Software Engineers are People Too: Applying Human Centered Approaches to Improve Software Development

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 programming 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
- 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 expert, novice and end-user programmers
End User Programming

- People whose primary job is *not* programming
- *[Scaffidi, Shaw and Myers 2005] (and confirmed recently)*
  - 90 million computer users at work in US
  - 55 million will use spreadsheets or databases at work (and therefore may potentially program)
  - 13 million will describe themselves as programmers
  - 3 million professional programmers
Goal: Gentle Slope Systems

Program Complexity and Sophistication

- **Low Threshold**
- **Goal**

**Web Development**
- Jupyter notebook
- Scikit-learn API
- Pandas
- Numpy API

**Machine Learning**
- Java
- Python
- CSS & HTML

**Machine Learning techniques**
- C or C# Programming
- Scikit-learn API
- Matplotlib.pyplot API
- Numpy API
- Pandas

**Visual Basic**
- Swing
- Python

**Client-side APIs**
- Server-side APIs

**Basic**
- Email
- Mail client
- Basic editor

**Difficulty of Use**
- High Ceiling
- Low Threshold
Why Human Centered Approaches?

- Developers are people
- HCI can impact **everything** the developer encounters:
  - **Tools** – IDEs & their user interfaces
  - **Languages** themselves
    - Not necessarily just “taste”, “intuition”
    - Error-proneness
  - **APIs**
    - “Interface” between developer and functionality
    - “Languages” by themselves can do very little these days
  - **Documentation** for all of the above
  - **Processes** & context of development
- New as well as legacy systems
- **DX** or **DevX** – developer experience (as in UX)
Dangers of *Not* Applying Human Centered Approaches

• Tools may show **no measurable impact**
  – Desired advantage overwhelmed by problems with other parts
  – Example: Emerson Murphy-Hill found that refactoring tools are under-utilized and programmers do not configure them due to usability issues
  

• Hard-to-use APIs and tools have resulted in **bugs and security problems**

• Usability and quality are **key influencers** for the decision about which APIs and tools to use
Many HCI Methods for answering different questions

- Contextual Inquiry
- Contextual Analysis
- Paper prototypes
- Think-aloud protocols
- Heuristic Evaluation
- Affinity diagrams
- Personas
- Wizard of Oz
- Task analysis
- A/B testing
- Cognitive Walkthrough
- Cognitive Dimensions
- KLM and GOMS (CogTool)
- Video prototyping

- Body storming
- Expert interviews
- Questionnaires
- Surveys
- Interaction Relabeling
- Log analysis
- Storyboards
- Focus groups
- Card sorting
- Diary studies
- Improvisation
- Use cases
- Scenarios
- “Speed Dating”
- Journey Maps
- ...

© 2018 – Brad A. Myers
Many HCI Methods

for answering

different

questions

• Contextual Inquiry
• Contextual Analysis
• Paper prototypes
• Think-aloud protocols
• Heuristic Evaluation
• Affinity diagrams
• Personas
• Wizard of Oz
• Task analysis
• A/B testing
• Cognitive Walkthrough
• Cognitive Dimensions
• KLM and GOMS (CogTool)
• Video prototyping

• Body storming
• Expert interviews
• Questionnaires
• Surveys
• Interaction Relabeling
• Log analysis
• Storyboards
• Focus groups
• Card sorting
• Diary studies
• Improvisation
• Use cases
• Scenarios
• “Speed Dating”
• Journey Maps
• ...

Human-Computer Interaction Institute
Product Lifecycle

Source: http://www.accordtech.co.in/Product%20Development%20Lifecycle.htm
Product Lifecycle

Field Studies
- Logs & error reports

Evaluative Studies
- Expert analyses
- Usability Evaluation
- Formal A/B Lab Testing

Formative Studies
- Contextual Inquiries
- Interviews
- Surveys
- Lab Studies
- Corpus data mining

Design Practices
- “Natural programming”
- Prototyping
Formative Studies

- Identify what is really happening
- Discover important problems
- Quantify need
- Develop use cases based on data
Contextual Inquiry (CI)


• Watch developers while they are performing their real tasks
• Objective, concrete data about real activities
• May be followed by a survey, to establish generality of the issues
• Reveals many barriers and problems in current practice
• API Designers told us that identifying developer’s real needs was a key barrier  
  [Murphy, Kery, Alliyu, Macvean & Myers, VL/HCC'18]
Example of Contextual Inquiry & Surveys

“Developers Ask Reachability Questions”
[Thomas D. LaToza, Brad Myers, ICSE’2010, Cape Town, South Africa, 2-8 May 2010. pp. 185-194.]

