Software Engineering Principles
Qualities of Software

Reading Assignment
Chapters 1-2

Design Goals and Principles

- Design goals
  - Robustness
  - Adaptability
  - Reusability
- Design principles
  - Abstraction
  - Encapsulation
  - Modularity

Software qualities

- **Software engineering is concerned with software qualities**
- **Qualities (a.k.a. "ilities") are goals in the practice of software engineering**
- **External qualities**
  - **visible to the user**
  - **reliability, efficiency, usability**
- **Internal qualities**
  - **the concern of developers**
  - **they help developers achieve external qualities**
  - **verifiability, maintainability, extensibility, evolvability, adaptability**
- **Product qualities**
  - **concern the developed artifacts**
  - **maintainability, understandability, performance**
- **Process qualities**
  - **deal with the development activity**
  - **products are developed through process**
  - **maintainability, productivity, timeliness**

Robustness

- Handling unexpected input (i.e., string instead of number, entering zero and subsequently dividing by zero)
- Checking input for range and accuracy (e.g., Therac-25)
- Preventing operator mistakes due to alarm overloads (e.g., nuclear power plants)
- Redundancy checks (e.g., Mars lander had imperial and metrics system data mixed together in the same calculation)
- Reaching memory and array limits and other boundaries (e.g., expand and shrink data structures, java.util.Vector)
- One strategy to make programs robust is to use the exception mechanism to deal with unexpected data and situations
- Robustness, reliability and correctness work together and must be designed into the program from the beginning

Adaptability

- The programs we are writing today might last for 30 years
- For programs to stay useful, they must adapt over time
- Different parts of the program evolve at different times and at a different pace
  - Database
  - User interface
- Programs are expected to run on many platforms
  - Windows
  - Mac
  - Unix/Linux
  - Solaris
  - Network-centric
  - Web-centric
- Laws of software evolution

Laws of software evolution

- First Law of Lehman [Leh80]:
  - "Software which is used in a real-world environment must change or become less and less useful in that environment."
- Second Law of Lehman [Leh80]:
  - "As an evolving program changes, its structure becomes more complex, unless active efforts are made to avoid this phenomenon."
- Third Law of Lehman [Leh80]:
  - "Program evolution is self-regulating process. System attributes such as size, time between releases, and the number of reported errors are approximately invariant for each system release."

Laws of software evolution …

- Fourth Law of Lehman [Leh80]:
  - "Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development."
- Fifth Law of Lehman [Leh80]:
  - "Over the lifetime of a system, the incremental system change in each release is approximately constant."
- What can we say about the complexity of the software systems developed over the past 40 years?
  - Constant?
  - Increase?

Software qualities …

- Correctness
  - Ideal quality
  - Established wrt. Requirements specification
  - absolute
- Reliability
  - Statistical quality
  - Probability that software will operate as expected over a give period of time
  - Relative
- Robustness
  - Reasonable behaviour in unforseen circumstances
  - Subjective
  - A specified requirement is an issue of correctness
- Usability
  - Ability of end-users to easily use software
  - Extremely subjective

Software qualities …

- Understandability
  - Ability of developers to understand produced artifacts easily
  - Internal product quality
  - Subjective
- Verifiabilty
  - Ease of establishing desired properties
  - Performed by formal analysis or testing
  - Internal quality
- Performance
  - Equated with efficiency
  - assessable by measurement, analysis, and simulation
- Evolvability
  - ability to add or modify functionality
  - addresses adaptive and perfective maintenance
  - problem: evolution of implementation is too easy
  - evolution should start at requirements or design

Software qualities …

- Reusability
  - ability to construct new software from existing pieces
  - must be planned for
  - occurs at all levels: from people to process, from requirements to code
- Interoperability
  - ability of software (sub)systems to cooperate with others
  - easily integratable into larger systems
  - common techniques include APIs, plug-in protocols, etc.
- Scalability
  - ability of a software system to grow in size while maintaining its properties and qualities
  - assumes maintainability and evolvability
  - goal of component-based development

Software qualities …

- Heterogeneity
  - ability to compose a system from pieces developed in multiple programming languages, on multiple platforms, by multiple developers, etc.
  - necessitated by reuse
  - goal of component-based development
- Portability
  - ability to execute in new environments with minimal effort
  - may be planned for by isolating environment-dependent components
  - necessitated by the emergence of highly-distributed systems
  - an aspect of heterogeneity
- Maintainability
  - the ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment
  - Addresses corrective, adaptive, and perfective maintenance

Design Goals and Principles

- Design goals
  - Robustness
  - Adaptability
  - Reusability
- Design principles
  - Abstraction
  - Encapsulation
  - Modularity

Design principles: Abstraction

- To emphasize the important aspects and deemphasize immaterial aspects
- For example, a program is a string of bits, characters, tokens, syntax tree, classes, logical units, subsystems, application
- Levels of abstraction
  - Application
    - Concepts, business rules, policies
  - Function
    - Logical and functional specifications, non-functional requirements
  - Structure
    - Data and control flow, dependency graphs
    - Structure and subsystem charts
    - Software architectures
  - Implementation
    - AST's, symbol tables, source text

Design Principles: Encapsulation

- **Packaging code and data that belong together**
- **Information hiding**
  - **Information hiding allows details to be hidden from those who do not need to see them.**
  - **Different components of a software system should not reveal the internal details of their respective implementations**
  - **Gives programmers freedom on how to implement the details of a system**
  - **Rule of thumb:**
    - **usually we hide the internal data/state associated with an object**
    - **usually we hide the implementation details of an object's methods**
- **Localize change**

Design principles: modularity

- An organizing structure in which different components of a software system are divided into separate functional units
- Separation of concerns
- The architecture of a house can be viewed as several interacting units
  - Electrical, heating and cooling, plumbing, structural subsystem
- The elements of susbsystems can be readily replaced if certain standards are followed
  - Facilitates reuse and understanding
- Subsystems are organized into hierarchies of subsystems
  - Part-of hierarchies (i.e., packages, classes, fields, methods, statements, local variables)
  - Is-a hierarchies (i.e., inheritance)