BDSA Assignment 2

jakst, lawu & tbav

September 2022

Link to github repo: https://github.com/ThomasBavn/BDSA-assignment-02

C#

0.1 Types

Classes and structs share almost all of the same functionality. However, whereas classes are reference types, structs are value types. This means that structs are copied where a reference would be created for a class. This is desirable when you want value-type logic for your data, for example if it primarily represents numeric values. Furthermore, structs do not instantiate objects on the heap, which can lead to significant memory savings when dealing with many instances of the same type and should be used there instead of classes if possible. Structs do not support inheritance however, so are less useful in more object-oriented programming projects where inheritance is a more readable and easy to understand design choice.

Records of both classes or structs are designed to be immutable - so data cannot be changed or modified in a record of a class or struct. Records only support nondestructive mutation - which means that the developer must create a new record and copy all values except the ones that must be changed in order to "modify" a record. Records are useful if you want to implement structural equality instead of referential equality. Structural equality checks if the values, not the references, are the same. This is very useful to simplify equality checking, so this should be implemented in systems that do this often, if possible. Most classes and structs in any project benefit from the ability to easily change their internal data however, so records might simply be too clunky a solution in most cases.

Extension Methods

```
2 references
public static IEnumerable<T> Flatten<T>
(this IEnumerable<IEnumerable<T>> source)
=> source.SelectMany(x => x).ToList();

2 references
public static IEnumerable<T> Filter<T>
(this IEnumerable<T> source, Func<T, bool> filter)
=> source.Where(filter);
```

Delegates/Anonymous Methods

```
1 reference
public static Action<string> ReverseOrder = (string i)
=> Console.WriteLine(i.Reverse().ToArray());

2 references
public static Func<double,double, double> ProductOfTwoDoubles
= (double x, double y) => x * y;

2 references
public static Func<int,string, bool> NumericallyEqual
= (int i, string k) => i == double.Parse(k);
```

Exercise 1

Use cases describe a single interaction between the user and a system, visualized in a diagram whereas a scenario is a more general diagram of the entire solution and how the different parts of the solution are supposed to work together. You would usually use scenarios to get an overview of the solution's general structure and see how the parts are interconnected.

Both scenarios and use cases are made when developing a solution, but the scenario is created first to get the rough outline of a system's structure and after that the use case diagrams are created from the different parts making up the scenario.

Exercise 2

- Identify and briefly describe four types of requirements that may be defined for a computer-based system

Functional Requirements

These are the requirements that define specific functionality - in short, what the program should do. These can vary in detail and type, but all focus on what type of functionality is to be included.

Opposite those are the nonfunctional requirements. These will, more or less, specify or put constraints on the functional requirements. There is a bigger variety of subcategorizations here than for functional requirements, but all of them are important in some way. We will now describe three types of requirements:

Efficiency Requirements

Efficiency requirements are for both performance and space requirements. In short, these are the constraints on the amount of space the system should take, and the constraints on the performance of the system itself. This is a subcategory of the product requirement type.

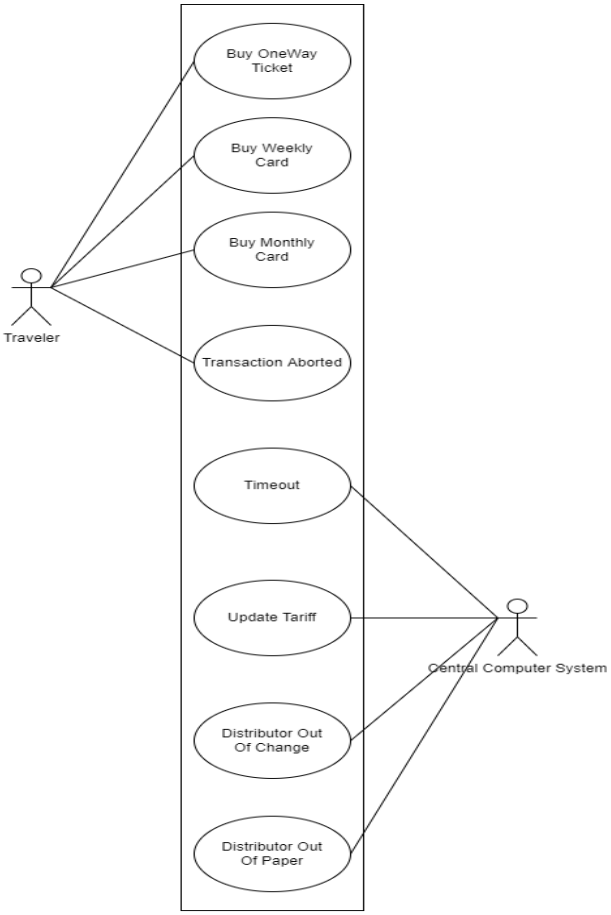
Developmental Requirements

These requirements relate to how the developers actually develop the system - which programming language is to be used, which process is used to plan out and develop the system, etc. A subcategory of organizational requirements.

