Software Engineering
Topics

- Computer science v. software engineering
- Definition of software engineering
- Types of software
- The nature of software
- Software history and the “software crisis”
- Software quality
- Elements of a software engineering discipline
- Software development personnel
- Software development process models
Science vs. Engineering

The difference between science and engineering:

- *Science* seeks to explain phenomena through *theory*, *hypothesis*, and *experiment*, in an effort to ascertain *natural laws*
  - For example, chemistry investigates the structure of chemicals and their interactions
- *Engineering* seeks to apply natural laws to the solution of practical problems
  - For example, chemical engineering might use the results of chemistry to come up with a better way of refining gasoline
Computer Science as a Science

- Theory:
  - Computability, automata, discrete computational structures
  - Algorithm analysis

- Hypothesis:
  - That a certain algorithm will solve a problem

- Experiment:
  - Run a program implementing the algorithm
Computer Science as a Science (cont'd)

Theory of Computation

Algorithms & Data Structures

Programming Languages

Theory  Hypothesis  Experiment
Adding a Customer

Customer Problem

Requirements

Computer Science

- Theory of Computation
- Algorithms & Data Structures
- Programming Languages

...
The Difference Between Computer Science and Software Engineering

Computer Science
- Theory of Computation
- Algorithms & Data Structures
- Programming Languages

Software Engineering

Customer

Problem

Tools and Techniques to Solve Problem
Definition of Software Engineering

The process of solving customers’ problems by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints.
Solving Customers’ Problems

- This is the goal of software engineering
- Sometimes the solution is to buy, not build
- Adding unnecessary features does not help solve the problem
- Software engineers must communicate effectively to identify and understand the problem
Definition of Software Engineering (again)

The process of solving customers’ problems by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints.
Systematic Development and Evolution

- An engineering process involves applying well understood techniques in an organized and disciplined way
- Many well-accepted practices have been formally standardized
  - e.g. by the IEEE or ISO
- Most development work is evolution
Definition of Software Engineering (again)

The process of solving customers’ problems by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints.
Large, High-Quality Software Systems

- Software engineering techniques are needed because large systems cannot be completely understood by one person
- Teamwork and co-ordination are required
- Key challenge: Dividing up the work and ensuring that the parts of the system work properly together
- The end-product that is produced must be of sufficient quality
Definition of Software Engineering (again)

The process of solving customers’ problems by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints.
Cost, Time and Other Constraints

- Finite resources
- The benefit must outweigh the cost
- Others are competing to do the job cheaper and faster
- Inaccurate estimates of cost and time have caused many project failures
Types of Software

• Custom
  – For a specific customer

• Generic
  – Sold on open market
  – Often called “COTS” (commercial off-the-shelf) or “shrink-wrapped”

• Embedded
  – Built into hardware
  – Hard to change
More Types of Software

• Real time
  – E.g., control and monitoring systems
  – Must react immediately
  – Safety often a concern

• Data-processing
  – Used to run businesses
  – Accuracy and security of data are key
The Nature of Software

• Software is intangible
  - Hard to understand development effort

• Software is easy to reproduce
  - Cost is in its development
  - In other engineering products, manufacturing is the costly stage

• The industry is labor-intensive
  - Hard to automate
The Nature of Software (cont'd)

- Untrained people can hack something together
  - Quality problems are hard to notice

- Software is easy to modify
  - People make changes without fully understanding it

- Software does not ‘wear out’
  - Deteriorates through design change that increases its complexity and decreases its maintainability
The Nature of Software (cont'd)

• Conclusions
  – Much software has poor design and is getting worse
  – Demand for software is high and rising
  – We are in a perpetual ‘software crisis’
  – We have to learn to ‘engineer’ software
History of the Role of Software

- In the 1950's software development was de-emphasized, because it only contributed to about 20% of overall system cost.
History of the Role of Software

• In the 1950's software development was de-emphasized, because it only contributed to about 20% of overall system cost

• Programmers moved from machine language, to assembly language, to high-level language
History of the Role of Software

- In the 1950's software development was de-emphasized, because it only contributed to about 20% of overall system cost
- Programmers moved from machine language, to assembly language, to high-level language
- In 1968, a NATO report coined the term "software engineering"
History of the Role of Software

- In the 1950's software development was de-emphasized, because it only contributed to about 20% of overall system cost
- Programmers moved from machine language, to assembly language, to high-level language
- In 1968, a NATO report coined the term "software engineering"
- Hardware became faster and cheaper, outpacing the ability of software to keep up
History of the Role of Software

• In the 1950's software development was de-emphasized, because it only contributed to about 20% of overall system cost

• Programmers moved from machine language, to assembly language, to high-level language

• In 1968, a NATO report coined the term "software engineering"

• Hardware became faster and cheaper, outpacing the ability of software to keep up

• By the 1980's the software cost of a system had risen to 80%, and many experts pronounced the field "in crisis"
Elements of the Continuing Software Crisis

- Software is **not delivered on time**
Elements of the Continuing Software Crisis

- Software is **not delivered on time**
- Software is **over budget** (usually by a factor of 2 or more)

Dramatic example: in the early 1980's the IRS hired Sperry to automate tax form processing for $103 million. By 1985 the cost had tripled, the system could not handle the workload, and it had to be replaced.
Elements of the Continuing Software Crisis

- Software is **not delivered on time**
- Software is **over budget** (usually by a factor of 2 or more)

Dramatic example: in the early 1980's the IRS hired Sperry to automate tax form processing for $103 million. By 1985 the cost had tripled, the system could not handle the workload, and it had to be replaced.

