Introduction

What is Software?

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

Characteristics of Software

- Engineered; NOT manufactured
- Typically, custom-built
  - Any reuse possibility?
- No wear-out

Failure Curve for Hardware
Wear-out vs. Deterioration

- Failure rate due to side effects
- Actual curve
- Idealized curve
- Change
- Time

Software Application Domains

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line software
- WebApps (Web applications)
- AI software
- Batch processing systems
- Systems of systems
- Also ...
  - Data mining
  - Grid computing
  - Cognitive machines
  - Software for nanotechnologies

New Challenges

- Ubiquitous computing
  - Pervasive, distributed computing across networks
- Netsourcing
  - 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.
Problems in Engineering Quality Software

- Cost overruns
- Late deliveries
- Inadequate performance
- Unreliability
- Maintenance cost and difficulty

Software Crisis!!
- Software quality was unacceptably low and overran schedule and budget
- A few examples
  - Therac-25, C-17 project, Ada and DoD – Success?

Engineering Quality Software was Hard in 1982

It was Still Difficult in 1992

In 2007

Source: Stuart Anderson
Software Engineering

- **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

  **Software must be carefully engineered!**

- **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. – Fritz Bauer 1969 @ NATO conference

The IEEE definition (1993):

- 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).


- Software Engineering is about creating high-quality software in a systematic, controlled, and efficient manner.
  - Consequently, there are important emphases on analysis and evaluation, specification, design, and evolution of software.
  - In addition, there are issues related to management and quality, to novelty and creativity, to standards, to individual skills, and to teamwork and professional practice that play a vital role in software engineering.

A Layered Technology

- Automated support (CASE tools)
- Technical guides for modeling, design, ...
- Glue governing activities, artifacts, and actors

A Process Framework

- Software process is not a rigid prescription, but rather agile and adaptable
  - Activities: for broad objectives, domain independent
  - Actions: a set of tasks for creating artifacts
  - Tasks: for small, well-defined objective
  - Development team may pick an appropriate set of actions and tasks

  **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

- Occur throughout the software process
  - Software project management
  - Formal technical reviews
  - Software quality assurance
  - Software configuration management
  - Work product preparation and production
  - Reusability management
  - Measurement
  - Risk management

Adapting a Process Model

- Adaptation is almost always needed
  - Flow of activities/actions/tasks and interdependencies
  - Degree to which actions and tasks are defined within each framework activity
  - Degree to which work products are identified and required
  - Manner which quality assurance activities are applied
  - Manner in which project tracking and control activities are applied
  - Degree of detail & rigor with which the process is described
  - Degree to which the customer and other stakeholders are involved with the project
  - Level of autonomy given to the software team
  - Degree to which team organization and roles are prescribed

The Essence of Practice

- Polya suggests:
  - Understand the problem (communication and analysis).
  - Plan a solution (modeling and software design).
  - Carry out the plan (code generation).
  - Examine the result for accuracy (testing & QA).
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 sub-problems be defined? If so, are solutions readily apparent for the sub-problems?
- 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?
Hooker’s General Principles

- The Reason It All Exists
- KISS (Keep It Simple, Stupid!)
- Maintain the Vision
- What You Produce, Others Will Consume
- Be Open to the Future
  - Never design yourself into a corner. Think of “What if”.
- Plan Ahead for Reuse
- Think first!
  - Complete thought before making actions

Software Myths

- Affect managers, customers (and other non-technical stakeholders) and practitioners
  - Looks reasonable because they often have elements of truth, but invariably lead to bad decisions
- Management myth examples
  - We are behind schedule. Add more programmers...
  - Let’s simply outsource.
- Customer myth examples
  - Objectives are sufficient to begin coding
  - Software is soft. So, requirement changes can be accommodated easily
- Practitioner myth examples
  - If it works, I’m done
  - SE is about unnecessary documentation, and slows us down.

Coming Up Next....

- Software Process
  - Process models
  - CMMI
- Object-Oriented Development Methodology
  - UML
  - Design Pattern
- Software testing researches
- Other topics
  - Component-based software engineering
  - Aspect-oriented programming
  - Project cost estimation methods
  - ...