L11 - from requirements to design

CS3028 - Principles of Software Engineering

Ernesto Compatangelo

Department of Computing Science

UNIVERSITY OF ABERDEEN

11.1 Reminding past issues and mapping them to current topics

Where are we now?

Software development paradigms

⇒ The Unified Process (UP) paradigm

⇒ UP phases and UP disciplines (activities) within each phase

⇒ Inception (first UP phase)

⇒ Elaboration (second UP phase)

⇒ Elaboration requirements

⇒ From requirements to design

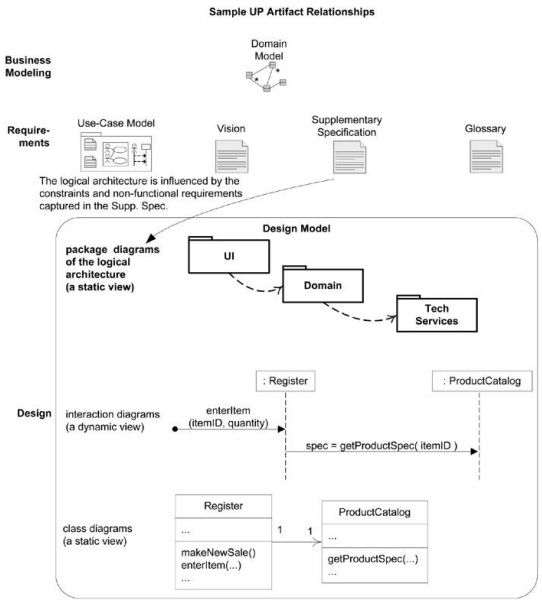
⇒ . . . . . .

11.2 Design objectives and artifacts

Design objectives

- The **analysis (problem) model** describes the system from the actors' viewpoint, without any reference to its internal structure
- Design transforms a **problem model** defined by the software requirements into a **solution model** defined by the software structure
- The design model describes the system in terms of its **internal structure**, specifying **how** this should be realised
- Similarly to requirements — where in-width elicitation is followed by in-depth analysis — in-width **system (architectural) design** is followed by in-depth **detailed (structural) design**

From requirements to design



System design artifacts

System design results in the following artifacts:

- A **software architecture**, describing the system decomposition into subsystems and highlighting their
  - responsibilities
  - mutual dependencies
  - mapping to HW
- A set of **policy decisions** associated to the software architecture, such as and control flow, access control, and data storage
- A set of **design goals**, derived from non-functional requirements and describing the system qualities to be optimised during design
- A set of **boundary use cases**, describing system configuration, startup, shutdown, and exception handling issues

11.3 Subsystems: packages, dependencies, services, and interfaces

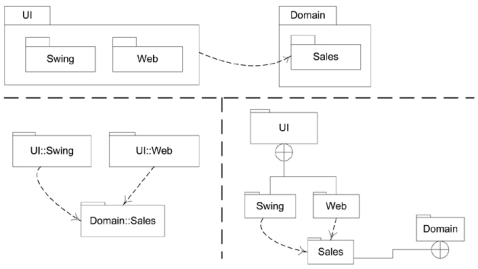
Software architecture: the notion of subsystem

Subsystems (often represented as **architectural packages**)

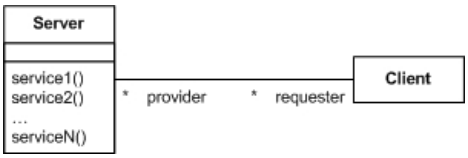
- Group together system elements that share **common properties** (*e.g.*, user interface, data management. . .)
- Result in **development units** made of solution domain classes
- Have clearly specified **boundaries** and fully defined **interfaces** with other sub-systems
- Provide **services** to other sub-systems
- Correspond to the **amount of work** that either a single developer or a single development team can tackle

Subsystems: UML packages and package dependencies

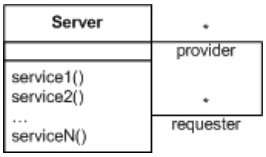
- Subsystem = UML package =
  general container (construct) that groups units together
- Subsystems used to reduce system (model) complexity
- A UML package dependency (A → B, *i.e.*, A *depends on* B)
  indicates that a change in B could cause a change/problems in A
- UML package ≠ { Java | C# | C++ } package in principle
  e.g., it can also include use cases and other documentation



How package dependency works

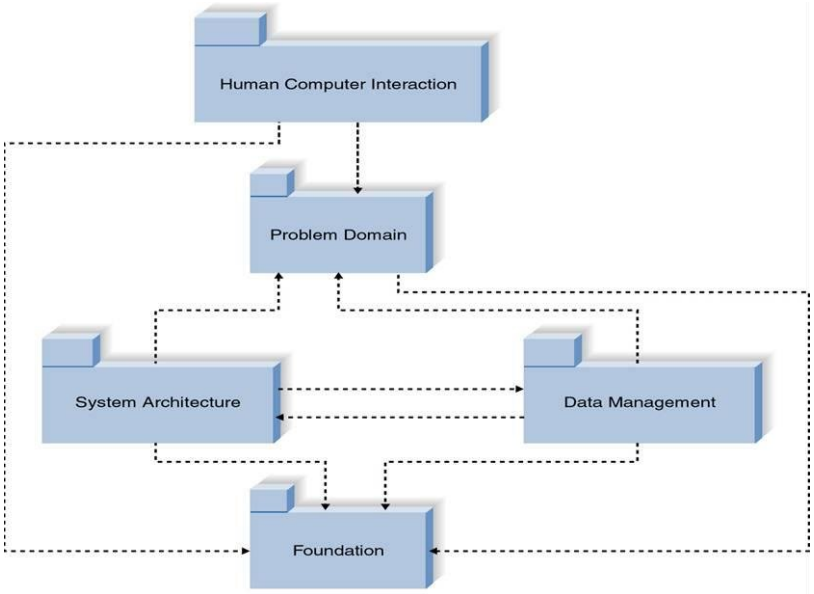


Client→server: client uses server services, hence client package depends on server package (but not vice-versa)



Peer↔to↔peer: each subsystem may request services from the other, hence packages are mutually dependent

Package dependencies: a general example



Subsystems: services and interfaces

- **Service**: set of **related operations** that share a common purpose
- **Subsystem interface**: set of operations (*i.e.*, services) of a subsystem available to other subsystems
- Subsystem interfaces only enumerate operations, their parameters and their high-level behaviour
- Detailed design (*i.e.*, object design) refines and extends subsystem interfaces developing the Application Program Interface (API). This also include the parameters and the return value of each operation (*i.e.*, the operation *signature*)

11.4 Preparing for the topic ahead

Next lecture. . .

Design principles:

More specifically, we will focus on:

- Modularity
- Coupling, cohesion, esthetics
- Architectural design