Term Paper Assessment

Professional Software Development
UC2PSD051

Benjamin Hagen

November 2022

Contents

1 Introduction

In this term paper i will answer the following *six* questions.

* Question 1: LaTeX
* Question 2: Requirements Engineering and System Design
* Question 3: Software Testing Methodologies
* Question 4: Software Development Life Cycle
* Question 5: Version Control
* Question 6: Working with Git

1.1 Question 1: LaTeX

Question 1 is to write all the answers to this term paper in LaTeX. The assessment should use the article documentClass, and should be appropriately structured. To complete this assignment, the following LaTeX functions must be used:

* Sections
* **Bold**,*Italics*, and Underlining.
* Figures
* References
* Table of Contents

2 Question 2: Requirements Engineering and System Design

bookTrainTicket scenario.

2.1 Write four functional requirements for this system.

1 The system shall allow the client to input their name, passport number, and contact information.

2 The system shall allow the client to select the travel dates, departure and arrival train stations, and seating preference.

3 The system shall allow the client to select additional facilities such as meal, tea or coffee, and luggage inclusion.

4 The system shall send a booking confirmation email to the client upon successful payment.

2.2 Write one non-functional requirement

Product Requirements The train ticket booking system shall be easy to use.

Organizational Requirements The train ticket booking system shall be able to integrate with the organization's existing systems.

External Requirements The system shall be compatible with the company's existing booking system.

2.3 Design the client class as a parent class for both persons and organizations

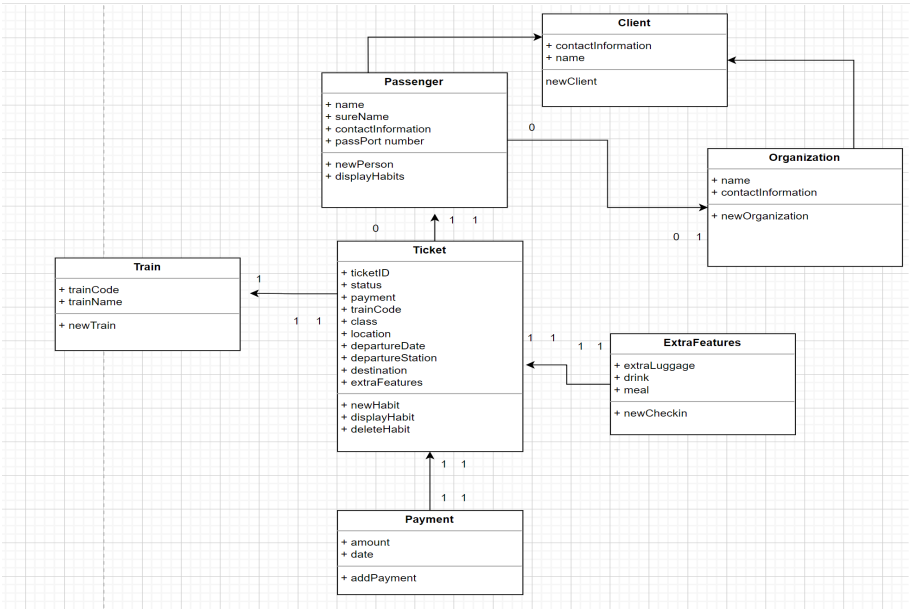

Figure 1: Client class as a parent class for both persons and organizations

2.4 One-to-one relationships and one-to-many relationships with UML

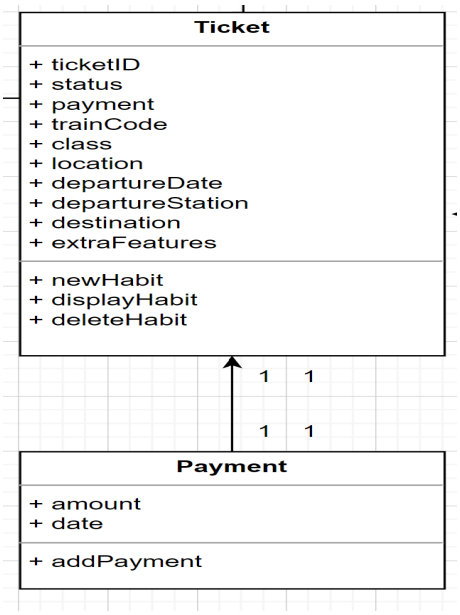

Figure 2: One-to-One

One-to-One: One-to-One: This picture contains one to one relationship between Ticket and Payment, as each ticket can have one payment associated with it, and one payment can be associated with one ticket.

