Electronics and
Computer Science

University of Southampton

Detailed Design

COMP6226: Software Modelling Tools and Techniques for Critical Systems

Dr A. Rezazadeh (Reza)
Email: ra3@ecs.soton.ac.uk or ar4k06@soton.ac.uk

October 24

University of Southampton

Overview

- Design Principles
- Design Principles – SOLID
- Software Modelling
- Types of System Models
- What is detailed design?
- Main tasks in detailed design
- Object-Oriented Design
- UML diagram types

Design Principles

- What are Software Design Principles?
  - Software Design Principles are a set of guidelines that helps developers to make a good system design.
- Why are Software Design Principles important?
  - You can write code without Software Design Principles. That's the truth. But if you **want to become a Senior level** you should **understand and apply Software Design Principles** in your work.
  - We have **many recommended set of principles to apply** Software Design Principles to your project.

Design Principles

- KISS: is an acronym for Keep It Simple, Stupid.
  - The acronym reminds us to avoid unnecessary complexity in our designs.
  - Our design need contain only enough complexity to achieve our requirements, and no more.
- DRY (Do Not Repeat Yourself)
  - We try to avoid repetition in software development.
  - Repetition means multiple- source code fragments performing a similar task.
  - This becomes a challenge when maintenance is needed, since changes must be made in more than one place.
  - The DRY principle applies to all aspects of our development work and includes scripts, tests, databases as well as source code.

Design Principles – Cont.

- YAGNI (You Aren't Gonna Need It)
  - Some software engineers have the habit of predicting future needs of clients and implementing software features in anticipation of those future requirements.
  - This is not a good practice because sometimes we invest effort in preparing for future features that never come.
  - This results in bloated software source code.
  - Instead, only functionality needed now must be implemented to boost your productivity.

University of Southampton

Design Principles – Cont.

- GRASP
  - The General Responsibility Assignment Software Patterns (GRASP) principles, proposed by Craig Larman, provide a mental model to help object-oriented design [*].

    [*] Larman, C.: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd ed. Prentice Hall PTR, Upper Saddle River, NJ (2004)

- The GRASP pattern comprises:
  - Controller – Creator – Indirection – Information expert
  - Low coupling – High cohesion – Polymorphism – Protected variations
  - Pure fabrication

Design Principles – SOLID

- The SOLID acronym was introduced around 2004 by Michael Feathers, to help you remember good principles of object-oriented design [*].

  [*] *Martin, R.: Clean Code: A Handbook of Agile Software Craftsmanship, 1st ed. Prentice Hall, Upper Saddle River, NJ (Aug 2008)*

- The SOLID principles have some overlap with Larman's GRASP patterns.
- The SOLID acronym is derived from:
  - Single responsibility
  - Open-closed
  - Liskov substitution
  - Interface segregation
  - Dependency inversion

University of Southampton

SOLID Design Principles – Cont.

- Single responsibility: every class should have only one responsibility
  - Consequently, it should only have one reason to change.
  - Less functionality in a single class will have fewer dependencies and this means lower coupling.
- Open-closed: Objects or entities should be open for extension but closed for modification.
  - In doing so, we stop ourselves from modifying existing code and causing potential new bugs in an otherwise happy application.

SOLID Design Principles – Cont.

- Liskov substitution: Let q(x) be a property provable about objects of x of type T. Then q(y) should be provable for objects y of type S where S is a subtype of T.
  - If class A is a subtype of class B, we should be able to replace B with A without disrupting the behaviour of our program.
- Interface segregation: A client should never be forced to implement an interface that it doesn't use, or clients shouldn't be forced to depend on methods they do not use.
  - Larger interfaces should be split into smaller ones.
  - By doing so, we can ensure that implementing classes only need to be concerned about the methods that are of interest to them.

SOLID Design Principles – Cont.

- Dependency inversion: Entities must depend on abstractions, not on concretions. It states that the high-level module must not depend on the low-level module, but they should depend on abstractions.
  - The principle of dependency inversion refers to the decoupling of software modules.
  - This way, instead of high-level modules depending on low-level modules, both will depend on abstractions.

