SWEN30006
Software Design and Modelling

Review &
Exam Information

"Striving for success without hard work is like trying to harvest where you haven't planted."

—David Bly

Disclaimer

- Today lecture will summarise the key topics taught in this subject
- This slide may ***NOT*** cover all the important details related to the key topics
- You should NOT solely use this review for a review for your exam

To prepare for this exam, it is highly recommended to review:

- Lecture slides
- Workshop supplementary materials
- Workshop exercises
- Textbook: Applying UML and Patterns (Larman 3rd Ed.); See Textbook Sections Covered

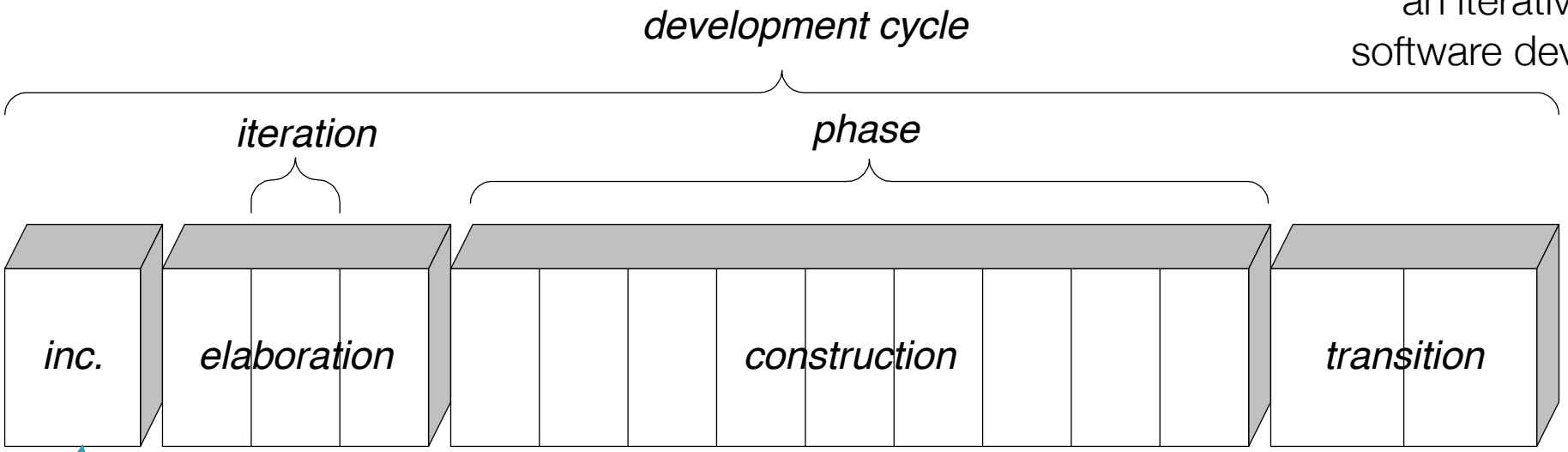
Whilst this exam is open book, you will not have enough time to complete the exam if you research all the questions during the exam time. We recommend you study for this exam like you would any other and aim for a good understanding of the key areas of focus, so you can focus on crafting the best answer you can during the exam time.

Subject Aims & Objective

- The **aim** of the subject is to teach you about ‘Software Modelling’ and ‘Software Design’.
- Software Design is all about *purposefully* choosing the **behaviour** and **structure** and of your software system.
  - System behaviour: how your systems responds to inputs and events
  - System structure: how parts of the system collaborate to achieve the goals of the system
- Software Modelling is the creation of tangible, but abstract, representations of a system so that you can communicate your design ideas, critique them and explore viable alternatives

