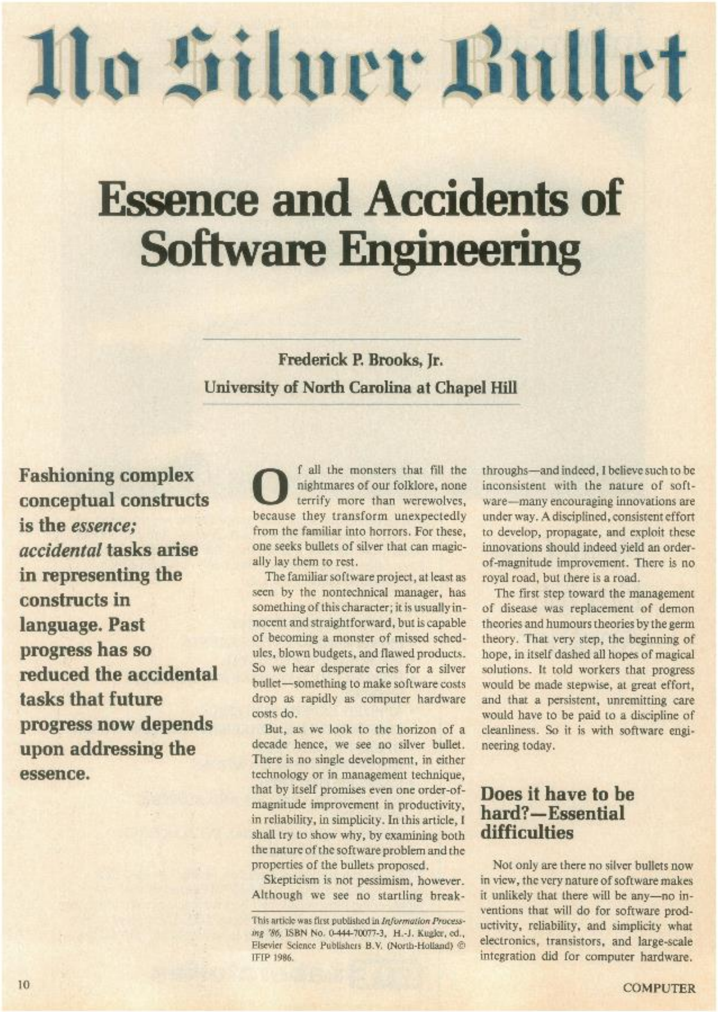
CS 490 – Review of Software Engineering

Instructor: Yongjie Zheng

Software is essentially hard to build – Fred Brooks

- **Complexity**: *software systems differ profoundly from computers, buildings, or automobiles, where repeated elements abound*.
  - E.g., Windows NT - 1.8 million SLOC; Windows XP - 45 million SLOC.
- **Conformity**: *software is designed by different people and must conform to different interfaces*.
  - There is no unifying rules to follow.
- **Changeability**: *The software entity is constantly subject to pressures for change*.
  - Changes may be made throughout the software development lifecycle, even after deployment.
- **Invisibility**: *software is invisible and unvisualizable*.



No Silver Bullet

Essence and Accidents of Software Engineering

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

Fashioning complex conceptual constructs is the *essence; accidental* tasks arise in representing the constructs in language. Past progress has so reduced the accidental tasks that future progress now depends upon addressing the essence.

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 familiar software project, at least as seen by the nontechnical 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. So we hear desperate cries for a silver bullet—something to make software costs drop as rapidly as computer hardware costs do.

But, as we look to the horizon of a decade hence, we see no silver bullet. There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity. In this article, I shall try to show why, by examining both the nature of the software problem and the properties of the bullets proposed.

Skepticism is not pessimism, however. Although we see no startling breakthroughs—and indeed, I believe such to be inconsistent with the nature of software—many encouraging innovations are under way. A disciplined, consistent effort to develop, propagate, and exploit these innovations should indeed yield an order-of-magnitude improvement. There is no royal road, but there is a road.

