End-User Design and Development

Brad Myers
Human Computer Interaction Institute
Carnegie Mellon University

www.bradamyers.com

Copyright © 2009 – Brad A. Myers
Definitions

- "Program"
  - "A set of statements that can be submitted as a unit to some computer system and used to direct the behavior of that system"
    - *Oxford Dictionary of Computing*

- "Programming"
  - "The process of transforming a mental plan of desired actions for a computer into a representation that can be understood by the computer"
    - *Jean-Michel Hoc and Anh Nguyen-Xuan*
Definitions, cont.

- "Professional Programmer"
  - Someone whose primary job function is to write or maintain software
  - Typically have significant training in programming (e.g., BS in CS)

- "Novice Programmer"
  - Someone who is learning to be a professional programmer
“End-User Programmer” (EUP)

- People who write programs, but *not* as their primary job function
- Instead, they must write programs in support of achieving their main goal, which is something else
- Covers a wide range of programming expertise
  - Business executives and secretaries
  - Physicists
Definitions, cont.

“Design”

- Deciding things about the program
- What to program
  - “Getting the right design” – Buxton
- How the program will work
  - Architecture, algorithms
- User interface from the program
  - “Getting the design right” – Buxton
Definitions, cont.

- "Development"
  - All the activities involved in software (except use)
  - Includes design and programming
  - Also, requirements analysis, testing, documentation, management, etc.

- "End-User Development" (EUD)
  - End users doing any of these activities themselves
Other Names

- Europeans prefer “End User Development” (EUD)
  - As in European Commission’s (2002-3)

Some “Domain-Specific Languages” (DSL)

- Often created for end-user developers

Visual (Graphical) Programs

- Sometimes created for EUD

“Scripting” languages, “Macros”
Examples of EUD

- Accounting (spreadsheets)
- Analysts using MatLab
- Creating a web page with an interactive tool
- Recording Macros in Word
- Automating office tasks
- Business software (SAP programming)
- “Programming” VCRs, Microwaves
- Scientific research
- Kids authoring simulations and interactive worlds
- Creating email filters
- Musicians configuring synthesizers
- Entertainment (e.g., behaviors in The Sims)
- Web 2.0: Mashups
How Many EUDs?

- A very large group
- In 2012: — [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

*(based on data from US Bureau of Labor Statistics)*
Languages Being Used

- For the 12 millions self-described programmers
- Caveats:
  - “Popularity”, not LOC
  - Doesn’t count the 50,000,000 spreadsheet programmers
  - Doesn’t count interactive web authoring as “programming”
  - Cobol → SAP’s ABAP

Languages for Interaction Designers

- [Myers 2008] survey

Languages:
- HTML
- Javascript
- PHP
- Actionscript for Flash
- VisualBasic
- Java
- C++
- Lingo for Director
- .Net
- C#
- Ruby on Rails
- CSS
- Processing
- Perl
- Cocoa & Objective C
- XML
- Python
- ASP
- Pascal
- SQL
- HyperTalk
- JSP
- VRML
- Flex
- Matlab

Copyright © 2009 – Brad A. Myers
History

- Long History:
  - Original HCI!
    - 1971 “Psychology of Computer Programming”
  - “Software Psychology”
    - Ben Shneiderman book, 1980
  - “Empirical Studies of Programming” (ESP)
    - Workshops from 1986 through 1999
  - “Psychology of Programming”
    - Psychology of Programming Interest Group (PPIG)
      - from 1987 and PPIG’09 = 21th workshop
- But mostly focused on novice or professional

Copyright © 2009 – Brad A. Myers
Allen Newell and Stuart Card, 1985:

“Millions for compilers but hardly a penny for understanding human programming language use. Now, programming languages are obviously symmetrical, the computer on one side, the programmer on the other. In an appropriate science of computer languages, one would expect that half the effort would be on the computer side, understanding how to translate the languages into executable form, and half on the human side, understanding how to design languages that are easy or productive to use. The human and computer parts of programming languages have developed in radical asymmetry.”
Renewed Interest Recently

- New book from Springer:

- Significant numbers of papers at CHI, ICSE, VL/HCC, UIST and many other conferences!
  - 12 presentations at CHI’09
  - Much of IEEE VL/HCC

- *This overview!* 😊
Consequences of Lack of Attention

- Lots of errors attributed to End-User Programming of spreadsheets:
  - Columbia Housing Authority admitted to overpaying by $118,387 due to a spreadsheet error (Feb. 22, 2006)
  - Auditor, major accounting firm:
    “...in 6 years work, checking literally hundreds of business-critical models, ... my team have never failed to find errors.”
  - .... (many more!)
  - See http://eusesconsortium.org/euperrors/
Why is Programming Difficult?

