Faculty of Applied Science & Engineering
University of Toronto

Midterm Exam

Department: Electrical & Computer Engineering
Instructor: Matt Medland
Date & Time: Tuesday, Oct. 27, 2015 – 3:10 p.m.

Conditions: Closed Book
Duration: 50 minutes

This test counts for 20% of your final grade

Name: SOLUTIONS.
(Please underline last name)

Student Number: _____

Question Marks

1 _____ /50

2 _____ /10

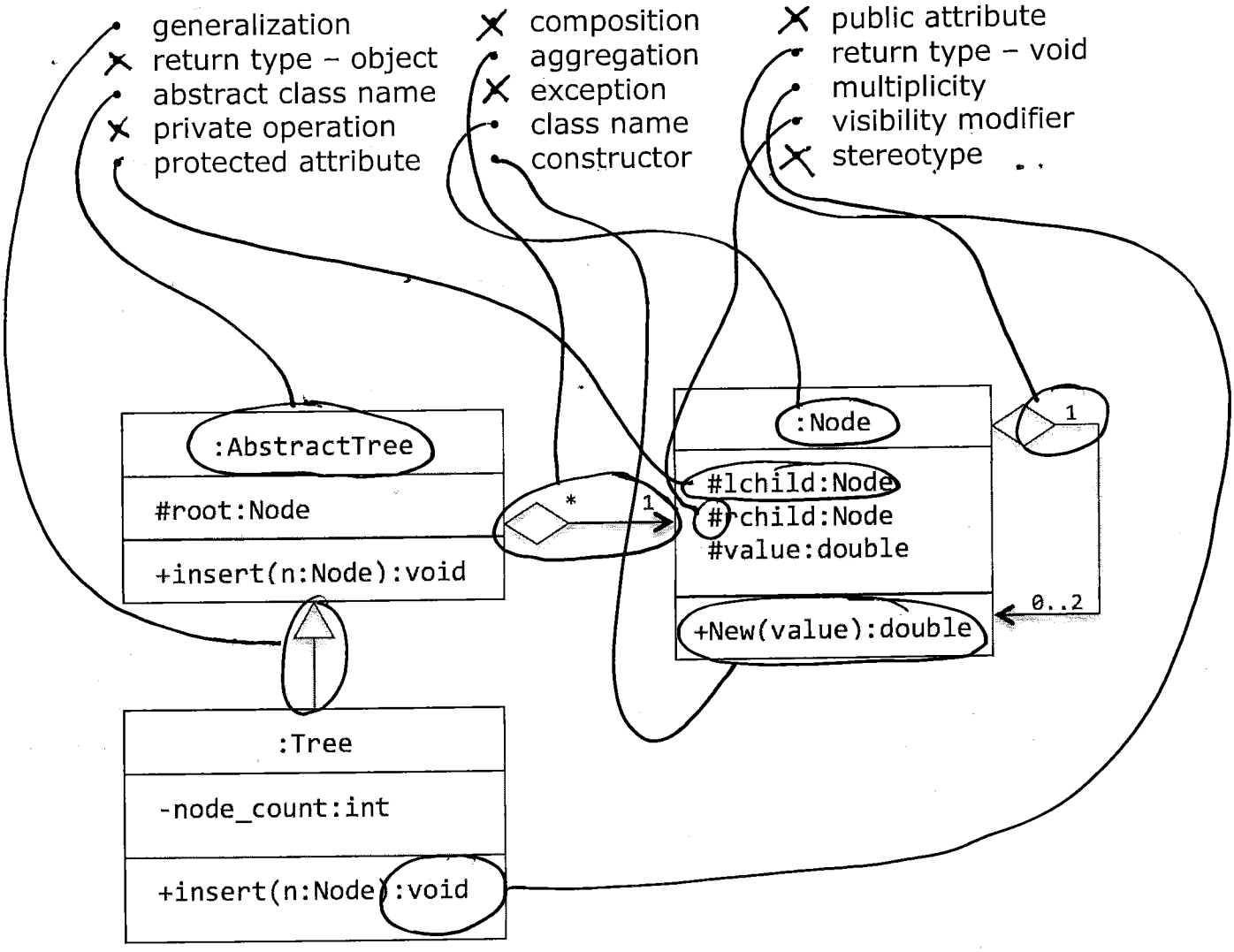
3 _____ /20

Total _____ /80 = _____ %

1. [Modeling & UML Questions, 50 marks Total]

