Software Design

Massimo Felici

Room 1402, JCMB, KB

0131 650 5899

mfelici@inf.ed.ac.uk

Software Design

- IEEE standard glossary: "the **process** of defining the **architecture**, **components**, **interfaces** and other **characteristics** of a system or component."
  - The Link to **Requirements**
  - Key Design **techniques** and **issues**
  - **Structure** and **architecture**
    - the main elements of software that need to be managed
    - design in the large and design in the small
  - Design notations
  - Design quality and evaluation
- Design is a pervasive activity
  - often there is no definitive solution
  - solutions are highly context dependent
  - **No "magic bullet" in general**

The Link to Requirements

- Design links requirements to "implementable specifications"
- Traceability - retaining the link from requirements to components
  - By allocating a particular requirement to a particular component as we decompose, e.g., in VolBank, we might require a log
  - By decomposing requirements into more refined requirements on particular components, e.g., a particular function in VolBank might be realized across several components
  - Some requirements (e.g., usability) are harder to decompose, e.g., it takes 30 minutes to become competent in using the system
- We might require traceability back from the design

Traceability

- There are four basic types of traceability:
  - **Pre-traceability** (e.g., requirements-sources, requirements-rationale, etc.)
    1. **Forward-to** requirements traceability links other documents preceding requirements (e.g., users document)
    2. **Backward-from** requirements traceability links requirements to their sources (e.g., rationale)
  - **Post-traceability** (e.g., requirements-architecture, requirements-design, requirements-interface, etc.)
    3. **Forward-from** requirements traceability links requirements to design and implementation
    4. **Backward-to** requirements traceability links design and implementation back to requirements.
- To manage requirements, you need to maintain traceability information (e.g., Traceability Tables)
  - Requirements Management Tools support traceability practice (e.g., IBM Rational RequisitePro or Telelogic DOORS)

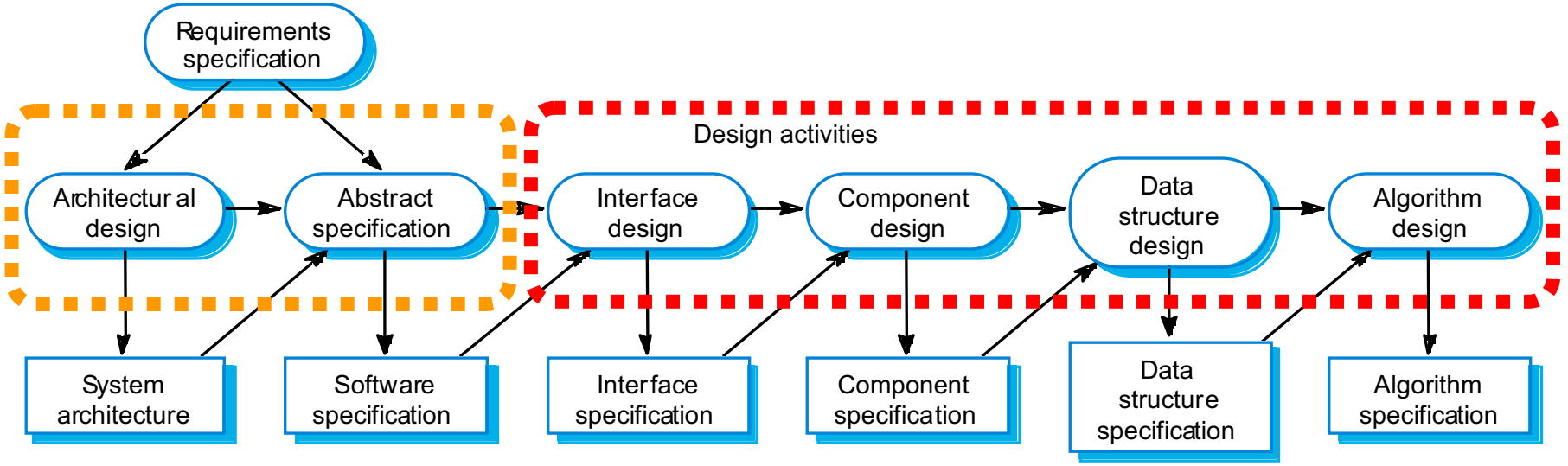
Key Design Techniques

- **Abstraction**
  - ignoring detail to get the high level structure right
- **Decomposition** and **Modularization**
  - big systems are composed from small components
- **Encapsulation/information hiding**
  - the ability to hide detail (linked to abstraction)
- Defined **interfaces**
  - separable from implementation
- Evaluation of structure:
  - **Coupling**: How interlinked a component is
  - **Cohesion**: How coherent a component is

Key Issues in Software Design

- **Concurrency**
  - Often there is significant interaction that needs management
  - What are the main concurrent activities?
  - How do we manage their interaction?
  - VolBank: matching and specifying skills and needs goes on concurrently
- **Workflow** and **event handling**
  - What are the activities inside a workflow?
  - How do we handle events?
- **Distribution**
  - How is the system distributed over physical (and virtual) systems?
- **Error handling** and **recovery**
  - Action when a physical component fails (e.g., the database server)
  - How to handle exceptional circumstances in the world
  - VolBank: a volunteer fails to appear
- Persistence of **data**:
  - Does data need to persist across uses of the system, how complex?
  - How much of the state of the process?
- Can you think through some of these issues for VolBank?

