Invited Research Overview:

# End-User Programming

Brad Myers, Andrew Ko, and Margaret Burnett

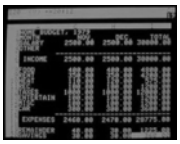
Human Computer Interaction Institute
Carnegie Mellon Univ.

School of Elec. Engr. & Computer Science
Oregon State Univ.



## Empowering Users

- One of the key features of computers is *programmability*
  - Perform the specific actions desired
  - But only if know how
- Spreadsheets enable people to define their own computations
  - Invented late 1970's
  - One of the key reasons personal computers became popular for business



- How to generalize to other areas?

## Malleability is Key Today

- Hottest new thing on the web is end-user *authoring*
  - Blogs
  - Flickr
  - MySpace
- Key is personalization
  - End users shape the artifact
- Raises expectations for the level of personalization, customization generally

## 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

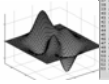
 

## Definitions, cont.

- "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

## Examples of EUP



- Accounting (spreadsheets)
- Analysts using MatLab
- Creating a web page
- Recording Macros in Word
- Automating office tasks
- Business software (SAP programming)
- "Programming" VCRs, Microwaves
- Scientific research
- Authoring educational software
- Creating email filters
- Musicians configuring synthesizers
- Mashups
- Entertainment (e.g., behaviors in The Sims)

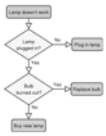


## Other Names

- Also called "End User Development" (EUD)
  - As in European Commission's

NETWORK OF EXCELLENCE ON END-USER DEVELOPMENT

- Some "Domain-Specific Languages" (DSL)
  - Often created for end-user programmers
- Visual (Graphical) Programs
  - Sometimes created for EUP
- "Scripting" languages, "Macros"
- Rapid Application Development (RAD)

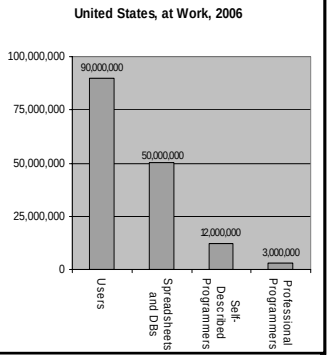


## How Many Today?

- Most people who write programs today are not professional programmers

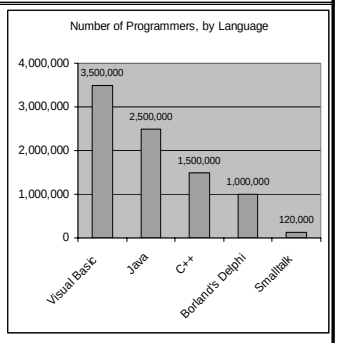
*(based on data from US Bureau of Labor Statistics)*






## Languages Being Used

- For the 12 millions self-described programmers
- Caveats:
  - Probably outdated
  - Doesn't count the 50,000,000 spreadsheet programmers
  - Cobol → SAP, etc.
  - .Net (C#) is rising



## History



- Long History:
  - *Original* HCI!
    - 1973 "Psychology of 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'06 = 18th workshop
- But mostly focused on novice or professional



## 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


- Significant numbers of papers at CHI, VL/HCC, ICSE, UIST and many other conferences!
- New book from Springer
- Areas like End-User Software Engineering (EUSE)
  - End-users *are* and *will* program
  - How to make their software more reliable?
  - EUSES – NSF funded consortium
  - 3 papers and a workshop at CHI'06
- *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 data-entry error (February 22, 2006)
  - New York Times, Oct 30th, 2003 - $1.2 Billion Spreadsheet Error at Fannie Mae
  - TransAlta Corp. took $24 million charge to earnings due to cut-and-paste error in an Excel spreadsheet (June 3rd, 2003)
  - 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/

## Consequences, 2

- Also, errors in:
  - Web pages
  - Email filtering rules
- From the WEUSE II workshop:
  - Clinical customization package used by medical personnel reports the need for better reuse and debugging support
  - SysAdmins need better testability of database and other sorts of scripts
  - Issues with reuse of MATLAB applications
- Difficulty of *learning*
  - Potentially millions of people who try to learn HTML, Flash, Visual Basic, Javascript, spreadsheets, etc., but give up because of one or two insurmountable errors