“Search across feasible paths through a program for target statements matching search criteria”

- Watched 17 developers investigating unfamiliar code
- Also surveyed 460 developers
- Over 100 other hard-to-answer questions
Many hard-to-answer questions about code

Rationale (42)

Why was it done this way? (14) [15][7]

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)

In what circumstances does this bug occur? (3) [15]

Which team’s component caused this bug? (1)

In what situations does this bug occur? (10)

How do I debug this bug in this environment? (3)

Why didn’t this happen? (3)

Is there functionality or code that could be refactored? (4)

Is it possible to refactor this? (9)

Is the existing design a good design? (2)

Should I refactor this? (1)

Are the benefits of this refactoring worth the time investment? (3)

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? (7)

Should I refactor this? (1)

Are the benefits of this refactoring worth the time investment? (5)

Testing (20)

Is this code correct? (6) [13]

Is the test or code responsible for this test failure? (1)

Is this tested? (3)

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/ post conditions can be made? (2)

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 (5) 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)

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 original source of this data? (2) [13]

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) [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)

Which preconditions 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)

History (23)

When, how, by whom, and why was this code changed or inserted? (12)

Has this code always been this way? (2)

Why didn’t this happen? (3)

Where have changes been made? (1)[15][7]

Have changes in another branch been integrated into this branch? (1)

Concurrency (9)

What threads reach this code (4) or data structure (2)? (6)

Is this class or method thread-safe? (2)

Which part of this code takes the most time? (4)

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)? (2) [15][24]

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)

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)

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)? (2) [15][24]
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
Example of Interviews: Immutability

- Experts recommend making classes *immutable* so instances cannot change accidentally [*Coblenz, et al, ICSE’2016*]
  - Thread safe, more secure, no unexpected state changes, etc.
- Many relevant language features
  - C++ *const*, Java *final*, Obj-C immutable collections, .NET *Freezable*, etc.
- Usability studies suggest programmers prefer classes that *can* change
- Semi-structured interviews with a convenience sample of 8 software engineers
  - Agreed that mutability is a frequent source of bugs
  - But *none* of these features are what is needed
  - Preferred *transitive, class-based immutability*
- So we provided this in the *Glacier* language extension [*Coblenz, et al, ICSE’2017*]
  - *Great* Languages *Allow* Class *Immutability* *Enforced* *Readily*
- User study showed *none* made errors enforcing immutability, vs *all* in control condition using Java *final*
Exploratory Lab Studies

- To understand what is happening
- More controlled than Contextual Inquiries
  - Can compare multiple people on same tasks
- Example: studying Eclipse for maintenance tasks
  - Detailed study of fixing bugs and adding features
  - Dataset used for 3 different award-winning papers: interruptions, navigation, code editing behaviors

<table>
<thead>
<tr>
<th>Interactive Bottleneck</th>
<th>Overall Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigating to fragment in same file (via scrolling)</td>
<td>~ 11 minutes</td>
</tr>
<tr>
<td>Navigating to fragment in different file (via tabs and explorer)</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>
Corpus Data Mining

• Studied 11 million Java try/catch blocks from GitHub using Boa tool
• 12% of catch blocks were completely empty. [Kery, Le Goues, & Myers, 2016]
• 25% of all exceptions caught are simply Exception
• Motivated a new tool to help programmers write better exception handling code
  – **Moonstone**: Making Object Oriented Novel Software Tools Optimized for Noting Exceptions [Kistner, Kery, Puskas, Moore & Myers, VL/HCC'17]

Figure 3: Exceptions caught by catch blocks on GitHub. Exceptions that occur more than 1% of the time are labeled. The rest, in purple, are thousands of exceptions that only rarely occur.
Data Mining Stack Overflow

- Want to help developers **organize** information after finding it
- Study what kinds of questions people express on Stack Overflow
  - 51% involved **decisions** with multiple **options** and **criteria**
- Motivated new web & IDE tool: **Unakite: Users Need Accelerators for Knowledge for Implementations in Technology Environments**
  - Programmers clip **snippets** of information from the web
  - Organizes information into a table
  - When paste into code, creates back pointers to the table as **design rationale**

