Principles of Software Design Methods

Lecture 2B

In this lecture you will learn:

•How to make good designs?

•The causes of difficulties

•The basic vehicles to deal with difficulties

•Design process and strategies

•Design objectives

•Design methodology

Nature of Design Problems

- Design problems are "ill-structured" and "wicked"
- Nature of Design Problems
  - No definitive formulation of the problem
  - No definitive solution to the problem
  - No definitive way of solving the problem

No definitive formulation of the problem

The initial goals are usually vague

Many constraints and criteria are unknown

The context of the problem is often complex and poorly understood

No definitive solution to the problem

Solutions are often not true or false, but good or bad

Often no objective criterion for the evaluation of a solution

Often no best solution, even criteria that can be used as a "Stopping Rule"

No definitive way of solving the problem

Resolving a discrepancy or inconsistency may pose another problem in its turn

The formulation of a problem often depends on the way of solving it

Many assumptions and uncertainty can be exposed only by proposing solution concepts

Many constraints and criteria emerge as a result of evaluating solution proposals

Sub-solutions of the design sub-problems can be found to be inter-connected with each other in ways that form a pernicious circular structure to the problem

Major Causes of Difficulties in Software Design

- From the article “No Silver Bullet: Essence and Accidents of Software Engineering”, major causes of difficulties in software design:
  - Complexity
  - Conformity
  - Changeability
  - Invisibility

Brooks, F. P. Jr,
No Silver Bullet: Essence and Accidents of Software Engineering,
IEEE Computer, 1987, pp10~19

- **Complexity -** It is an essential property of software.
- **Conformity -** Software is expected to conform to the standards imposed by other components, such as hardware, or by external bodies, or be existing software.
- **Changeability -** Software suffers from constant needs of changes.
- **Invisibility -** Any forms of representations that are used to describe software will lack any form of visual link that can provide an easily grasped relationship between the representation and the system.

Common Design Errors

- There are four common design errors:
  - Incorrectness
  - Inconsistency
  - Ambiguity
  - Inferiority

•**Incorrectness -** The design does meet the users’ requirements on its functionality and features.

•**Inconsistency -** Different parts or aspects of the design conflict with each other. Consequently, it does not work. (For example, if two design statements make conflicting assumptions about the functionality of a component or the meaning of a data item).

•**Ambiguity** - The design specification may be interpreted in several different ways, or it is not clear enough.

Ambiguity causes errors in the implementation of the design due to inconsistent interpretations made in the implementation process.

•**Inferiority** - The design does not address quality requirements adequately.

Typically inefficiency and inflexibility, etc. Inflexibility causes the designed software to be difficult to change.

Dealing with Complexity

- Witt, Baker & Merritte's axioms

Separation of concerns:
  - **The Axiom of Separation of Concerns:** A complex problem can best be solved by initially devising an intermediate solution expressed in terms of simpler independent problems.
  - **The Axiom of Comprehension:** The mind cannot easily manipulate more than about seven things at a time.

Abstraction:
  - **The Axiom of Translation:** Design correctness is unaffected by movement between equivalent contexts.
  - **The Axiom of Transformation:** Design correctness is unaffected by replacement of equivalent components.

Software Design Objectives

- This property is neither a product-oriented quality attribute, nor a process-oriented quality attribute. It should be considered as as a good guideline for how to make a good design.
  - Modularity
  - Portability
  - Malleability
  - Conceptual Integrity
  - Intellectual Control

Witt, Baker & Merritte, 1994

Modularity

- The design should be composed of replaceable, self-contained assemblies of elementary parts, thereby aiding both the initial development and the later maintenance.

Portability

- Individual parts of the design, as well as the design as a whole, should be capable of reuse in different environments.
- The designed product should be able to be moved unchanged from test environments to operational environments, and from one operational environment to another.

Malleability (also known as Modifiability and Flexibility)

- The design should facilitate adaptation to changing end-user requirements, for example, changes based on new problems in the end user's world, the discovery of a need for information not previously anticipated or included in the original specifications.

Conceptual Integrity

- The design should exhibit harmony, symmetry and predictability.
- The system should appear to reflect the mind of a single person, and to faithfully adhere to a single concept.
- There should be no surprises for its user or its maintainer; knowledge gained in one use or change should be immediately transferable to the next.

Intellectual Control

- The design process should be under intellectual control.
- An evolving design is under intellectual control if, despite its complexity, it is deeply understood by those responsible for its correctness; they have mastery of its form and content.
- Managers may understand cost and schedules; but those responsible for the design itself must understand the manner in which the parts interrelate, the rationale and criticality of design choices, and effect of proposed change.

