Inherent vs. Accidental vs. Intentional Difficulties in Programming

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

Carnegie Mellon

Key Questions

- To what extent are the difficulties faced by developers avoidable or fixable?
  - Clear that *some* difficulties are avoidable or fixable
  - Would be useful to know which ones are *not*
  - *Inherent* vs. *accidental* difficulties

- To what extent are the difficulties faced by developers a result of explicit design decisions?
  - *Intentional* difficulties

"Programming"

- All the activities involved with programming
  - Issues with the programming language itself
  - Issues with libraries, frameworks, and other APIs
    - Little can be accomplished with *just* the language
    - To the developer, they are indistinguishable
  - Issues with tools such as IDEs
    - May interact with language features and APIs
  - Professional, Novice *and* End-User programmers
- Focus on programming = developing = coding

"Difficulties"

- Any barriers to success when the developer is programming
  - (but *not* those due to bad specifications)
- Evaluated like other usability breakdowns
  - Problems with learning or remembering how to do it
    - Again, for all kinds of developers
  - Slow-downs while coding
    - Efficiency of the programming itself
  - Error-proneness while coding
  - Lower quality results
    - Quality of the resulting code
- Measured using various kinds of user studies
  - Bug reports, CIs & other field research, lab evaluations, …

Are any Difficulties "Inherent"?

- "Inherent" – "part of the very nature of something, and therefore permanently characteristic of it or necessarily involved in it"
  -- Encarta Dictionary
- Means that research on eliminating the difficulties can never succeed

Complex Algorithms

- **Google search for "inherent difficulties in programming" includes:** "Researchers … have never ceased to emphasize the inherent difficulties in solving **stochastic programming** problems… Recent developments in the theory of **computational complexity** allow us to establish the theoretical complexity of most stochastic programming models studied in the literature. Our results **confirm** the general feelings alluded to above."

  -- [Dyer & Stougie, 2003]

- *(Not very surprising)*

"Essential complexity"

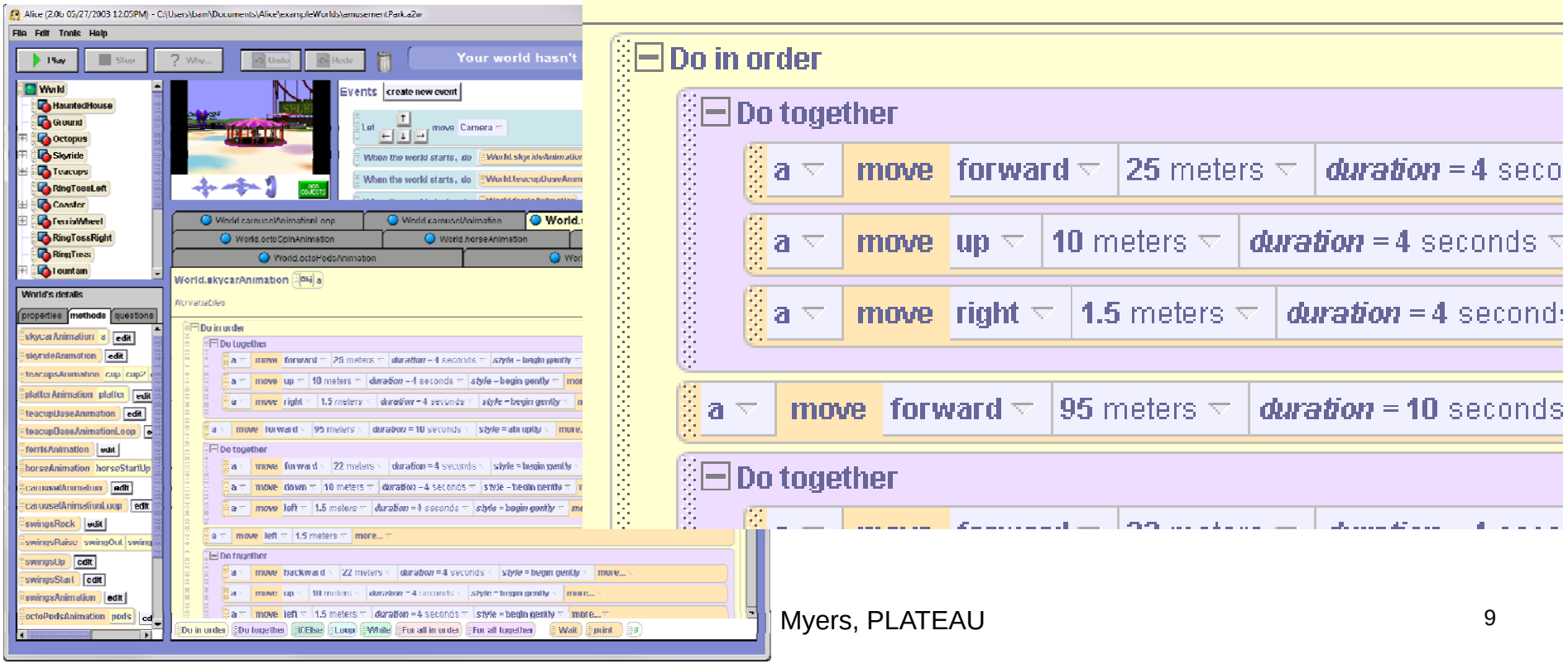
- "All reasonable solutions to a problem must be complicated (and possibly confusing) because the 'simple' solutions would not adequately solve the problem" -- Wikipedia
- Complexity comes from the size and interactions and the constraints on the parts
- More interesting to look at difficulties with *individual* aspects
  - And difficulties that arise from limitations of the building blocks

