Assignment1

Analysis, Design and Software Architecture 2021

September 2021

1 C#

1.1 Generics

In the first method the constraint requires T to be a type that implements the interface IComparable. This means that all instances of type T can be compared to each other.
The constraints in the second method requires U to implement the interface IComparable, while T is a naked type that inherits from U and therefore can be any type that is a subclass of U. Hence, T indirectly implements IComparable<U>, and can be compared to other instance that are the same type as U or a type inherit from U.

2 Software Engineering

2.1 Exercise 1

Knowledge acquisition is a non-linear process and therefore not sequential. When collecting information about our system during the modelling of the application and solution domain, we cannot expect this process to be linear as the addition of a new piece of information may invalidate our model(s) and force us to start from scratch, thereby going back to a prior step/process. For example feedback given from the end users of the system, during a usability test of the user interface may force the developers to update the solution model with this new information thus forcing them to go back to the modelling step/process.

2.2 Exercise 2

The following decision was made during the system design process as it breaks the system into smaller subsystems that can be realised by the development teams:

- "The ticket distributor is composed of a user interface subsystem, a subsystem for computing tariff, and a network subsystem managing communication with the central computer."

The following decision was made during requirement elicitation. The requirement is functional as it describes an interaction between the user and the TicketDistributor:

- "The ticket distributor provides the traveller with an on-line help."

The following decision can either be made during requirement elicitation by the client or during system design by the system designer:

- "The ticket distributor will use PowerPC processor chips."

2.3 Exercise 3

> "Assume you are developing an online system for managing bank accounts for mobile customers. A major design issue is how to provide access to the accounts when the customer cannot establish an online connection. One proposal is that accounts are made available on the mobile computer, even if the server is not up. In this case, the accounts show the amounts from the last connected session."

Figure 1: Exercise 3: Account belongs to the application domain when underlined with red and the solution domain when underlined with blue

As seen on figure 1 the term account belongs to the application domain, when using it as a means to describe problems or concepts from the real world that are relevant to the system.
While the term account belongs to the solution domain when using it to describe possible solutions to the problem area of the system.

2.4 Exercise 4

There is not as much material waste if a program fails, and we can typically use what has already been made, whereas if an airplane fails and crashes, it is harder to recycle what is left. So besides the power to the computers, it is not as bad for the climate to develop a word processing program. In addition, we can test the program along the way, which means that we can work iteratively on a different level than with the bridge or the airplane. Of course, we can build prototypes of bridges or airplanes, but it will most likely be small versions of them, whereas the word program can be tested along the way, and we can test features and expand with new features. We cannot test whether a wing works without the rest of the airplane unless we have another airplane to test it on or whether the bridge can carry cars. There is probably a lot of calculation of that before it is built.

2.5 Exercise 5

The following requirements are non-functional as they are concerned with the implementation, usability and reliability of the TicketDistributor respectively and therefore are not directly related to the functional aspect of the system:

- "The TicketDistributor must be written in Java."
- "The TicketDistributor must be easy to use."
- "The TicketDistributor must always be available."

The following requirements are functional as they describe interactions between the user and the TicketDistributor:

- "The TicketDistributor must enable a traveller to buy weekly passes.
- "The TicketDistributor must provide a phone number to call when it fails."

2.6 Exercise 6

Modelling allows us to work with and understand complex problems as the model, that is an abstract representation of our system, simplifies the problem by filtering out details that are irrelevant to our system/problem. We can then gradually add more detail to the model, so that the model approaches the real world phenomenon.
Furthermore modelling the application domain lets the developers gain an understanding of concepts from the application domain that are important to the system but otherwise unknown to the developers. Similarly modelling the solution domain lets the developers evaluate and compare different solutions that would otherwise be too expensive to build or to complex to understand.
Moreover the model can be used as a tool during the early stages of the development process to get feedback from the client before writing any code. Making changes after code has been written can be a very time and resource consuming endeavour.