How to Achieve Design Objectives

- The Principle of Modular Designs
- The Principle of Portable Designs
- The Principle of Malleable Designs
- The Principle of Conceptual Integrity
- The Principle of Intellectual Control

The Principle of Modular Designs

•Modularity can be achieved by

Dividing large aggregates of components into units having loose inter-unit coupling and high internal cohesion

Abstracting each unit's behavior so that its collective purpose can be known

Recording each unit's interface so that it can be employed

Hiding its design details so that it can be changed

The Principle of Portable Designs

•Portability can be achieved by employing abstract context interfaces

The Principle of Malleable Designs

•Malleability can be achieved with designs that model the end-user's view of the external environment

The Principle of Conceptual Integrity

•Conceptual integrity can be achieved by uniforming application of a limited number of design forms

The Principle of Intellectual Control

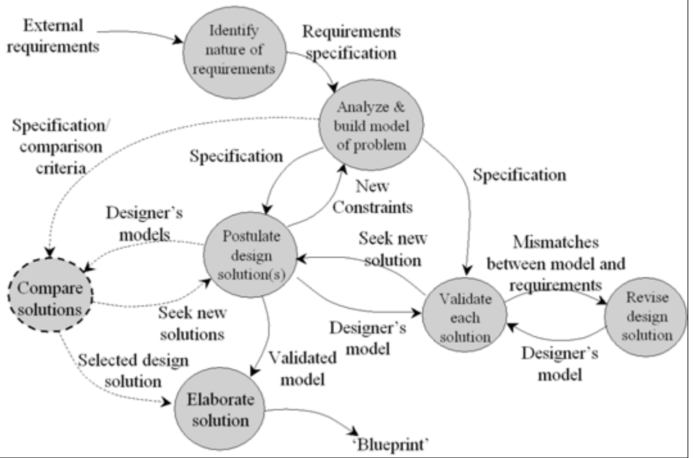
•Intellectual control can be achieved by recording designs (after developing a design strategy) as hierarchies of increasingly detailed abstractions.

Design Processes

- Involve a wide range of activities, which consists of at least 4 aspects
  - The action carried out in the activity
  - The participants
  - The input information
  - The output or the result of the action
- Process model
  - The activities involved in the design process
  - The interrelationships between the activities
- Generic process models
  - Apply to the design of all kind of products.
- Specific Process models apply to specific types of systems
  - Software process model: for software development and design
  - Safety system lifecycle: for safety related systems

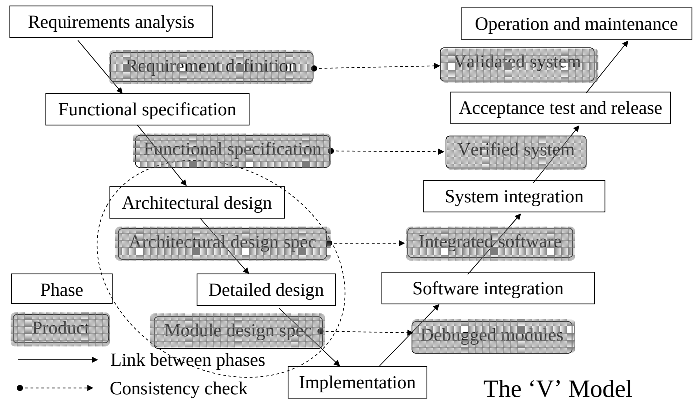
There are also other aspects, such as the conditions and constraints on which the activity to be carried out.

A Generic Process Model of Designs



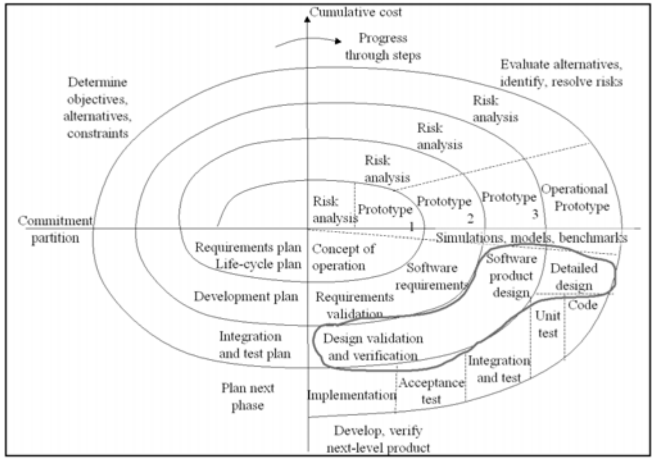
Descriptive modes: Describe what designers do in design.

