About 50 minutes
Engineering

Engineering is the discipline, art and profession of acquiring and applying technical, scientific and mathematical knowledge to design and implement materials, structures, machines, devices, systems, and processes that safely realize a desired objective or inventions.

– Wikipedia

Engineering

- Engineering is the discipline, art and profession of acquiring and applying technical, scientific and mathematical knowledge to design and implement materials, structures, machines, devices, systems, and processes that safely realize a desired objective or inventions.

  – Wikipedia

Interactive Programming Systems

- Focus on systems for developers:
  - Programming languages themselves
  - Application programming interfaces (APIs) such as Toolkits, Libraries, Software Development Kits (SDKs), Frameworks, etc.
  - Integrated Development Environments (IDEs)
    - Writing code
    - Debugging code
  - Documentation tools
Why Programmers?

- By 2012, 30% of new jobs may require programming skills – *US Department of Labor*

- But programming and learning to program have only become more complex, with increasingly complex APIs

- Increasing demand for programmers to be efficient and create correct code
What Knowledge?

- Engineering = “acquiring and applying ... knowledge to design and implement ... systems”

- Radical notion:

  Treating programmers as if they are people!
  Gain knowledge from studying them!

- Use proven human-computer interaction (HCI) methods to study programmers
  - Professional programmers
  - Novice programmers
  - End-User Programmers (EUP)
End User Programming

- People whose primary job is *not* programming
- There are lots! — Scaffidi, Shaw and Myers 2005
  - 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
- We should make better tools for all of these people!

(based on data from US Bureau of Labor Statistics)


Not so Natural!

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&

Java
Visual Basic
Originally appeared in:
HCI Techniques We Have Used

- Discovering the real problems:
  - Contextual Inquiry
  - Surveys
  - Lab studies of existing systems

- Create models from results

- Prototypes of novel approaches

- Lab studies of the prototypes

- Can get *real measurements* for the effectiveness of the tools (=Engineering!)

- Don’t have to rely just on intuition
“Contextual Inquiry” method
- [Beyer & Holtzblatt, 1998]
- Study programmers while doing actual tasks
- CIs can help discover what the real problems are
- Surveys can document how wide-spread, and how important people think this problem is.


Left out: Scaffidi study of information workers
Novices Programming in Alice

- Study of novice errors in programming
  - 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
    - Two-thirds of novices’ bugs were inserted due to incorrect guesses

Brad A. Myers, CMU

Model of invalid assumptions led to what other kinds of barriers. Don’t include barriers where use stuck (insurmountable barriers)
Professional Programmers

- Studies at Microsoft, e.g.:
  - [LaToza 06]
    - Percent of time on tasks
    - Reliance on implicit code knowledge
    - Face to face preferred for getting design rationale
  - [Ko’07]
    - Many task switches, about every 5 minutes
    - 21 different information types needed
      - Often unavailable, especially when from co-worker


Lab Studies

- “Think aloud” protocols in lab of target populations doing “realistic” tasks
  - Of existing systems
- Gain deeper insight into details of the problems
- Observations that would not arise from surveys, introspection and field studies
- Natural expression for programming concepts

Note: not of a new prototype system

Reference for Think-aloud:
Examples of Results

- Rule-based style
  "If PacMan loses all his lives, its game over."

- Set operations instead of iterations
  "When PacMan eats all of the dots, he goes to the next level."

- "And", "Or", "Not" don’t match computer interpretation
  - ... men and women, ...(not an apple) or pear

- Most arithmetic used natural language style
  "When PacMan eats a big dot, the score goes up 100."

- 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
To better illustrate a working set, here is all of the code in the Paint program. The grey regions indicate the code that one programmer included in their working set as part of the stroke thickness task.

This makes sense: in modern software development, dependencies are distributed and non-local.

Given this 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 helped or hindered them.

## 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 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>

= **35% of uninterrupted work time!**
Object Method Placement

- Where to put functions when doing object-oriented design of APIs
  - 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

Create & Test Prototypes

- Address the observed barriers
- More likely to solve real problems
- Test the prototypes in user studies
  - Measure impact compared to existing situation
- But most importantly first...
See http://www.cs.cmu.edu/~bam/acronyms.html
Whyline

- Allow users to directly ask “Why” and “Why not”
- Collect a trace and replay
- Whyline = Workspace that Helps You Link Instructions to Numbers and Events
- Ask about specific output that is in error
  - Accurate about what is wrong, but not when guess why
- Visualization of combined control and data flow
- Highlighting of relevant parts of the code
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 things associated with other things
  - E.g., classes that are often used together
  - Classes that implement or are used by a method

The right diagram shows a tree structure with classes and their relationships, such as:
- Packages
  - java.lang
  - java.util
- Classes
  - Object
  - String
  - Arrays
  - Properties

The left diagram appears to be a visual representation of the above concepts, possibly showing a search or classification process.

The right diagram includes comments like:
- "The points method is often used in..."
Euclase tool for authoring behaviors

EUCLASE = End User Centered Language, APIs, System, and Environment
Conclusions

- HCI techniques can provide significant new knowledge about developers
- Use this knowledge to guide new systems
- HCI techniques can provide evidence that systems work
Thanks to:

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

- Many students:
  - Htet Htet Aung
  - Jack Beaton
  - Ruben Carbonell
  - John R. Chiang
  - Polo Chau
  - Luis J. Cota
  - Michael Coblenz
  - Dan Eisenberg
  - Brian Ellis
  - Andrew Fauling
  - Aristiwidya B. (Ika) Hardjanto
  - Sae Young (Sophie) Jeong
  - Andy Ko
  - Thomas LaToza
  - Joonhwan Lee
  - Leah Miller
  - Mathew Moity
  - Gregory Mueller
  - Yoko Nakano
  - Stephen Oney
  - John Pane
  - Sunyoung Park
  - Choulral (Anri) Ralanamialatana
  - Christopher Scattidi
  - Jeff Stylos
  - David A. Weitzman
  - Yingyu (Clare) Xie
  - Zizhuang (Zizzy) Yang
About 65 minutes
SLIDES NOT USED – NOT ENOUGH TIME
Required Constructors

- Compared create-set-call (default constructor)
  ```javascript
  var foo = new FooClass();
  foo.Bar = barValue;
  foo.Use();
  ```

- vs. required constructors:
  ```javascript
  var foo = new FooClass(barValue);
  foo.Use();
  ```

- All participants assumed there would be a default constructor
- Did not insure valid objects – passed in null
- Preferred to not use temporary variables


Topes, cont.

- Toped: Interactive editor for specifying Topes
- Can learn from examples or directly specified
- User test
  - Successful for novices
  - Significantly better at finding errors compared to regular expressions or previous research tools
  - Compared to manual checked, user generated topes were quite accurate
  - Faster to author a tope and check when >50 items to check
- Topes were able to easily express most formats found across a corpus of web forms and spreadsheets