(a) [UML Class Diagrams – 15 marks] Identify items from the list below in the given UML Class Diagram by circling the item in the diagram and drawing a line to the name in the list of elements. Marks are awarded for correct answers and **marks are subtracted for incorrect answers.** Not all elements in the list appear in the diagram. For the elements that do not appear place an "X" beside them:

Class Diagram Elements



(b) [Sequence Diagram – 15 marks] The three-phase commit protocol (3PC) is a non-blocking, distributed algorithm, that lets all nodes in a distributed system commit a transaction. A simplified version of 3PC, with one coordinator and one member in the cohort (normally there can be any number in the cohort), is described below:

3PC Process

The 3PC process, as seen from both the coordinator's and cohort's point of view, is outlined below:

Coordinator:

1. Coordinator sends **canCommit** message to cohort.
2. Coordinator receives either a **Yes** or a **No** message from the cohort. From this point forward, we will assume the response received was a **Yes**.
3. Coordinator sends a **preCommit** message to the cohort and receives an **ACK** in reply.
4. The coordinator commits the transaction and sends a **doCommit** message to the cohort and receives a **haveCommited** message in reply.

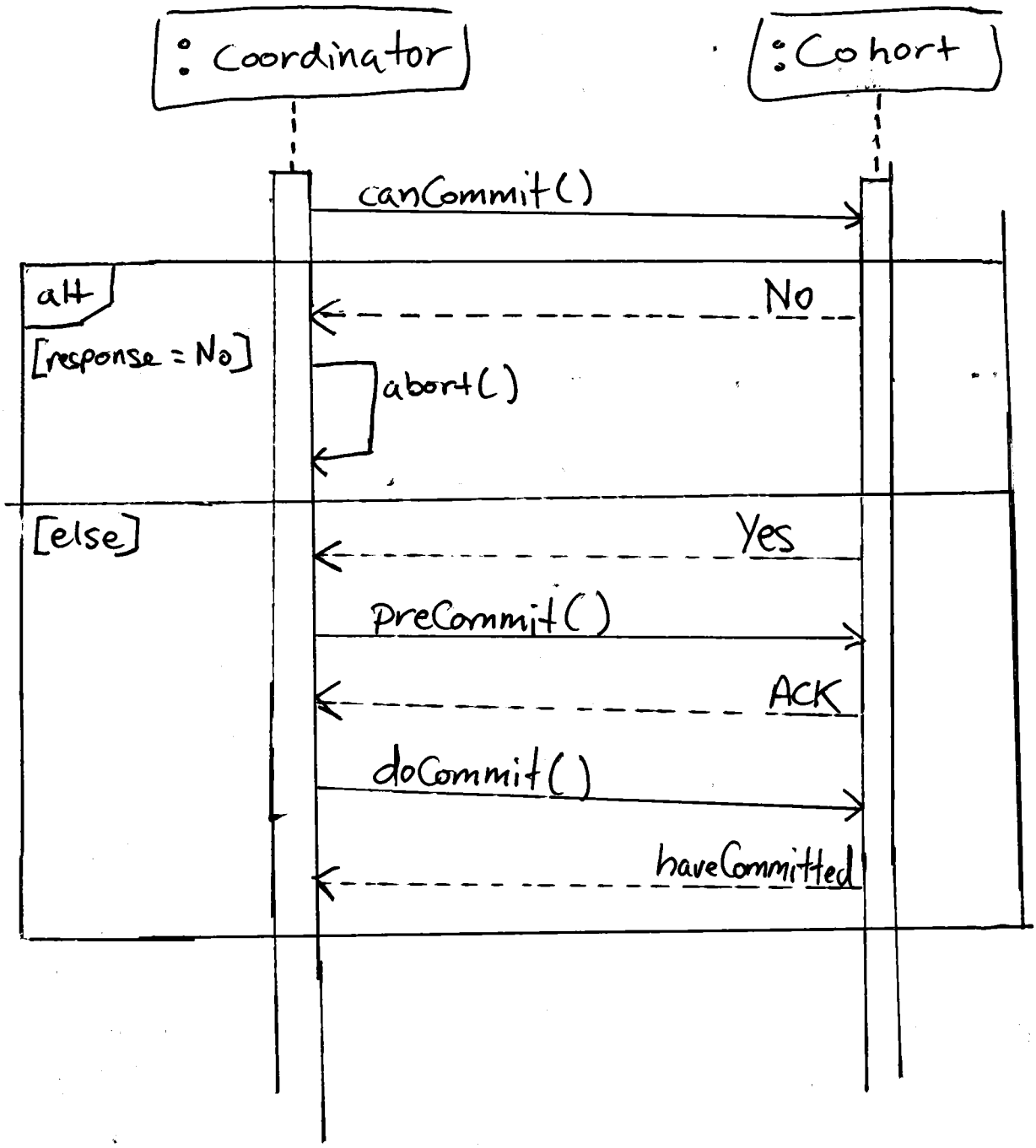
Cohort:

1. Cohort receives a **canCommit** message from the coordinator and replies either **Yes** or **No**.
2. Receives **preCommit** message from coordinator and replies with an **ACK** message.
3. Receives **doCommit** message from coordinator, commits the transaction, and replies with **haveCommitted** message.

At this point, both the coordinator and the cohort have committed the transaction.

On the next page, draw a UML sequence diagram representing the 3PC protocol.

[answer question 1(b) here]



(c) [Sequence Diagram Extensions – 5 marks] In the previous question, where you drew a sequence diagram for 3PC, we specified that the cohort only contained a single member. In the case where the cohort contained multiple members (i.e. the normal case) in what ways could we extend the sequence diagram to make that clear? Give at least two (2) suggestions.

① could use a loop frame to show iteration among all members of the cohort

② could use a par frame to show that each cohort member can be consulted asynchronously by the coordinator

In general, #2 is probably more efficient on a parallel system with multiple processors.

(d) [Use Case Diagram – 15 marks] Spishak Airlines has contracted you to build a kiosk-driven application that allows travellers to purchase airline tickets, check bags, and print their boarding passes. A detailed use case for the kiosk software is given below:

Book a Flight

Main Success Scenario:

1. enter destination & select flight
2. enter traveller's personal information
3. traveller scans passport & verify with external authority
4. traveller books a ticket on plane
5. [optional] traveller selects seat, regular or first class
6. [optional] traveller weighs and checks bags
7. process payment for flight
8. authorize transaction with bank
9. bank processes transaction
10. print boarding passes

Extensions:

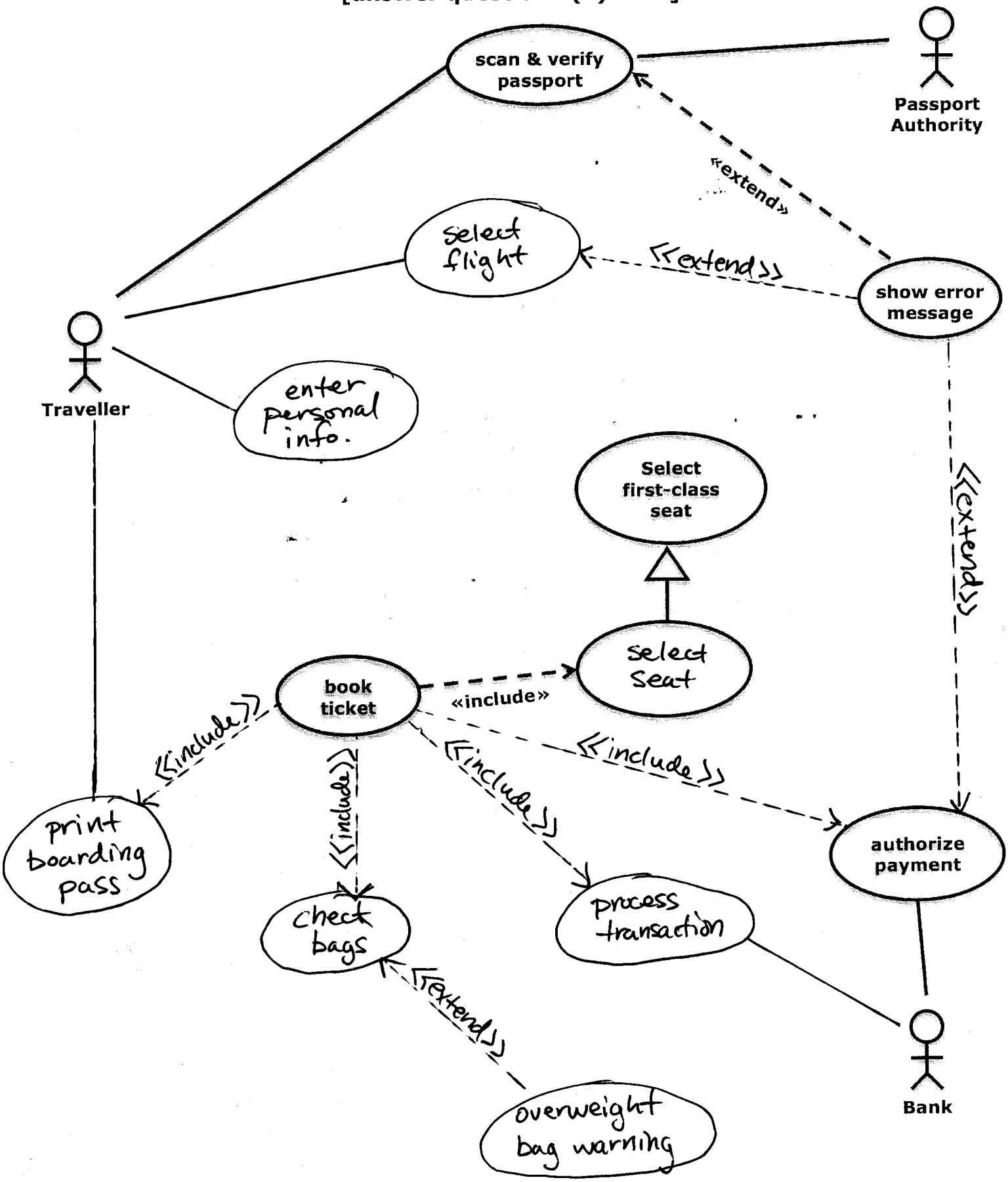
1a. no flight to selected destination, show error message
3a. invalid or expired passport, show error message
6a. bags overweight, show warning for increased price
8a. payment authorization failed, show error message

Complete the use case diagram for this detailed use case, which is already started on the next page.

Some hints and suggestions:

- use the scratch paper at the back of this package to practice laying out your diagram so you don't make a mess
- you shouldn't have more than a dozen (12) use cases in the diagram in total – write them down on the scratch paper before starting your drawing
- the "book ticket" use case will have the most **«include»** relationships, and in general is the most interesting

[answer question 1(d) here]



2. [Software Architecture, 10 marks Total]
In lecture, we discussed many different types of software architectures and architectural patterns. Give a brief description of the following software architecture models:

- client-server

  two main parts to system. one (the client) sends requests to the other (server) and handles results.

- open layered architecture

  multiple layers where each layer above can call into all layers below. Hard to maintain!

- closed layered architecture

  multiple layers where each layer can only use services exposed by the one layer immediately below