Reference:

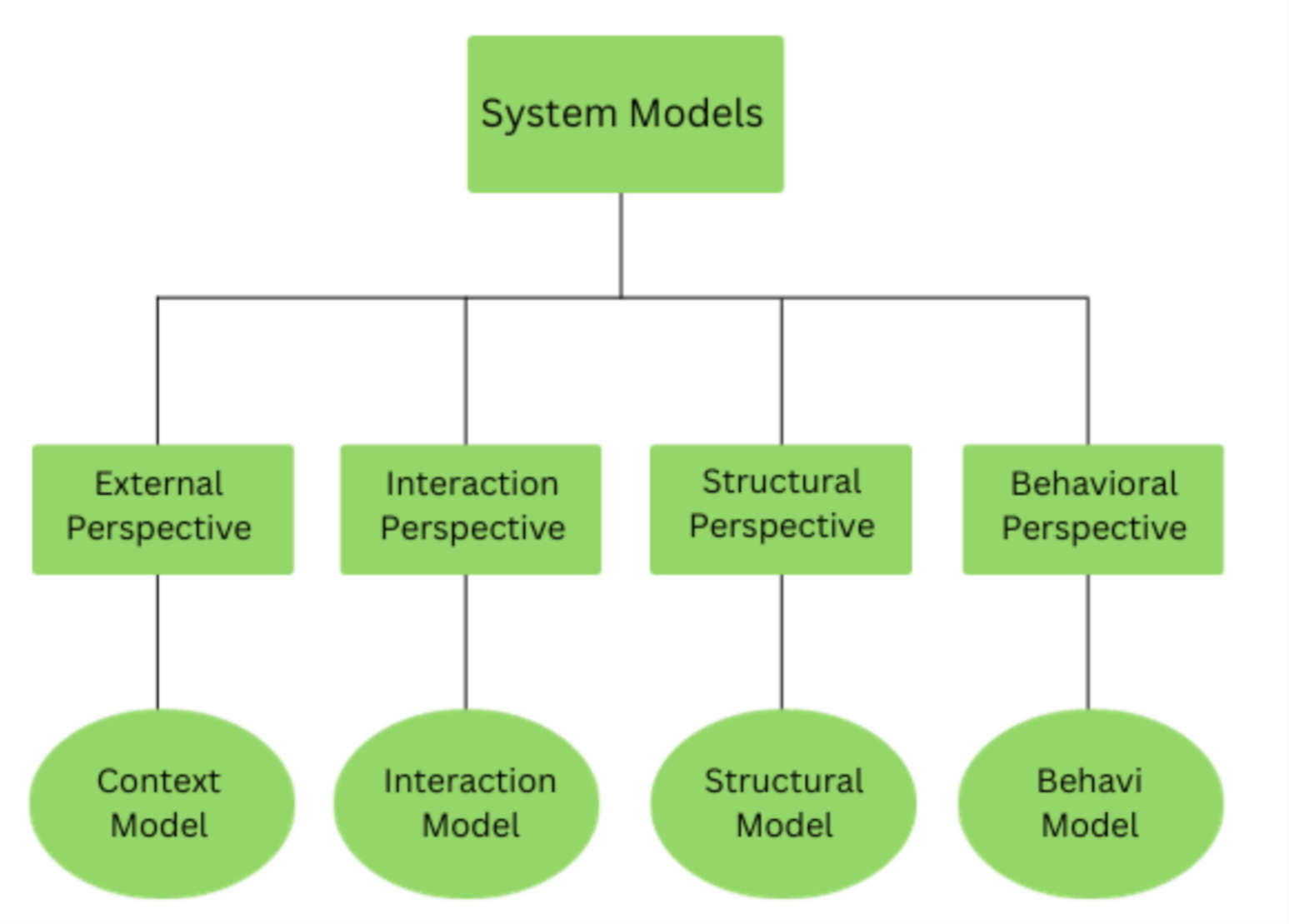
A Solid Guide to SOLID Principles

Software Modelling

- For software modelling, we use models that are based on some kind of *graphical* or *textual* notation.
- The *Unified Modelling Language* (*UML*) is a commonly used graphical representation.
- The *two main types* of model: *structural* and *Behavioural*.
  - *Structural modelling* is used to illustrate a software application's physical or logical model from the perspective of its composition, architecture, componentization, and/or organization.
  - *Behavioural modelling* is a model type that focuses on identifying and defining the dynamic behavioural aspects of software components.
  - The goal is to represent how software functions, features, and system elements behave when in operation.

