


SWINBURNE
* *
UNIVERSITY OF
TECHNOLOGY


SWE30003
Software Architectures and Design

Lecture 12
Summary of Main Topics,
Final Assessment Info

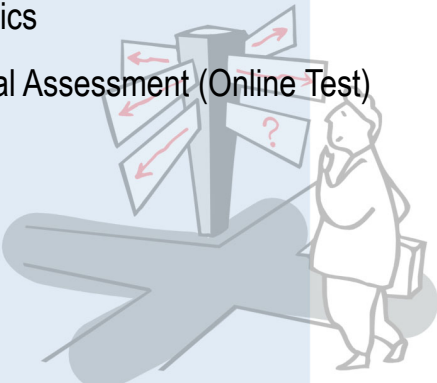


1

Outline



- Summary of Main Topics
- Information about Final Assessment (Online Test)



2

2

Major Topics covered this semester



- *Requirements Analysis and Specification*

-
-
-

- *Object Oriented Design*

-
-
-

- *Software Architecture Design*

-
-
-
-

3

3




Communication of information is
the key concept behind a
requirements specification!

In fact, this applies to any document produced in the SDLC!!

4

4

Who is the intended Audience?



The Goal Design Scale for software requirements - The intended audience will determine which of the following levels is/are most appropriate:

- Goal
- Domain
- Product
- Design


Questions:

- What do you do if the audience is mixed?
- What is the purpose of the SRS?

5

5

The Goal-Design Scale (cont.)



(Ship repair quotation system)

R1. Our pre-calculations shall hit within 5%

R2. Product *shall support* cost recording and quotation with experience data

R3. Product shall have recording and retrieval functions for experience data

R4. System shall have screen pictures as shown in app. xx

Goal-level requirement

Domain-level requirement

Product-level requirement

Design-level requirement

Which requirement to choose?

If the supplier is

- A vendor of business applications?
- A software house - programmers?
- PriceWaterhouseCoopers?

6

6

Types of Requirements

Quality reqs:
Performance
Usability
Maintainability
...

Other deliverables:
Documentation
Install, convert, train ...

Managerial reqs:
Delivery time
Legal
Development process ...

Helping the reader:
Business goals
Definitions
Diagrams ... 7

Data requirements:
System state: Comm. states,
Input/output formats, Persistent Data

Functional requirements, each interface:
Record, compute, transform, transmit
Theory: $F(\text{input, state}) \rightarrow (\text{output, state})$
Function list, pseudo code, activity diagrams
Screen prototypes, user tasks

7

Capture Functional Requirements: Tasks & Support

Task: 1.2 Checkin Purpose: Give guest a room. Mark it . . . Frequency: . . .	
Sub-tasks:	Example solution:
1. Find room. Problem: Guest wants neighbour rooms; price bargain.	System shows free rooms on floor maps. System shows bargain prices, time and day dependent.
2. Record guest as checked in.	(Standard data entry)
3. Deliver key. Problem: Guest forgets to return the key; guest wants two keys.	System prints electronic keys. New key for each customer.
Variants:	
1a. Guest has booked in advance. Problem: Guest identification fuzzy.	System uses closest match algorithm.

Past:
Problems

Domain
level

Future:
Computer part

From: Soren Lauesen: Software Requirements © Pearson / Addison-Wesley 2002

8

Identify Quality Attributes?



- Identify the top 3 to 5 quality attributes/factors of the domain
 - ☞ Assume that qualities of similar applications will also be of relevance for the application under consideration
- Systematically go through all identified user task
 - ☐ Identify these tasks that make explicit mention of the identified quality attributes
 - ☐ Note: “critical” of user tasks generally imply qualities!
- Spell out the resulting non-functional requirements in a **verifiable** form.
 - ☞ *Maybe not be 100% perfect, but a good start...*

9

9

Function vs. Quality



- Many requirements have **both functional and quality** aspects.
- Example: “*Halt garage door within 0.5 seconds when an obstacle is detected.*”
 - ☐ Function: detect obstacle, halt garage door
 - ☐ Quality: within time limit (0.5 seconds)

10

10

Quality Criteria for Requirements



- *Correct*: each requirement reflects a need
- *Complete*: all necessary requirements are included
- *Verifiable*: possible to see whether is/can be met
- *Ranked for importance/stability*: priorities
- *Understandable* by customer and developers

11

11

General View of Abstraction



Real World

- ☞ Phenomenon with messy details
- ☞ Facts and laws governing the real world

Abstract World

- ☞ Model with “simple” properties
- ☞ Representation subject to “formal” manipulation



Abstraction: *mapping* from *real world* to *abstract world*

12

12

Software Abstractions



- Abstraction is the most important concept in computer science (and probably also the oldest):
 - any kind of software generally consists of several *layers* of abstraction.
- A software abstraction has two levels:
 - *Specification* (i.e. what it offers to clients)
 - *Realization* (i.e. the lower-level building blocks used to support the specification)

13

13

Software Abstractions (cont.)