[Liu, et al, 2019]
Design Methods

• Now know the problem, what is the solution?
• How do I design it so it is easy to learn and effective?
“Natural Programming” Elicitation Method

• Technique developed by my group to discover developer’s “natural” expressions
  – Mental models of tasks, vocabulary, etc.
• A form of participatory design
• Blank paper tests
• Must prompt for the tasks in a way that doesn’t bias the answers
• Examples:
  – PacMan before and after
    • Mostly rule-based (if-then)
  – API designs
    • Architecture, names used, which methods are on which classes
Example of use of Natural Programming

- **Obsidian** is a new domain specific language (DSL) for blockchains [Coblenz, et al, 2019]
  - Object-oriented Blockchain State Interaction and Development Implementation And Notation
- Combining state transition language ("TypeStates") with resources ("linear types") all checked statically
- 11 different NatProg studies on how to present these complex concepts
Prototyping

- Try out designs with developers before implementing them
  - Paper
    - “Low fidelity prototyping”
    - Often surprisingly effective
    - Experimenter plays the computer
    - Drawn on paper → drawn on computer
  - Implemented Prototype (“Click through”)
    - Balsamiq, Axure, PowerPoint, Web tools (even for non-web UIs)
    - (no database)
  - Real system

- Need to test these with users!
- Better if sketchier for early design
  - Use paper or “sketchy” tools, not real widgets
  - People focus on wrong issues: colors, alignment, labels
  - Rather than overall structure and fundamental design
Example of Early Prototyping

- Thomas LaToza designing new visualization tool to try to help answer Reachability Questions [LaToza & Myers, VL/HCC'11]
- Prototypes created with Omnigraffle and printed
- Revealed significant usability problems that were fixed
  - Basic concepts
  - Graphical presentation
  - Controls
Another Example: Variolite

- How to support data scientists with exploratory programming?
- What kind of version control support would be useful?
  - Interviews and CIs showed that conventional approaches like Git are too heavy-weight

- Showed dozens of sketches to target users to get feedback on which seemed usable and useful
- Final design appeared at CHI’2017

Variations Augment Real Iterative Outcomes Letting Information Transcend Exploration
Evaluation Methods

- Does my tool work?
- Does it solve the developer’s problems?
- How much better than alternatives?
- “If the user can’t use it, it doesn’t work!”
  – Susan Dray
Expert Analyses

- Usability experts evaluating designs to look for problems
  - Heuristic Analysis – [Nielsen] set of guidelines
  - Cognitive Walkthroughs – evaluate a task
- Can be inexpensive and quick
- However, experienced evaluators are better
  - 22% vs. 41% vs. 60% of errors found [Nielsen]
- Disadvantage: “just” opinions, open to arguments
Our Use of Expert Analyses

- Collaborating with SAP on their APIs and tools
- We studied SAP’s Enterprise Service-Oriented Architecture (eSOA) APIs & Documentation
  

- Naming problems:
  - Too long
  - Not understandable
Usability Evaluations

• Let representative target users use the system or a prototype of the system
• Also called “think aloud” protocols
• Different from formal A/B “user testing”
  – Understand usability issues
  – Get qualitative feedback about issues; not numbers
• Should be done early and often
  – Doesn’t have to be “finished” to let people try it
• Demonstrates that users can use the system
  – Show that novel features of the UI are understandable
Example Use of Usability Analysis

- **Sugilite**: Smartphone Users Generating Intelligent Likeable Interfaces Through Examples
- Allow end-users to create automations on Smartphones
- Initiate with speech commands
- Record scripts by example
- Generalizes from one or more examples
- 19 participants attempted 5 tasks
  - All completed at least 2 tasks successfully
  - 8 (42.1%) succeed in all 4 tasks
  - Overall, 65 out of 76 (85.5%) scripts worked
  - Feedback on what we need to improve

[Li, Azaria, & Myers, CHI'2017]
Formal A/B testing

- Formal A/B lab user tests are “gold standard” for academic papers – to show something is **better**
- But many issues in the study design
  - “Confounding” factors which were not controlled and are not relevant to study, but affect results
  - Tasks or instructions are misunderstood
  - Use prototypes & pilot studies to find these
- Statistical significance doesn’t mean real savings
- Be sure to collect qualitative data too
  - Strategies people are using
  - Why users did it that way
  - Especially when unexpected results
Example of A/B testing