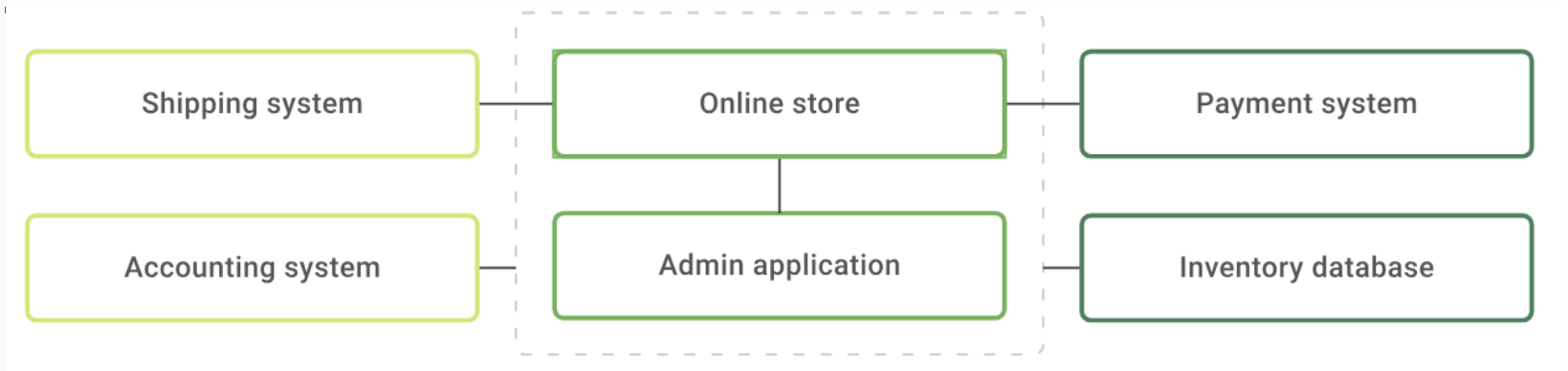
University of Southampton

Types of System Models



Four views of the system

- External perspective
  - An external perspective, where you model environment the context or of the system



- Interaction perspective
  - An interaction perspective, where you model the interactions between a system and its environment between the components of a system , or between the components of a system.

Four views of the system

- Structural perspective
  - A structural perspective, where you model organization of a system the or the structure of the data is processed by the system.
- Behavioural perspective
  - A behavioural perspective, where you model that the dynamic behaviour of the system and how it responds to events.

University of Southampton

A Reminder from previous lectures

- What constitutes a good model?
  - A model should
    - use a standard notation
    - be understandable by clients and users
    - Help software engineers to gain insights about the system
    - provide abstraction, modularisation, ..
  - Models are used:
    - to help communicate with stakeholders.
    - to permit analysis and review of those designs.
    - as the core documentation describing the system.
    - to generate code

University of Southampton

What is detailed design?

- The process of *refining* and *expanding* the *software architecture* of a system or a component to the extent that the design is *sufficiently complete* to be implemented.
- During *Detailed Design* designers go deep into each component to define its internal *structure* and *behavioral* capabilities.
  - the resulting design leads to efficient construction of software.
- *Architecture is design, but not all design is architecture.*
  - Detailed design is *closely related* to *architecture*;
  - therefore, designers are required to have or acquire a full understanding of the *system's requirements* and *architecture*.

University of Southampton

Main tasks in detailed design

- The major tasks identified for carrying out the detailed design activity include:
  - Understanding the architecture and requirements
  - Creating detailed designs
  - Evaluating detailed designs
  - Documenting software design
  - Monitoring and controlling implementation
- It can be especially tough for large-scale systems, built from scratch without experience with the development of similar systems.

Object-Oriented Design

A discipline that utilises the object-oriented paradigm to achieve the aims of software engineering

A discipline its aims are:

- To provide an effective approach to cope with ever-increasing *complexity* of systems
- The production of a relatively fault-free software,
- Delivered on time and within budget,
- That satisfies the client's needs
- Furthermore, the software must be easy to modify when it needs to change