Parallel Programming?

- **Google search for "inherent difficulties in programming" includes:**
  "The ever-growing prevalence of parallel architectures has pushed the inherent difficulties in **parallel programming** to the front stage **...**" **http://faspp.ac.upc.edu/**

Parallel Programming?

- But Alice claims that their approach makes it easy
  - At least for simple situations



What else?

- Recursion?
- How unification works in Prolog?
- For some people, the requirement to decompose the task into primitives is a barrier
  - "Programming is the process of transforming a mental plan into one that is compatible with the computer."
    — *Jean-Michel Hoc*

Inherently Hard Because Currently Unsolved

- No one knows how to express these
- Language support for distributed systems
- API support for flexible user input handling
  - Beyond low-level event-handlers

Accidental Difficulties

- Difficulties that can be avoided
- "Accidental complexity" – "non-essential to the problem to be solved" -- Wikipedia
  - "due to mistakes such as ineffective planning …"
  - Should "be minimized in any good architecture, design, and implementation"
- Difficulties ⊃ complexity
  - Can arise from things that might be "simple"

Example

- Difficulties in handling multiple objects
- F(o) – easy to operate on one object

  `F(o);` or `o.F();`
- Now suppose we have a bunch of o's

```
for (int i=1; i<11; i++)
    {
        o_array[i].F();
    }
```

Two

- ~~Three~~ kinds of parentheses, extra array data structure, extra variable `i`

```
for (Otype i : o_array)
    {
        i.F();
    }
```

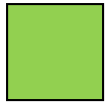
- vs. HANDS: `F(o)` works on either. Also: `F(all o)`

Restrictions on Built-ins

- `puts "Hello World"` in Ruby or tcl
- vs. Java console output:

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

  - 9 special words and 3 types of parentheses

- But how output ***bold, red script***? Or  ? Or sounds?
- Compare to PowerPoint or Visual Basic

Unfortunate Language Designs

- Confusing keyword choices:
  - STOP in Logo – exits the current procedure (doesn't stop the program)
  - `static` (vs. `const`, `final`) – multiple meanings
  - AND in all programming languages
    - men *and* women
- Precedence
  - E.g., `A+++B` means? `(A++)+ B` *or* `A + (++B)`
    - `A++++B, A++-+B  A--+-++B`
  - Bring home a fruit that is(not an apple)or a pear
- See Java Puzzlers [Joshua Bloch and Neal Gafter]
  - *"traps, pitfalls, and corner cases"*