• User testing of InterState compared to JavaScript
  [Oney, Brandt, Myers, UIST'2012]

```javascript
var ms_until_advance;
window.setInterval(function ()
  ms_until_advance = 5000 - \\
  if (diff <= 0) {
    set_selected_index((sel
    reset_timer();
  }

```

time taken
(minutes)

<table>
<thead>
<tr>
<th>Task 1*</th>
<th>Task 2**</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript</td>
<td>InterState</td>
</tr>
<tr>
<td>p &lt; .01**</td>
<td>p &lt; .05 *</td>
</tr>
</tbody>
</table>

smaller is better
Another Example of A/B testing

• **Gneiss:** Gathering **Novel End-user Internet Services** using **Spreadsheets** [Chang & Myers, CHI'2016]
  - Novel spreadsheet interface for investigating hierarchical (e.g., JSON) data
    - Investigate using conventional spreadsheet formulas and drag-and-drop of columns
  - Gneiss users significantly outperformed Excel users and programmers (p<.001)
Field Studies of System in Use

- Find out what happens when the tool is really used
- Requires significant effort to make the tool sufficiently solid
Logging Actual Use

- Easier if instrument your tools
- Objective use data better than users’ recollections and opinions
- Many levels of data can be collected
  - Privacy issues
- Easy to log and analyze web data
- Example: Fluorite logger for Eclipse
  - Fluorite: Full of Low-level User Operations Recorded In The Editor
  - Records all edits and events, including scrolling operations & source code,
  - Has been used by multiple studies

[Yoon & Myers, PLATEAU 2011]
Summary of Insights

• Formative field and lab studies can reveal the real questions
  – Answering these questions creates tools that are actually useful
• Human centered design methods help insure workable designs
• Researcher’s intuitions about what might be useful may be 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.
More on This Topic

• Our current tools: www.natprog.org


• Brad A. Myers and Jeffrey Stylos, "Improving API Usability", *Communications of the ACM*, vol 59, No. 6, June, 2016, pp. 62-69.

• In general: CHASE and WAPI workshops at ICSE; PLATEAU at SPLASH/OOPSLA; IEEE VL/HCC conference


**Acronyms are fun!**

And there are lots of Gemstones!!