## Why is Programming Difficult?

- Some difficulty may be intrinsic to programming
  - Problem solving
  - Precise specification of algorithms
- How much difficulty can be attributed to usability problems?
  - Programming languages are a kind of user interface
  - Most language designs do not emphasize usability

## 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 & Flash
  - Increasing language complexity and features

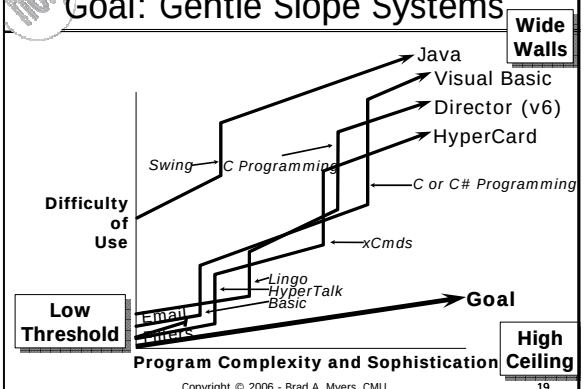
## Hello World!

```
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"

## Goal: Gentle Slope Systems



## Outline

- *NOTE: Not Comprehensive*
- 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)

## Empirical Studies of Programming

- Studies of why programming difficult to learn
  - Identified collections of issues with languages
  - Mostly relevant to EUP
  - Survey: [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 that can avert a lot of problems at design time
  - Very influential in a number of language/environment design efforts
- "Natural Programming" approach [Myers, Pane, Ko]
  - 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

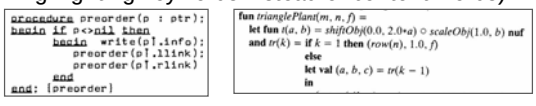
## Empirical Studies of Programming, cont.

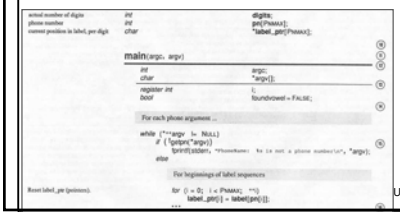
- Many Syntax Problems observed
  - E.g., `if (a = 0)`
- Small typos easily missed → wrong programs
  - Incorrect indentation → code is in a control structure [du Boulay1989a].
  - Syntax, idioms, strategies for programming & debugging
- Wrong words: STOP doesn't mean halt & exit (Logo) [Kurland 1989]

## Empirical Studies of Programmers, cont.

- Inappropriate formatting hinders reading (e.g., highlighting keywords instead of content words)




Baecker, R.M. and Marcus, A., *Human Factors and Typography for More Readable Programs*, ACM Press, Addison-Wesley, 1990

## Empirical Studies of Programmers, cont.

- Many studies about the differences between novices and experts
  - E.g., experts know more "schemas" or "plans" and how to put them together [Soloway]
    - E.g., Running-Total-Loop Plan (sum up a set of numbers); Dirty-Bit Flag Plan (a flag is set if some data needs to be rewritten out to disk)
  - Novices do not know debugging strategies