- *Software Architectures*
 - Subsystems
 - Components
 - Modules, Packages
 - *Objects*, Interfaces
 - Abstract Data Types
 - Functional abstractions (i.e. procedures)
 - Primitive data types
 - Simple arithmetic expressions and control structures
 - Macros
 - Symbolic names (for instructions and memory cells)
 - Machine instructions, memory cells
- “high-level”
- “mid-level”
- “low-level”

14

14

Responsibility Driven OO Design



■ Finding Classes

- Class Selection Rationale

■ (CRC Cards)

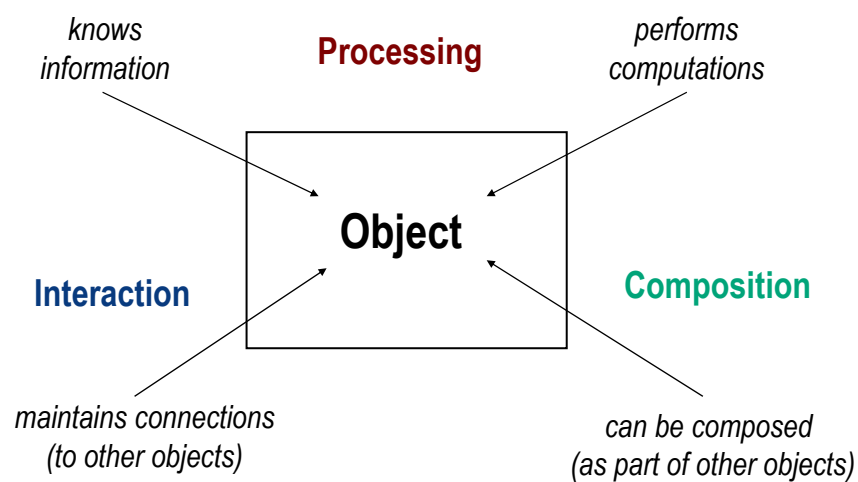
■ Identifying Responsibilities

■ Finding Collaborations

15

15

What is an Object?



16

16


Class Responsibility Collaboration Cards

Use CRC cards to record candidate classes:

Text Creation Tool	subclass of Tool
Editing Text	

Record the candidate *Class Name* and *superclass* (not yet known)
Record each *Responsibility* and the *Collaborating classes*

- ☐ compact, easy to manipulate, easy to modify or discard!
- ☐ easy to arrange, reorganize
- ☐ easy to retrieve discarded classes




17

17

Object-Oriented Applications

An *application* = a set of interacting *objects*
An *object* = an implementation of one or more *roles*
A *role* = a set of related *responsibilities*
A *responsibility* = an obligation to perform a task or know certain information
A *collaboration* = an *interaction* of objects or roles



18

18

Refinement of OO Design



- Classes and class hierarchies
- Other class relationships
- Roles and protocols
- Association and Aggregation

19

19

Abstract State vs. Concrete State



- Every object (bar a few rare exceptions) contains statefull information (*an object has state*).
- Externally visible properties of this state defines its *abstract state*
 - ☞ Stack: size, order of elements, top element
- Internal representations define the *concrete state*
 - ☞ Stack may use a list or array to “store” its elements
- Clients should only ever rely on an object’s abstract state, not its concrete state.
- Object equality should only ever be defined over the abstract, but never on the concrete state.

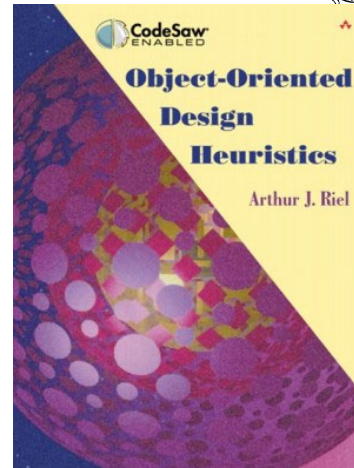
20

20

Object-Oriented Design Heuristics



- Seminal work, describing app. 60 heuristics to improve object-oriented design and implementation
- A selected few of the heuristics will be introduced in this lecture.
... abstractions, transformations, etc



21

21

What are (Design) Patterns?