An Overview

Unified Process:
an iterative & incremental
software development process



Inception Phase:
- Analysis of ~10% of use cases
- Go or no Go decision

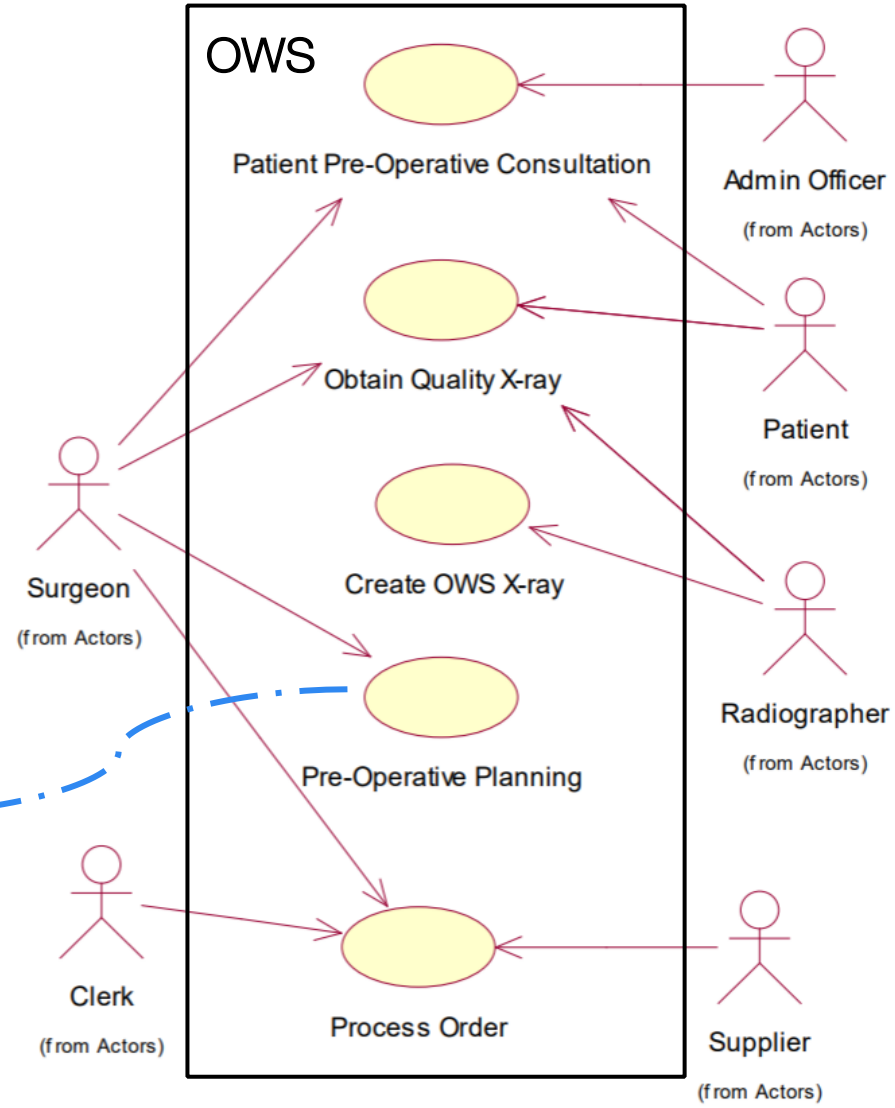
Outcome:
- Common Vision
- Use case model (Use case text + Use case diagram)

Use Case Model

Definition: Use Case is a list of actions or event step that define how the user (or some actors) interact with the system to achieve the goal

- **Use case text** is a written description (a story) of some actors using a system
- **Use case diagram** shows the relationships between actors and use cases in the system

Use Case: Pre-Operative Planning (brief)
The surgeon inspects and annotates an OWS x-ray; judgement is made regarding the femur neck resection lines and location of replacement components. Decisions are also made by the surgeon regarding type and size of replacement components, and the accessories needed for the operation. An operation plan is prepared to reflect decisions made during Pre-Operative Planning. Orders for the components and accessories are generated. A Fad may also be prepared and some details of the Pre-Operative Plan are added to Patient Details.



