BDSA - Assignment 01

Alaa Abdul-Al (alia)
Shuja Hussain (shhu)
Mohammad Hasham (mhas)
Gustav Sølvsteen Olesen (guol)

September 17th, 2021

1 C#

The code can be found in following Github repo:
https://github.itu.dk/alia/Assignment1.git

1.1 Generics

In the first example T must implement the interface *IComparable¡T¿*. In the second example, T is a subset of U, furthermore U implements the interface IComparable <U>

2 Software Engineering

2.1 Exercise 1

Knowledge acquisition is basically learning new information. By saying it is sequential, it means that the new knowledge we learn, can disprove what we already knew.
An example is that many years ago people had the thought that Earth is flat. By gaining new information, we had to disprove the previous knowledge, as we realised Earth actually is an irregularly shaped ellipsoid.

2.2 Exercise 2

The system design decisions are coloured in blue while the requirements are coloured in green.

- "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 ticket distributor will use PowerPC processor chips."
- "The ticket distributor provides the traveler with an on-line help."

2.3 Exercise 3

The application domain concept is coloured red while the system domain concept is coloured blue.

"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."

When used in an application domain concept, it means we are talking about the context in the real world. On the other hand, when we talk about the system domain concept, the context suddenly switches over to a modelling of the real world, where we talk about classes or functionalities for example.

2.4 Exercise 4

The difference between developing an aircraft and a bridge, compared to a word processor, according to Brooks [1], is that software is characterised by the following:

1. Conformity
    - When building an aircraft, you have to keep in mind the different physical laws etc. Whereas in software, there are no real laws to keep in mind.
2. Complexity
    - Building software is more complex than building an aircraft which is why the delivery date of software may get postponed, as it was the case with the word processor.
3. Changeability
    - this characteristic encompasses that software often changes due to new technology, new requirements proposed by the user etc. It is often the case with software that customers/users ask for a refinement of the requirements of the software system which usually leads to refactorization of many parts of the developed software. This often entails that the deliverance of the final product (software) is delayed.

[1] Brooks, F. P. 'No silver bullet: essence and accidents of software engineering', 1995.

4. Invisibility

    - When building an aircraft, you have something physical to take as your starting point, whereas in software development, you cannot really see what you are building.

2.5 Exercise 5

Functional requirements are marked in orange while non-functional requirements are marked in blue.

- "The TicketDistributor must enable a traveler to buy weekly passes."
- "The TicketDistributor must be written in Java."
- "The TicketDistributor must be easy to use."
- "The TicketDistributor must always be available."
- "The TicketDistributor must provide a phone number to call when it fails."

A Functional requirement can be understood as a specific functionality or feature that should be included in the system. A non-functionality requirement can be understood as a constraint that the system as a whole should live up to. Based on these definitions, we have classified the requirements above.

2.6 Exercise 6

Modeling helps us deal with complexity by enabling us to incrementally refine simple models into more detailed ones that are closer to reality. Models are made to make bigger and complex systems more clear and understandable. Models can also be used to understand no longer used systems. More specific to software development can we make use of UML, which make use of Object-oriented methods. Modelling with the OO methods is the combination of the application domain and solution domain modeling activities into one.