Review of Basic Software Design Concepts

Fethi Rabhi
SENG 2021
Topics

• The development process
  – Planning
  – Designing
  – Implementing
1. The development process

• How to organise activities related to the creation, delivery and maintenance of software

• There are some fairly common processes
  – Traditional methods: Waterfall
  – Agile methods: Scrum

• 3 major steps: A. Requirements analysis
  B. Design
  C. Build and Deploy

Our focus is here
A. Requirements analysis and specification

• Can involve
  – Formal specifications
  – Use cases (UML use cases)
  – Epics and Stories (Scrum)

• Which one(s) to use
  – Can be a mixture
  – Depends on type of system
B. Design

• Next phase after requirements analysis
• First step in building the system
• Design is a process
• Design operates according to a specific methodology (e.g. OO)
• The design can be represented using a notation (e.g. UML)
• Methodologies/notations are usually supported by tools
Requirements analysis

Design

Implementation

Software architecture

Detailed design

Testing

Maintenance
Software architectures

• Equivalent to design at the highest level
• Essential for large applications: defines “parts” of the system and how these parts are assembled
• Architecture satisfies design goals e.g.:
  – Extensibility (ability to add new features)
  – Adaptability (accommodating changing reqs.)
  – Simplicity (ease of understanding/implementing)
  – Efficiency (time/space)
Decomposition criteria

• Decomposition into modules/components/packages etc. is of critical importance in the design activity
  – Cohesion is the degree to which communication takes place among the module’s elements
  – Coupling describes the degree to which modules communicate with each other.

• Low coupling/high cohesion is essential for managing changes
Architectural patterns

• Used for designing software architectures, as a first step in the design activity

• An architectural pattern expresses a fundamental structural organization schema for software systems

• Provides:
  – set of predefined components (subsystems)
  – a specification of their relationships
  – rules and guidelines for organizing the relationships between them
Common architectural patterns

• Façade
• Layers
• Observer
• Adapter
Façade pattern

- Useful for modelling *client-server* architectures
Characteristics of the pattern

- Low coupling between the two participating components
- Secure. Easy to maintain.
Layers pattern

- Useful for large systems requiring decomposition
Characteristics of the pattern

- Decomposition based on levels of abstraction
- Easier to manage and maintain than monolithic code
- Allows the construction of complex components out of simpler ones
- Examples: OSI layers in networking, CORBA, multi-tier architectures
Example: 3-tier software architecture
Observer pattern

- Useful for modelling client updates
Characteristics of the pattern

- Allows observers to be added/removed without disrupting other observers
- Potential to reduce network traffic
Adapter pattern

• Useful for the integration of legacy code
Detailed Design

• Not one but many!
  – Successions of refinements from the overall architecture
  – Involves modelling of certain elements to make implementation easier

• Design coverage
  – Function
  – Structure
  – Behaviour
  – Information
Detailed Design

• Detailed design is usually closely associated with the implementation language used

• If using an OO language (e.g. Java)
  – Some design elements can be expressed in the Unified Modelling Language (UML)
  – UML is an OMG standard
  – UML contains lots of notations

• If using a database
  – Information structure can be modelled using a conceptual modelling notation
UML Notations

Use case view

Design view
- Class/Object diagrams

Process view
- Sequence/Statechart/Activity diagrams

Implementation

Deployment
- Deployment diagrams

Software Architecture

UML
- Use-case diagrams
- Component/Package diagrams
Typical OO design

• Expand use cases (real use cases)
• Develop collaboration or sequence or statechart diagrams (just use 1 of them!)
• Defining classes (in parallel)
• Each class needs *responsibilities* assigned to it
Data modelling

• Identify domain objects, their attributes and associations between objects
• Normally, combination ER-diagram/sequence diagrams
• Relational models (for SQL databases)
• Formal methods: more powerful and unambiguous
User Interface and visualisation design

• Helping users to interact with systems
  – Intuitive interfaces
  – Diversity in user devices
  – Easy to learn

• Visualisation
  – Viewing complex data
  – Infographics
C. Build and Deploy phase

- Start thinking about implementation
- Identify languages/platforms to be used
- “Packaging” of classes into programs.
- Allocating packages to platforms (deployment)
There are two ways of constructing a software design: one way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.

C.A.R. Hoare
Focus of this workshop

- System Scope and Specification
- Interface Design
- Software Architecture
- Software Design
- Implementation Considerations
What needs to be in the design

• Software architecture
  – Diagram showing components

• Software Design
  – Data Model (e.g. ER Diagram)
  – Process View (e.g. sequence diagrams)
Example of process view

Create Invoice

Create Invoice

PriceList
despAdv
Invoice

Invoice: createInvoice(despAdv, PriceList)

Invoice: produces
despAdv: uses
PriceList: uses

User story

INITIAL
SEQUENCE
DIAGRAM

Create Invoice

Invoice

createInvoice(DA, PriceList)

Return(i)
i: Invoice

DETAIL SEQUENCE DIAGRAM

Architectural
component
Design tools

• Functions
  – Modelling design artefacts
  – Managing/sharing models

• Tools
  – For architecture, people tend to use a drawing tool
  – For detailed design, several UML tools exist (e.g. https://www.draw.io/)
  – Benefits: consistency checks, automatic code generation
  – Disadvantages: steep learning curve
Conclusions

• Design is next activity after requirement analysis
• Divided into 2 stages: architectural design and detailed design
  – Architectural design facilitated by the use of design patterns
  – Detailed design is an iterative activity: check requirements satisfaction, think about implementability
Further reading

- Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.