An Example of use case diagram

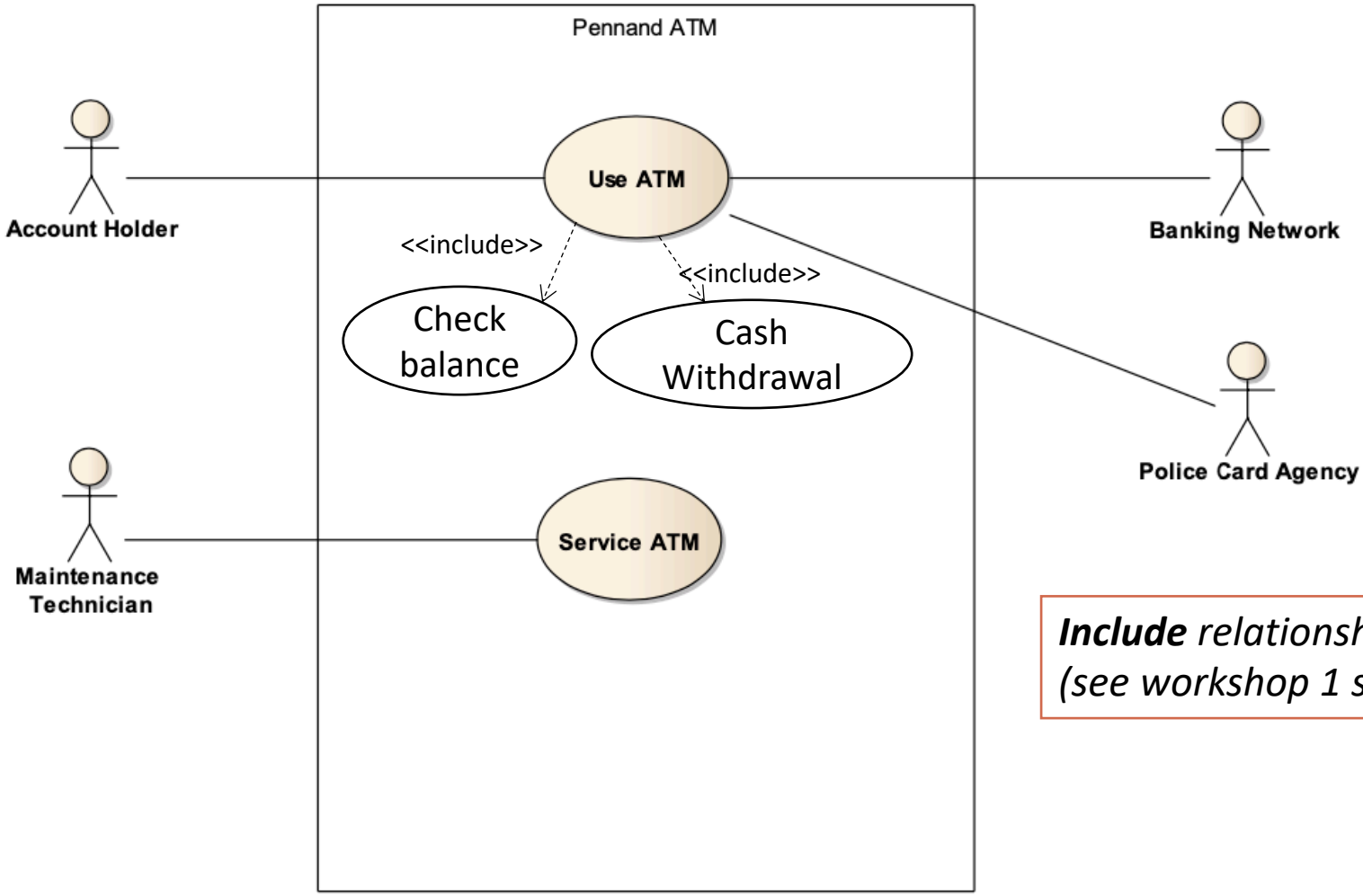
Example: Exam 2018 S1

Q81. Draw a use case diagram, covering the Pennand ATM description above.

> 1. The ATM allows an account holder to check the balance or make a withdrawal on one of their accounts.
> 2. When the ATM is idle, they can insert their transaction card and enter their PIN; the ATM will authenticate (via the banking network) the id from the card and their PIN.
> *…. <See complete version in LMS or Practice Exam> …*
> 12. When a customer first inserts their card, the ATM will photograph the customer and check the card details against a list of stolen cards (stored locally); a stolen card will be kept in the ATM and a message sent to the Police Card Agency, before the ATM returns to the idle state.
> 13. If the ATM at any stage detects a fault, it switches itself into out-of-service mode; a maintenance technician can then use this mode to conduct diagnosis/repairs, before shutting down and restarting the ATM.

