Chapter 1

What is Software Engineering

Shari L. Pfleeger
Joanne M. Atlee

4th Edition
Contents

1.1 What is Software Engineering?
1.2 How Successful Have We Been?
1.3 What Is Good Software?
1.4 Who Does Software Engineering?
1.5 A System Approach
1.6 An Engineering Approach
1.7 Members of the Development Team
1.8 How Has Software Engineering Changed?
Objectives

• What we mean by software engineering
• Software engineering’s track record
• What we mean by good software
• Why a system approach is important
• How software engineering has changed since 1970s
1.1 What is Software Engineering
Solving Problems

• Software products are large and complex
• Development requires analysis and synthesis
  – *Analysis*: decompose a large problem into smaller, understandable pieces
    • abstraction is the key
  – *Synthesis*: build (compose) a software from smaller building blocks
    • composition is challenging
1.1 What is Software Engineering
Solving Problems (continued)

- The analysis process
1.1 What is Software Engineering
Solving Problems (continued)

• The synthesis process
1.1 What is Software Engineering
Solving Problems (continued)

- **Method**: refers to a formal procedure; a formal “recipe” for accomplishing a goal that is typically independent of the tools used
- **Tool**: an instrument or automated system for accomplishing something in a better way
- **Procedure**: a combination of tools and techniques to produce a product
- **Paradigm**: philosophy or approach for building a product (e.g., OO vs structured approaches)
1.1 What is Software Engineering
Where Does the Software Engineer Fit In?

- **Computer science**: focusing on computer hardware, compilers, operating systems, and programming languages

- **Software engineering**: a discipline that uses computer and software technologies as problem-solving tools
1.1 What is Software Engineering
Where Does the SW Engineer Fit in? (continued)

- Relationship between computer science and software engineering
1.2 How Successful Have We Been?

- Perform tasks more quickly and effectively
  - Word processing, spreadsheets, e-mail

- Support advances in medicine, agriculture, transportation, multimedia education, and most other industries

- Many good stories

- However, software is not without problems
1.2 How Successful Have We Been?

Sidebar 1.1 Terminology for Describing Bugs

- **A fault**: occurs when a human makes a mistake, called an error, in performing some software activities.
- **A failure**: is a departure from the system’s required behaviour.
1.2 How Successful Have We Been?

Examples of Software Failure

- IRS hired Sperry Corporation to build an automated federal income tax form processing process
  - An extra $90 M was needed to enhance the original $103M product
  - IRS lost $40.2 M on interests and $22.3 M in overtime wages because refunds were not returned on time
- Malfunctioning code in Therac–25 killed several people
- Reliability constraints have caused cancellation of many *safety critical* systems
  - *Safety–critical*: something whose failure poses a threat to life or health
1.3 What is Good Software?

- Good software engineering must always include a strategy for producing quality software
- Three ways of considering quality
  - The quality of the product
  - The quality of the process
  - The quality of the product in the context of the business environment
1.3 What Is Good Software? 
The Quality of the Product

- Users judge external characteristics
  - (e.g., correct functionality, number of failures, type of failures)
- Designers and maintainers judge internal characteristics (e.g., types of faults)
- Thus different stakeholders may have different criteria
- Need quality models to relate the user’s external view to developer’s internal view
1.3 What Is Good Software?

The Quality of the Product (continued)

- McCall’s quality model

```
Factors to specify  | Quality criteria to build

Correctness       | Traceability, Completeness, Consistency, Accuracy, Error Tolerance, Execution Efficiency, Storage efficiency, Access control, Access audit, Operability, Training
Reliability       | Communicativeness, Simplicity, Conciseness, Instrumentation, Self-descriptiveness, Expandability, Generality, Modularity, Software System Independence, Machine Independence, Communications commonality, Data commonality
Efficiency         | Product Oper., Product Revision, Product Transition
Integrity          | Maintainability, Testability, Flexibility
Usability          | Portability, Reusability, Interoperability

Product Oper.  | Maintainability, Testability, Flexibility
Product Revision | Portability, Reusability, Interoperability
Product Transition | Correctness, Reliability, Efficiency, Integrity, Usability
```

1.3 What Is Good Software?
The Quality of the Process

- Quality of the development and maintenance process is as important as the product quality
- The development process needs to be modeled
- Modeling will address questions such as
  - Where to find a particular kind of fault
  - How to find faults early
  - How to build in fault tolerance
  - What are alternative activities
1.3 What Is Good Software?
The Quality of the Process (continued)

- Models for process improvement
  - SEI’s Capability Maturity Model (CMM)
  - ISO 9000
  - Software Process Improvement and Capability dEtermination (SPICE)
Capability Maturity Model (CMM)

- **Level 1: Initial**
  - ad hoc
  - success depends on people

- **Level 2: Repeatable**
  - track cost, schedule, functionality

- **Level 3: Defined**
  - use standardized processes

- **Level 4: Managed**
  - collect detailed metrics

- **Level 5: Optimizing**
  - continuous process improvement
  - “built-in” process improvement

Software Engineering Institute:
http://www.sei.cmu.edu/cmm/
1.3 What Is Good Software?
The Quality in the Context of the Business Environment

- Business value is as important as technical value
- Business value (in relationship to technical value) must be quantified
- A common approach: *return on investment (ROI)* – what is given up for other purposes
- ROI is interpreted in different terms: reducing costs, predicting savings, improving productivity, and costs (efforts and resources)
1.4 Who Does Software Engineering?

