or “usually” true
      - “USA phone area code never ends in 11”
      - “USA Last names usually start with a capital letter”
- Library for verifying & transforming values
  - Can be used from JavaScript for web and from VB for Excel
- Editor for specifying
Study of Errors

• Study of novice errors and debugging
  – Created a new model of barriers & kinds 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

© 2012 – Brad A. Myers
Whyline

- Andy Ko, PhD 2008
- Allow users to directly ask “Why” and “Why not”
Whyline User Studies

• Initial study:
  – Whyline with novices outperformed experts with Eclipse
  – Factor of 2.5 times faster
    • (p < .05, Wilcoxon rank sums test)

• Formal study:
  – Experts attempting 2 difficult tasks
  – Whyline over 3 times as successful, in $\frac{1}{2}$ of the time
Crystal

- **Crystal: Clarifications Regarding Your Software using a Toolkit, Architecture and Language**
- Apply WhyLine idea to regular desktop applications (Word 2003)
- Lots of complexity in powerful features that people generally like
- Ask “Why” about what recently happened
- Architecture: supports adding to application with small overhead
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 ’2012
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
A Programmer’s Working Set

- A collection of task-relevant code fragments
- In modern software development, dependencies are distributed and non-local
# 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>~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!**
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
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, ...
- Which programming patterns are most usable?
- Barriers to use 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 getting using factories in APIs
Object Method Placement

- Where to put functions when doing object-oriented design of APIs when multiple classes work together
  - `mail_Server.send( mail_Message )`
  - `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 ($p < 0.05$)
- Starting class can be predicted based on user’s tasks

![Graph showing Time to Find a Method](image)
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
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
SAP’s NetWeaver® Gateway Developer Tools

• Plug-in to Visual Studio 2010 for developing SAP applications
• We used heuristic evaluation and cognitive walkthroughs to evaluate early prototypes
• Our recommendations were quickly incorporated due to agile software development process
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
Jadeite: Improved JavaDoc

- Jadeite: Java API Documentation with Extra Information Tacked-on for Emphasis
  - 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

Most common way to construct:
- `SSLSocketFactory factory = ...;
  String host = ...;
  int port = ...;
  SSLSocket socket = (SSLSocket)factory.createSocket(host, port);
Based on 38 examples
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

```java
SSLSocket s = ???
```
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:** **GR**aphical **P**alettes **H**elp **I**nstantiate **T**ypes in the **E**ditor.
- Pop up a custom palette for specialized constants (literals) in Eclipse
  - Color palettes
  - Regular expression strings
- Customizable

(ICSE’2012)
Apatite Documentation Tool

- **Apatite**: Associative Perusing of APIs That Identifies Targets Easily
  - [http://www.cs.cmu.edu/~apatite](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
Studies of Code Understanding

• Thomas LaToza, PhD 2012
• Studies about how experts learn unfamiliar code
• Programmers investigate reachability questions
  – How can this code be reached, either upstream or downstream
  – E.g., control flow from user scrolling → update status line
• Identified over 100 hard-to-answer questions that developers asked
  – E.g., “What method implements this trigger?”
  – “Why was this designed this way?”
• Survey shows such control flow questions are difficult and important
• No easy way to discover with current tools
  – Call graphs are too general
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
Fluorite Logger

- PhD work of YoungSeok Yoon (in progress)
- **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

![Fluorite Logger Table](image)