*“Patterns describes a problem which **occurs over and over again** in our environment, and then describes the **core of the solution** to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.”*

— Alexander et al., *A Pattern Language*, 1977

22

22

Software Architecture



The architecture of a software system is described as:

- the *structure(s) of its high-level processing elements*,
- the *externally visible properties* of the processing elements, and
- the *relationships and constraints* between them.

in other words:

The set of *design decisions* about any system (or subsystem) that keeps its implementors and maintainers from exercising "*needless creativity*".

23

23

Architectural Styles



*"An **architectural style** defines a **family of software systems** in terms of their structural organization. An architectural style expresses components and the relationships between them, with the constraints of their application, and the associated composition and design rules for their construction."*

— Dewayne Perry and Alexander Wolf, 1992

24

24

Popular Architectural Styles



- **Data-flow architectures:**
 - Batch sequential, *Pipes-and-Filter*
- **Call-and-Return architectures:**
 - Main program and subroutine, Object-oriented
 - *Layered*
 - *Client-Server*
- **Data-centred architectures:**
 - *Repository*, Blackboard
- **Independent component architectures:**
 - *Peer to Peer*
 - Communicating processes
 - Event systems: implicit invocation, explicit invocation
- **Virtual machine architectures:**
 - Interpreter, rule-based systems
- **Service oriented architecture:**
 - Software services, WS*(SOAP) Web services, REST services, Microservices

25

25



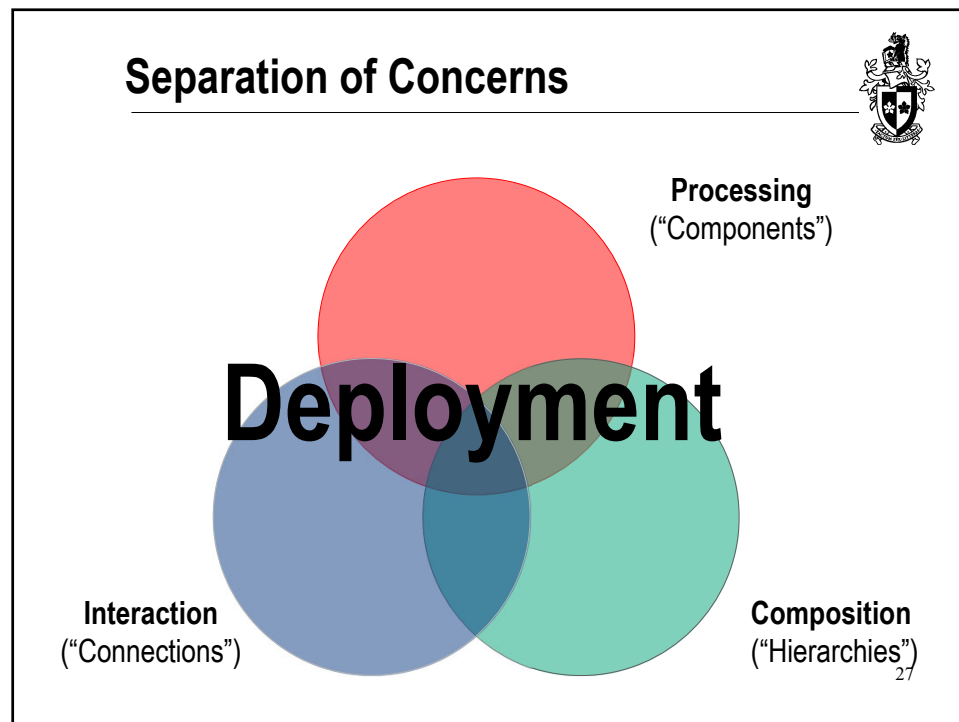
s/Object/Processing Element/g

(Almost) everything that applies to objects can be
used at higher levels of abstraction!