- **Customer**: the company, organization, or person who pays for the software system
- **Developer**: the company, organization, or person who is building the software system
- **User**: the person or people who will actually use the system
1.4 Who Does Software Engineering? (continued)

- Participants (stakeholders) in a software development project

Diagram:

- CUSTOMER
  - Sponsors system development
  - $$$ needs
  - Contractual obligation

- DEVELOPER
  - Builds system

- USER
  - Uses system
  - Needs software system

Pfleeger and Atlee, Software Engineering: Theory and Practice (edited by B. Cheng)
1.5 Systems Approach

- Hardware, software, interaction with people
- Identify activities and objects
- Define the system boundary
- Consider nested systems, systems interrelationship
Use–Case Diagrams (POST)

POST: Point of Sale Terminal

System Boundary

POST

- Buy Item
- Log In
- Refund a Purchased Item

Cashier

Customer

Adapted from Larman “Applying UML and Patterns”
A Different Boundary

- Let us view the whole store as our system
1.6 Engineering Approach
Building a System

- Requirement analysis and definition
- System design
- Program design
- Writing the programs
- Unit testing
- Integration testing
- System testing
- System delivery
- Maintenance
Waterfall Process Model

Requirements → Design → Coding → Testing → Maintenance
1.7 Members of the Development Team

- **Requirement analysts**: work with the customers to identify and document the requirements.
- **Designers**: generate a system-level description of what the system is supposed to do.
- **Programmers**: write lines of code to implement the design.
- **Testers**: catch faults; detect errors.
- **Trainers**: show users how to use the system.
- **Maintenance team**: fix faults that show up later.
- **Librarians**: prepare and store documents such as software requirements.
- **Configuration management team**: maintain correspondence among various artifacts.
1.7 Members of the Development Team (continued)

- Typical roles played by the members of a development team
Why is software development so difficult?

• Communication
  – Between customer and developer
    • Poor problem definition is largest cause of failed software projects
  – Within development team
    • More people = more communication
    • New programmers need training

• Project characteristics
  – Novelty
  – Changing requirements
    • 5 x cost during development
    • up to 100 x cost during maintenance
  – Hardware/software configuration
  – Security requirements
  – Real time requirements
  – Reliability requirements
Why is software development difficult? (cont.)

• Personnel characteristics
  – Ability
  – Prior experience
  – Communication skills
  – Team cooperation
  – Training

• Facilities and resources
  – Identification
  – Acquisition

• Management issues
  – Realistic goals
  – Cost estimation
  – Scheduling
  – Resource allocation
  – Quality assurance
  – Version control
  – Contracts
1.8 How Has Software Engineering Changed?  
The Nature of the Change

• Before 1970s
  – Single processors: mainframes (e.g., IBM, VAX)
  – Designed in one of two ways
    • as a transformation: input was converted to output
    • as a transaction: input determined which function to apply

• After 1970s
  – Run on multiple systems
  – Perform multi-functions

• ...

• After 2000:
  – Distributed (global) development teams
1.8 How Has SE Changed?  
Wasserman's Seven Key Factors (continued)

- The key factors that have changed the software development
1.8 How Has SE Changed?
Wasserman's Discipline of Software Engineering

- Abstractions
- Analysis and design methods and notations
- User interface prototyping
- Software architecture
- Software process
- Reuse
- Measurement
- Tools and integrated environments
1.8 How Has SE Changed?

Abstraction

- A description of the problem at some level of generalization
  - Hide details

Diagram:

```
  Electrical device
   ↓          ↓
   Sensor     Sensor
    ↓          ↓
  Water sensor Air sensor
```
1.8 How Has SE Changed? Analysis and Design Methods and Notations

- Provide documentation
- Facilitate communication
- Offer multiple views
- Unify different views
- Example: Unified Modeling Language
  - de facto standard for OO modeling
1.8 How Has SE Changed?
User Interface Prototyping

- Prototyping: building a small version of a system
  - Help users identify key requirements of a system
  - Demonstrate feasibility
- Develop good user interface
1.8 How Has SE Changed?
Software Architecture

- A system’s architecture describes the system in terms of a set of architectural units and relationships between these units

- Architectural decomposition techniques
  - Modular decomposition
  - Data-oriented decomposition
  - Event-driven decomposition
  - Outside-in-design decomposition
  - Object-oriented decomposition

- Architectural Styles:
  - Pipe and filter; client–server, star–based, p2p
Pipes-and-Filter

- The system has
  - Streams of data (pipe) for input and output
  - Transformation of the data (filter)
Ex: Pipeline Topology (Architecture)

Compiler:

source program → Lexical analyzer → token stream → Semantic analyzer

abstract syntax tree → Code generator → code sequence → Code optimizer → object code
Ex: Star Topology (Architecture)

Monitoring system:

- **Sensors**
  - Sensor status

- **Control panel**
  - Commands, data
  - Display information

- **SafeHome software**
  - Sends sensor status to **Alarm**
  - Command and data

- **Alarm**
  - Sends On/Off signals and alarm type
  - Number tones

- **Telephone line**
  - Receives number tones
Client–Server

• Two types of components:
  – Server components offer services
  – Clients access them using a request/reply protocol

• Client may send the server an executable function, called a callback
  – The server subsequently calls under specific circumstances
1.8 How Has SE Changed?
Software Reuse

- Commonalities between applications may allow reusing artifacts from previous developments (e.g., product lines)
  - Improve productivity
  - Reduce costs
- Potential concerns
  - It may be faster to build a smaller application than searching for reusable components
  - Generalized components take more time to build
  - Must clarify who will be responsible for maintaining reusable components
  - Generality vs specificity: always a conflict
1.11 What this Chapter Means for You

• Given a problem to solve
  - Analyze it
  - Synthesize a solution
• Understand that requirements may change
• Must view quality from several different perspectives
• Use fundamental software engineering concepts (e.g., abstractions and measurements)
• Keep system boundary in mind