Project Software Engineering:
CO2-friendly printing system

1Ba INF 2023-2024

1 Introduction

The University of Antwerp currently owns and manages an extensive systems of printers and scanners. These devices can be found in the libraries, study areas, and hallways of the various campuses of the University, and are accessible both to students and employees through the myPrint system.

Figure 1: Printer devices in the library.

With an eye on a climate-friendly and sustainable future, the University of Antwerp has developed an ambitious plan to become climate-neutral by 2030. As part of this initiative, the University wishes to get a clear view on the CO2-emissions of the printing devices across the different campuses.

In order realise this, the lab of Serge Demeyer, part of the AnSyMo research group, has been tasked with developing new printing software that is user-friendly to both students and employees while at the same time providing a good overview of all CO2-emissions of the system.

Most importantly, this system needs to be **state-of-the-art**. This means that all best practices of Software Engineering and C++ programming have to be applied. This includes: good planning, excellent communication between the customer, developers, and management, extensive testing, object-oriented programming, as well as good documentation and clearly defined contracts.

2 Use case format

Important: It is possible that some things in this specification are unclear. In the field of software development, this is the norm. If you are not sure about the meaning of something, or if you suspect it is an error, you are expected to do the following:

1. **Reflect**: What is missing? Why do you think it is an error? Why is it not clear?
2. Try to come up with some **alternatives**: what are the different ways to interpret the specification? In what different ways can it be implemented?
3. Present your question, together with the different interpretations and alternatives you propose, to one of the assistants.

The requirement specification consists of so-called use cases. Each use case describes a specific functionality of the system. Throughout the project, you will have to implement multiple use cases. It might also be possible that some use cases are modified or revised in later versions of the specification. A use case may contain the following components:

- **Use case number and title:**
  Every use cases has a title and a number. This is used to refer to that use case.
- **Priority:**
  This specification will contain too many use cases to implement during the semester. This is why the use cases come with priorities. Possible priorities are: REQUIRED (implementing these is obligatory), IMPORTANT (not obligatory, but highly preferred), USEFUL (interesting, but may be left out).
- **Goal:**
  A small description of the use case: what it is supposed to do, why is it there.
- **Precondition:**
  What needs to be present in the system before the use case functionality can be used. This will tell you exactly under what conditions the use case should work.
- **Postcondition:**
  Description of what the use case will have accomplished in case of the "happy day scenario". That is, everything went well and no errors occurred.
- **Steps:**
  A sequential description of the steps that the use case will take as part of the "happy day scenario". The steps are numbered, and may contain control flow (e.g., WHILE, IF, ELSE, …).
- **Exceptions:**
  A list of possible problems that might occur, and how they should be handled. Each exception will specify the following:
  (a) the number of the step where the problem can occur,
  (b) a condition that specifies when the problem has occurred,
  (c) a short description of how the exception should be handled.
- **Example:**
  An example of the input or output of the use case.

Some use cases will be extensions of other use cases. In order to implement them, you will have to adjust functionalities that you have implemented earlier. In that case, the following things will be specified:

- **Extension:**
  A reference to the original use case.
- **Steps:**
  A description of how the steps of the original use case should be modified.

3 Overview

Use-Case	Priority
1: Input	
`Use Case 1.1: Reading printers and jobs`	REQUIRED
2: Output	
`Use Case 2.1: Simple output`	REQUIRED
3: Simulation	
`Use Case 3.1: Manual processing`	REQUIRED
`Use Case 3.2: Automated processing`	IMPORTANT

Use Case 1.1: Reading printers and jobs

Priority: REQUIRED

Goal: The printing system should be read from an XML file. The file reader should be able to handle all the information in the file, read it correctly, check for errors, and inform the user of any errors in the input.

Precondition: An XML file in ASCII format needs to be presented to the system. The XML file needs to contain information about the devices and jobs. See appendix A for a detailed description of the input format.

Postcondition: The system contains a printing system with printer devices and printing jobs. All devices and jobs in the system need to conform to the information specified in the XML file.