## More studies, cont.

- Incremental testing important to understanding
  - Rapid test, revise cycle with good feedback
    - Spreadsheets provide immediate feedback
- Appropriate metaphor important
  - "von Neumann machine" model has no physical world counterpart, which is an important stumbling block for novices [du Boulay]
    - E.g., variables as "box", but can't hold more than one value
    - Value still in J after I = J [Putnam 1989, Sleeman 1988]
  - Spreadsheet metaphor works better [Lewis 1987]

## More Recent Empirical Studies

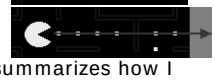
- [Pane and Myers, 2000]: how people express algorithms

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
- 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

## 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?

## 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.
  - Use these directly

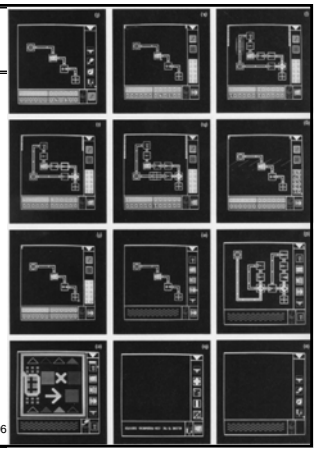
## Examples of Visual Programming

- Flowcharts and Data flow
  - Earliest: Grail [Ellis, 1969]
  - Pict [Glinert 1984]
  - Prograph [Pietrzykowski 84]
  - LabView [National Instruments, 1986]
  - Lego Mindstorms [1998]
  - Apple’s Automator
- Spreadsheet systems
  - Forms3
- Before and after pictures
  - Agentsheets [Repenning 91]
  - Kidsim/Cocoa/Stagecast Creator [Smith 94]
- Studies of VP – Green & Petre

## Pict

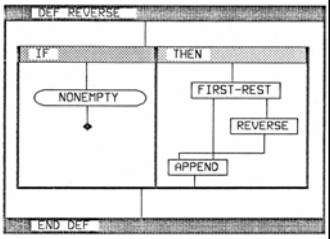
- [Glinert 1984]
- Flowchart
- Only 4 variables
- Animate execution





## Prograph

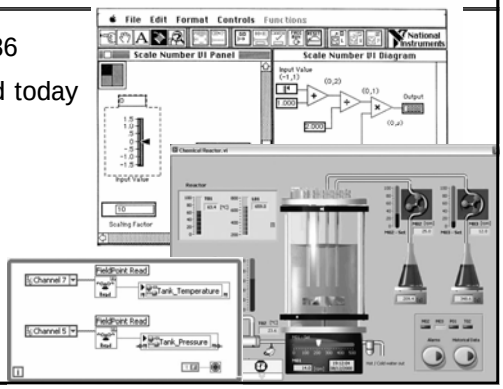
- Innovative data-flow format
- 1983
- TGS → Prograph, Inc → “Pictorius” → ☹



## National Instruments Labview

- 1986
- And today




## Lego Mindstorms

- 1998
- “Nxt” version coming fall’06
  - “Powered by LabView”

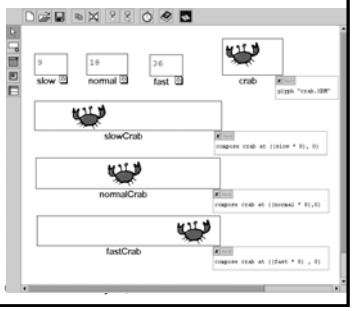
## Apple's Automator

- 2005
- Sequence of operations
- Transform data
- No control structures



## Spreadsheet Systems

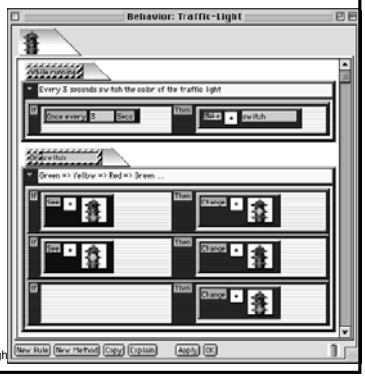
- Leverage power and success of spreadsheets for other domains
- E.g., Forms3
  - Burnett, 1991
  - More general code for formulas
  - Graphics in cells






## Agentsheets

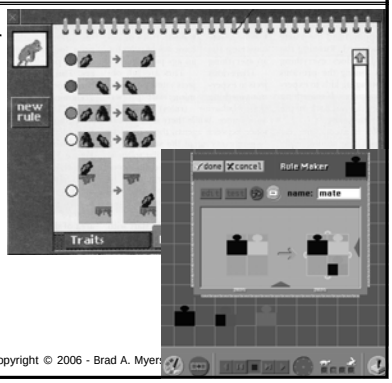
- [Repenning 91]
- Agentsheets.com