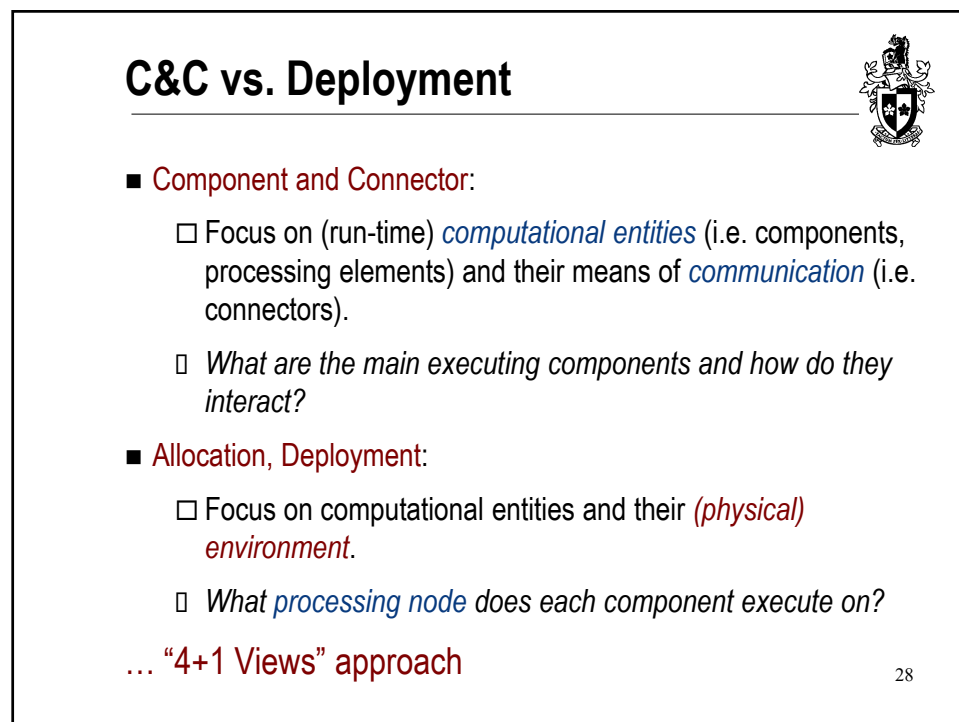
[Modules, Components, Sub-Systems ...]

26

26



27



28



“Notation used?”

29

29



Important to consider

- Various *design factors* determine how to approach the design of a large-scale software system,
- **Abstraction** is the most important concept in software,
- Quality attributes, not functional requirements, determine the main design decisions.
- Patterns and Architectural tactics assist in addressing quality attributes in designs!
- Tactics require a certain level of experience to use them effectively,
- Up to system designer to decide when and how to use what kind of tactics and/or patterns.

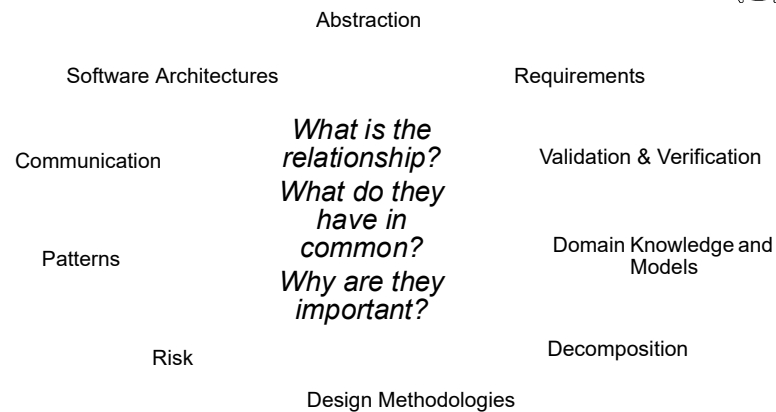
☞ *Software development is “always” reuse driven!*

☞ *Pattern-based reuse vs code reuse!*

30

30

To think about...



31

31

“Vocabulary” for System Design



Software architectures, patterns, tactics, software components, etc.:


- are all “**ingredients**” for (large-scale) system design,
- define a *vocabulary* to enhance communication amongst the different stakeholders of a system,
- increase level of *abstraction*, enable system design at various levels of abstraction,
- enable *encapsulation* of system concerns into self-contained, reusable entities.

32

32



thinking



... but think the problem through!!