<table>
<thead>
<tr>
<th>Commands</th>
<th>Keystrokes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type char.</strong></td>
<td>17092 (31.8%)</td>
</tr>
<tr>
<td><strong>Line down</strong></td>
<td>5795 (10.8%)</td>
</tr>
<tr>
<td><strong>Delete prev.</strong></td>
<td>5692 (10.6%)</td>
</tr>
<tr>
<td><strong>Move caret</strong></td>
<td>4686 (8.7%)</td>
</tr>
<tr>
<td><strong>Line up</strong></td>
<td>4491 (8.4%)</td>
</tr>
<tr>
<td><strong>Col. next</strong></td>
<td>3544 (6.6%)</td>
</tr>
<tr>
<td><strong>Col. prev.</strong></td>
<td>2715 (5.1%)</td>
</tr>
<tr>
<td><strong>Select text</strong></td>
<td>1975 (3.7%)</td>
</tr>
<tr>
<td><strong>Sel. col. next</strong></td>
<td>1035 (1.9%)</td>
</tr>
<tr>
<td><strong>File open</strong></td>
<td>907 (1.7%)</td>
</tr>
<tr>
<td><strong>Sel. col. prev.</strong></td>
<td>857 (1.6%)</td>
</tr>
<tr>
<td><strong>Save</strong></td>
<td>852 (1.6%)</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td>576 (1.1%)</td>
</tr>
<tr>
<td><strong>Paste</strong></td>
<td>459 (0.9%)</td>
</tr>
<tr>
<td><strong>Assist(auto)</strong></td>
<td>456 (0.8%)</td>
</tr>
<tr>
<td><strong>Run</strong></td>
<td>391 (0.7%)</td>
</tr>
<tr>
<td><strong>Copy</strong></td>
<td>314 (0.6%)</td>
</tr>
<tr>
<td><strong>Undo</strong></td>
<td>294 (0.5%)</td>
</tr>
<tr>
<td><strong>Assist(manual)</strong></td>
<td>213 (0.4%)</td>
</tr>
<tr>
<td><strong>Sel. line down</strong></td>
<td>212 (0.4%)</td>
</tr>
<tr>
<td><strong>Others</strong></td>
<td>1113 (2.1%)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>53669</td>
</tr>
<tr>
<td><strong>Down arrow</strong></td>
<td>5797 (13.7%)</td>
</tr>
<tr>
<td><strong>Backspace</strong></td>
<td>5693 (13.5%)</td>
</tr>
<tr>
<td><strong>Up arrow</strong></td>
<td>4495 (10.6%)</td>
</tr>
<tr>
<td><strong>Right arrow</strong></td>
<td>3586 (8.5%)</td>
</tr>
<tr>
<td><strong>Left arrow</strong></td>
<td>2751 (6.5%)</td>
</tr>
<tr>
<td><strong>Shift</strong></td>
<td>1645 (3.9%)</td>
</tr>
<tr>
<td><strong>Enter</strong></td>
<td>1641 (3.9%)</td>
</tr>
<tr>
<td><strong>T</strong></td>
<td>1289 (3.1%)</td>
</tr>
<tr>
<td><strong>E</strong></td>
<td>1250 (3.0%)</td>
</tr>
<tr>
<td><strong>S</strong></td>
<td>1021 (2.4%)</td>
</tr>
<tr>
<td><strong>N</strong></td>
<td>1003 (2.4%)</td>
</tr>
<tr>
<td><strong>I</strong></td>
<td>881 (2.1%)</td>
</tr>
<tr>
<td><strong>Space</strong></td>
<td>859 (2.0%)</td>
</tr>
<tr>
<td><strong>A</strong></td>
<td>790 (1.9%)</td>
</tr>
<tr>
<td><strong>O</strong></td>
<td>750 (1.8%)</td>
</tr>
<tr>
<td><strong>L</strong></td>
<td>610 (1.4%)</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td>576 (1.4%)</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>557 (1.3%)</td>
</tr>
<tr>
<td><strong>.</strong></td>
<td>546 (1.3%)</td>
</tr>
<tr>
<td><strong>R</strong></td>
<td>510 (1.2%)</td>
</tr>
<tr>
<td><strong>Others</strong></td>
<td>5970 (14.1%)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>42220</td>
</tr>
</tbody>
</table>
Backtracking Results

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

• 30 studies; 17 systems in 16 years

• Doing studies first provides new insights that can inspire significantly new designs for programming languages and environments

• Need to understand software engineers’ real issues

• New designs shown to be better
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
  • 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
Thank You!

Improving Software Development through Human-Centered Approaches

Brad A. Myers

Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University

http://www.cs.cmu.edu/~bam
bam@cs.cmu.edu
Better Tools for Authoring Interactive Behaviors: ConstraintJS

Brad Myers & Stephen Oney
Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
Interactive Software

• Today: programmed with callbacks & side effects
• Result: interdependent, complex code
Constraints

• Relationships declared once and maintained automatically
• Can help reduce the complexity of interactive code
• In GUI programming, constraints have caught on for:
  – Data bindings (example: WPF, Silverlight)
  – Layout controllers (example: CSS)
ConstraintJS

• Constraints for building interactive software
• Integrates constraints with Finite-State Machines (FSMs)
  – Makes it easy to create constraints that sometimes hold
  – Result: Cleaner, clearer code
• Works with Web languages (JavaScript, HTML, & CSS)
• (paper to appear at UIST’2012)
Motivating Example

Corey Smith

Ellyn Todd

Sarah Kelly

Keith Malcom

Karen Collins

Eric Marshall

© Carnegie Mellon - 2012
JavaScript implementation

• Requires:
  – Four nested callback functions using side-effects to handle asynchronous communication
    • Ensuring correct scoping for nested callbacks is difficult
  – Significant code to ensure view is in sync with model
  – Significant error handling code
ConstraintJS implementation

- Requires fewer callbacks and no side-effect code
- Clearer and less interdependent code
- Enhances HTML syntax to add flexibility while maintaining clarity

```javascript
friends = cjs.async(fb_request("/me/friends"));
pics    = friends.map(function(friend) {
    return cjs.async(fb_request( "/" + friend.id + "/picture"));
});

//...

{{#diagram friends.state}}
  {{#state pending }} Loading friends...
  {{#state rejected}} Error
  {{#state resolved}}
    {{#each friends friend i}}
      {{#diagram pics[i].state}}
        {{#state pending }} <img src = "loading.gif" />
        {{#state resolved}} <img src = "{{pics[i]}}" />
        {{#state rejected}} <img src = "error.gif" />
      {{/diagram}}
      {{friend.name}}
    {{/each}}
  {{/state resolved}}
{{/diagram}}
```
Current Work

- Many interactive behaviors can be specified using *only* a combination of FSMs and constraints
- Interactive tool for specifying FSMs & constraints
  - Spreadsheet-like for constraints, with columns for FSM states
Acknowledgements

• Microsoft SEIF Award, 2011
• Joel Brandt & Adobe
• Ford Foundation
• National Science Foundation

Website: www.constraintjs.com