- Before and after pictures as rules






## KidSim/Cocoa/Stagecast Creator

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






## Studies of VP

- Claims that VP would be better due to 2-D more "natural" and no syntax
- 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][Moher 1993]
  - Visual programs are usually very difficult to edit ("high viscosity") [Green 96]
  - Take more space than text

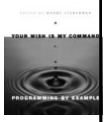
## Outline

- 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

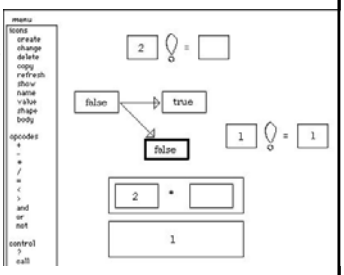
- Create program by performing the steps by example
  - Assumes user knows how to do the problem concretely
  - Avoids problems of *abstraction*
  - [Cypher 93], [Lieberman 2001]
- Pygmalion [Smith 77]
- Smallstar [Halbert 81, 84]
- Peridot [Myers 86]
- Comic strip:
  - Chimera [Kurlander 92]
  - Pursuit [Modugno 93]
- Gamut [McDaniel 96]




## Pygmalion

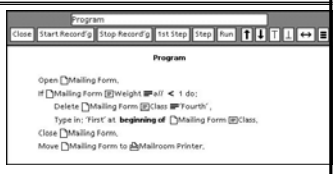
- [Smith 77]
- Show the computer the desired steps

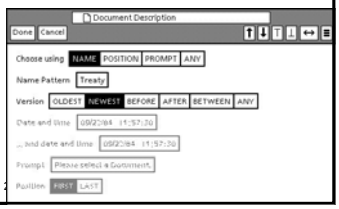





## SmallStar

- Halbert 81,84
- By example in simulation of the Star
- Property sheets for data generalizations

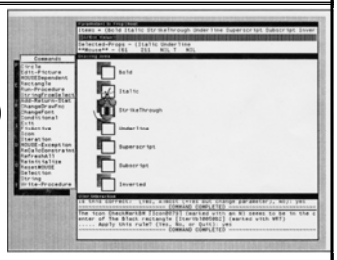





## Peridot

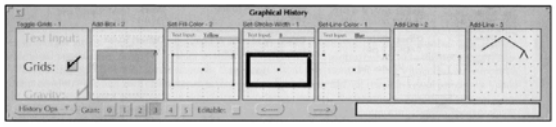
- [Myers 86]
- Show behavior of controls (widgets) by example
- Leverage power of Direct Manipulation
  - Directly build dynamic parts of interface
- Inferred constraints and mouse behaviors

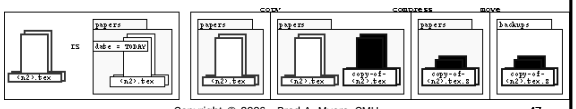





## As a "Comic Strip"

- Chimera [Kurlander 1988]
- Pursuit [Modugno 1993]

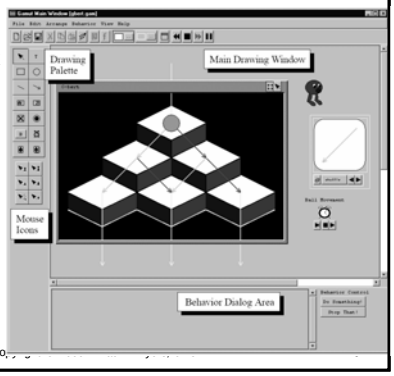





## Gamut

- [McDaniel 96]
- Inferred complex behaviors
- "Do Something" and "Stop That"
- Various kinds of hints



## Gamut Video

## 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

## 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)
- Hands (2002)
- Chickenfoot (2005)

## Basic

- Designed in 1963, by John George Kemeny and Thomas Eugene Kurtz at Dartmouth College
- Beginner's All-purpose Symbolic Instruction Code
- To allow students not in science fields to use computers
- Timesharing and then personal computers
- (Microsoft's first product, in 1975)