33

33



MAGIC

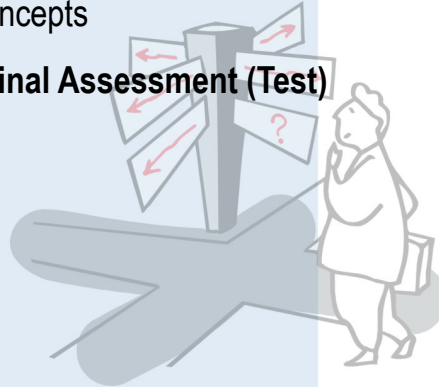
34

34

Outline



- Summary of Main Concepts
- Information about Final Assessment (Test)



35

35

Information about Test



- Online on Canvas
- Date: **Saturday, 7 June 2025**
 - **2.5 hours from 2pm (AEST).**
- Worth **20%** of your final mark – **with hurdle of 40%**
- 10~15 questions
- Open book, but **your own work/words** (ie, no consultation, no chapGPT/tools, no discussion with others)
- More specific info will be announced in week before test ...
- Topic: **all** material covered during **lectures, readings** (pre-/recommended), **tutorials, weeklies**, and three **assignments**
- No really “tricky” or difficult questions, but relying on “understanding/application”!

36

36

Information about Test (cont.)



- Main focus: *being able to apply knowledge in a problem situation.*
- Coverage of questions:
 - ☐ About software requirements
 - ☐ About OO design
 - ☐ About architecture design (more)
- Types of questions:
 - ☐ Extended-answer questions (with examples)
 - ☐ Case studies/applications – small

37

37

Information about Test (cont.)



- Best preparation:
 - ☐ having *seriously* done all assignments & all parts,
 - ☐ done all required readings,
 - ☐ practiced writing answers each week,
 - ☐ *actively* participated in tutorials.
- (Note that marks given for weekly answers do not represent full appreciation of the questions ...)*
- ☞ *Do not memorize answers to “sample” questions!!!
– no “standard” answers (not provided)*

38

38

Consultation & discussion before Test



****** The following consultation sessions (via Collaborate on Canvas) are indicative, and subject to change:

- ☐ 2:30pm-3:30pm, Monday 2 Jun 2025;
- ☐ 2:30pm-3:30pm, Wednesday, 4 June 2025;
- ☐ 2:30pm-3:30pm, Friday 6 June 2025;
- ☐ Or use Canvas discussion forum.

In any case, please send me an email (before the session) when you intend to come to a consultation session – indicating approx. time!

39

39

Sample Exam Questions: Requirements (I)



- Software requirements can be described at four levels. Name these four levels and give an example requirement for each.
- Using six sentences, summarize the importance/relevance of a software requirements specification (SRS) in the software development lifecycle (SDLC).
- In your own words, explain the difference between functional and non-functional requirements and give a sample requirement for each.
- In your own words, characterize the concept of software quality and give two examples why specifying software quality generally is a non-trivial task.

40

40

Sample Exam Questions: Requirements (II)



- What are the main differences between a *reliable* and a *robust* software system?
- List the *major topics* which you would expect to find covered in a good requirements specification.
- “The system shall exhibit acceptable performance.” This is an example of a common non-functional requirement. Why is this not an acceptable requirement, and how can it be modified to make it better?
- Why is it important to make the distinction between the *inner domain* and the *outer domain*? When/where in the Software Development Lifecycle would you need this information?

41

41

Sample Exam Questions: Design (I)



- The principle of information hiding is generally regarded as a good design principle and has been successfully applied in many software systems. However, there are design situations where hiding information is not a good choice. Under which circumstances is information hiding appropriate and under which might it not be recommended?
- Does the data-flow architectural style provide support for portability? If so, how? If not, which architectural styles do and why?
- What role does the concept of Software Architecture play in increasing the quality of a system's design and accommodate non-functional requirements?

42

42

Sample Exam Questions: Design (II)



- A project leader asks you to come up with reference implementations for a given set of design patterns. From a technological point of view, what is the appropriate answer to this demand? Explain!
- Explain how a good object-based decomposition of a given problem results in weak coupling and strong cohesion? When will an object-based decomposition result in strong coupling?
- What are the advantages and disadvantages of a “data-centric architectural style”? For what kinds of projects/problem scenarios is it appropriate to use this style?

43

43

Sample Exam Questions: Case Study



- Assignments 1, 2 & 3.
- Lectures, tutorials, and readings – **in particular**, the part concerning **software architectures** (limited coverage by assignment 3)

44

44

Sample Exam Questions: Case Study (II)



Given a particular business context/scenarios,

- Identify, describe and explain a task (Task & Support),
- Identify, state and explain some quality requirements
- Identify, describe and explain some objects/classes (CRC),
- Identify, describe and justify a suitable architectural style(s) for a high-level design solution (C&C view, physical deployment view).
- Identify, describe and explain some objects/classes (CRC), in a detailed OO design solution.

45

45

More Sample Questions ...



See

- weekly questions,
- tutorial questions,
- Assignments

Any Questions?

46

46



**Best wishes for
the Test!!**