- Software is **too complex**. Complexity does not scale linearly: a 1000 line program is more than 10 times as complex as a 100 line program
Elements of the Software Crisis (cont'd)

- Software is **unmaintainable** due to:
  - poor design
  - poor documentation (most software can be understood only by its author, and then only within a few months of writing it)
Elements of the Software Crisis (cont'd)

- Software is **unmaintainable** due to:
  - poor design
  - poor documentation (most software can be understood only by its author, and then only within a few months of writing it)

- Software is **inefficient** (new versions of complex software require machine upgrades)
Elements of the Software Crisis (cont'd)

- Software is **unmaintainable** due to:
  - poor design
  - poor documentation (most software can be understood only by its author, and then only within a few months of writing it)

- Software is **inefficient** (new versions of complex software require machine upgrades)

- Software is **unreliable** due to:
  - poor design (Therac-25 disaster)
  - inadequate testing (market pressures, *beta* releases)
  - impossible testing (SDI)
Elements of a Software Engineering Discipline

1 Abstraction: Identifying hierarchical classes of objects to reason about, ignoring detail
Elements of a Software Engineering Discipline

1 **Abstraction**: Identifying hierarchical *classes* of objects to reason about, ignoring detail

2 **Analysis and Design Methods and Notations**: For example, use of design patterns and UML
Elements of a Software Engineering Discipline

1 Abstraction: Identifying hierarchical *classes* of objects to reason about, ignoring detail

2 Analysis and Design Methods and Notations: For example, use of design patterns and UML

3 User Interface Prototyping: To help user and developer agree on requirements and software functions
Elements of a Software Engineering Discipline

1. **Abstraction**: Identifying hierarchical classes of objects to reason about, ignoring detail.

2. **Analysis and Design Methods and Notations**: For example, use of design patterns and UML.

3. **User Interface Prototyping**: To help user and developer agree on requirements and software functions.

4. **Software Architecture**: Striving for independence of parts through modularity.
Elements of a Software Engineering Discipline (cont'd)

1 Software Process (see later)
Elements of a Software Engineering Discipline (cont'd)

1. **Software Process** (see later)

2. **Reuse**: Not just of software, but also sets of requirements, parts of designs, or groups of test scripts
Elements of a Software Engineering Discipline (cont'd)

1. **Software Process** (see later)

2. **Reuse**: Not just of software, but also sets of requirements, parts of designs, or groups of test scripts

3. **Measurement**: Trying to quantify project goals to evaluate progress (for example, number of bugs per 100 lines of code)
Elements of a Software Engineering Discipline (cont'd)

1. **Software Process** (see later)

2. **Reuse**: Not just of software, but also sets of requirements, parts of designs, or groups of test scripts

3. **Measurement**: Trying to quantify project goals to evaluate progress (for example, number of bugs per 100 lines of code)

4. **Tools and Integrated Environments**: For example, CASE (computer-aided software engineering) tools
Software Engineering Stakeholders

- **Users**
  - Those who use the software
- **Customers**
  - Those who pay for the software
- **Software developers**
- **Development managers**

Software quality means different things to different stakeholders.
The Goal of Software Engineering: Software Quality

Three perspectives on software quality:

1) **Manager view**: does product conform to a good process, and does it meet specs? (externally measured)
The Goal of Software Engineering: Software Quality

Three perspectives on software quality:

1) **Manager view**: does product conform to a good process, and does it meet specs? (externally measured)

2) **User/customer view**: product characteristics measured by fitness for purpose, e.g., correctness, reliability, maintainability (externally measured)
The Goal of Software Engineering: Software Quality

Three perspectives on software quality:

1) **Manager view**: does product conform to a good process, and does it meet specs? (externally measured)

2) **User/customer view**: product characteristics measured by fitness for purpose, e.g., correctness, reliability, maintainability (externally measured)

3) **Developer view**: quality measured by *internal* characteristics (software "metrics") thought to be indicators of good *external* characteristics
User/Customer Views of Quality

- Correctness
- Reliability
- Efficiency
- Integrity
- Usability
- Maintainability
- Testability
- Flexibility
- Portability
- Reusability
- Interoperability
# Developer Views of Quality