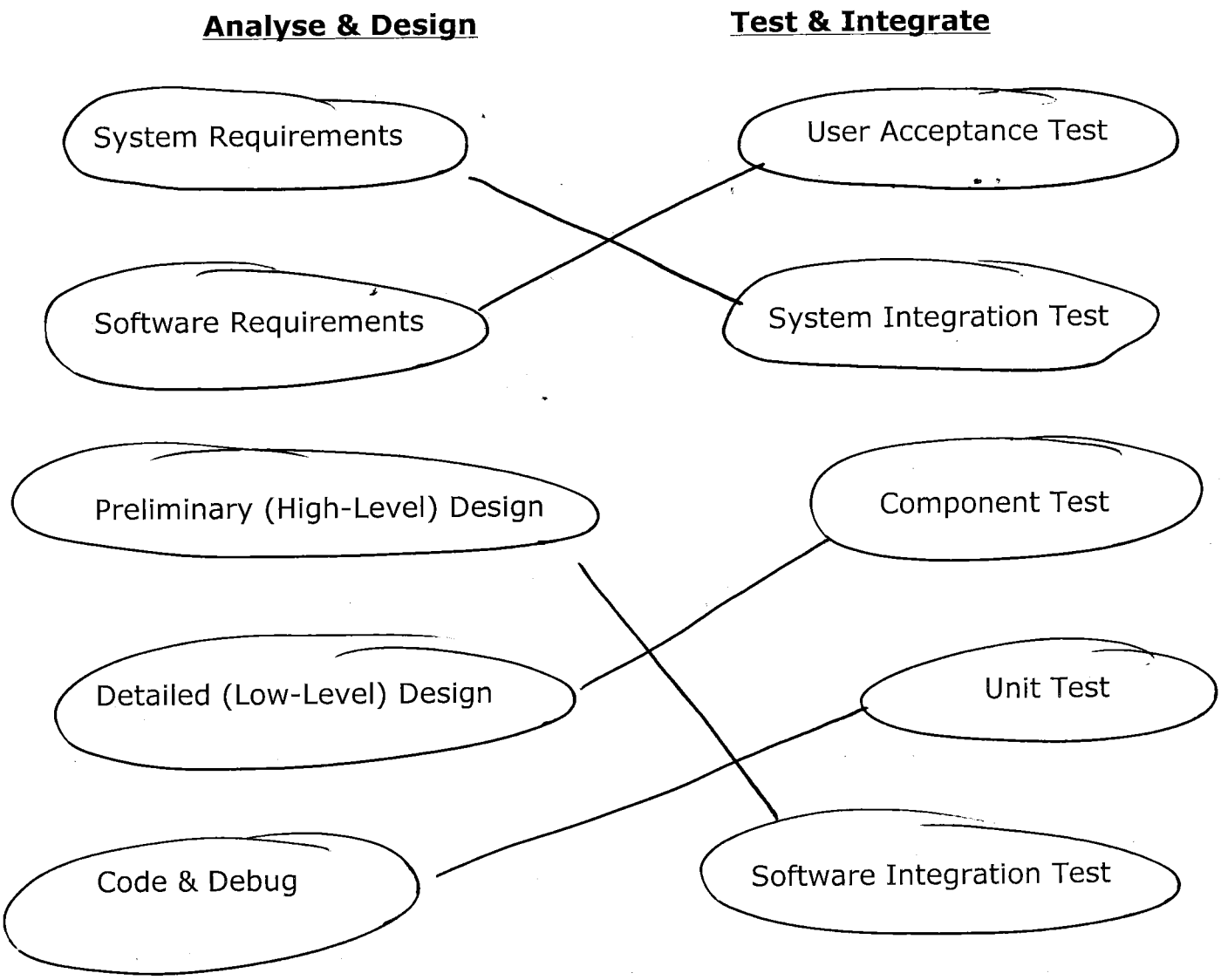
- pipe-and-filter

  multiple processing units or modules forming a chain. each unit's output is the input into the next & so on.

- model-view-controller

  a user-interface pattern for systems where many different views can display data from the same underlying model.

3. [Software Development Lifecycle, 20 marks Total]
(a) [V-Model – 10 marks] In the V-model for software development there are a number of levels of abstraction in the **"analyze and design" phase** that are paired with specific tests carried out during the **"test and integrate" phase**. Draw lines below connecting the design levels with their corresponding test steps.



(b) [Agile Horizon Planning – 10 marks] Explain the concept of **Horizon-based planning** for software development. Describe a specific case, where the software development team plans in **two-week sprints**, and the rest of the company plans their activities on **quarterly boundaries**. How many sprints should development management be planning in advance in this case?

The horizon, in terms of software development, is a timeframe ending on a date when a specific set of new features are expected to be delivered.

If the R&D team work in short sprints, some of the new features may be "live" before the end of the horizon. This is really just seen as a bonus.

In the case of a company that plans quarterly (every 3 months) and sprints every two weeks, the R&D management should be planning roughly six (6) two-week sprints in advance.

[scratch paper]

[scratch paper]