Context of Software Design



To understand the role of software design, it is important to understand the context in which it fits, the software engineering life cycle. Thus, it is important to understand the major characteristics of software requirements analysis vs. software design vs. software construction vs. software testing.

The Spiral Model



Spiral Development

•Process is represented as a spiral rather than as a sequence of activities with backtracking.

•Each loop in the spiral represents a phase in the process.

•No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.

•Risks are explicitly assessed and resolved throughout the process.

Spiral Model Sectors

•Objective Setting - Specific objectives for the phase are identified.

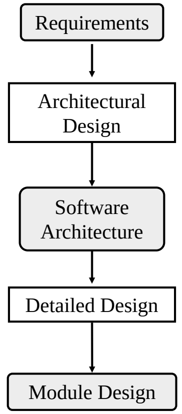
•Risk Assessment and Reduction - Risks are assessed and activities put in place to reduce the key risks.

•Development and Validation - A development model for the system is chosen which can be any of the generic models.

•Planning - The project is reviewed and the next phase of the spiral is planned.

Design Stages

- Requirement
- Architectural Design
- Software Architecture
- Detailed Design
- Module Design



Design Stages – Architectural Design

•The structure, i.e. the composition of components

•The global control structures

•The protocols for communication, synchronisation, and data access

•The assignment of functionality to components

•Physical distribution

•Scaling and performance

•The dimensions of evolution

•The selection among design alternatives

Design Stages – Detailed Design

•The data structures and the algorithms for each component

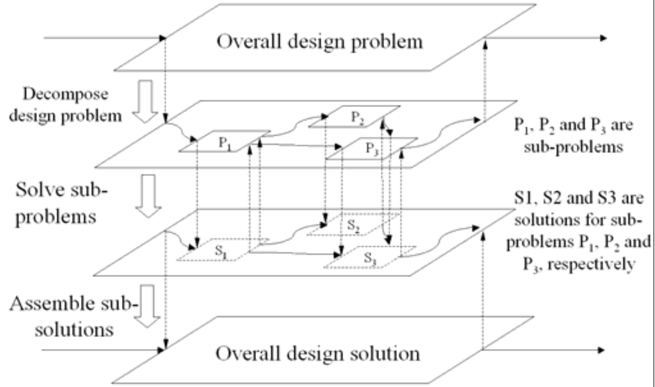
•The details of the user interface, and input/output formats

•The selection of language, libraries and development tools

Design Strategies

- A design strategy is used in a design method to guide the process of building up design models.
  - Decompositional Design (Top-down)
  - Compositional Design (Bottom-up)
  - Design Patterns (Reuse of Designs)
  - Evolutionary Design (Trial and Error)
- Each design strategy can be considered as a **Prescriptive** design process, which are
  - Guidelines for the creation of designs
  - Ideal processes that usually lead to good designs

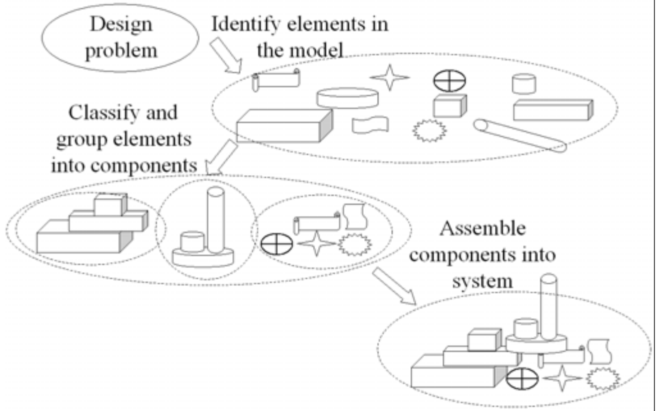
Decompositional Methods



Decompositional Methods take a top-down approach to the design process.

- It starts with an original description of the problem or a model of the original problem.
- The original problem is, then, decomposed into a number of sub-problems.
- These sub-problems are then solved separately.
  - The sub-problem is solved directly if possible
  - If a sub-problem is still too complicated to be solved directly, it is further decomposed.
- The solutions of the sub-problems are put together to form a solution of the original problem.