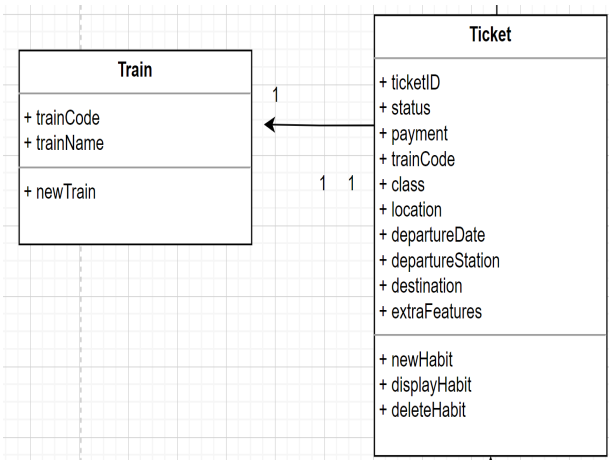

Figure 3: One-to-many

One-to-many: Train and Ticket can have One to many relationships as Train can have many tickets associated with it while one ticket can have only one train associated with it.

2.5 Draw the composite use case diagram from the client actor with five interactions that can be performed

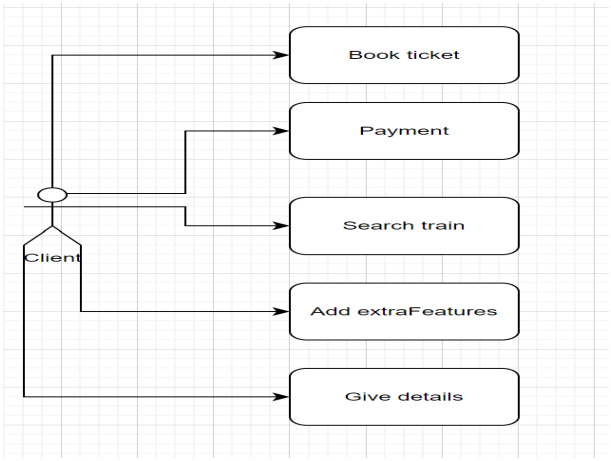

Figure 4: Use case diagram

2.6 UML activity diagram for the bookTrainTicket system

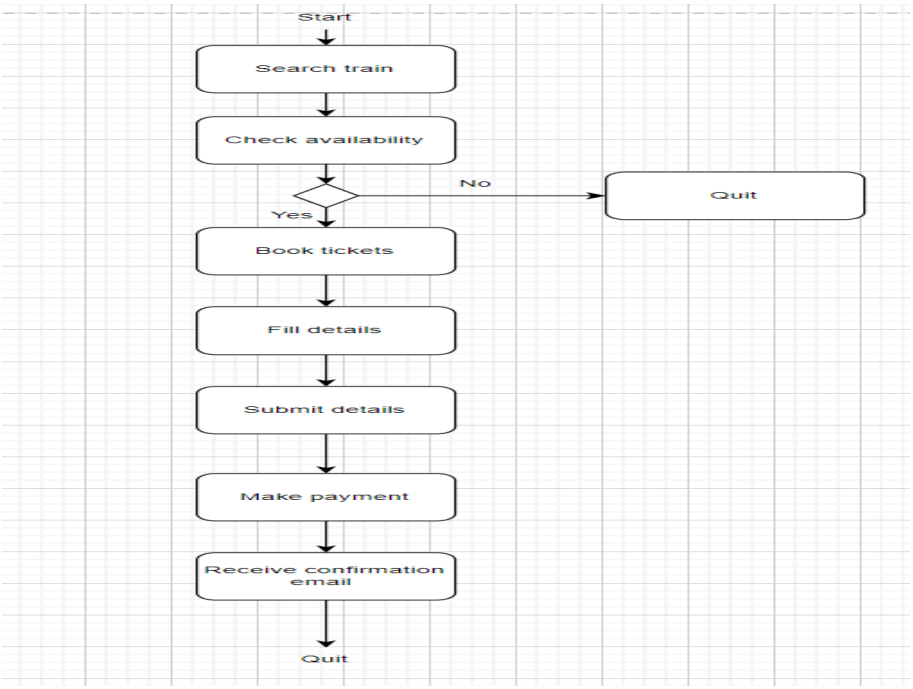

Figure 5: bookTrainTicket system

2.7 Draw the sequence diagram which depicts the client logging into the booking system portal. Write a short summary.

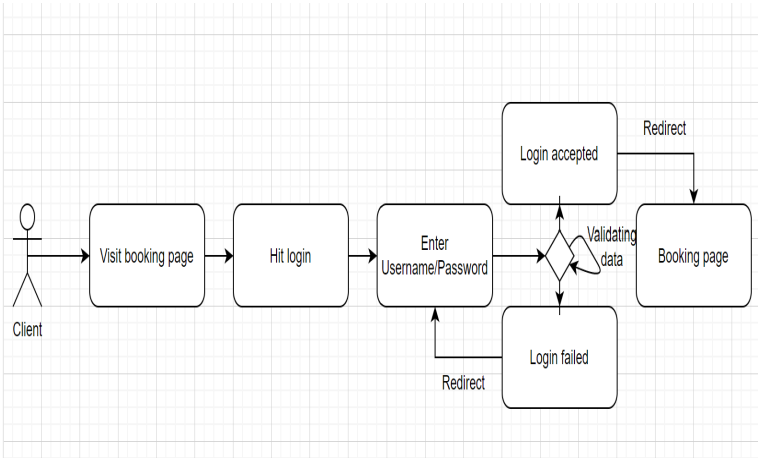

Figure 6: Sequence diagram

The client visits the booking page and hits the “login” button. Here, the client needs to enter the username and password. The username and password will be validated and if the data is correct, it will be redirected to the “Booking page”. However, if the username and password is incorrect, the client will be redirected back to the login page.

3 Question 3: Software Testing Methodologies

3.1 Scenarios

Design 4 scenarios in order to perform "Scenario Testing" of the system from Question 2. For each of your test scenarios create at least two Test Cases

3.1.1 Scenario 1 Test Case 1

Objective: To test the functionality of the train ticket booking system from the perspective of the client.

Pre-requisites:

* The client has all the necessary information about the passenger.
* The client has all the necessary information about the travel.

Test data:

* Client information: name, passport number, email address, contact number.
* Passenger information: name, passport number.
* Travel information: departure date and time, departure train station, arrival train station.

Test steps:

* The client enters the train ticket booking system.
* The client enters the necessary information about the passenger.
* The client enters the necessary information about the travel.
* The client chooses the seating preference.
* The client chooses the additional facilities.
* The client makes the payment.

Expected results:

* The booking confirmation email is delivered to the client.
* The receipt for the booking is delivered to the client.

3.1.2 Scenario 1 Test Case 2

Objective: To test the functionality of the train ticket booking system from the perspective of the passenger.

Pre-requisites:

* The passenger has all the necessary information about the travel.

Test data:

* Client information: name, passport number, email address, contact number.
* Passenger information: name, passport number.
* Travel information: departure date and time, departure train station, arrival train station.

Test steps:

* The passenger enters the train ticket booking system.
* The passenger enters the necessary information about the travel.
* The passenger chooses the seating preference.
* The passenger chooses the additional facilities.
* The passenger makes the payment.

Expected results:

* The booking confirmation email is delivered to the passenger.

3.1.3 Scenario 2 Test Case 1

Objective: To test the functionality of the train ticket booking system in the case of an unsuccessful payment.

