Introduction to Software Design and Architectural Styles

Design as a verb - Relates to the design process

Design as a noun - The form of the model or plan produced from the design process

Technical debt - A form of risk, and one that needs to be recognised and managed when designing

Designing involves using abstraction to help:

- Model our design solution in terms of the essential aspects
- Separate the logical and physical aspects of a solution
- Make decisions where there are choices

The ultimate criterion for any decision should be fitness for purpose, in that a design should be well structures, should do its job well, and have no unnecessary features.

1 Knowledge Schemas

Schematic knowledge complements knowledge about the domain and about syntax and semantics, and relates to the way ideas about possible solutions are organised in memory.

Schematic knowledge is typically transferred via:

- Plan driven design methods
- Design patterns
- Software Architectures

2 Some complications

Significant problems that we encounter and must cope with when designing software are that

- It has both static and dynamic characteristics
- We need to use static forms of description to model these characteristics
- Software is invisible, and so we have no intuitive ways of visualising its form or describing it diagrammatically

3 The outcomes of designing

The outcome of the design process is usually in the form of a model or plan which can be used to guide the task of implementation, and that describes

- The characteristics of the set of elements that will make up the ”solution”
- Structure (how these elements are related to one another)
- How the elements will interact when the system is executed

The form of the model reflects the type of design elements we employ and is influenced by the choice of architectural style

Plan driven approaches create much of the model before implementation begins, whereas for agile forms, the details of the model evolve dynamically.

4 Architectural Style

Architectural style - Concerned with a high-level concept of widely used structures and relationships.
Architectural form - Concerned with the high-level design of a particular system

The concept helps analysis and understanding, but as with so many aspects of software, this has proved a difficult one to describe in a systematic manner

4.1 What does software architecture do for us?

- Assists understanding, by giving a vocabulary that descries a system's high-level design
- Enables reuse of software, by identifying where we might "match" different system elements
- Aids system evolution, by exposing the dimensions along which a system is expected to evolve. The maintainers can use this as a guide as to how the system can best be executed
- Provides a framework for analysis - allowing us to assess how well the solution will meet user needs

4.2 Classifying architectural style

A simple and useful classification scheme is one based on the forms of the following three elements:

- Components: the basic building blocks used in a system built in this style
- Connectors: the way the components interact
- Context: how the elements and their interactions are organised and managed at run-time

The context can also be characterised in terms of other features of a system, sich as:

- Topology of information and control flow
- Synchronisation mechanisms
- Binding time

4.3 Evolution of architecture

- Early architectural styles tended to mirror the way that a computer operated, with a strong emphasis upon function
- Later styles show more balance between function and data
- But all are still used

4.4 Major architectural styles

Form	Dominated By	Examples
Data Flow	Motion of data, with no upstream content control by recipient	Batch sequential, pipe-and-filter
Call-and-return	Order of computation, with a single thread of control	Main program and subprograms, abstract data types
Interacting-processes	Communication among independent, concurrent processes	Communicating processes, distributed objects
Data-centered repository	Complex central data store	Transactional databases, client-server, blackboard
Data-Sharing	Direct sharing of data among components	Hypertext, lightweight processes

4.5 Data Flow examples

4.5.1 Batch sequential systems

No user interaction, the program reads data from files and outputs to new files:

- Components are the program itself any files it users
- Connectors are the run-time links to files

4.5.2 Pipe-and-Filter

A chain of processes, which each filter the input data in some way, but which also have clearly independent and reusable roles

- Components are the filter processes
- Connectors are the mechanism linking the standard output from one filter to the standard inpit of another

Unix processes and pipes are a classical example of this form