- Some difficulty may be intrinsic to programming
  - Problem solving
  - Precise specification of algorithms
  - Concepts like abstraction, recursion

- How much difficulty can be attributed to usability problems?
  - Programming languages, IDEs, APIs are all user interfaces
  - Most designs do not emphasize (or even consider!) usability
Unnecessary (?) Complexity

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!”
Evidence That Difficult

- End User Programming is still research goal
- Researchers have tried many approaches
  - Surveyed next
- Many commercial attempts have moved away from addressing end users
  - E.g., Visual Basic & ActionScript for Flash
    - Leave “programming” to the professionals
    - Clever interactive tools for the designers
  - Increasing language complexity and features
Goal: Gentle Slope Systems

Program Complexity and Sophistication

- Swing
- Backend Programming
- XCmds
- ActionScript
- HyperTalk
- Basic
- HyperCard
- Flash
- Visual Basic
- Java

Difficulty of Use

Low Threshold

High Ceiling

Copyright © 2009 – Brad A. Myers
Outline

- NOTE: Not Comprehensive
- Empirical studies of programming
  - Novices, professionals, EUD
- Approaches for EUD:
  - Visual Programming
  - Programming by Example
  - Simpler Textual Languages
  - Better Development Environments
- Recent: Focus on Reliability
  - End-User Software Engineering (EUSE)
Empirical Studies of Programming

- Studies of why programming difficult to learn
  - Identified collections of issues with languages
  - Most are relevant to EUP
  - Survey of early work [Pane 1996]
Techniques for Studies

- Many observations and intuition based on teaching
- Green & Petre’s “Cognitive Dimensions” [1996]
  - 13 criteria for evaluating programming systems
  - E.g., “Viscosity” – how hard to change
  - “Consistency”, “Premature Commitment”, etc.
  - Low-cost analytical tool
  - Very influential in a number of systems
- “Natural Programming” approach [Myers]
  - See how people think about a task
  - Design a tool to support the way they are thinking
  - Evaluate how well the tool works with user studies
- Lab and field studies
Our Empirical Studies

- [Pane and Myers, 2001]: how people express algorithms

**Usually Pacman moves like this.**

**Now let's say we add a wall.**

**Pacman moves like this.**

**Not like this.**

**Do this:** Write a statement that summarizes how I (as the computer) should move Pacman in relation to the presence or absence of other things.
Examples of Results [Pane]

- Rule-based style
  
  “If PacMan hits the wall, he stops.”

- 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
  - ... many others
Barriers in Novice use of VB

- Studied 40 novices using Visual Basic.NET [Ko & Myers 2004]
- Analyzed 74 barriers that were not able to overcome
  - Design – inherently hard algorithm, e.g., sorting
  - Selection – can’t find how to do it
  - Use – can’t figure out how it is used
  - Coordination – how to use 2 things together
  - Understanding – what just happened?
API Usability Studies

- PhD work of Jeff Stylos
- Libraries, frameworks, SDKs
- Which programming patterns are barriers to usability?
- Measures: learnability, errors, preferences
- Findings:
  - Indirect creation of objects has penalty of taking 2.1 to 5.3 times longer
  - When desired method is on the class that they start with, users were between 2.4 and 11.2 times faster
  - Documentation can overcome some of these problems
Design of Interactive Behaviors

- Contextual Inquiry field study of 13 interaction designers, followed by survey of 231
- Prototyping behaviors is harder than the appearance
- Sketches (used by 97%) and storyboards (88%), are popular, but not adequate
  - Annotations are key
- Designer explore *alternatives*
  - Multiple versions sketched
  - Design emerges through creation, exploration
  - More difficult with behaviors
- Desired behaviors are quite complex
Outline