Pre-requisites:

* The client has all the necessary information about the passenger.
* The client has all the necessary information about the travel.

Test data:

* Client information: name, passport number, email address, contact number.
* Passenger information: name, passport number.
* Travel information: departure date and time, departure train station, arrival train station.

Test steps:

* The client enters the train ticket booking system.
* The client enters the necessary information about the passenger.
* The client enters the necessary information about the travel.
* The client chooses the seating preference.
* The client chooses the additional facilities.
* The client makes the payment.

Expected results:

* The booking confirmation email is not delivered to the client.
* The receipt for the booking is not delivered to the client.

3.1.4 Scenario 2 Test Case 2

Objective: To test the functionality of the train ticket booking system in the case of a successful payment.

Pre-requisites:

* The client has all the necessary information about the passenger.
* The client has all the necessary information about the travel.

Test data:

* Client information: name, passport number, email address, contact number.
* Passenger information: name, passport number.
* Travel information: departure date and time, departure train station, arrival train station.

Test steps:

* The client enters the train ticket booking system.
* The client enters the necessary information about the passenger.
* The client enters the necessary information about the travel.
* The client chooses the seating preference.
* The client chooses the additional facilities.
* The client makes the payment.

Expected results:

* The booking confirmation email is delivered to the client.
* The receipt for the booking is delivered to the client.
* Client information: name, passport number, email address, contact number.

3.1.5 Scenario 3 Test Case 1

Objectives: To test whether the system is able to successfully book a train ticket when all the required information is provided

Pre-requisites:

* The user should have all the required information, such as the name and surname of the passenger, passport number, travel details, etc.

Test data:

* Name: John
* Surname: Smith
* Travel details: Train date and time: 01/01/2020, 10:00 AM
* Arrival train station: ABC
* Seating preference: Class: First Class
* Location: Near window
* Additional facilities: Meal included: Yes
* Additional facilities: Tea/coffee included: Yes
* Additional luggage included: Yes

Test steps:

1 The user opens the train ticket booking system and enters the required information.

2 The user clicks on the "Book" button.

3 The system processes the booking request.

4 The user receives a booking confirmation email.

Expected results:

* The system should successfully book the train ticket and send a confirmation email to the user.

3.1.6 Scenario 3 Test Case 2

Objectives: To test whether the system is able to handle incomplete information and provide an error message to the user when a user tries to book a train ticket with incomplete information.

Pre-requisites:

* The user should have incomplete information, such as the name and surname of the passenger, passport number, travel details, etc.

Test data:

* Name: John
* Surname: Smith
* Travel details: Train date and time: 01/01/2020, 10:00 AM
* Arrival train station: ABC
* Seating preference: Class: First Class
* Location: Near window
* Additional facilities: Meal included: Yes
* Additional facilities: Tea/coffee included: Yes
* Additional luggage included: Yes

Test steps:

1 The user opens the train ticket booking system and enters the required information.

2 The user clicks on the "Book" button.

3 The system processes the booking request.

4 The user receives an error message.

Expected results:

* The system should not be able to book the train ticket and should send an error message to the user.

3.1.7 Scenario 4 Test Case 1

Objectives: To test if an invalid booking will be made when an invalid credit card is used

Pre-requisites:

* An invalid credit card is required.

* Train ticket is available.

Test data:

* Use an invalid credit card.

* Check train ticket availability.

Test steps:

1 Enter credit card information.

2 Enter passenger information.

3 Enter travel information.

4 Enter seating preference.

5 Enter additional facilities.

6 Click on “Book Train Ticket” button.

Expected results:

* Error message is displayed indicating that the credit card is invalid.

* Booking is not confirmed.

3.1.8 Scenario 4 Test Case 2

Objectives: To test if an invalid booking will be made when a valid credit card is used

Pre-requisites:

* A valid credit card is required.
* Train ticket is available.

Test data:

* Use a valid credit card.
* Check train ticket availability.

Test steps:

1 Enter credit card information.

2 Enter passenger information.

3 Enter travel information.

4 Enter seating preference.

5 Enter additional facilities.

6 Click on “Book Train Ticket” button.

Expected results:

* The system should successfully book the train ticket and send a confirmation email to the user.