Source of Accidental Difficulties

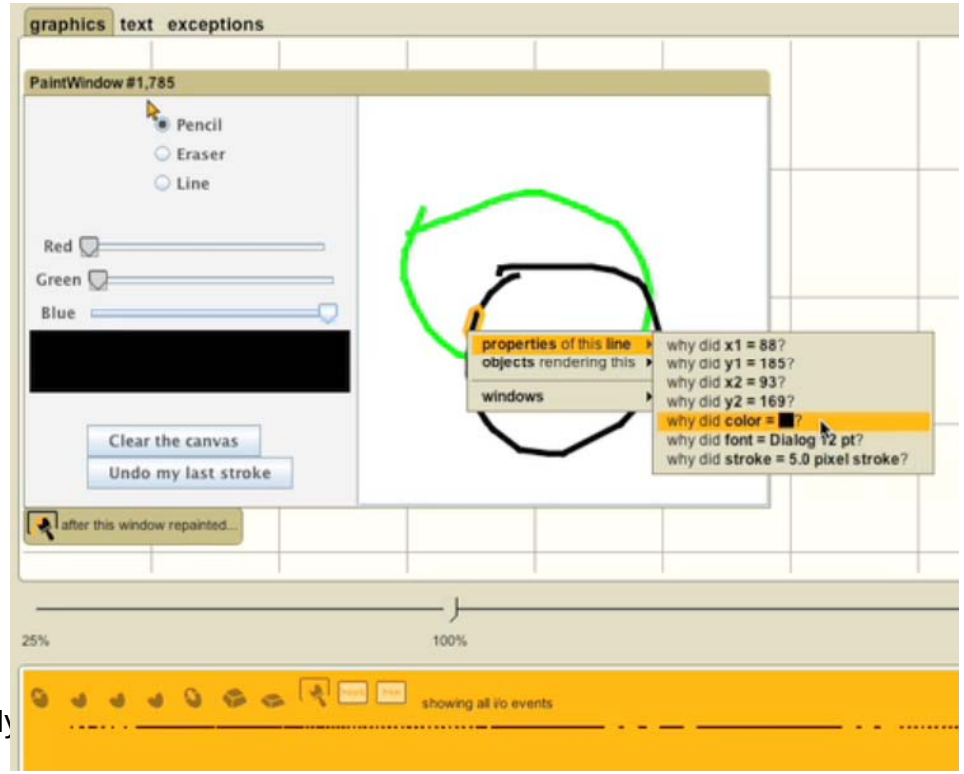
- Some causes:
  - Consistency with previous languages
    - Syntax in Javascript, C#, Java, … based on C (1969)
      - `switch (month) {`
        `case 1: …; break;`
        `case 2: …; break;}`
      - Inline conditional: `d = c ? a : b;`
  - Constraints that are no longer relevant
    - Reducing amount of typing:
      - `for(;;)` syntax; `{}` instead of `begin end`
    - Only use pure ASCII in programming languages
      - `=` and `==` instead of ← and `=`
      - No place for meta-information, history, etc.

Due to Programmer Stubbornness

- "Whatever I learned first is best"
  - Resistance to learning new tools/methods/languages
- "Macho"→"Tools? We don't need no stinkin' tools!"
- Examples:
  - Auto-complete is one of the most popular tools in IDEs
    - But many people are still programming without it (e.g., in Notepad or VI -- but VIM has completions)
  - Debugging is still done the same way as 60 years ago
    - Breakpoints, print statements, watching variables