Legislative Requirements

The legal constraints that a system must adhere to. These are usually related to matters like security and handling of private personal data, although these demands can expand as the law regarding data or development of software evolves. A subcategory of external requirements.

Exercise 3


Buy OneWay Ticket
Buy Weekly Card
Buy Monthly Card
Transaction Aborted
Timeout
Update Tariff
Distributor Out Of Change
Distributor Out Of Paper
Traveler
Central Computer System

Exercise 4

Ambiguous requirements

- How is the solution going to make the user proud to work for the company?
- How much of the solution should be accessible on mobile devices?

Most of the third paragraph in its entirety is rather confusing. It's stated that the solution should be able to be used in pretty much all scenarios.

Missing information

- What are the core services that should be taken care of?
- What parts of the system should be accessible on mobile devices?

Problem with the non-functional requirements

First of all, the description is very long, with a lot of unnecessary details, that could be summed up a lot quicker.
They pretty much make it one of their requirements that the solution should make it more attractive to work for the local authority and linking this to professional pride, however the connection here is only very loosely stated.
The requirements are very loose in the sense that they want it to be a magic solution where everyone should be able to use it anywhere including doing demanding tasks.

The problem with these requirements is that they say a lot about what they want in terms of it being usable in every scenario, but there are no mentions of HOW any of these things should be achieved or any way that the requirement should be tested, as almost all of the mentioned requirements are too vague to be thoroughly tested.

Rewritten description

Da løsningen skal bruges af mange forskellige målgrupper i mange forskellige situationer, er det vigtigt at

- Den digitale løsning skal være intuitiv og kunne bruges af personer mellem 20 og 65
- 85% af løsningen skal kunne bruges på en mobil enhed
- Brugeren skal føle sig godt understøttet ved at bruge løsningen. Dette skal forstås som at brugeren skal
  - Føle sig sikker i at dele data igennem løsningen
  - Kunne arbejde effektivt uanset hvilken opgave de arbejder på

Exercise 5

Identify actors that interact with a music tracker software system

- The primary actor is the composer that interacts with a music tracker software system to write music.
- The database where the music is extracted and stored from the music tracker software is also an actor

Formulate three use cases in structured language that a software music tracker system has to support

Use case nr:	01
Name:	Make a new piece
Description and purpose:	A composer wants to make a new piece of music from scratch, using different instruments, percussion and pitches overlaying each other in different tracks. He does this by using the musical tracker to pick an instrument via the instrument button, and then pressing the buttons until he finds a key he is satisfied with. He then proceeds to enter this into his track.

Use case nr:	02
Name:	Extract the piece and save it
Description and purpose:	A composer wants to extract his new piece in a format which he will be able to play on his phone. He does this by saving the music he has composed and the extracting it to 3rd party software, which will then convert to mp3.

Use case nr:	03
Name:	Use and modify sample
Purpose:	A composer wants to use a sample from an instrument and being able to choose a certain a pitch of that instrument.

Express three non-functional requirements for a music tracker software system.

- A user must be able to use samples from different instruments.
- A user must be able to export the music.

- A user must be able to change the pitch of the samples of the instruments.

Exercise 6

Use case nr:	01
Name:	Manage payment for customers.
Description and purpose:	A customer in the ITU Canteen wants to buy food from the canteen via the payment system. There are several kinds of foods to buy, and the payment system should account for all those choices. The conclusion should be a succesful payment.
Initializing actor:	Customer.
Initializing event:	A customer wishes to pay for an item in the ITU Canteen.
Starting conditions:	Nothing, any customer should be able to use the system.
Progression:	The customer selects an item on display in the canteen. The customer confirms this selection in the menu provided by the system. The system calculates the final price.
End result:	The customer has succesfully bought an item from the Canteen.
Ending state:	The transaction modifies the accounts of both the customer and the canteen.
Remarks:	Most choices should be available from the menu provided by the system. However, some items require weighing to determine the final price. This choice should be implemented as well.

Three requirements:

Functional requirement: The payment system must include a weight and an algorithm that can determine the cost of weighed food.

Nonfunctional requirement: (Product Requirement - Dependability) The payment system should be implemented robustly, with minimum risk of data loss in case of errors or a system failure.

Nonfunctional requirement: (Product Requirement - Efficiency) Payment should take no more than two seconds (to accommodate long queues).