**Software Design Principles**

“Producing the software blueprint”

**What Is Design?**

- Explaining the idea/concept of something
- Usually with graphical diagrams
- With the intention to build from the explanation
- The design is a representation of a product or a system with sufficient detail for implementation

**Lecture Objectives**

- To understand the importance of design in developing quality software
- To describe the translation from the requirements analysis model to the design model
- To understand the principles that guide proper design of software

**The Second Task**

```
Analysis → Problem
Design → Models
Development → Testing
Testing → Solution
```

**Designing A House**

- If you are asked to design a house...

```
D  W  |  W  
 Kitchen  |  Room 2

D  D  |  WC
Living Room  |  Room 1

D  W  |  W
```

**Designing Software**

- From our understanding of the problem, we start building the software
- Translate the analysis model into the design model
- Map the information from the analysis model to the design representations - data design, architectural design, interface design, procedural design
Translation Model

Design Principles

- Design process should not suffer from “tunnel vision”
- The design should be traceable to the analysis model
- The design should not reinvent the wheel; Time is short
- The design should “minimize intellectual distance” between the software and the problem in the real world

Design Principles (Continued)

- The design should exhibit uniformity and integration
- The design should be structured to accommodate change
- The design should be structured to degrade gently.

Design Concepts

Fundamental concepts which provide foundation to design correctly:
- Abstraction
- Refinement
- Modularity
- Software Architecture
- Control Hierarchy

Abstraction

- Identifying important features for representation
- There are many levels of abstraction depending on how detailed the representation is required
- Data abstraction - representation of data objects
- Procedural abstraction - representation of instructions
Refinement

**Stepwise refinement** - top-down design strategy by Niklaus Wirth
**Starting at the highest level of abstraction, every step of refinement 'decompose' instructions into more detailed instructions**
**Complementary to abstraction**

Software Architecture

**Modules can be integrated in many ways to produce the system**
**Software architecture is the overall structure of the software**
**The hierarchy of components and how they interact, and the structure of data used by the components**
**Use of framework models, and possible reuse of architectural patterns**

Modularity

**Software is divided into separately named and addressable modules**
**“Divide and conquer” approach - problem is broken into manageable pieces**
**Solutions for the separate pieces then integrated into the whole system**

Software Architecture Patterns

**Recurring pattern help designers reuse successful designs by basing new designs on prior experience.**
**A designer who is familiar with such patterns can apply them immediately to design problems without having to rediscover them.**

Divide And Conquer

**Why use Design Patterns?**
**Reuse successful practices.**
**Not new** – recognised that this is something that engineers have done for years.
**Improve communication**
**Step towards a software engineer’s handbook**
Examples of Software Architecture

Control Hierarchy

- Hierarchy of modules representing the control relationships
- A *super-ordinate* module controls another module
- A *subordinate* module is controlled by another module
- Measures relevant to control hierarchy: depth, width, fan-in, fan-out

Software Procedure

- Processing details of individual modules
- Precise specification of processing, including sequence of events, exact decision points, repetitive operations, and data organization/structure
- Procedure is layered - subordinate modules must be referenced in processing details

Structure Terminology

Information Hiding

- Information (procedure and data) contained within a module is inaccessible to other modules that have no need for such information
- Effective modularity is achieved by independent modules, that communicate only necessary information
- Ease of maintenance - testing, modification localized and less likely to propagate
References


“Software Engineering” by Ian Sommerville, Addison-Wesley, 2001