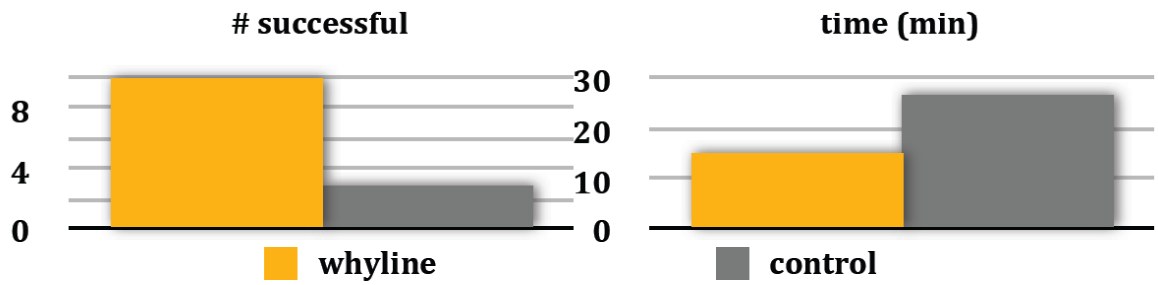
Whyline

- PhD work of Andy Ko
- Allow users to directly ask “Why” and “Why not”
- Collect a trace and replay trace within Whyline
- Algorithm contributions
  - Complete tracing of Java programs
    - Slow-down, about a factor of 5
    - Comparable to profilers
    - Size ≅ 2mb/sec for interactive programs
  - Incremental, real-time static and dynamic slicing
    - Causality of events
- Novel UI for asking questions



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 ½ of the time



Accidental Difficulties Due to Bad APIs

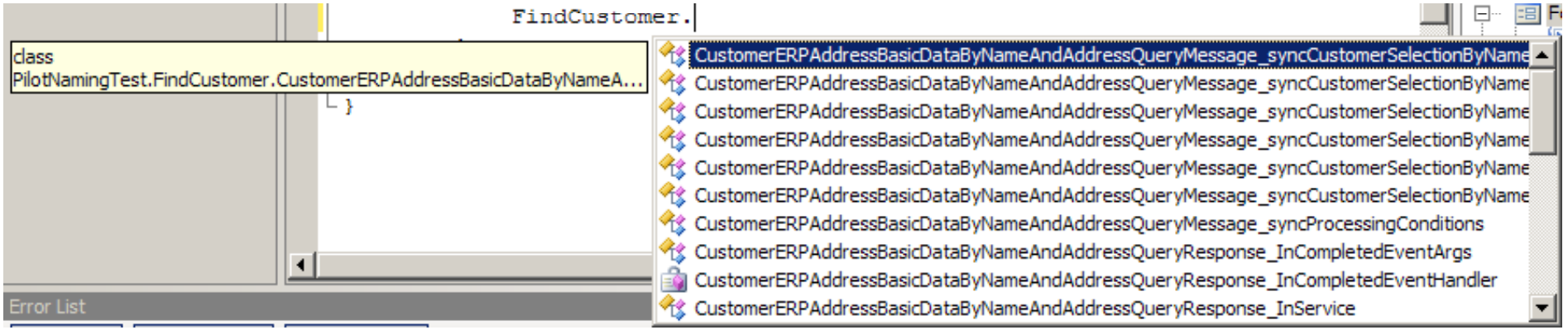
- Inconsistent parameter orders:
  - Java Interface XMLStreamWriter:

```
writeStartElement(namespaceURI, localName)
writeStartElement(prefix, localName, namespaceURI)
```

- Bad names, e.g. in SAP eSOA
  - Too long

```
MaterialSimpleByIDAndDescriptionQueryMessage_syncMaterialSimpleSelectionByIDAndDescriptionSelectionByMaterialDescription
```

  - Names which are not understandable



More Causes of Difficulties

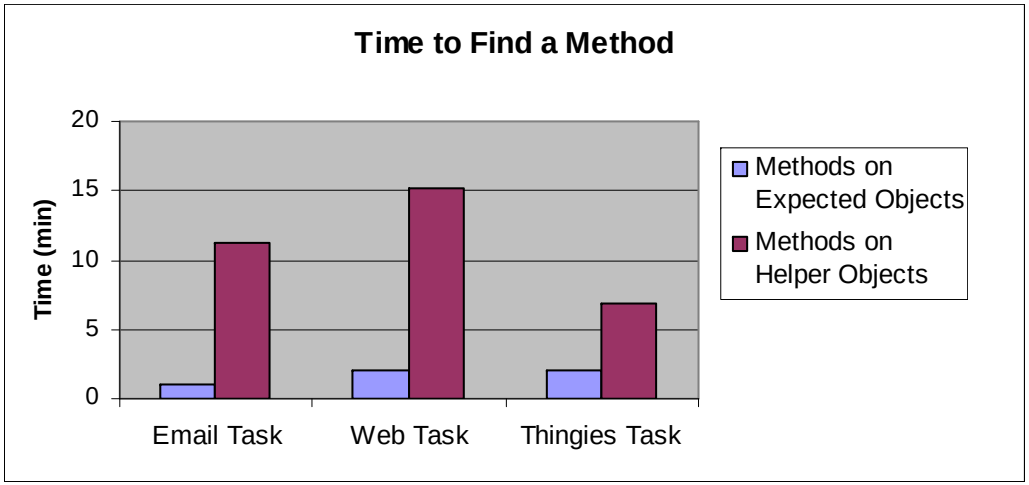
- Names that are not distinguishable
  - org.xml.sax.ext -> Attributes vs. Attributes2
- Unnecessarily exposing underlying mechanisms