Example: Exam 2018 S1

Use the Boss Test, EBP, or Size test to justify the level to express use case

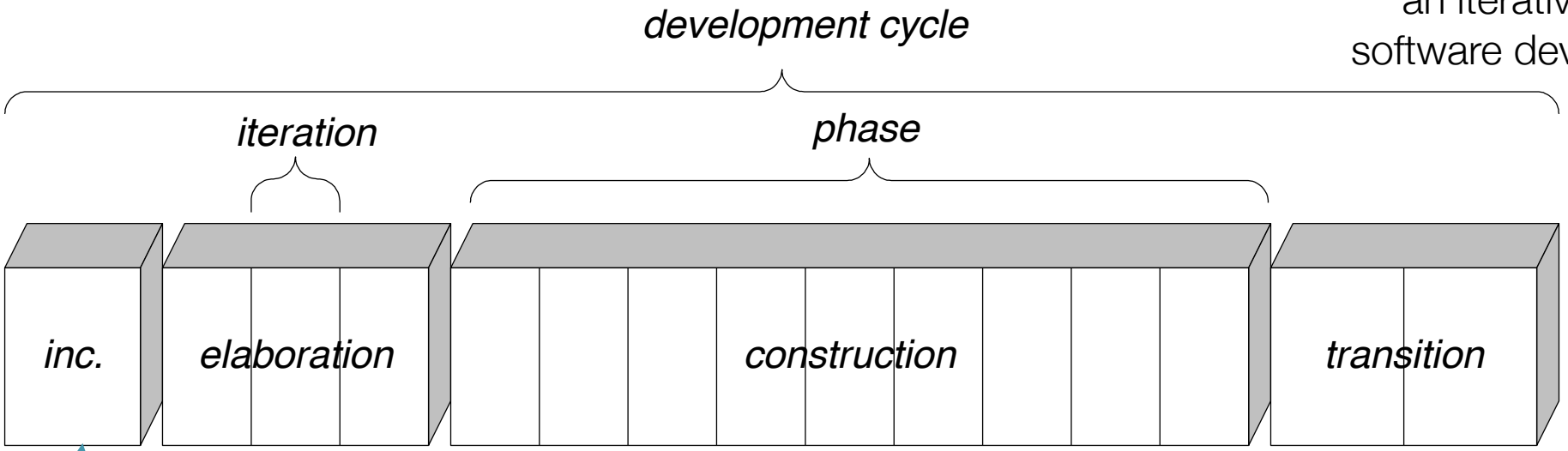


***Include** relationship can be added. (see workshop 1 sup materials)*

Answers for Exam 2018 S1 is available in Sample Exam page under the LMS exam module

An Overview

Unified Process:
an iterative & incremental
software development process



Inception Phase:
- Analysis of ~10% of use cases
- Go or no Go decision

Outcome:
- Common Vision
- Use case model (Use case text + Use case diagram)

Elaboration Phase:
- Building the core architecture, analyse the high-risk elements
- Discover and stablilise most of the requirements
- Start to think about the design

Outcome:
- Software Architecture Doc (e.g., component diagram, factor table, technical memo)
- Domain and Design models for some requirements

Architectural Analysis

Definition: An activity to identify *factors* that will influence the architecture, to understand their variability and priority, and resolve them

Architectural Analysis includes identifying and analysing:

- *Architecturally significant requirement*: The requirement that can have a large impact on the design (especially when it was not considered at the beginning)
- Variation points
- Potential evolution points

Goal: To reduce risk of missing a critical factor in the *design* of a system

Architecturally Significant Requirements

Functional Examples	Description
Auditing	Provide audit trails of system execution.
Localization	Provide facilities for supporting multiple human languages.
Printing	Provide facilities for printing.
Workflow	Provide support for moving documents and other work items, including review and approval cycles.

