

Software Engineering
CSE 335, Spring 2006

No Silver Bullet

Stephen Wagner

Michigan State University

Electronic Voting

- Many different aspects to the problem:
 - Technological challenges: security, reliability, accuracy
 - “Political” challenges: “proving” that the machines are secure, reliable, etc.
 - Human interaction challenges: machines have to be usable by just about anyone.
- Tremendous pressure to get it right.
- Existing machines failed on something as simple as the counting

An Aside

"Misunderstandings and neglect occasion more mischief in the world than even malice and wickedness. At all events, the two latter are of less frequent occurrence."

Goethe

"Never ascribe to malice, that which can be explained by incompetence."

Robert Hanlon(?)

Silver Bullets

- Software development is a very complex and difficult process
 - Lots of failed projects
 - Lots of buggy products
- The hope has been that there exists some new technology that will make everything easy
 - A new language
 - A new programming environment
 - Bigger, faster computers
- Why or why not?

Accidental vs. Essential Difficulties

- There are two types of difficulties
 - Accidental difficulties: difficulties that are not an essential part of the problem, e.g. the programming environment
 - Essential difficulties: difficulties that are intrinsic to the problem being solved
- The 60's and 70's saw great advances that minimized accidental difficulties

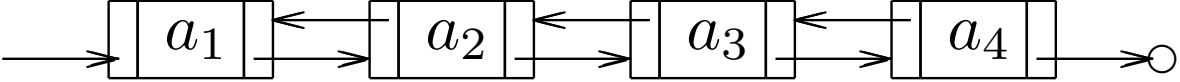
Accidental Difficulties

- Some difficulties that people use to have to deal with
 - Assembly Language
 - Compilation time
 - Memory Management
 - Programming Environment
- These difficulties have been greatly reduced
 - Higher Level languages let the programmer focus more on the problem to be solved than on the language
 - Faster compilers promote “immediacy”
 - Modern OS’s make memory management easy, make file sharing easy

Essential Difficulties

- Some problems are inherently complex and difficult
- These complexities and difficulties cannot be ignored or removed
- The solution is going to be correspondingly complex and difficult
- Brooks argues that technological advances will only help with the accidental difficulties, not the essential difficulties

Accidental Difficulties

- Linked Lists 
- `list<Employee *>;`
- Reusing existing code is one way to reduce accidental difficulties

Accidental Difficulties

- Code Generators
 - tools for parser generation
 - tools for GUI construction
 - tools for code reuse e.g. adapting a class
- Code generators leave “hooks” for the user to insert their own code
- What code does the user have to insert?

Essential Difficulties

- Some problems are inherently large and complex
- The solution is also going to be inherently large and complex
- Creating a large and complex solution is going to require time and will be error prone
- How much help can we expect in this area?
- How do you describe the problem to make automated help more feasible?

Essential Difficulties

- Complexity: software is amazingly complex for its size
 - not bound by physical constraints
 - no two parts are alike
 - software is discrete, i.e. the details matter
- Must conform to existing standards
- Changeability: because software is perceived as easy to change, it is subject to pressures for change
- Invisibility: software is invisible
 - geometric modeling, visualization tools are very useful in other fields
 - software does not occupy space
 - diagramming things is more challenging

Flexibility: Blessing and a Curse

- Software is or can be
 - amazingly complex
 - easy to reproduce
 - easy to modify “physically”
 - not subject to aging, etc.
- All of these are good things, but can encourage bad behavior
 - Software engineers face a harder problem,
 - but in general did not adopt many good engineering practices

Some Ideas from 20 years ago

- Ada and other high-level language advances
- Object-oriented programming
- Artificial Intelligence
 - Expert Systems
 - Automatic Programming
- Program verification

Ada

- High level programming language developed in 80's, still used by military

```
-- Function F is a simple recursive routine
-- A test program is provided

with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;

procedure Fig1_2 is

    function F( X: Integer ) return Integer is
    begin
        if X = 0 then -- Base case
            return 0;
        else
            return 2 * F( X - 1 ) + X ** 2;
        end if;
    end F;

begin
    Put( F( 5 ) ); New_Line;
end Fig1_2;
```

Object Oriented Programming

- 20 years ago Object Oriented Program was a new idea
 - SmallTalk first widely released in 1980
 - C++ written 1983-1985
- C++ has been popular for 15 years
- Added potential for code reuse
- How much has it helped?

Why has OO not been as successful as hoped?

The answer is simple. It is because OO has been tied to a variety of complex languages. Instead of teaching people that O-O is a type of design, and giving them design principles, people have taught that O-O is the use of a particular tool. Unless we teach people how to design, the languages matter very little. The result is that people do bad designs with these languages and get very little value from them. If the value is small, it won't catch on.

David Parnas, 1995

Artificial Intelligence

- The only field in computer science that has not far exceeded the predictions for it 40 years ago
- A.I. successes get moved out of A.I.
 - pattern recognition, vision, speech recognition
- There will likely not be any breakthroughs any time soon
- How would you present a problem to an A.I.?

Expert Systems

- Try to capture how an expert solves a problem
- Medical diagnoses were one area this technique worked well in
 - What questions does a doctor ask?
 - What do they learn from the answers?
 - A very elaborate version of the 'Animal Game'.
- Not clear how to apply this to software engineering

Automatic Programming

- The ultimate goal in programming
 - tell the computer what you want
 - do not tell the computer how to do it
- Not clear how it differs from an even higher level programming language
 - Is this automatic programming? `sort(students);`
- How do you describe what you want?
- Assisted Automated Theorem Proving

Buy vs. Build

- There is a lot more general purpose software readily available today than 20 years ago
- Many applications can be built out of existing parts
 - A payroll system needs a database and some code to implement some standard queries and updates
 - Modern Database Management Systems include the ability to write programs
- There is now a large market for applications that are designed to build applications
 - There are still(?) applications that need to be built from scratch
 - There are still difficulties using a complex piece of software

Other changes since the 80's

- Personal Workstations
 - Current day PC compares favorably with supercomputers of the 70's
 - Reduces compile time
 - Supports graphical programming environments, language specific smart editors, etc.
- Integrated Programming Environments
- Better networks

Specification

- Many of the above techniques would require you to somehow describe the problem
 - For some sort of code generation tool there has to be some form of input
 - The input to an automatic programming system needs to be a description of the problem
- *Specification* describe the problem that needs to be solved
- A very important, but often under-appreciated, part of software development, regardless of what tools you use.

Specification

- **Defn:** The precise definition of the tasks to be performed by the system
- The task that lies “between” requirements elicitation and design
- Culminates in some form of written document
 - Should be a combination of English text
 - and formal (or even semi-formal) model
- Task is very difficult and rarely amenable to automation