University of Southampton

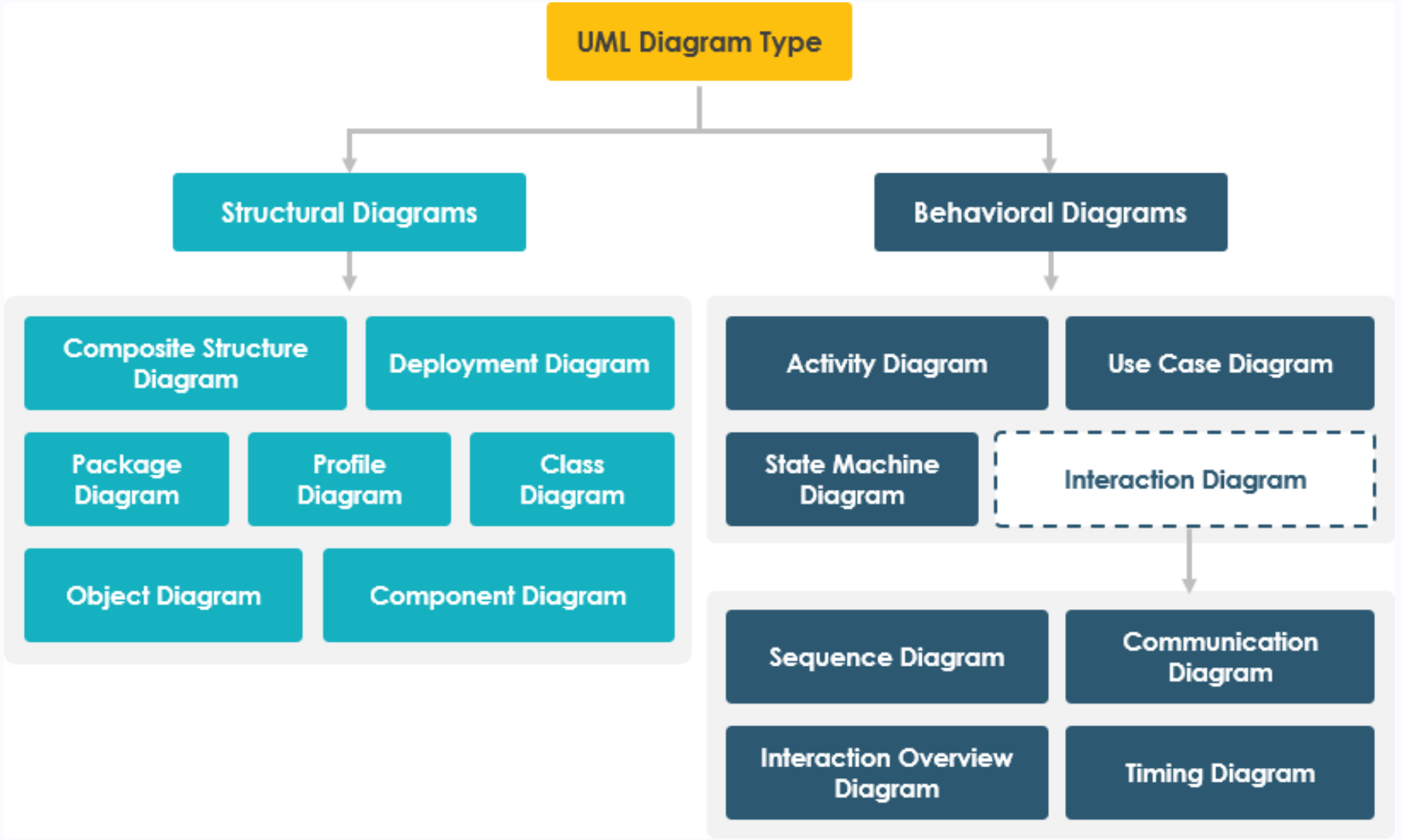
Object-Oriented Design – Various approaches

- In heavyweight software development processes, the entire Design is completed before coding/implementation begins.

- In lightweight software development processes, an outline design is made before coding, but the details are completed as part of the coding process.

University of Southampton

UML diagram types



UML diagrams – Cont.

Models used mainly for requirements

- Use case diagram shows a set of use cases and actors and their relationships.
- Activity diagram (flowchart) shows the flow from one activity to another activity within a system.

Models used mainly for systems architecture

- Component diagram shows the organisation and dependencies among a set of components.
- Deployment diagram shows the configuration of processing nodes and the components that live on them.

University of Southampton

UML diagrams – Cont.

Models used mainly for detailed design

- Class diagram: shows a set of classes, interfaces, and collaborations with their relationships.
- Sequence diagrams: time ordering of messages
- State diagrams and activity diagrams also are widely used.

University of Southampton

UML Models - Interactive Aspects of Systems

- These models can be used for requirements analysis or detailed design.
  - Sequence diagrams: time ordering of messages
  - activity diagrams shows the flow from one activity to another activity within a system.

University of Southampton

Resources

- The Unified Modeling Language

  https://www.uml-diagrams.org/

- Software Engineering, 10th edition, Ian Sommerville, Chap. 7
- Software Engineering Design: Theory and Practice , Carlos E. Otero Chap. 5
- Software Engineering: Principles and Practice, Hans van Vliet Chap. 12

University of Southampton

YOUR QUESTIONS