```
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$
100 REM
110 INPUT "Do you want more stars? "; A$
120 IF LEN(A$) = 0 THEN GOTO 110
130 A$ = LEFT$(A$, 1)
140 IF (A$ = "Y") OR (A$ = "y") THEN GOTO 40
150 PRINT "Goodbye ":
160 FOR I = 1 TO 200
170 PRINT U$; " ";
180 NEXT I
190 PRINT
```

## Logo

- Created in 1966 at BBN by Wally Feurzeig and Seymour Papert
- Like Lisp without parentheses
- First turtle was physical device with wheels and a pen

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

spiral 0 91

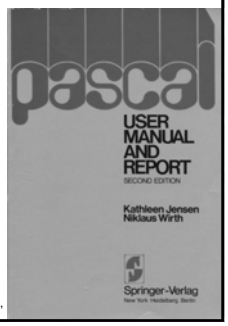
 

## Pascal

- Created in 1970 by Niklaus Wirth to teach structured programming

```
program HelloWorld(output);

begin
  writeln('Hello, World!')
end.
```






## 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

```
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"

## HANDS Video

## Chickenfoot

- [Bolin, 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

## 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

## Better Support in the Environment

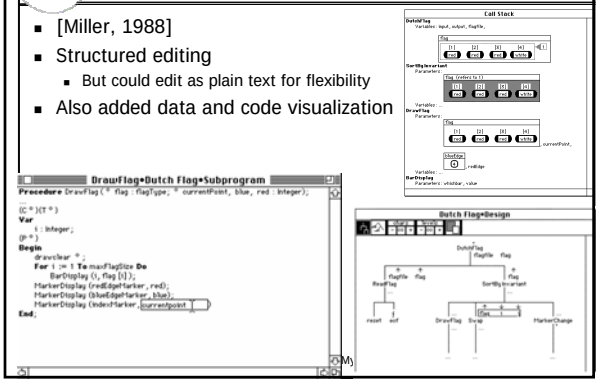
- Original: Cornell Program Synthesizer, 1981
- Structured Editing
  - MacGnome, 1988
  - Alice, 2002
- HyperCard, 1987
- Director, 1988
- Visual Basic, 1991
- WhyLine, 2004

```
DO WHILE ( condition );
    [{]statement}
END;
```

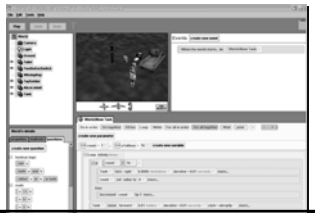
## MacGnome

- [Miller, 1988]
- Structured editing
  - But could edit as plain text for flexibility
- Also added data and code visualization

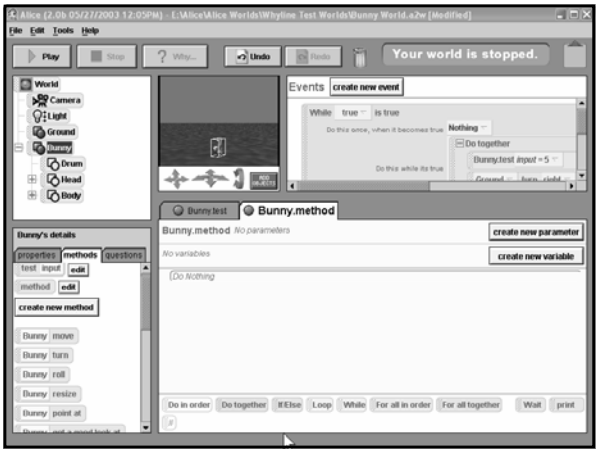



## Alice

- Alice 2 [Pausch] (2002)
- Drag-and-drop program parts
- Pop-up menus for parameters
- Dramatic impact on learning and attitude








## 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

## HyperCard

- Atkinson (1987) tried to make user's first experience with the tool effective ("low threshold")
- Metaphor of designing cards
  - Background, foreground objects
  - Change cards in-place
  - Now familiar from WWW and PowerPoint
- Programmed in HyperTalk (discussed earlier)
- Successfully enabled significant EUP

## Visual Basic

- Microsoft, first released, 1991
  - 1997, VB5 Debuts – replaces Word Basic, Excel Basic, etc.
  - 2002, VB.NET Debuts
- For scripting, connecting components, database access, etc.
- Interactive tool for placing widgets (controls) such as buttons (= "Interface Builder")
- Event-based version of the Basic language