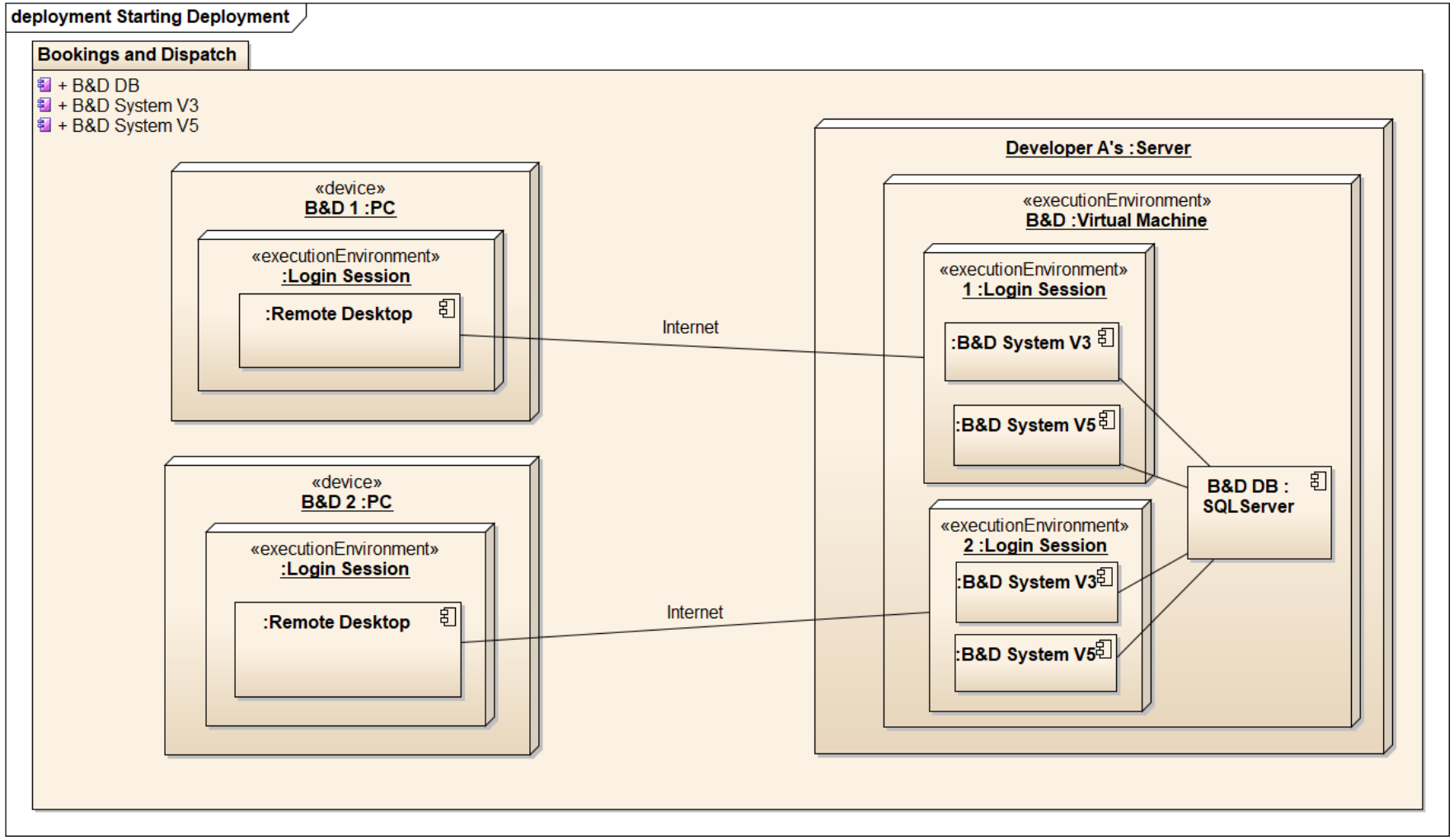
Non-Functional Examples	Description
Usability	Incl. UX, aesthetics, and consistency in the UI.
Reliability	Incl. availability, accuracy of calculations, failure recovery.
Performance	Incl. throughput, response time, recovery time, start-up time.
Supportability	Incl. testability, adaptability, maintainability.

Case Study: Chauffeured Car Company

Client's Requirement:
Want the system to be more ***responsive***

Current Issues:
Responsiveness

- Remote Desktop over internet to remote server
- Sharing of limited bandwidth: UI freezes



