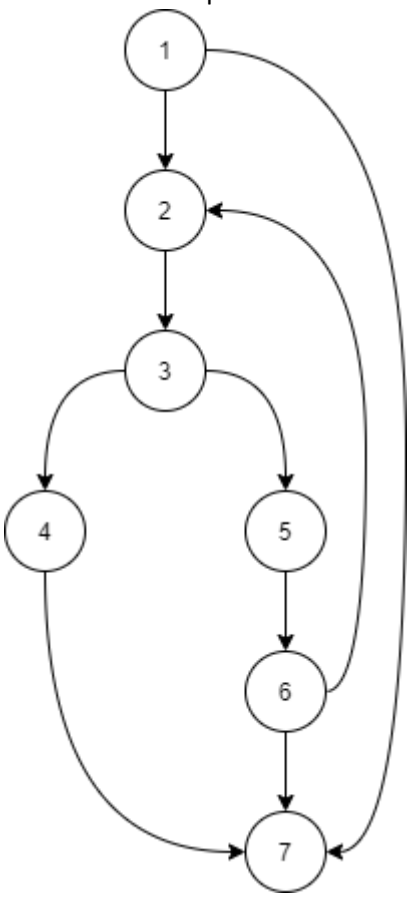
SECM: Semester Two Exam Answer Sheet

From Section A of the exam, I will be attempting **Question 1**.

From Section B of the exam, I will be attempting **Question 3** and **Question 4**.

Question 1

a) Control Flow Graph



b) Cyclomatic Complexity

The formula to calculate cyclomatic complexity is E (number of edges) – N (number of nodes) + 2P (number of connected components). Since all of the nodes are connected there is only one component so we will add 2 to our number. There are 9 edges and 7 nodes, so 9 - 7 + 2 = 4.

The cyclomatic complexity is 4.

c) Linearly independent paths

- 1-2-3-4-7
- 1-2-3-5-6-7
- 1-2-3-5-6-2-3-5-6-7
- 1-7

d) Test cases

1. For GetDateConflict to return **"None"**, in the preference set of the Response object:
   Calendar date: 23 July 2020
   Start time: 10:00
   Finish time: 12:00

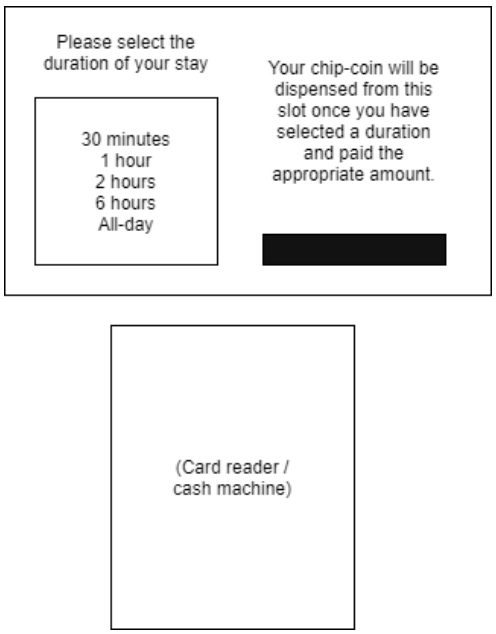
2. For GetDateConflict to return **"Weak"**, in the exclusion set of the Response object:
   Calendar date: 24 July 2020
   Start time: 14:00
   Finish time: 17:00
3. For GetDateConflict to return **"Strong"**, in the exclusion set of the Response object:
   Calendar date: 23 July 2020
   Start time: 11:00
   Finish time: 14:00

e) To accommodate for the use of Microsoft Fakes to test this unit of code, direct references to the Response class should be removed and replaced with references to an intermediate class that can either return real or test values, unbeknownst to the code referencing it. Once Microsoft Fakes has been installed and set up on the development system, the code directly accessing the Response class would be amended as such (line numbers referencing the code given in the exam script):

[3] if (IConflictChecker.isExcluded(response, possibleDate));
[5] if (IConflictChecker.IsPreferred(response, possibleDate) == false)

Question 3

i. Secure Car Park System
   i. Five pertinent questions:
      i. Would you prefer to pay by cash or card/contactless?
      ii. Would you pay more for a secure parking spot?
      iii. Do you carry a wallet/purse/handbag/something that a chip-coin could be stored in?
      iv. How often are you likely to utilise this facility to park?
      v. How long do you think you would stay parked here on average?
   ii. Two user stories:
      i. As a driver, I want to be able to pull into the facility and pay to access the car parking facility, and be assigned a slot in which to park my car.
         Acceptance test: The user stops their car alongside the booth in the entry and selects a duration they want to stay for. The booth dispenses a coin and grants access to the facility, informing the user where they should park their car.
      ii. As a driver who has parked their car in the facility, I want to be able to regain access using my chip-coin, retrieve my car, and leave.
         Acceptance test: The user arrives at the pedestrian entrance to the building and enters their chip-coin into a slot. The door unlocks to grant access to the facility, and they are reminded of the bay assigned to their car earlier. They go to their car and drive it out of the facility, where an exit barrier is automatically raised.
   iii. A Brief Design:
   Shown below is a low-fidelity prototype I have designed for the entry booth where the chip-coin is dispensed. The left-hand side of the main screen would be a software touchscreen so it can be updated if necessary. The card / cash machine would be a standard issue device linked to the system.



iv. When the system is nearly finalised and undergoing checks before opening to the public, the chip-coin system will be examined against user requests obtained from part i. and user stories and acceptance tests (which can be re-enacted using the real system to test the outcomes). The final look and feel of the system can also be judged as to whether it aligns with the prototypes produced by the design team in the early stages (the design in part iii.)

Question 4

a) Changes to the requirements late in the development

The aim of agile development is to work closely around the requirements to produce a product that fully satisfies them in the least amount of time possible.

As the focus is on the requirements, it is essential to always welcome changes to the requirements – at any point in the development – because if the desires of the user(s) changes during the development, but the product is still built around their original desires as listed in the requirements then the agile technique has failed.

If such a late change were to be requested, the agile development team should take on board the new requirements and see how they align with earlier planning techniques such as user stories and storyboards.

It is possible that the requirements may have only changed slightly and these plans can be amended; the changes can then flow through the design process again until the software product's code is altered.

If the changes are more major then it may be worth a more high-level reconsideration of the product in its current state, as it may save more time to simply return to the start, or near the start, of the designing and planning stage – after all, a half-developed then heavily modified body of code may start to break down with all sorts of unexpected bugs which could add even more time to the development due to faulty software.

b) Criticality of time-to-market for commercial products

c) Zero-defect programming

As many software engineers are passionate about their roles in producing code, they strive to make what they produce as error-free and “elegant” as possible. However, the fact of the matter is that sometimes this should not necessarily be the priority.

It is obvious, of course, that error-prone software is not and should not be seen as a good thing, however when considering the requirements laid out in the design phase of a project, and comparing them against the amount of time invested into the product, there may be certain parts of the system that demand more attention in the production phase as they are more critical requirements in the eyes of the user.

If the project is on a tight schedule, and the developer(s) is aware that there are bugs in the software that may not be resolved before release, they may either attempt to postpone the release, or release it knowing that it contains bugs. It is worth considering that these bugs may be very minor i.e. rarely occurring, or inconsequential upon occurrence to the user achieving what they want to do.

> For example, in many programs there are error messages which may appear due to a bug in the code, however the user’s goal is not adversely affected. If this was the kind of bug that a software engineer knew was present in their code and there were tight time-limiting factors, it may be worth releasing the software knowingly and quickly working towards a bug-fix update to amend the issue soon after release.