<table>
<thead>
<tr>
<th>Completeness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consistency</td>
</tr>
<tr>
<td>Accuracy</td>
</tr>
<tr>
<td>Error tolerance</td>
</tr>
<tr>
<td>Execution efficiency</td>
</tr>
<tr>
<td>Storage efficiency</td>
</tr>
<tr>
<td>Access control</td>
</tr>
<tr>
<td>Operability</td>
</tr>
<tr>
<td>Training</td>
</tr>
<tr>
<td>Communicativeness</td>
</tr>
<tr>
<td>Simplicity</td>
</tr>
<tr>
<td>Conciseness</td>
</tr>
<tr>
<td>Instrumentation</td>
</tr>
<tr>
<td>Self-descriptiveness</td>
</tr>
<tr>
<td>Expandability</td>
</tr>
<tr>
<td>Generality</td>
</tr>
<tr>
<td>Modularity</td>
</tr>
<tr>
<td>Software system independence</td>
</tr>
<tr>
<td>Machine independence</td>
</tr>
<tr>
<td>Communications commonality</td>
</tr>
<tr>
<td>Data commonality</td>
</tr>
</tbody>
</table>
Software Development Steps

- Requirements Analysis and Definition
- System Design
- Program Design
- Program Implementation
- Unit Testing
- Integration Testing
- System Testing
- System Delivery
- Maintenance
Software Developer Roles

- Analyst
- Designer
- Programmer
- Tester
- Trainer

Requirements Analysis and Definition
- System Design
- Program Design
- Program Implementation
- Unit Testing
- Integration Testing
- System Testing
- System Delivery
- Maintenance
Programming Team Organization

Chief Programmer

Assist. Chief Programmer

Senior Programmers

Librarian

Administration

Test Team

Junior Programmers
Software Process Models

A *process* is a series of steps involving activities, constraints, and resources that produce an intended output of some kind.

Two approaches to software process models:

1 *Prescribe* the way that software development *should* progress
2 *Describe* the way that software development *is* done in actuality

In theory, these two kinds of models should be similar or the same. In practice, they are not.

Every software development model has requirements as input and a delivered product as output.
The Opportunistic Model

- Why this model is inadequate:
  - Ignores requirements and design
  - No plan
  - No systematic testing
  - Resulting maintenance costs will be high
The Waterfall Model

- Requirements Analysis
- System Design
- Program Design
- Coding
- Unit & Integration Testing
- System Testing
- Acceptance Testing
- Operation & Maintenance
Elements of the Waterfall Model

• Proposed around 1970
• Assumes that one development stage completes before the next begins
• Has been used to prescribe software development activity
  • For example, Department of Defense Standard 2167-A
• Useful for explaining steps to customers who are not familiar with software development
Drawbacks of the Waterfall Model

• Incorrectly treats software as a manufacturing process

• Manufacturing produces a particular item and reproduces it many times

• Software is not like this, incorporating both problem solving and creativity

• Creativity involves trying alternative solutions, contrasting designs, and learning from failure

• Except for well understood problems, the software process is characterized by iteration
The Uncontrolled Software Process

Requirements Analysis

System Design

Program Design

Coding

Unit & Integration Testing

System Testing

Acceptance Testing

Mainenance
Prototyping

- In practice, neither users nor developers know in advance all the factors that affect desired outcome
- Thus considerable "thrashing" between one activity and back may take place in order to gain knowledge about the problem
- One way to control such thrashing: develop a partial product and allow customers and developers to examine it for feasibility
- Advantage: revisions made at requirements stage rather than the more costly testing stage
- The partially developed product is a prototype
Waterfall Model With Prototyping

Prototyping

- Requirements Analysis
  - System Design
    - Program Design
      - Coding
        - Unit & Integration Testing
          - System Testing
            - Acceptance Testing
              - Operation & Maintenance
Validation and Verification

- Prototyping attempts to minimize surprises encountered during system testing

- Goals of system testing:
  1. **Validation**: ensuring that the system has implemented all of the requirements (coverage)
  2. **Verification**: ensuring that each function works correctly (quality)
Software Development Focus

Two Views:

- The Waterfall Model is a **product-oriented** focus, regarding software as a product on its own, consisting of a set of programs and defining texts.

- This approach "does not permit us to treat systematically questions pertaining to the relationship between software and the living human world" (C. Floyd).

- The **process-oriented** approach "views software in connection with human learning, work, and communication" (C. Floyd).

- The V Model is an example of a process-oriented model.
The Prototype as a Central Element

If a prototype is used in the Waterfall Model, it is usually thrown away. In the **Prototyping Model**, the prototype becomes the product:

```
Prototype Requirements \(\Rightarrow\) List of Revisions \(\Rightarrow\) Prototype Design \(\Rightarrow\) List of Revisions \(\Rightarrow\) Prototype System \(\Rightarrow\) Test \(\Rightarrow\) Delivered System
```

- **System Requirements** (sometimes informal or incomplete)
- User/Customer review
- revise prototype
- List of Revisions
- List of Revisions
- List of Revisions

Note: In the Prototyping Model, the prototype is revisited and revised based on user/customer feedback, leading to a final product that is delivered.
The Spiral Model

Explicitly embraces prototyping and iterative development
Reducing "Cycle" Time: Phased Development
Evolution of Phased Development
Two Approaches to Phased Development

Incremental Development

Iterative Development
"Automatic" Programming: AI Meets Compiler Theory

Formal Specification

Program Code

Test

Compare with requirements; update as needed

System Requirements (sometimes informal or incomplete)

This transformation has automated support

Delivered System