Current architecture of the system

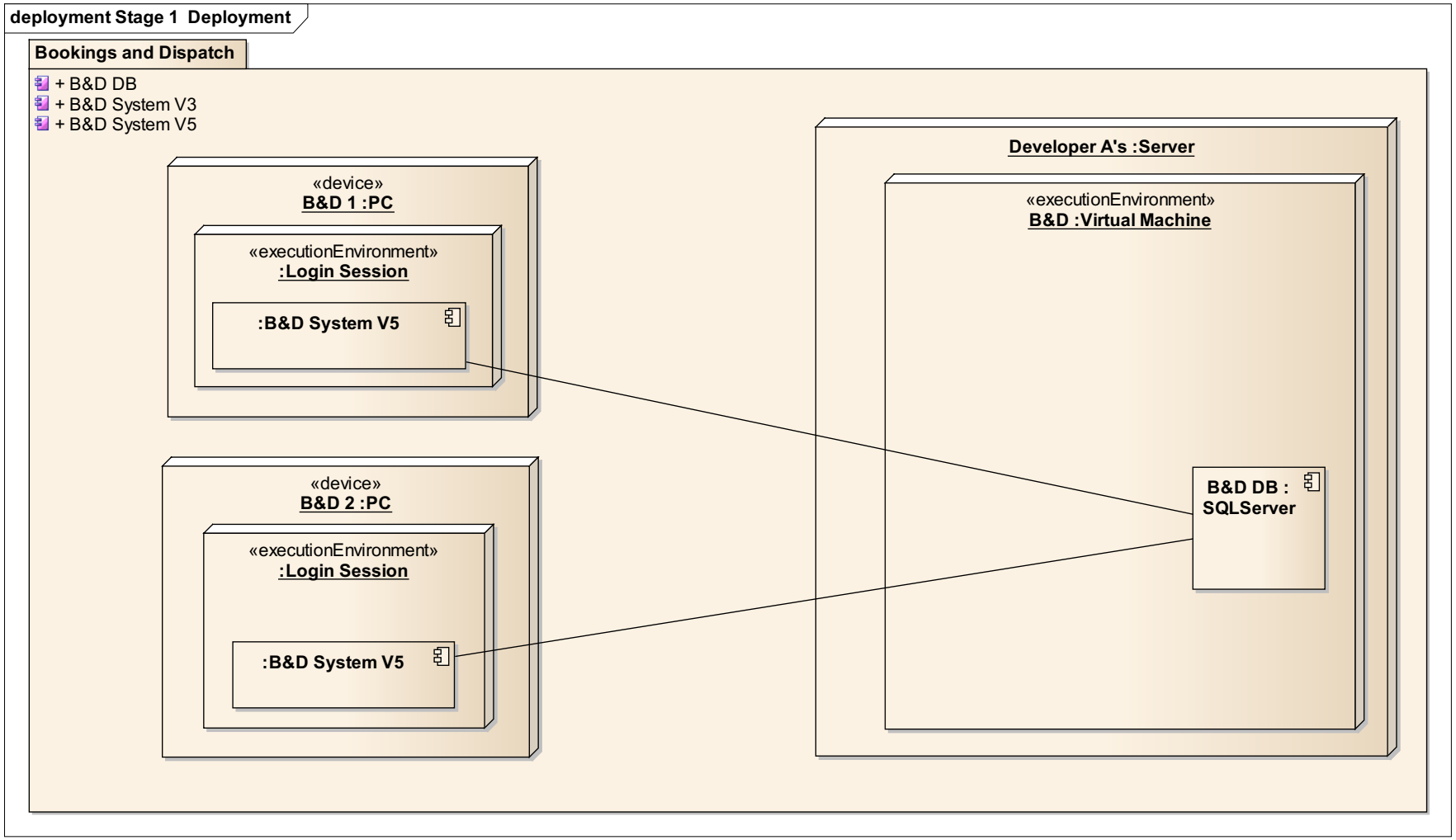
Case Study: Chauffeured Car Company

Client's Requirement:
Want the system to be more ***responsive***

Resolution:
Responsiveness

- Change from remote desktop + remote server to local system

Architecture is significantly changed to address the issues



Revised architecture of the system (Stage 1)

Example: Architecture Factor Table

Reliability—Recoverability of POS

Factor	Recovery from remote service (e.g., Tax Calculator) failure
Measures and quality scenarios	When remote service fails, re-establish connectivity with it within 1 min. of its detected re-availability, under normal store load in a production environment.
Variability (current flexibility and future evolution)	current flexibility - our SME says local client-side simplified services are acceptable (and desirable) until reconnection is possible. evolution - within 2 years, some retailers may be willing to pay for full local replication of remote services (such as the tax calculator). Probability? High.
Impact of factor (and its variability) on stakeholders, architecture and other factors	High impact on large-scale design. Retailers really dislike it when remote services fail, as it prevents them from using a POS to make sales.
Priority for Success	High
Difficulty or Risk	Medium

Example: Technical memo

Technical Memo: Issue: Reliability—Recovery from Remote Service Failure

Factors

Robust recovery from remote service failure, e.g., tax calculator, inventory

Solution

To satisfy the quality scenarios of reconnection with the remote services ASAP, use smart Proxy objects for the services, that on each service call test for remote service reactivation, and redirect to them when possible.

Where possible, offer local implementations of remote services. For example, implementing a small cache to store data (e.g., tax rates)

Achieve protected variation with respect to location of services using an Adapter created in a ServicesFactory.

Example: Technical memo (cont)

Motivation

Retailers really don't want to stop making sales! Therefore, if the NextGen POS offers this level of reliability and recovery, it will be a very attractive product, as none of our competitors provide this capability.

The small product cache is motivated by very limited client-side resources.

The real third-party tax calculator is not replicated on the client primarily because of the higher licensing costs, and configuration efforts (as each calculator installation requires almost weekly adjustments).

This design (Adapter and ServiceFactory) also supports the evolution point of future customers willing and able to permanently replicate services such as the tax calculator to each client terminal.