Alpha value in Java vs. “fade”

- Inappropriate models
  - 3D transforms using matrices in radians vs. Alice’s object-centered commands
    - Obj.Turn(left , 1/8), Obj.TurnTo (otherObj)

Object Method Placement

- Where to put functions when doing object-oriented design of APIs
  - `mail_Server.send( mail_Message )`
    vs.
    `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



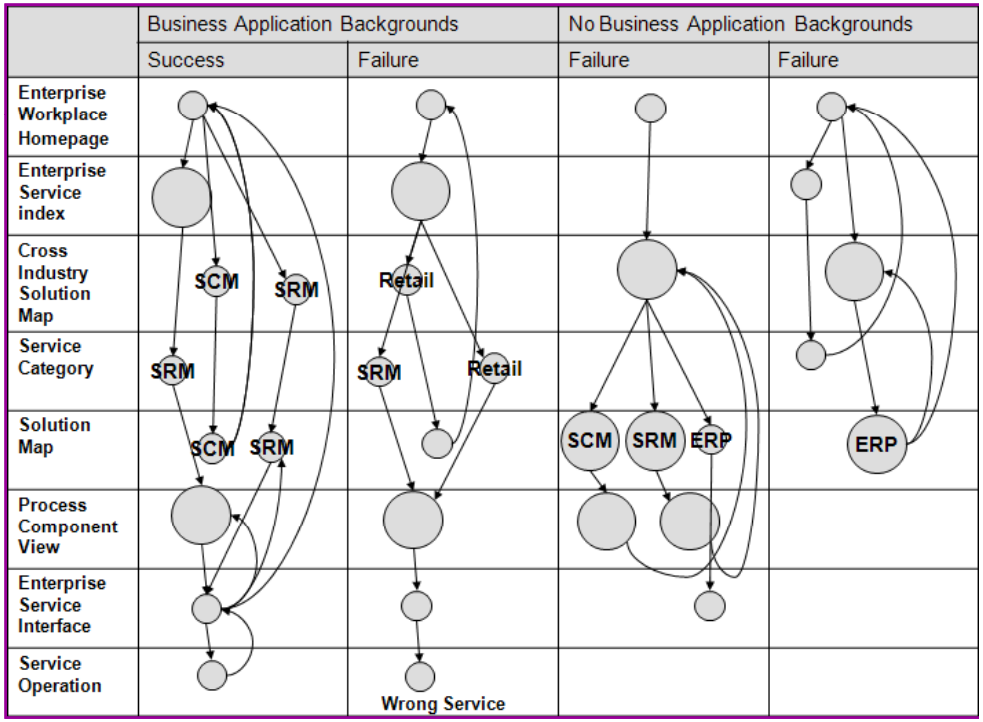
More API Difficulties

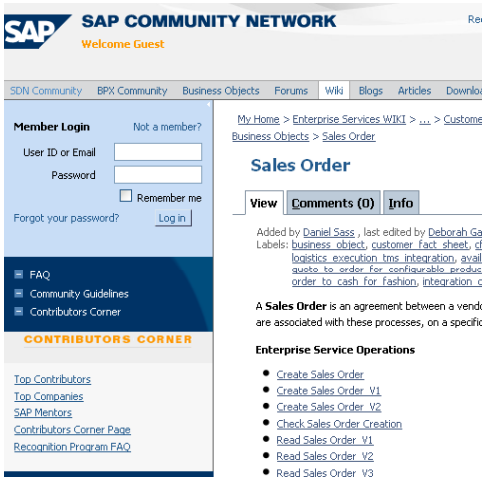
- Apparently missing functionality
  - Java `File` class has no `read` or `write`
- Actual missing functionality
  - `Select` in C# .NET 1.1 [Henning, "Api Design Matters." *ACM Queue,* 5(4), 2007]
    - Overwrites argument lists so requires extra copying
    - Does not tell if succeeds or times out
  - Original Java APIs had to use applet class to do audio
    - http://www.javaworld.com/javaworld/javatips/jw-javatip24.html (1997)
  - String handling difficulties in many languages
    - E.g., Imploding and exploding arrays of strings in PHP vs. Java
    - Much user-centered data must be represented as strings
      - Ref: Chris Scaffidi's "Topes"

Accidental Difficulties Due to Bad Documentation

Human-Computer Interaction Institute

- SAP eSOA documentation study
- 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 info





Intentional Difficulties

- Designer made something difficult on purpose
- Difficult by being silly
  - "Esoteric programming languages" – Wikipedia
    - INTERCAL, from 1972
    - FALSE from 1992
    - Turing tarpit – programmed like a Turing machine with minimal commands: `++++++++++[>+++++++>++++++++++>+++>+<<<<-]>++.>+.+++++++..+++.>++.<<++++++++++++++.>.+++.------.--------.>+.>.`