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fluorite</td>
<td>Full of Low-level User Operations Recorded In The Editor</td>
</tr>
<tr>
<td>Azurite</td>
<td>Adding Zest to Undoing and Restoring Improves Textual Exploration</td>
</tr>
<tr>
<td>Euclase</td>
<td>Eclipse Users’ Keystrokes Lessened by Attaching from Samples</td>
</tr>
<tr>
<td>Variolite</td>
<td>Variations Augment Real Iterative Outcomes Letting Information Transcend Exploration</td>
</tr>
<tr>
<td>Apatite</td>
<td>Java API Documentation with That Identifies Targets Easily</td>
</tr>
<tr>
<td>Jadeite</td>
<td>Java API Perusing of APIs That Identifies Targets For Emphasis</td>
</tr>
<tr>
<td>Crystal</td>
<td>Clarifications Regarding Your Software using a Toolkit, Architecture and Language</td>
</tr>
<tr>
<td>Obsidian</td>
<td>Object-oriented Blockchain State Interaction and Development Implementation And Notation</td>
</tr>
<tr>
<td>Calcite</td>
<td>Construction And Language Completion Integrated Throughout</td>
</tr>
<tr>
<td>Mica</td>
<td>Makes Interfaces Clear and Accessible</td>
</tr>
<tr>
<td>Euclase</td>
<td>Eclipse Users’ Keystrokes Lessened by Attaching from Samples</td>
</tr>
<tr>
<td>Gneiss</td>
<td>Gathering Novel End-user Internet Services using Spreadsheets</td>
</tr>
<tr>
<td>Aquamarine</td>
<td>Allowing Quick Undoing of Any Marks And Repair Improving Novel Editing</td>
</tr>
<tr>
<td>Jadeite</td>
<td>Java API Perusing of APIs That Identifies Targets Easily</td>
</tr>
<tr>
<td>Jasper</td>
<td>Java Aid with Sets of Pertinent Elements for Recall</td>
</tr>
<tr>
<td>GARNET</td>
<td>Generating an Amalgam of Real-time, Novel Editors and Toolkits</td>
</tr>
<tr>
<td>Apatite</td>
<td>Java API Perusing of APIs That Identifies Targets Easily</td>
</tr>
<tr>
<td>Calcite</td>
<td>Construction And Language Completion Integrated Throughout</td>
</tr>
<tr>
<td>Mica</td>
<td>Makes Interfaces Clear and Accessible</td>
</tr>
<tr>
<td>Pebbles</td>
<td>PDAs for Entry of Both Bytes and Locations from External Sources</td>
</tr>
<tr>
<td>Sugilite</td>
<td>Smartphone Users Generating Intelligent Likeable Interfaces Through Examples</td>
</tr>
<tr>
<td>Gneiss</td>
<td>Gathering Novel End-user Internet Services using Spreadsheets</td>
</tr>
<tr>
<td>Aquamarine</td>
<td>Allowing Quick Undoing of Any Marks And Repair Improving Novel Editing</td>
</tr>
<tr>
<td>Jadeite</td>
<td>Java API Perusing of APIs That Identifies Targets Easily</td>
</tr>
<tr>
<td>Jasper</td>
<td>Java Aid with Sets of Pertinent Elements for Recall</td>
</tr>
<tr>
<td>GARNET</td>
<td>Generating an Amalgam of Real-time, Novel Editors and Toolkits</td>
</tr>
<tr>
<td>Apatite</td>
<td>Java API Perusing of APIs That Identifies Targets Easily</td>
</tr>
<tr>
<td>Calcite</td>
<td>Construction And Language Completion Integrated Throughout</td>
</tr>
<tr>
<td>Mica</td>
<td>Makes Interfaces Clear and Accessible</td>
</tr>
<tr>
<td>Pebbles</td>
<td>PDAs for Entry of Both Bytes and Locations from External Sources</td>
</tr>
<tr>
<td>Sugilite</td>
<td>Smartphone Users Generating Intelligent Likeable Interfaces Through Examples</td>
</tr>
<tr>
<td>Glacial</td>
<td>Great Languages Allow Class Immutability Enforced Readily</td>
</tr>
</tbody>
</table>

For more, see: [www.cs.cmu.edu/~bam/acronyms.html](http://www.cs.cmu.edu/~bam/acronyms.html)
Thanks to:

- **Funding:**
  - NSF under IIS-1814472, CCF-1814826, CNS-1423054, IIS-1314356, IIS-1116724, IIS-0329090, CCF-0811610, IIS-0757511, IIS-0329090, ITR CCR-0324770, IRI-9900452
  - SAP
  - Adobe
  - IBM
  - Microsoft Research
  - Yahoo! InMind
  - Google

- **>40 students & visitors:**
  - Oluwatosin (Tosin) Alliyu
  - Htet Htet Aung
  - Jack Beaton
  - Ruben Carbonell
  - John R. Chang
  - Kerry S. Chang
  - Polo Chau
  - Luis J. Cota
  - Michael Coblenz
  - Christian Dörner
  - Dan Eisenberg
  - Brian Ellis
  - Andrew Faulring
  - Aristiwidya B. (Ika) Hardjanto
  - Erik Harpstead
  - Jane Hsieh
  - Amber Horvath
  - Sae Young (Sophie) Jeong
  - Mary Beth Kery
  - Andy Ko
  - Thomas LaToza
  - Joonhwan Lee
  - Toby Li
  - Michael Xieyang Liu
  - Leah Miller
  - Steven Moore
  - Mathew Mooty
  - Gregory Mueller
  - Lauren Murphy
  - Yoko Nakano
  - Daye Nam
  - Stephen Oney
  - John Pane
  - Sunyoung Park
  - Michael Puskas
  - Marissa Radensky
  - Chotirat (Ann) Ratanamahatana
  - André L. Santos
  - Christopher Scaffidi
  - Jeff Stylos
  - David A. Weitzman
  - Yingyu (Clare) Xie
  - Zizhuang (Zizzy) Yang
  - YoungSeok Yoon
Software Engineers are People Too: Applying Human Centered Approaches to Improve Software Development

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