Steps:

1. Open input file
2. WHILE the file is not yet fully read:
   2.1. Recognize element (DEVICE, JOB)
   2.2. Read information about the element (see appendix A)
   2.3. IF the information is correct
      2.3.1. THEN add element to the printing system
      2.3.2. ELSE print error message; go to next element in the input file.
3. Verify the consistency of the printing system (see appendix A).
4. Close input file

Exceptions:

Step 2.1. [Unrecognizable element] Print error message. Go to step 2 and begin reading the next element.
Step 2.2. [Invalid information] Print error message. Go to step 2 and begin reading the next element.
Step 3 [Inconsistent printing system] Print error message. Go to step 4.

Example: Example printing system with one printer, and two printing jobs.

```
<SYSTEM>
    <DEVICE>
        <name>Office_Printer5</name>
        <emissions>3</emission>
        <speed>40</speed>
    </DEVICE>
    <JOB>
        <jobNumber>89751</jobNumber>
        <pageCount>2</pageCount>
        <userName>SergeDemeyer</userName>
    </JOB>
    <JOB>
        <jobNumber>2189</jobNumber>
        <pageCount>3</pageCount>
        <userName>anonymous_user</userName>
    </JOB>
</SYSTEM>
```

Use Case 2.1: Simple output

Priority: REQUIRED

Goal: The system should print an easy-to-read status report.

Precondition: A valid printing system is loaded.

Postcondition: The system has created a text file (ASCII) that contains all the necessary information about the system.

Steps:

1. Create output file.
2. WHILE printers left
   2.1. Print information about the printer to the output file.
3. WHILE jobs left
   3.1. Print information about the job to the output file.
4. Close output file.

Example:

```
NEW-Printer (CO2: 5g/page):
    * Current:
        [#789|KasperEngelen]
    * Queue:
        [#423|Peter Selie]
        [#98712|New-user]
```

Use Case 3.1: Manual processing

Priority: REQUIRED

Goal: The printing system should be able to take a job and have to processed by a printer.

Precondition: A valid printing system is loaded.

Postcondition: All the pages of the job have been printed.

Steps:

1. WHILE pages left:
   1.1. Print page.
2. Print a message to the screen.

Example: If a job with 3 pages and job number 13989 was submitted by "John Doe" to the printer "Library Printer 5", the following message would be printed after the job was finished:

```
Printer "Library Printer 5" finished job:
    Number: 13989
    Submitted by "John Doe"
    3 pages
```

Use Case 3.2: Automated processing

Priority: IMPORTANT

Goal: The system should be able to automatically run multiple jobs one after the other.

Precondition: A valid printing system is loaded.

Postcondition: All jobs are finished.

Steps:

1. WHILE unfinished jobs left
   1.1. Pick job
   1.2. Perform `Use Case 3.1: Manual processing`

A Input format

The input format for the printing system is made in such a way that new elements and attributes easily be added. All input files will be specified as XML files.

```
File = "<SYSTEM>" ElementList "</SYSTEM>"
ElementList = Element { Element ... }
Element = "<" ElementType ">" AttributeList "</" ElementType ">"
ElementType = (see table)
AttributeList = Attribute { Attribute ... }
Attribute = "<" AttributeName ">" AttributeValue "</" AttributeName ">"
AttributeName = (see table)
AttributeValue = Integer | String | Float
Integer = Digit { Digit ... }
Float = Digit { Digit ... } "." Digit { Digit ... }
Digit = "0" | ... | "9"
String = Character { Character ... }
Character = "a" | ... | "z" | "A" | ... | "Z"
```

There are different types of elements that can be added to the input file. Each type of element has some required attributes, which have to be specified, as well as optional attributes, that the user may or may not specify.

Element type	**Attributes (required)**	**Attributes (optional)**
`DEVICE`	`name`, `emissions`, `speed`	N/A
`JOB`	`jobNumber`, `pageCount`, `userName`	N/A

Each attribute has a specific type of value that can be assigned to it:

Attribute	**Type of value**
`name`, `userName`	String
`jobNumber`, `emissions`, `speed`, `pageCount`	Integer

The `emissions`, `speed`, and `cost` are specified in the following units:

Attribute	**Unit**
`emissions`	gram CO2 per page
`speed`	pages per minute

Note that the opening tag has to correspond to the closing tag. For more information, see for example the Wikipedia page of the XML format.

There is the additional requirement that every input file, has to be **consistent**:

- The attributes `emissions`, `pageCount`, `speed`, and `jobNumber` must not be negative.
- Every `JOB` element needs to have a unique `jobNumber`.

Any and all input files that do not conform to the above requirements are not valid.