Assignment 1

Joanna Laursen, Kasper F. Kristensen , Mille M. Z. Loo

September 13, 2021

1 Generics

Compare the following two methods:

```
int GreaterCount<T, U>(IEnumerable<T> items, T x)
where T : IComparable<T>

int GreaterCount<T, U>(IEnumerable<T> items, T x)
where T : U
where U : IComparable<U>
```

Both methods returns the amount of elements in `items` which are *greater than* `x`.

Explain in your own words what the type constraints mean for both methods.

In the first method, the type constraint means that to compare anything with T, T must in fact be comparable. `Where` means that the method can only be applied to types T, that fulfil the given restriction. Meaning that T must implement IComparable.

In the second method, T must be of type U and U must implement IComparable. T's are still what is compared, but T is now inherited from U. For instance, if U is a Car and T is a Ferrari, Ferraris must be Cars.

2 Iterators

For our solution, please refer to https://github.com/jsklitu/Ass01.git

3 Regular Expressions

For our solution, please refer to https://github.com/jsklitu/Ass01.git

4 Software Engineering

4.1 Exercise 1

What is meant by "knowledge acquisition is not sequential"? Provide a concrete example of knowledge acquisition that illustrates this.

What is meant by "*knowledge acquisition is not sequential*", is that gaining knowledge is non-linear. Instead you have to acknowledge that times are changing and so is what we perceive as factual. This is prevalent throughout the scientific field since it is impossible to prove that a given fact is true, but possible to disprove it. Therefore, no matter how much we understand something, with time we might find something that nullifies our understanding.

An example could be writing software for a submarine that is driven by a diesel engine. The software is almost "done" and mentally, you're ready to finish the project, but you learn that they updated the engine from a diesel engine to a nuclear engine. This completely nullifies your understanding of the engine since the two have a similar purpose but don't function the same, and, therefore, you would need to restart the entire process.

Simplified, what you knew as fact might be proven wrong, and you will have to relearn said fact - and will likely do so again in the future.

4.2 Exercise 2

Specify which of the following decisions were made during requirements or system design:

- "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."

Made during requirements (what the system must do):
"Provides the traveller with on-line help"

Made during System Design (how to obtain functionality):
"Composed of a user interface [...] with the central computer"
"Will use PowerPC processor chips"

4.3 Exercise 3 - we would very much like to have feedback on this exercise, as we found it very difficult.

In the following description, explain when the term account is used as an application domain concept and when as a solution domain concept:

> "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."

An *application domain* is the domain in which the solution is applied. To create a good and effective solution one must understand the *application domain*. An example would be: if you were to build a software for a boat to use i.e. the *solution domain*, you must understand how the boat functions i.e. the *application domain*.

Application:
[...] managing bank accounts [...]
[...] provide access to the accounts [...]
We would argue that this *account* is the application domain, since it describes the actual physical bank accounts that the solution must be developed for, whereas "online system" refers to the solution domain.

Solution:
[...] accounts are made available [...]
[...] the accounts show the amounts [...]

The latter two occurrences of the term *account* are referring to the idea of accessing the mobile bank account via. the software. Since they refer to the software representation of an account rather than it's real life counterpart these two instances use the term *account* as a solution domain concept.

4.4 Exercise 4

A passenger aircraft is composed of several millions of individual parts and requires thousands of persons to assemble. A four-lane highway bridge is another example of complexity. The first version of Word for Windows, a word processor released by Microsoft in November 1989, required 55 person-years, resulted into 249,000 lines of source code, and was delivered 4 years late. Aircraft and highway bridges are usually delivered on time and below budget, whereas software is often not. Discuss what are, in your opinion, the differences between developing an aircraft, a bridge, and a word processor, which would cause this situation.

A bridge is made with a single function - allowing people, vehicles, etc. to cross it. Specialist bridge-builders are hired (engineers, architects, budget-managers etc), and a specific plan is executed. Everything is planned out beforehand in blueprints, with a certainty that it will work.

Bridges and air planes are well known and well described in how to construct them. There are very precise standards, as well as physics you can look up and know for certain, based on many years of experience in the field. Therefore, they are, in general, very technically regulated areas. On the other hand, the field of software is constantly and hurriedly developed.

Furthermore, people need a certain education to be allowed to build a bridge or an air plane. "Anyone" can call themselves software developers.

With software and air planes, understanding all elements in complete detail is almost impossible as both are extremely complex. However, if you want to build a bridge or air plane faster, you can hire more people to build it because it is so well defined. With software, which is at the beginning less defined, adding more people quickly becomes too many cooks.

Furthermore, software is created by software specialists, however, these specialists might be required to develop software for a field they have no expertise in. Therefore, they must first become specialists in that field as well - in this case word processor specialists.

Another consideration one should think of, is that a faulty bridge or plane could potentially result in casualties, whereas a faulty text editor can at most lead to irritation. Therefore, planes and bridges are tested thoroughly.

4.5 Exercise 5

Specify which of these statements are functional requirements and which are nonfunctional requirements:

- "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."

Functional requirements are what defines what the system must and must not do. Meaning, they specify what behaviour the system has.

Non-functional requirements, on the other hand, define a set of constraints on the system that is not related directly to a function of the system.

Functional:
Must enable the traveller to buy weekly passes
Must provide a phone number

Non-functional:
Must be easy to use
Must be written in Java
Must always be available

4.6 Exercise 6

What is the purpose of modeling?

The *modeling activity* is at its' core knowing the *application domain* before creating a *solution domain*. *Object-Oriented methods* combine the two modeling activities into one. This can be done with tools such as *CRC cards* and the *noun-verb approach*. In both of these instances the application domain is modeled as sets of objects and relationships representing real-world constructs. This allows the programmer to question the model and the decisions made whilst creating the model before the programming starts.