## Visual Basic Picture

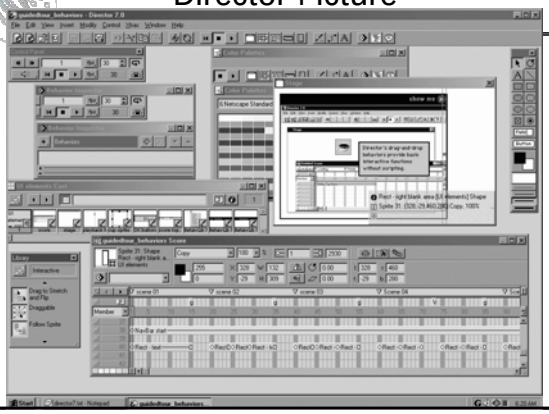
- VB.Net

## Director

- MacroMedia (now Adobe) 1988
  - Most people now use Flash
- Scripting language ("Lingo") for animations, with IDE
- Metaphor of a timeline "Score", for when animations start and stop
  - Awkward for user-driven interactions

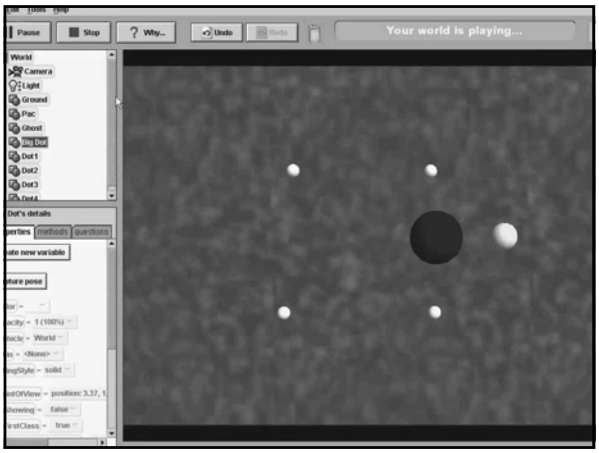
## Director Picture



## WhyLine

- Debugging tool [Ko & Myers, 2004]
  - Surprising lack of support for debugging, even in EUP tools
- Observation from studies: All of the observed debugging problems could be addressed by "Why" questions
  - 32% were "why did"; 68% were "Why didn't"
  - Allow directly asking these questions in the UI
  - Searches code and execution history for answers
  - "Why didn't" questions are answerable because only ask about what was plausible to have happened.
- Answers use:
  - Text message
  - Visualization of the time line ("WhyLine"), and
  - Highlighting of code and data




## Review of Results of User Study

- Subjects with WhyLine got *40%* more tasks completed
  - 3.20 vs. 2.25, *(p<.02)*
- In matched situations, subjects with the WhyLine debugged about *8 times faster*
  - Average: 20 seconds vs. 155.7 seconds, *(p<.02)*

## 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)

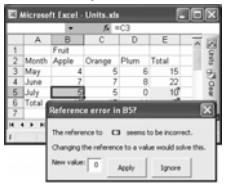
## End-User Software Engineering

- Initiative to make software created by end users more reliable and correct
- Motivation:
  - Spreadsheet errors
  - Difficulty of debugging
- 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 on EUSE (WEUSE 1 at ICSE'05, WEUSE II at CHI'06)
  - Connections: Researchers + Industry

## EUSE Examples, 1

- UCheck [Abraham 2004]
- Infers units based from layout and headers
- Identifies formulas that try to combine incompatible units

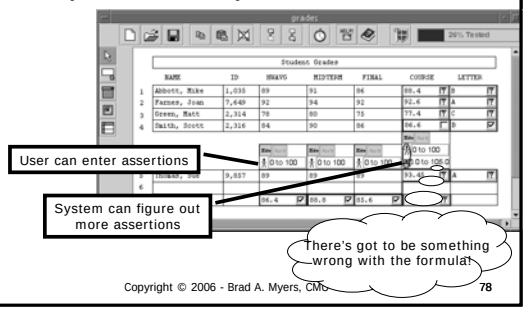



| | A | B | C | D |
|---|---|---|---|---|
| 1 | | Fruit | | |
| 2 | Month | Apple | Orange | Total |
| 3 | May | 8 | 11 | 19 |
| 4 | June | 75 | 5 | 80 |
| 5 | Total | 83 | 16 | 99 |

## EUSE Examples, 2

- WYSIWYT [Burnett 1997]
  - What you see is what you test



## 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?
- Will Artificial Intelligence (AI) help?
  - Reduce need for programming?
- Still enormous opportunities for research and new ideas

# Thank You!

## End-User Programming

Brad Myers, Andrew Ko, and Margaret Burnett

Funded by NSF