Unresolved Issues

none

Alternatives Considered

A “gold level” quality of service agreement with remote credit authorization services to improve reliability. It was available, but much too expensive.

Domain Model

Domain models capture the concepts, attributes, and associations in the domain

- **Domain class diagram** provides a *static* context of system: A visual dictionary of noteworthy abstractions, domain vocabulary, and information content of the domain
- **System Sequence Diagrams** provides *dynamic* context of system: How actors interact with system (as a black box) and how the system respond

- Domain models & SSDs should be expressed at the ***abstract level of intention***
  - Provide essential abstractions in information required to understand the domain in the context of the current requirements
  - Aid people in understanding the domain

Design Model

Design Model is a tangible representation of software classes to illustrate the structure, collaboration, and responsibilities of software objects

- **Design class diagram** provides a static view of the class names, attributes, and method signatures
- **Design sequence diagram** provides a dynamic view of how software objects interact via messages
- **State Machine Diagram** is a behaviour model that captures the dynamic behaviour of an object (or a system) in terms of states, events, and state transitions

To create Design model:

- Use the domain models to **inspire** the design of software objects
  - To reduce representational gap between how stakeholders conceive the domain and its representation in software
- Use a responsibility-driven design: GRASP and GoF design patterrns

GRASP

GRASP concerns on assigning responsibility to a software object:

Generic Problem: "Who should be responsible for Task *X*?"

Pattern/Principle	Brief Solution
Creator	If an object A contains or often use an object B, an object A should be responsible for creating an object B (Task X = Creation of B)
Information Expert	If an object A has a relevant information for Task X, an object A should be responsible for Task X
Low Coupling	Assign the responsibility for Task X to an object to make the coupling of the system remains low
High Cohesion	Assign the responsibility for Task X to an object that has strongly related responsibilities (the cohesion of the system remains high)
Pure Fabrication	If the solutions offered by other patterns (e.g., Creator, Expert) violate Low Coupling and High Cohesion, assign the responsibility for Task X to an artificial class that was represented in the domain model

GRASP

GRASP concerns on assigning responsibility to a software object:

"Who should be responsible for Task *X*?"

Pattern/Principle	Brief Solution
Indirection	To avoid direct coupling, an *intermediate* object to mediate between other components or services should be responsible for Task X.
Controller	What first object beyond the UI layer receives and coordinates ("controls") a system operation? -> Assign responsibility to a class representing one of: (a) a "root" object; or (b) a use case scenario that deals with the event. Note: • *Motivated by Indirection, Controller helps to avoid direct coupling between app logic and UI levels* • *Controller can be an artificial class motivated by Pure Fabrication*
Protected Variation	If there are variations or instability in these elements, assign responsibilities to create a stable interface
Polymorphism	If the behaviour of Task X is varied based on type or domain classes have various forms, assign responsibility for the behavior—using polymorphic operations—to the types for which the behavior varies

GoF Design Patterns

GoF Design Patterns: A set of patterns for specific problems

Pattern	Brief Description
Adapter	**Q**: How to provide a stable interface to similar components with different interfaces? **A**: Convert the original interface of a component into another interface, through an intermediate adapter object.
Factory	**Q**: Who should be responsible fore creating objects when there are special consideration? A: Create a Pure Fabrication object called a Factory that handles the creation
Singleton	**Q**: If there is an object A that many other objects need to access, who should be responsible for creating/containing an object A? **B**: Define a static method of the class A that returns the singleton object A.
Strategy	**Q**: How to design for varying, but related, algorithms, strategies or policies? **A**: Define each algorithm/policy/strategy in a separate class, with a common interface.
Composite	**Q**: How to treat a group or composition structure of objects the same way (polymorphically) as a non- composite (atomic) object? **A**: Define classes for composite and atomic objects so that they implement the same interface.

GoF Design Patterns

GoF Design Patterns: A set of patterns for specific problems

Pattern	Brief Description
Facade	**Q**: How to design for a common, unified interface to a disparate set of implementations or interfaces—such as within a subsystem—is required. **A:** Create a Pure Fabrication object called a Façade object to define a single point of contact to the subsystem
Decorator	**Q**: How to dynamically add behaviour to individual objects at run-time without changing the interface presented to the client? **A**: Encapsulating the original concrete object inside an abstract wrapper interface. Then, let the *decorators* that contain the dynamic behaviours also inherit from this abstract interface.
Observer	**Q**: Different kinds of subscriber objects are interested in the state changes or events of a publisher object, and want to react in their own unique way when the publisher generates an event. **B**: Define a “subscriber” or “listener” interface. Subscribers implement this interface. The publisher can dynamically register subscribers who are interested in an event and notify them when an event occurs.