- Empirical studies of programming
  - Novices, professionals, EUP

- Approaches:
  - Visual Programming
  - Programming by Example
  - Simpler Textual Languages
  - Better Environments

- Recent: Focus on Reliability
  - End-User Software Engineering (EUSE)
Visual Programming

- Harness human visual system
  - Should be more “natural”
- Avoid syntax
- People were already using graphical notations
  - Flowcharts and Data flow, State-Transition Diagrams, Wiring Diagrams, Petri nets, etc.
  - Code using these directly
  - Graphical Program Editor [William Sutherland 1966]
Pict

- [Glinert 1984]
- Flowchart
- Only 4 variables
- Animate execution
National Instruments Labview

- 1986
- And today
Lego Mindstorms

- 1998
- “Nxt” version released Fall’06
  - Powered by LabView
KidSim/Cocoa/Stagecast Creator

- [Smith, Cypher & Spohrer, 94]
- Before and after pictures
- Stagecast '97

Copyright © 2009 – Brad A. Myers
Scratch

- Mitch Resnick’s (MIT Media Lab), 2003
- New language for kids
- Jigsaw pieces fit together
- Explicit support for sharing
- 470,969 projects, 11,894,895 scripts, 71,767 contributors, 319,484 members
Studies of VP

- Formal studies show some benefits for novices
- But:
  - Not a panacea: every notation has advantages and disadvantages
  - Graphical programs are no better for understanding than text [Green 91, 92]
  - Visual programs are usually very difficult to edit (“high viscosity”) [Green 96]
  - Take more space than text
Outline

Empirical studies of programming
  - Novices, professionals, EUP

Approaches:
  - Visual Programming
  - Programming by Example
  - Simpler Textual Languages
  - Better Environments

Recent: Focus on Reliability
  - End-User Software Engineering (EUSE)
Programming by Example

- Create program by performing the steps by example
  - Assumes user knows how to do the problem concretely
  - Avoids problems of abstraction
  - [Cypher 93], [Lieberman 2001]
PBE Examples:

- Pygmalion [Smith 77]
- Smallstar [Halbert 81, 84]
- Tinker [Lieberman 82]
- Peridot [Myers 86]
- Comic strip:
  - Chimera [Kurlander 93]
  - Pursuit [Modugno 93]
- Gamut [McDaniel 98]
Evaluation of PBE

- Systems often need examples of *different* cases
  - People are not good at giving good examples
- Sometimes by example is harder than expressing desired result: sorted, A AND B
- Need a way to *represent* code for confirmation, understanding, editing
  - If can understand code, why not just write it
  - Most of coding is editing, not starting from scratch
Outline

- Empirical studies of programming
  - Novices, professionals, EUP
- Approaches:
  - Visual Programming
  - Programming by Example
  - Simpler Textual Languages
  - Better Environments
- Recent: Focus on Reliability
  - End-User Software Engineering (EUSE)
Simpler Textual Languages

- Basic (1963)
- Logo (1966)
- Pascal (1970)
- Hypertalk (1987)
- Chickenfoot (2005)

```plaintext
10 INPUT "What is your name: "; U$
20 PRINT "Hello "; U$
30 REM
40 INPUT "How many stars do you want: "; N
50 S$ = ""
60 FOR I = 1 TO N
70 S$ = S$ + "*"
80 NEXT I
90 PRINT S$
```

to spiral :size :angle
if :size > 100 [stop]
forward :size
right :angle
spiral :size + 2 :angle
end

spiral 0 91
HyperTalk

- Created in 1987 for Apple’s HyperCard by Bill Atkinson
- Targeted at EUP
- Programmers were called “authors” and programs called “scripting”
- Event-based programming model
- HyperTalk designed to be similar to English
  - Studies inconclusive on whether this helps
  - Lots of problems with consistency
- Evolved into AppleScript

```
o on mouseUp
  put "100,100" into pos
  repeat with x = 1 to the number of card buttons
    set the location of card button x to pos
    add 15 to item 1 of pos
  end repeat
end mouseUp
```
**HANDS**

- PhD of John Pane, 2002
- Designed based on studies

**Properties:**
- All data visible on *cards*
- Metaphor of agent (Handy the dog) operating on cards
- Natural language style for code
- Domain-specific operations, like movement in a direction
- All operations can operate on single items or sets of items
- Sets can be dynamically constructed and used
  - “Set all bees direction to 90”