    - Malbolge (8th circle of hell) 1998: `('&%:9]!~}|z2Vxwv-,POqponl$Hjig%eB@@>}=<M:9wv6WsU2T|nm-,jcL(I&%$#"`CB]V?Tx<uVtT`Rpo3NlF.Jh++FdbCBA@?]!~|4XzyTT43Qsqq(Lnmkj"Fhg${z@>`
    - Shakespeare Programming Language

More Seriously…

- Features made difficult because designer does *not* want novices to use them
  - No `enum`s in original Java because abused in C
    - So used `final static int`
  - No `null` value in ML

More Difficult due to a Tradeoff

- Designer decides that some other requirement is more important than making it easier
  - Or maybe didn't consider developer difficulty at all
  - Malice vs. laziness vs. ignorance?
- Classic tradeoff of high vs. low level control
- Usually: flexibility & versatility vs. usability
  - Multiple steps to perform an action allows programmer to do it in different ways
    - E.g., Ruby on Rails makes it easy to create websites of specific styles, vs. Java Spring with more overhead

"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 throughout Java (>61) and .NET (>13) and SAP
- Advantages
  - No memory allocation
  - Indirection: easier to have other implementations
- Results of lab study with expert Java programmers:
  - 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

Control vs. Automatic

- Automatic conversions of values
  - `"1" + 1` vs. `1 + "1"` in Smalltalk vs. Java
- All APIs have *protocols*
  - Series of methods called in a particular order
  - "Boiler plate"
  - Example: `file` must be `open`ed before `read`
    - Alternatively: `read` could automatically `open`
    - But more "magic" with objects fixing themselves
      - Less control for programmer
  - How many protocols could be eliminated?
    - When would that be a good idea?