3.2 Provide two validation tests and two defective tests that can be performed on Air Conditioner (AC). Provide assumptions that you have made for this test.

Assumptions:

1 The AC is plugged in and has power.

2 The AC is set to the correct temperature/mode.

Validation tests:

1 Check if the AC turns on and off as expected.

2 Check if the AC blows air as expected.

Defective tests:

1 Check if the AC blows air when it is not supposed to.

2 Check if the AC does not blow air when it is supposed to.

3.3 Unit testing, component testing and system testing

Unit testing, component testing, and system testing are all methods of testing software to ensure that it is functioning correctly. Unit testing focuses on individual units of code, component testing focuses on groups of units of code, and system testing focuses on the entire system. There are advantages and disadvantages to each level of testing. Unit testing is the most granular level of testing and has the advantage of being able to pinpoint exactly where a problem lies if a test fails. Unit tests, however, can be time-consuming to create and maintain, and they may not always provide an accurate representation of the behavior of the system. [SmartBear,] [geeksforgeeks, b]

Component testing is less granular than unit testing but more granular than system testing. Since component tests test groups of code units rather than individual units, they can provide a more realistic picture of how the system will perform. When the components under test are constantly changing, component tests can be difficult to create and maintain. System testing is the highest level of testing and has the advantage of testing the entire system. The creation and maintenance of system tests can, however, be time-consuming and expensive. [Black,] [Sommerville, 2016]

3.3.1 Example

As an example, if we create a browser, we will test each individual unit of the system, such as the "bookmarks" unit, and determine if it behaves as intended. In component testing, the complete component is tested after merging all the units of that component to evaluate the correct functioning of that component.

Let's assume that we integrate all the units and classes of the "save a bookmark" component and test it. In system testing, the complete system is put under observation after integrating all the components of the system. We will check that the Browser is working correctly after integration and on different OS devices with different screen sizes, etc.

4 Question 4: Software Development Life Cycle

a). Here, I will discuss the pros and cons of using waterfall method versus using an agile method with respect to: Managing Project Risk, Project Tracking, Software Architecture Design

4.1 Managing Project Risk

There is no single answer to this question, as it depends on the specific project and organization. One of the advantages of using the waterfall method is that it is a well-defined and structured approach that is easier to manage and control. The use of this method may also facilitate the communication of progress and milestones to stakeholders. Despite this, some of the disadvantages of the waterfall method include its inflexibility and inability to adapt well to changes. This can lead to an increase in risks. This approach can also make it difficult to accurately estimate the time and resources required for a project.[SDLC, b]

An advantage of using agile methods is that they are more flexible and adaptable to changes, which can reduce risks. Additionally, it can enhance communication and transparency among team members and stakeholders. However, some disadvantages of using the agile method include that it can be more difficult to manage and control, as it is less structured. Furthermore, it may be difficult to estimate the amount of time and resources required for a project with this approach.[SDLC, a]

4.2 Project Tracking

The appropriateness of using either the waterfall method or the agile method depends on the specific project being undertaken. However, there are some general considerations that can be made. The waterfall method is more traditional and typically features more linear and rigid project management. As a result, planning and execution can be more straightforward, as well as tracking progress can be easier. However, it can also make it more difficult to adapt to changes during the project, resulting in a less flexible product.

The agile method is more flexible and adaptive, features more frequent check-ins and iterations, and generally relies heavily on team input and collaboration. This can make it more difficult to plan and execute, as well as monitor progress. In addition, it can also result in a more flexible final product that is better able to adapt to changes as the project progresses.

4.3 Software Architecture Design

Regarding the software architecture design process, there are pros and cons to using the waterfall method and the agile method. The waterfall method has the advantage of being very linear and straightforward. This can make it easier to understand and follow for those who are inexperienced with the software design process. It is also beneficial to apply the waterfall method in order to ensure that all phases of the software design process are completed in a timely and efficient manner. One con of using the waterfall method is that it can be inflexible and does not allow for much change or adaptation once the design process has begun. As a result, problems may arise if the requirements of the project change during the design phase. Another con of using the waterfall method is that it can be difficult to track and manage progress. This is because each phase of the design process must be completed before the next can begin.

