"No Silver Bullet"

MONTANA STATE UNIVERSITY

Mountains & Minds

No Silver Bullet

- "No Silver Bullet" -- a paper by Fred Brooks, Professor of Computer Science at University of North Carolina in Chapel Hill
- (Best known as the father of IBM System/360.
- "No Silver Bullet – Essence and Accidents of Software Engineering", IEEE Computer, April 1987.

MONTANA STATE UNIVERSITY

Mountains & Minds

No Silver Bullet

- "Of all the monsters that fill our nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors.
- For these one seeks bullets of silver that can magically lay them to rest.
- The familiar software project, at least as seen by the non-technical manager, has something of this character; it is usually innocent and straightforward, but is capable of becoming a monster of missed schedules, blown budgets, and flawed products."

MONTANA STATE UNIVERSITY Mountains & Minds

No Silver Bullet

- A silver bullet – something to make software costs drop as rapidly as hardware.
- However, as we look over the horizon there is no one single development in either:
  - technology, or,
  - management techniques,

  that by itself will provide even an order of magnitude improvement in productivity.
- In this class we will try to find out why.
  - We will examine the nature of software.
  - We will examine the proposed bullets.

MONTANA STATE UNIVERSITY Mountains & Minds

Nature of Software

- Brooks argues that the very nature of software makes it unlikely that there will ever be any silver bullets.

1. Software progress is not slow,
   - but hardware has advanced even faster,
   - 6 orders of magnitude price-performance gain in 30 years.
2. Difficulties of software technology: why software progress isn't as fast as hardware.
   - essence: inherent in the nature of software
   - accidents: attending its production but not inherent

MONTANA STATE UNIVERSITY Mountains & Minds

Essence Difficulties

- "the hard part of building software is the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation"
- The essence of software is a construct of
  - interlocking components,
  - data sets,
  - relationships between data items,
  - algorithms,
  - invocations of functions.

MONTANA STATE UNIVERSITY Mountains & Minds

Essence Difficulties

- This essence is abstract: the conceptual construct is the same under many different representations.
- The inherent problems are:
  - complexity,
  - conformity,
  - changeability,
  - invisibility.

Complexity

- Software entities are more complex than perhaps any other human construct because no two parts are alike.
- Computers are themselves more complex than most things people build.
- Software systems have orders of magnitude more states than a computer.
- Scaling up software entities does not merely involve a repetition of the same elements in larger sizes.

Complexity

- Complexity increases more than linearly.
- Complexity is essential, not accidental:
- It just cannot be abstracted away:
  - difficulty of communication among people
  - difficulty of enumerating of program states
  - difficulty of using programs
  - difficulty of extending programs

MONTANA STATE UNIVERSITY Mountains & Minds

Conformity

- The physicist labors on in a firm faith that there are unifying principles put there by God.
- No faith faces the software engineer — software is not designed by God it is designed by people.
- In many cases software must conform because it is the newest thing on the scene.
- Much complexity comes from conformation to other interfaces.

MONTANA STATE UNIVERSITY Mountains & Minds

Changeability

- Software is constantly subject to pressures for change.
- All successful software gets changed.
  1. Successful software is applied to tasks for which it wasn't intended.
  2. Successful software outlives the hardware for which it was written.

MONTANA STATE UNIVERSITY Mountains & Minds

Invisibility

- Geometric abstractions are powerful tools:
  - floor plans for architect & client,
  - scale drawings/models of mechanical parts.
- Software is hard to visualize.
- A diagram may be many non-planer graphs:
  - flow of control,
  - data flow,
  - dependency patterns.
- This impedes the process of design.
- It also impedes communication among.

MONTANA STATE UNIVERSITY Mountains & Minds

Solutions to Accidential Difficulties

- High level languages:
  - Free programmer from accidental complexity
  - Language development approaches closer and closer to the sophistication of the users.
- Time sharing - as opposed to batch processing:
  - Preserves immediacy – slow turn-around is an accidental difficulty.
- Unified programming environments:
  - Attacks difficulties by providing integrated libraries, unified file formats, pipes and filters.

Hope from Technical Development

- Ada. Java, C++, .NET, … and other HLL advances
  - "Perhaps the Ada philosophy is more of an advance than the Ada language, for it is the philosophy of modularization, of abstract data types, of hierarchical structuring."
- Ada, Java, .NET, C++, … will not slay the monster, its only another HLL.
- The main payoff came in changing from accidental complexity to more abstract statement solutions.

Hope from Technical Development

- Expert Systems
  - "The most important advance offered by this technology is the separation of the application complexity from the program itself."
- Such systems can:
  - suggest interface rules,
  - advise on testing strategies,
  - remember bug-type frequencies,
  - offer optimization hints.
- This is no small contribution ... it comes from rich knowledge bases that reflect the real world.

MONTANA STATE UNIVERSITY Mountains & Minds

Hope from Technical Development

- Automatic programming
  - generation of a program from a statement of the problem specifications.
  - In most cases, it is the solution method, not the problem, whose specification has to be given.
  - exceptions: the technique of building generators is very powerful.

MONTANA STATE UNIVERSITY Mountains & Minds

Hope from Technical Development

- Graphical programming:
  - graphical, or visual programming,
  - Nothing convincing, has yet emerged:
    1. flowcharts are poor representations,
    2. screens are too small.

MONTANA STATE UNIVERSITY

Mountains & Minds

Hope from Technical Development

- Object-orientation - two separate ideas:
  1. Abstract data types - encapsulation.
  2. Hierarchical types - refinement.
- Program verification.
- Environments and tools.
- Workstations.

MONTANA STATE UNIVERSITY

Mountains & Minds

PROMISING ATTACKS ON THE ESSENCE

- Four hopeful directions:
  1. Buy don't build,
  2. Requirements refinement and rapid prototyping,
  3. Incremental development,
  4. Great designers.

MONTANA STATE UNIVERSITY

Mountains & Minds