THE UNIVERSITY OF MELBOURNE

Exam Information

Exam Information/Instructions

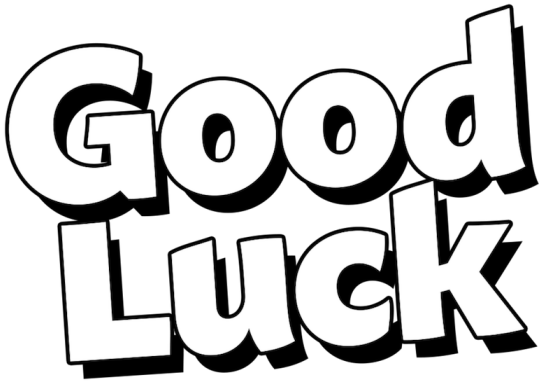
All the details are available at the **Exam Information & Instructions** page in LMS

Exam Consultation: We will mainly communicate via Piazza. I will schedule 2-3 online consultation sessions via Zoom during Nov 2-12 ☺

Important Note:

- Exam via Gradescope (Practice Exam is now available)
- Some questions will require you to draw a diagram
  - All diagrams must be ***legibly hand-drawn.*** *Include your student number on the top right of all diagrams.*
  - Before the exam, prepare paper + pencil + a facility to take a photo or scan your hand-drawn diagram *You can use a tablet to draw a diagram, but it must *be legibly hand-drawn* and *all text in diagrams must be hand-written*.

Exam Principles



- Is there reading or writing code in Exam?
  - *Short Answer*: Reading code -> Possibly, Yes; Writing code -> Unlikely
  - *Long Answer*: Coding is not the core focus of this subject, but coding can demonstrate the effectiveness of design

- What kind of questions/topics will be in the exam?
  - *Short Answer*: All the key topics mentioned today can be in the exam
  - *Long Answer*: Based on the learning objective of this subject, you should be able to “Apply the knowledge of modelling and design to solve computing problems” and “Evaluate and apply appropriate choices of software tools in the modelling and design process”. These are the key focus of the exam. And you should demonstrate these in your exam answers.

Q&A about the exam

Q: Why can’t we use draw.io for an exam answer?

A: The University is very serious about academic misconduct. We need to ensure that all the answers are genuinely yours. So, we need your hand-writing and hand-drawing to prove that.

Q: If we run into any issue with uploading a photo, can we get partial mark by describing our solutions in words?

A: There will be a period of 30-min after exam for you to ensure that all the photos are uploaded. But if you still encounter technical issues, please check the special consideration.

Q: Is there any recommendations for scan the hand-writing?

A: Please refer to this recommendation by the University. To ensure that everything will work during the exam, please try to do Practice Exam and upload your diagram

Q&A about exam

Q: Are we able to start writing during reading time?

A: In theory, No. But practically, yes. It's okay ☺

Q: Are the workshop supplementary material relevant to the exam?

A: Yes. However, the topics that will be in the exam should mainly be in the lecture slides and workshop exercises.

Q: Will TDD and refactoring be in the exam?

A: Possibly because TDD and refactoring are part of workshop supp materials.

Q: Do we need to know the specific tools to use, e.g., Amazon web service?

A: No. That's not the key focus of this subject

Q: Would there be a detailed case study like week 11 in the exam?

A: There will be a case study in the exam, but not as big as the week 11 or project assignments. The size and complexity should be reasonable given the fixed timeframe (2 hours of exam).

Q&A (Others)

Q: How detailed should our design sequence diagram be?

A: It should reflect how your code work. If your diagram is too large, you can use ref frame. So that you have another diagram for that part.

Q: What kind of classes should and shouldn’t be in the domain model?

A: The domain model capture the understanding about the domain and requirements. If the classes are more relevant to implementation (but don’t have meaning from the stakeholder point-of-view). Then, it shouldn’t be in the domain model.

Q: A class made for pure fabrication wouldn’t be included in the domain model, right?

A: Correct.

Q: Should architecture always go first, before the domain and software design?

A: Architecture, Domain model, and Design model can be developed concurrently in multiple iterations. It is not like a waterfall model where you have a solid architecture before doing something else.

Lecture Identification

Lecturer: Patanamon Thongtanunam

Semester 2, 2020

These slides include materials from:

Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition, by Craig Larman, Pearson Education Inc., 2005.