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.

Does it have to be hard?—Essential difficulties

Not only are there no silver bullets now in view, the very nature of software makes it unlikely that there will be any—no inventions that will do for software productivity, reliability, and simplicity what electronics, transistors, and large-scale integration did for computer hardware.

This article was first published in *Information Processing '86*, ISBN No. 0-444-70077-3, H.-J. Kugler, ed., Elsevier Science Publishers B.V. (North-Holland) © IFIP 1986.

What is software engineering?

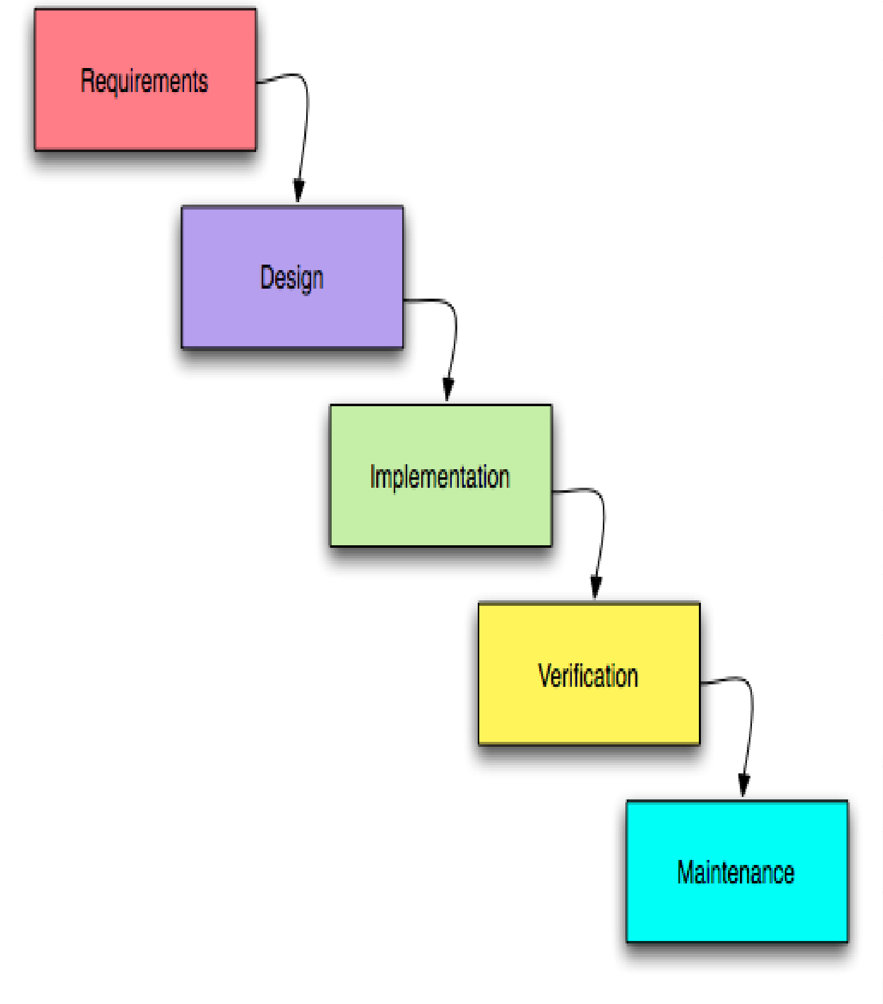
- **Software**: Computer programs and associated documentation (e.g., requirements specification, architectural models).
- **Software engineering**: The systematic application of practical theories, methods, and tools to the production (e.g., design, implementation, and testing) of large software systems.
  - **Programming-in-the-large** (thus, requires collaborations between people).
  - Software products are to be used by people different from the original authors.
    - Requirements analysis, design, and testing.
- Software engineering aims to improve **productivity**, **quality**, and **predictability** of software production.
- Software engineering is a **developing** discipline.
  - E.g., Software Engineering vs. Automotive Engineering

