Software Engineering

Chapter 12

Software Engineering
The big picture

- "A successful software organization is one that consistently deploys quality software that meets the needs of its users. An organization that can develop such software in a timely and predictable fashion, with an efficient and effective use of resources, both human and material, is one that has a sustainable business." (italics added)
  - Booch, Rumbaugh, & Jacobson
  - The Unified Modeling Language User Guide, p. 3

- Software Engineering
  - The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software
  - ANSI/IEEE standard 610.12-1990
  - Ju-An Wang, Towards Component-Based Software Engineering; JCSS

Kinds of Models
Two main classifications

- Conceptual models
  - Describe how a process works
  - Define concepts, illustrate relationships
  - Descriptive
- Procedural or operational models
  - Present algorithms, steps, or flows
  - Prescriptive
- Software engineering incorporates both kinds of models

Benefits of Modeling
Conceptualize, communicate, validate, refine, & construct

- Symbols in which to express ideas
  - A foundation for success
- A controlled environment for failing
- A language with which to share ideas
- Blueprint for construction

Managing Software Projects
Why is it hard to conduct and manage software projects?

- Intangibility. Software, unlike hardware, is intangible. As a result, software is difficult to manage because it contains no visible milestones to measure progress and quality.
- Complexity. The sheer complexity of software makes it difficult for people to comprehend it, creating not only technical, but management problems as well.
- Volatility of requirements. Software requirements are under constant pressure for change. Because software can be changed more easily than hardware, change is a way of life in software development.
  - Jaak Juntison, Software Project Management: The Manager's View, p. 4
  - Communications of AIS (1999), Volume 2, Article 17

The Software Life Cycle
A systematic approach to software management

- Life cycle
  - Divided into phases
  - Each phase is more manageable, controllable, and predictable
- Phase
  - The span of time between two major milestones
  - A well-defined set of objectives are met
  - Artifacts are produced
  - Focusses attention on main activity
  - The decision is made to move to the next phase
Software Life Cycle Phases

Typical phases

- Requirements
- Analysis
- Design
- Implementation
- Validation
- Maintenance
- Retire

Development

Analysis

Evaluating the problem domain (i.e., the "real world")

- Emphasis is placed on the problem, not the solution
- Creates an external model of the problem/application domain by abstracting essential aspects or features
- Defined in terms of the public or user interface
- Results should be understandable by customers, domain experts, and implementors
- Language/system independent
- Verifies that the requirements are sufficiently complete to proceed
- Called OOA when applied to the object model

Design

Describe the problem domain in an abstract language

- Creates a solution architecture or framework by transforming the analysis results into a form that can be implemented
- Forms a bridge between analysis and implementation
  - Adds data structures and other implementation features
  - Describes user interface
  - Describes data management
  - Describes task management
- Maintain language/system independence
- Called OOD when applied to the object model

Implementation

Building an emulation of the "real world"

- Creates or forms a usable tool or system
- Forms the most significant part of a project’s deliverables
- Final result may be represented as
  - Hardware
  - Software
  - Combination
- Called OOP (programming) when the implementation is in software based on the object model

Waterfall Model

IEEE process standard P1074, 1/1/91

Used in many industries
- Royce introduced to software engineering in 1970 as a flawed model

Pre-development process

Development process

Post-development process

Iterated Waterfall Model

Iteration permits corrections and refinements

System

Requirements

Problem domain

Design

Organization/structure

Implementation

Testing

Validation

Fix/update

Maintenance
The Baseball Model

"Think a little, program a little"

The Development Goal

From problem domain to working system

Creating A Design Document

Engineering "by the numbers"

Example Design Document

Origins of the UML

Major milestones

1990
1991
1992
1993
1994
1995
1996
1997

Object Modeling Technique (OMT)
James Rumbaugh

Object-Oriented Software Engineering (OOSE)
Ivar Jacobson

Object Management Group (OMG)
adopts UML as a standard methodology

Booch Technique
Grady Booch

Rumbaugh joins Rational Software

Jacobson joins Rational Software

Rational Software forms UML Partners Consortium

Inception
- idea is sufficiently well founded to warrant entering the elaboration phase
- strategic/tactical

Elaboration
- requirements articulated & prioritized
- testing planned
- OOA / OOD

Transition
- release the software to users
- evaluate improvements and corrections

Construction
- allocate resources
- requirements & evaluation criteria examined
- OOP
Graphical Class Representation

UML class notation

- **Circle**
  - Features:
    - x
    - y
    - radius
    - draw()
    - setVisible()

  - Class name
  - Attributes: Data, Variables
  - Behaviors: Operations, Services, Functions, Methods

Class Diagrams

- **Person**
- **Car**

  - Inheritance / generalization: An "is a" relation
  - Composition: A "has a" relation

Object and Use Case Diagrams

- **Object Diagram**
  - Instance of a class
  - Shows:
    - Object or instance name
    - Class name

- **Use Case Diagram**
  - System services from a user’s perspective
  - Shows:
    - Actor (initiates action)
    - Use case(s)

Statechart Diagrams

- **Statechart**
  - How an object changes over time
  - A state is a condition or configuration
  - Shows:
    - States (including start and stop)
    - Transitions
    - Events
    - Actions
    - Conditions

Interaction Diagrams

Collaboration & Sequence Diagrams

- **Collaboration diagram**
  - Object structure/organization
  - Shows: objects, links, and message sequences

- **Sequence diagram**
  - Time ordering of messages
  - Shows: objects, lifelines, messages, and control