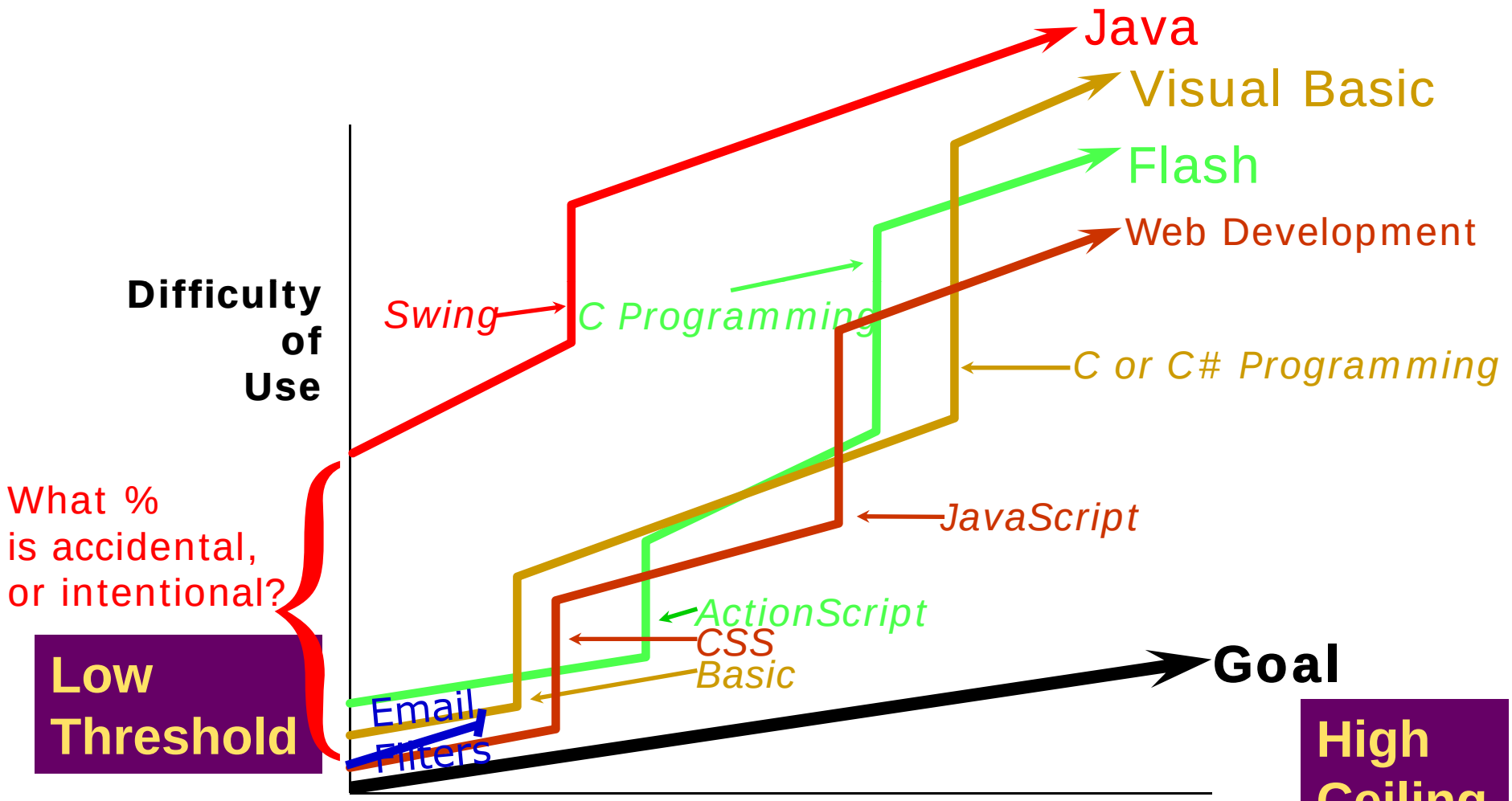
Classic Arguments

- Static vs. dynamic typing: more “difficult” to enter code for more error checking
  - Controversy: Which makes it less difficult to get correct code in the end?
- OO vs. functional
- …

Visual Basic developers

- "Make things simpler than possible. That is the contradiction hardcore Visual Basic programmers live with. … Our task is to seemingly simplify beyond the limits of the possible; to create a sort of garden where children play and then grow up and venture off into the woods. The point isn't that there aren't woods or that they shouldn't see trees. The point is that they start off in a garden."
  - Hide abstractions that provide flexibility wanted by professional programmers

Goal: Gentle Slope Systems



Implications

- Usability studies & general use can identify difficulties
- Designer should determine if difficulties are Inherent, Accidental, or Intentional
- Educate when inherent
- Fix when accidental
- Document when intentional

Thanks!

- To Andrew Faulring, Thomas LaToza, Donna Malayeri, and Jonathan Aldrich for help with this talk
- To >30 students
- To funding from NSF, SAP, Microsoft, Adobe, IBM
- http://www.cs.cmu.edu/~natprog
- http://ww.cs.cmu.edu/~bam
- bam@cs.cmu.edu