Software Engineering vs. Computer Science

- Computer science: data structure, programming languages, algorithm design and analysis (computation theory), operating system, computer architecture.
  - Computer science includes fundamental concepts, theories, and principles (e.g., data structure, algorithm) about **computer programs**.
  - Computer science is **essential** to software engineering.
- Software engineering focuses on **production** of large software systems. It combines computer science with the **engineering** discipline.
  - Software engineering covers **all the software development activities**: requirements analysis and specification, software design, implementation (e.g., programming), testing, maintenance, etc.
  - Software engineering emphasizes **documentation and tool support**, in addition to computer programs.
  - Software engineering involves **human beings** (e.g., project managers, software developers, software users, and customers) and related facets (e.g., management, psychology, social science).

Software Development Process

- Software process is a set of activities leading to the production of a software product.
  - What product we should work on next.
  - What criteria that work product must satisfy.
- There is no standard or ideal process!
  - For some systems, such as critical systems, a very structured development process is required.
  - For business systems, with rapidly changing requirements, a flexible process is likely to be more effective.
  - For large systems, a mixed process is often preferred.



Software Development Process Models

- **Structured processes (plan-based, document-driven, rigor)**: build a software system in a one-round strict process.
  - **Waterfall model**: each development phase must derive a complete artifact for the next phase to start with.
  - **V-Model**: a variation of waterfall model.
- **Iterative processes (flexible)**: grow a software system in multiple iterations.
  - **Incremental development**: the specification is developed in conjunction with the source code.
  - **Spiral model**: identification, evaluation, and resolution of development risks.
  - **Agile process**: focus on source code, instead of documents; embrace changes; frequent releases; customer involvement; incremental delivery.
- **Other process models:** Prototyping, Rational Unified Process, etc**.**

Requirements Analysis and Specification

- Identify and document functional requirements and non-functional requirements (e.g., performance, reliability) of a software system.
- ***The hardest single part of building a software system is deciding precisely what to build***.- Fred Brooks.
- **Requirements elicitation**
  - Prototyping, interviews, questionnaire, etc.
- **Requirements specification**
- **Requirements traceability**

Software Architecture and Design

- Make **principal design decisions** about the structure, behavior, and/or quality attributes of the system under development:
  - Decompose the system into components.
  - Select protocols for communication, synchronization, and data access.
  - Design components’ internal structure.
- A good architectural design is the key to a successful software product.
- **Design software architecture**
  - **Principles**: conceptual integrity, Information hiding, modularity, abstraction.
  - **Techniques**: Architecture patterns and styles (e.g., pipe-and-filter, MVC), function-oriented design, design patterns, object-oriented design and analysis, architecture recovery (extraction and clustering).
- **Model software architecture**
  - Architecture description languages, UML, metamodels, domain-specific languages.

Software Testing

- Execute the software using **representative data** samples and compare the actual results with the **expected results**.
- **White-box testing** – structural, program-based testing
  - Test cases are designed, selected, and run based on the structure of source code (control-flow coverage, data-flow coverage).
  - Tests the nitty-gritty.
  - Drawbacks: need access to source code.
- **Black-box testing** – functional, specification-based testing
  - Test cases are designed, selected, and run based on specifications.
  - Tests the overall system behavior.
  - Drawbacks: less systematic.
- Unit Testing, Integration Testing, System Testing, and Regression Testing.

Other Software Engineering Topics

- **Software Implementation**: software frameworks, (e.g., Spring, Apache Struts, Hibernate), integrated development environment (e.g., Emacs, Eclipse Plug-ins), code generation.
- **Software Maintenance**: version control, reengineering(e.g., refactoring), change impact analysis, consistency management.
- **Software Metrics:** coupling, cohesion, number of lines of code, constructive cost model (COCOMO).
- **Software Mining**: reverse engineering, program analysis, code search, program comprehension and visualization.
- **Software Reuse and Software Product Lines**
- **User Interface Design**
- **Self-Adaptive Software Systems**