A Design Process



[Sommerville, 2004]

- Main activity in design:
  - decomposing system (components) into smaller more manageable components
  - definitions of components that are easily codable
- Usually a two stage process: Architectural Design and Detailed Design
  - Architectural Design (or High-level Design)
    - What are the components and how do they relate?
    - How does the system architecture deal with issues that pervade the system?
  - Detailed Design deals with the function and characteristics of components and how they relate to the overall architecture.

Architecture and Structure

- **Architectural structures** and **viewpoints**
  - attempt to deal with facets separately, e.g., physical view, functional (or logical) view, security view, etc.
- **Architectural styles**, for example:
  - Three-tier architecture for a distributed system (interface, middleware, back-end database)
  - Blackboard
  - Layered architectures
  - Model-View-Controller
  - Time-triggered
- **Design patterns**
  - small-scale patterns to guide the designer
- **Families** and **frameworks**
  - component set and ways of plugging together
  - software product lines

Architectural Design

- **Advantages**:
  - Stakeholder Communication
  - System Analysis
  - Large-scale reuse
- **Design Strategies**
  - **Function Oriented**: sees the design of the functions as primary
  - **Data Oriented**: sees the data as the primary structured element and drives design from there
  - **Object Oriented**: sees objects as the primary element of design
- There is no clear distinction between Sub-systems and modules. Intuitively,
  - Sub-systems are independent and composed of modules, have defined interfaces for communication with other sub-systems
  - Modules are system components and provide/make use of service(s) to/provided by other modules

Architecture Models

- Architecture Models that may be developed may include:
  1. A **static structural model** that shows the sub-systems or components that are to be developed as separate units.
  2. A **dynamic process model** that shows how the system is organized into processes at run-time. This may be different from the static model.
  3. An **interface model** that defines the services offered by each sub-system through their public interface.
  4. A **relationship model** that shows relationships such as data flow between the sub-systems.

Quality Analysis and Evaluation

- The system architecture affects the quality attributes of a system
- **Quality attributes**:
  - Performance, security, availability,… modifiability, portability, reusability, testability, maintainability, etc.
- **Quality analysis**:
  - reviewing techniques, static analysis, simulation, performance analysis, prototyping
- **Measures (metrics)**:
  - Defined measure on the design
  - Predictive, but usually very dependent on the process in use

Architectural Design: Key Points

- The software **architecture** is the fundamental framework for **structuring the system**
- Different **architectural models** (e.g., system organizational models, modular decomposition models and control models) may be developed
- **Design decisions** enhance **system attributes**
  - **Performance**, e.g., localize operations to minimize sub-system communication
  - **Security**, e.g., use a layered architecture with critical assets in inner layers
  - **Safety**, e.g., isolate safety-critical components
  - **Availability**, e.g., include redundant components in the architecture
  - **Maintainability**, e.g., use fine-grain self-contained components

What are the Architect's Duties?

- Get it **Defined**, **documented** and **communicated**
  - Act as the emissary of the architecture
  - Maintain morale
- Make sure
  - everyone is **using** it (correctly)
  - management understands it
  - the **software** and system **architectures** are in synchronization
  - the right **modeling** is being done, to know that **quality attributes** are going to be met
  - the architecture is not only the right one for **operations**, but also for **deployment** and **maintenance**
- Identify
  - architecture timely **stages** that support the overall organization progress
  - suitable **tools** and **design** environments
  - (and interact) with **stakeholders**
- Resolve
  - disputes and make tradeoffs
  - technical problems
- Manage **risk** identification and risk **mitigation strategies** associated with the architecture
  - understand and plan for **evolution**

Comparing Architecture Design Notations

- **Modeling Components**:
  - Interface, Types, Semantics, Constraints, Evolution, Non-functional Properties
- **Modeling Connectors**:
  - Interface, Types, Semantics, Constraints, Evolution, Non-functional Properties
- **Modeling Configurations**:
  - Understandable Specifications, Compositionality (and Conposability), Refinement and Traceability, Heterogeneity, Scalability, Evolvability, Dynamism, Constraints, Non-functional Properties

UML Design Notations

- **Static Notations**:
  - Component diagrams
  - Class and object diagrams
  - Deployment diagrams
  - CRC Cards
- **Dynamic Notations**:
  - Activity diagrams
  - Collaboration diagrams
  - Statecharts
  - Sequence diagrams

VolBank: Example

- Suppose we consider two requirements:
  - That a request for a volunteer should produce a list of volunteers with appropriate skills.
  - The system shall ensure the safety of both volunteers and the people and organizations who host volunteers.
    - This may decompose into many more specific requirements:
      - That the organization has made reasonable efforts to ensure a volunteer is bona fide.
        - » That we have a confirmed address for the individual: i.e., the original address is correct, and only the volunteer can effect a change in address.

Reading/Activity

- Please read
  - Chapter 3 – Software Design - of the SWEBOK for an overview of the work on design
  - Nenad Medvidovic and Richard N. Taylor. *A Classification and Comparison Framework for Software Architecture Description Languages.*

Summary

- Design is a complex matter
- Design links requirements to construction, essential to ensure traceability
- Generally two stages:
  - Architecture Design (or High-level Design)
  - Detailed Design
- Many notations and procedures to support design
- More domain-specificity for easier design task