Large Scale Systems Design  
G52LSS

Lecture 18 – The Design Phase

- Elements of Design
- User Interface Design
- Data Storage Design
- Architecture Design
- Program Design

Learning outcomes: describe the most important aspects of the design phase in the SDLC; understand interface structure design as a tool for user interface design; understand structure chart as a tool for program design.

Three main design strategies:
- Develop a custom application in house – builds technical skills and functional knowledge within the organisation
- Develop the application by customising off-the-shelf software products – customisation by workaround or systems integration
- Outsource the development of the application – risk to compromise confidential information, no skills learn from development

Consideration aspects when selecting a design strategy:
- Business need – unique, common, critical for business?
- In-house experience – does functional/technical experience exist?
- Project skills – strategic/non-strategic/desirable?
- Project management – skilled/non-skilled?
- Time frame – flexible/tight?

Elements of Design

Design Selection
Architecture Design
Interface Design
Data Storage Design
Program Design

Physical Models

Logical process (DFDs) and data (ERDs) models are converted to physical process and data models, which show implementation details including:
- Actual, specific technology – available for the system?
- Format of information – files/databases/emails/reports?
- Human interaction with system – details

Transition from logical to physical DFD
- Add implementation references
- Draw a human-machine boundary
- Add system-related data stores, data flows and processes
- Update data elements in the data flows
- Update the metadata in the CASE repository
User Interface Design

Interaction of the system with external entities:
- user interfaces
- system interfaces

Aims of good user interface design:
- pleasing to the eye
- simple to use
- minimise user's effort to accomplish tasks

Principles of user interface design:
- Layout – consistency throughout the system
- Content awareness – for status of system
- Aesthetics – functional vs. pleasing the user
- User experience – easy to use vs. easy to learn
- Consistency – in functionality throughout the system
- Minimal user effort – simple to use

User Interface Design Principles

Navigation – the goal is to make the system as simple as possible to use.
- Prevent mistakes – e.g. don’t display commands that cannot be used
- Simplify recovery from mistakes – e.g. the typical ‘undo’ option
- Use consistent grammar order – e.g. object then action or vice-versa

The system should provide (clear, concise and complete) navigation messages to the user in order to respond to users’ action and inform of the status of the interaction: error messages, confirmation messages, acknowledge messages, delay message, help message.
**Input** – the goal is to simply and easily capture accurate information.
- Adequate online and batch processing – is real time information needed?
- Capture data at the source – avoid intermediary collection
- Minimise keystrokes – minimise cost in time and money
- Validate input – very important to ensure accuracy

**Output** – the goal is to present information to users so they can accurately understand it with the least effort.
- Understand report usage – design accordingly to users needs
- Manage information load – not all information, not a data dump
- Minimise bias – ways of sorting records, highlighting fields, etc.

---

**Data Storage Design**

**Step 1** Select format of data storage: files, databases or a combination of these two.

**Criteria for selection:**
- Data types
- Type of application system
- Existing software formats
- Future needs

**Step 2** Optimise performance of data storage.
- Optimise storage efficiency
- Optimise access speed
- Estimate adequate storage size

---

**Types of files:**
- master files – store core information to the business
- look-up files – store static values used commonly for validation
- transaction files – information to update the master file
- audit files – records the ‘before’ and ‘after’ images of data
- history files – stores past transactions

**Types of databases:**
- legacy – based on older technology
- relational – based on collection of tables each with a primary key
- object-oriented – based on the object oriented paradigm
- multi-dimensional – used for data warehousing for decision making support systems

---

**Architecture Design**

**Software** (basic functions)
- Data storage
- Data access logic
- Application logic
- Presentation logic

**Hardware** (basic components)
- Client computers
- Servers
- Network

The goal of architecture design is to decide on the **assignment of software components to hardware.**
Non-functional requirements play an important role in the architecture design:

- Operational requirements
- Operating systems, other information systems.
- Performance requirements
- Response time, processing/storage capacity, reliability
- Security Requirements
- Accessibility to data, encryption, authentication, viruses
- Cultural/political requirements
- Specific to location, multilingual, norms, legal requirements, disability

Program Design

Coding without prior planning and design often leads to inefficient programs, non-reusable code and systems that are not functional.

Traditional top-down modular approach:

- **Structure chart** – high-level diagram showing organisation and interactions of the various pieces of code. For each component, the structure chart shows sequence (order for calling the modules), selection (conditions for calling a module) and iteration (how often a component is repeated).
- **Program specifications** – communicate the basic logic and structure of the pieces of code to help reduce logical and syntactical errors during the implementation phase.

Guidelines for good program design:

- **Build modules with high cohesion** – cohesion refers to how well the lines of code relate to each other: module performing only one task for high cohesion
- **Build modules that are loosely coupled** – coupling refers to how closely modules are interrelated: design interrelation between modules for independency to avoid changes affecting multiple modules
- **Create high fan-in** – refers to the number of modules that call or use the given module: high fan-in indicates module is reused in many places indicating is well written generic code
- **Avoid high fan-out** – refers to the number of subordinate modules that are associated to a single control module: the less subordinate modules (around 8), the more effective control
- **Assess the structure chart for quality** – there are defined guidelines and standards based on the above recommendations

Example 18.2 Example of a structure chart with low fan-in and high fan-out (which is undesirable).
Additional Reading

Chapter 8 to 12 of (Dennis et al., 2006)

Chapter 16 of (Kendall and Kendall, 2005)