Chapter 1

Software & Software Engineering

What is Software?

Software is: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately manipulate information and (3) documentation that describes the operation and use of the programs.
What is Software?

- Software is developed or engineered, it is not manufactured in the classical sense.
- Software doesn’t “wear out.”
- Although the industry is moving toward component-based construction, most software continues to be custom-built.

Wear vs. Deterioration

- Increased failure rate due to side effects
- Change
- Actual curve
- Idealized curve
Software Applications

- system software
- application software
- engineering/scientific software
- embedded software
- product-line software
- WebApps (Web applications)
- AI software

Software—New Categories

- Open world computing—pervasive, distributed computing
- Ubiquitous computing—wireless networks
- Netsourcing—the Web as a computing engine
- Open source—"free" source code open to the computing community (a blessing, but also a potential curse!)
Legacy Software

Why must it change?

- software must be adapted to meet the needs of new computing environments or technology.
- software must be enhanced to implement new business requirements.
- software must be extended to make it interoperable with other more modern systems or databases.
- software must be re-architected to make it viable within a network environment.

Software Engineering

Some realities:

- a concerted effort should be made to understand the problem before a software solution is developed
- design becomes a pivotal activity
- software should exhibit high quality
- software should be maintainable

The seminal definition:

[Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.
Software Engineering

- The IEEE definition:
  - Software Engineering: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).

A Layered Technology

- tools
- methods
- process model
- a “quality” focus

Software Engineering
A Process Framework

Process framework
Framework activities
- work tasks
- work products
- milestones & deliverables
- QA checkpoints

Umbrella Activities

Framework Activities
- Communication
- Planning
- Modeling
  - Analysis of requirements
  - Design
- Construction
  - Code generation
  - Testing
- Deployment
Umbrella Activities

- Software project management
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Work product preparation and production
- Reusability management
- Measurement
- Risk management

The Essence of Practice

- Polya suggests:
  1. Understand the problem (communication and analysis).
  2. Plan a solution (modeling and software design).
  3. Carry out the plan (code generation).
  4. Examine the result for accuracy (testing and quality assurance).
Understand the Problem

- **Who has a stake in the solution to the problem?** That is, who are the stakeholders?
- **What are the unknowns?** What data, functions, and features are required to properly solve the problem?
- **Can the problem be compartmentalized?** Is it possible to represent smaller problems that may be easier to understand?
- **Can the problem be represented graphically?** Can an analysis model be created?

Plan the Solution

- **Have you seen similar problems before?** Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required?
- **Has a similar problem been solved?** If so, are elements of the solution reusable?
- **Can subproblems be defined?** If so, are solutions readily apparent for the subproblems?
- **Can you represent a solution in a manner that leads to effective implementation?** Can a design model be created?
Carry Out the Plan

- **Does the solution conform to the plan?** Is source code traceable to the design model?
- **Is each component part of the solution provably correct?** Has the design and code been reviewed, or better, have correctness proofs been applied to algorithm?

Examine the Result

- **Is it possible to test each component part of the solution?** Has a reasonable testing strategy been implemented?
- **Does the solution produce results that conform to the data, functions, and features that are required?** Has the software been validated against all stakeholder requirements?
Software Myths

- Affect managers, customers (and other non-technical stakeholders) and practitioners
- Are believable because they often have elements of truth,

*but* …
- Invariably lead to bad decisions,

*therefore* …
- Insist on reality as you navigate your way through software engineering