No Silver Bullet: Essence and Accidents of Software Engineering
Frederick P. Brooks

Of all the monsters that fill the 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 first step toward the management of disease was replacement of demon theories and humours theories by the germ theory. That very step, the beginning of hope, in itself dashed all hopes of magical solutions. It told workers that progress would be made stepwise, at great effort, and that a persistent, unremitting care would have to be paid to a discipline of cleanliness. So it is with software engineering today.

Proposed Silver Bullets

- Structured programming
- Modularity
- Data Abstraction
- Software Verification
- Object oriented
- Agile or Xtreme programming
- Aspect oriented programming

Complexity. Software entities are more complex for their size than perhaps any other human construct because no two parts are alike (at least above the statement level). If they are, we make the two similar parts into a subroutine—open or closed. In this respect, software systems differ profoundly from computers, buildings, or automobiles, where repeated elements abound.

Requirements refinement and rapid prototyping. The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.

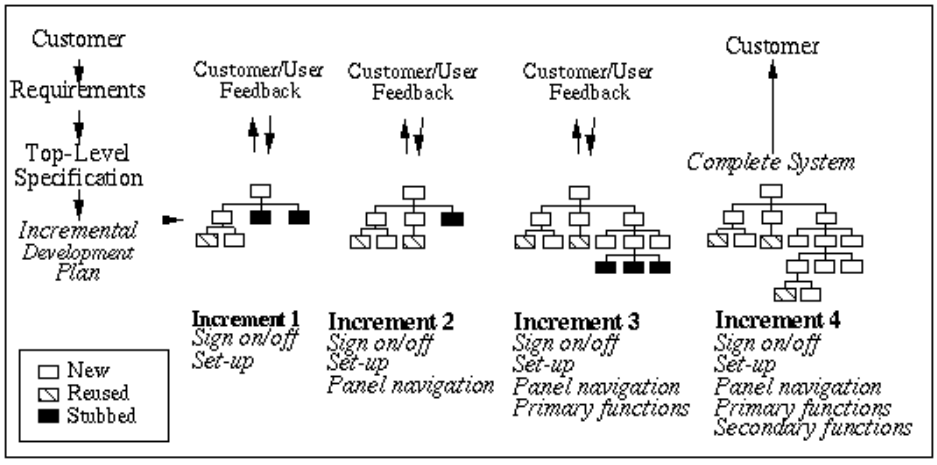
> Some years ago Harlan Mills proposed that any software system should be grown by incremental development.[10] That is, the system should first be made to run, even if it does nothing useful except call the proper set of dummy subprograms. Then, bit by bit, it should be fleshed out, with the subprograms in turn being developed—into actions or calls to empty stubs in the level below.

Advocates:

- Incremental development

Cleanroom

Incremental Development of a Small System



Benefits of Incremental Development

- Early feedback
  - on part of the system, at least
- Improves morale
  - Something tangible is working
- Improves chances of releasing on time
  - Incorporate high priority capabilities first
  - Low priority capabilities may miss release
  - Detect problems with high priority capabilities early
    - More time to react
- Often the requirements are not fully known
  - Provides an approach for incrementally learning the requirements (if done well)

High-level Goals of Software Engineering

- improve productivity
  - reduce resources
    e.g., time, cost, personnel
- improve predictability
- improve maintainability
- improve quality
- improve security
  - Most security problems would be eliminated by using good SE practices