Compositional Methods

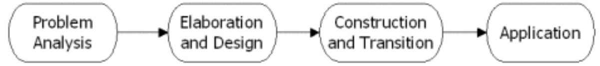


U08182 © Peter Lo 2010 15

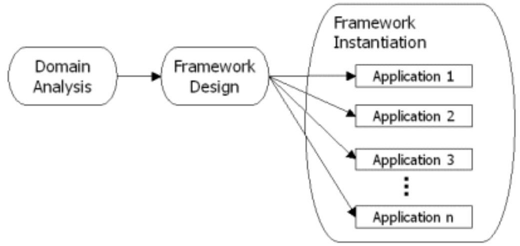
Compositional Strategy is a bottom-up approach.

- •It starts with identifying a set of particular entities or objects involved in the problem.
- •These entities and objects are described, classified and grouped.
- •For each group, the relationships between the entities are identified so that links between entities are established. Such groups form the components of the model.
- •These components are further classified and grouped. The relationships between the components are identified to make larger components.
- •This composition process continues until a complete model is built.

Design Patterns and Design Reuse



Traditional Object-Oriented Design



Framework Development Process

U08182 © Peter Lo 2010 16

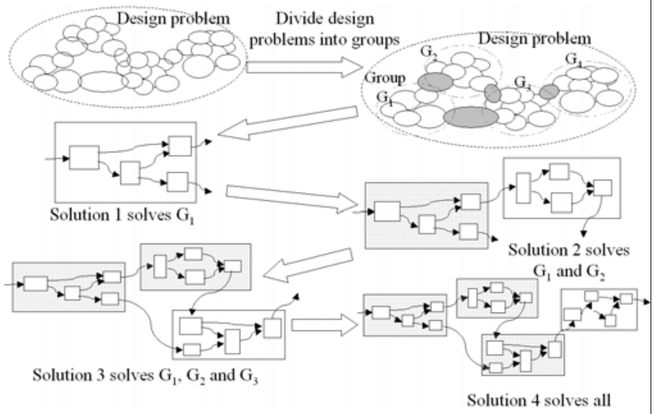
•Certain types of design problems in certain application domains may have a great deal of similarities in the design solutions that are proved to be good designs.

- •Common structure and other design features are abstracted into a template of designs.
- •Once a problem is identified to be an instance of such a class of problems, the design template can be instantiated and a good design can be relatively easily obtained.

•Such a template is called a design pattern.

•The use of design pattern is a reuse of design.

Evolutionary Design



•Trial-and-error is perhaps the most basic approach to all designs.

•It involves the creation of a design and evaluation of the design against the requirements and constraints.

•If some requirements and/or constraints are not satisfied, the design is modified and a new design, even new designs, is created.

•The cycle of creation and evaluation stops until a satisfactory design is obtained.

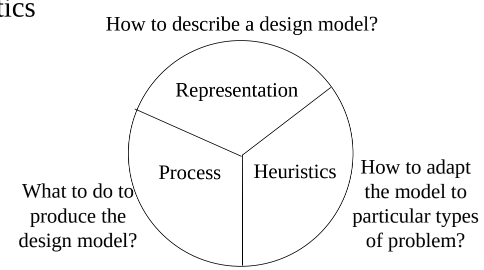
•However, there is no guarantee that a satisfactory design can always be obtained.

•Example of software evolutionary design methods:

•Program transformation

Design Methodology

- Representation
- Process
- Heuristics



•**Representation -** Consists of one or more forms of notations to describe and/or model both the structure of the initial problem and that of the solution. It should also facilitate the analysis of the model.

•**Process -** Describes the procedures to follow in developing the solution and the strategies to adopt in making choices.

•**Heuristics -** Provides guidelines on the ways in which the activities defined in the process part can be organized for specific classes of problems.

Well-Known Software Design Methods

- Jackson Structured Programming (JSP) and Jackson System Development (JSD) methods
- Structured methods
  - e.g. SSA/SD, SADT and SSADM;
- Object-oriented and Object-based methods:
  - HOOD
  - More recent developments in the UML and united process;
- Formal methods:
  - Model-oriented formal specification methods,
  - axiomatic and algebraic formal specification methods,
  - refinement calculus,
  - formal proof methods,
  - program transformation methods, etc.
- Design patterns and software architectural styles.

Further Readings

- Zhu, H., Software Design methodology. Chapter 3 (design principles).
- Budgen, D, Software Design, Addison-Wesley, 1994. Chapter 1~ 3 (design process), Chapter 8 (design methods), Chapter 9 (design strategies).
- Bernard Witt, Terry Baker and Everett Merritt, Software Architecture and Design, Van Nostrand Reinhold, New York, 1994, Chapter 1~2, pp1~35. (the principles of software design).