Copyright © 2009 – Brad A. Myers

---

```
for (int i = 0; i < 10; i++)
    cout << i << endl;
```
Chickenfoot

- [Bolin and Miller, 2005]
- EUP for the web
  - Automating repetitive operations
  - Integrating multiple web sites
  - Transforming a web site's appearance
- Simpler version of JavaScript
  - Adds pattern-matching to find parts of web page
- Descendant = Allen Cypher’s CoScripter language
Outline

- Empirical studies of programming
  - Novices, professionals, EUP
- Approaches:
  - Visual Programming
  - Programming by Example
  - Simpler Textual Languages
  - Better Environments
- Recent: Focus on Reliability
  - End-User Software Engineering (EUSE)
Better Environments

- Integrated development environment (IDE)
  - Help with creating, maintaining, debugging code
  - Somewhat independent of the particular language
- Interactive tools to (partially) avoid programming
  - “Interface Builders”, “Resource Editors”
Better Programming Environments

- Original: Cornell Program Synthesizer, 1981
- Structured Editing
  - MacGnome, 1988
  - Alice, 2002
  - Drag-and-drop program parts
  - Pop-up menus for parameters
  - V-3 now
  - Caitlin Kelleher will cover in depth
Structured Editing Studies

- Studies show such editors can help novices construct correct programs
- Acquiring language syntax is a barrier to novices, especially for children
- But, make it very difficult to edit programs after created
  - E.g., re-organizing code, re-using arbitrary-size pieces
WhyLine

- Debugging tool [Ko & Myers, 2004]
  - Surprising lack of support for debugging, even EUP
  - All of the observed debugging problems in our study could be addressed by “Why” questions
    - 32% were “why did”; 68% were “Why didn’t”
    - Allow directly asking these questions in the UI
  - Answers use visualization and highlighting of code
  - Versions for Alice 2 and full Java
  - Debugging 3 to 8 times faster

Question: Why didn’t Pac resize 0?

Answer: Actually, Pac resize 0 did happen. Maybe it looked like it didn’t for some reason?
Interface Builders

- Allow graphical parts of interface to be created using an interactive tool
  - Menulay – [Buxton 1983]
  - Peridot – [Myers 1986]
  - HyperCard, 1987
  - Director, 1988
  - Visual Basic, 1991
  - Flash Catalyst, 2009 – Ethan Eismann will cover in depth
- Drawing is up to 125 times faster than coding (4 min vs. 500 min)
- Limited to only parts of the development
Outline

- Empirical studies of programming
  - Novices, professionals, EUP
- Approaches:
  - Visual Programming
  - Programming by Example
  - Simpler Textual Languages
  - Better Environments
- Recent: Focus on Reliability
  - End-User Software Engineering (EUSE)
End-User Software Engineering

- Initiative to make software created by end users more reliable and correct
  - Bring “Software Engineering” principles to end users
    - But not necessarily SE methods
    - EUP will not follow strict processes, etc.
- Founded by Burnett, et. al. ~2002
  - NSF ITR 2003-2007
  - End Users Shaping Effective Software = EUSES consortium. www.eusesconsortium.org
- Workshops and SIGS on EUSE
  - WEUSE 1 - 5
EUSE Example

- WYSIWYT [Rothermel 2000]
  - What you see is what you test

System can figure out more assertions

User can enter assertions

There’s got to be something wrong with the formula!
Conclusions

- Increasing need to automate our systems
  - Increase productivity
  - Control our complex world
  - Author interesting behaviors

- Programming still too hard for most people
  - How can it be made easier?
  - Is there a way to avoid or to make understandable abstraction, iteration, conditions, recursion and other concepts?
  - How much can be covered by interactive tools?

- Will Artificial Intelligence (AI) help?
  - Reduce need for programming?

- Still enormous opportunities for research and new ideas
Thank You!

End-User Design and Development

Brad Myers
Human Computer Interaction Institute
Carnegie Mellon University

www.bradamyers.com

NPUC: 09 - The Future of Design and Software Development
Thursday, July 9, 2009 - IBM Almaden Research Center