When it comes to software architecture design, there are a few pros and cons to using the agile method. One of the advantages of agile methods is that they can reduce the overall risk of the project. This is because smaller and more frequent iterations allow for more feedback and course corrections. Additionally, agile methods can improve communication and collaboration among team members, since everyone is working together on a more frequent basis. A disadvantage of using agile methods includes the fact that it can be difficult to maintain a consistent architecture when changes are being made constantly, and that it can be challenging to effectively communicate the architecture to all team members when it is constantly evolving. [McCormick,] [Sommerville, 2016]

4.4 Spiral Development Life Cycle - Project Idea

The spiral development life cycle is ideal for projects where there is a lot of uncertainty, and where it is difficult to determine all requirements at the beginning. Additionally, it is an appropriate choice when the project is complex and there is a great deal of risk involved. Here are some examples of projects where the spiral development life cycle would be appropriate:

- Development of a new product
- Developing a new software application

The design of a new product

This type of project is well suited to the spiral development life cycle, as it allows for constant feedback and iteration. In this way, the project can be continuously improved as new information is gathered. As a result, risks can also be identified and addressed early in the project. There are a few reasons why the spiral development life cycle would work better for this project in comparison to the validation and verification life cycle method. In general, spiral development is better suited to projects with a high degree of uncertainty and risk. Since this project is focused on developing a new product, there are bound to be a number of unknowns and risks involved. Using the spiral development life cycle, the project team will be able to manage these risks more effectively and ensure that the project remains on schedule. The spiral development life cycle would also be more appropriate for this project due to its flexibility. As the validation and verification life cycle is very linear, it can be difficult to make changes along the way. With the spiral development life cycle, the project team can make adjustments as needed, which will ultimately lead to a better end product. [geeksforgeeks, c]

5 Question 5: Version Control

5.1 Four reasons for using a version control system

Four of the most important reasons to use a Version Control System (VCS) are as follows:

* VCS allows developers to collaborate on the same codebase. For larger projects with multiple developers, this is especially important.
* VCS makes it easy to track changes to your codebase. This is important for debugging and for understanding how your codebase evolves over time.
* VCS makes it easy to roll back changes. In the event that you make a mistake or if you wish to experiment with different changes to your code, this is very important.
* VCS allows you to create branches. This is important for developing new features or for working on experimental changes.

[geeksforgeeks, d]

5.2 Three advantages of distributed version control systems over centralized version control systems.

Distributed version control systems have the following advantages over centralized version control systems:

1 Using a distributed version control system, every developer has a local copy of the repository, which includes all previous versions of every file. In this way, past versions of files can be accessed much more quickly, as well as branching and merging can be accomplished more quickly.

2 In a distributed system, there is no single point of failure. In the event that the central server goes down, developers will still be able to collaborate and work using their local copies of the repository.

3 Distributed version control systems are generally more flexible than centralized ones. Developers can, for example, work offline or use a different repository for each project.

5.2.1 Example of a centralized control system and a distributed version control system.

A centralized control system would be a system where there is one central server that controls all the resources and information. Microsoft Team Foundation Server are example of a centralized version control systems. Distributed version control systems consist of multiple servers that control different parts of resources and information. Git is a distributed version control system. [geeksforgeeks, a]

5.3 What is branching?

Branching is the process of creating a new branch, or timeline of development, from an existing one. The branching process enables developers to create different versions of a software application or program, each with its own code base. This can be useful when developing new features or fixing bugs in existing code. Also, branching allows developers to experiment with different code bases without affecting the main development branch or the main code base. Branching can be used to manage code complexity and improve software quality. When using branching, it is important to use it wisely, as it can also lead to code duplication and increased maintenance costs.[Schiestl,]

5.3.1 Three uses of branching with examples

1 When a developer wants to add a new feature to a software project, they would create a new branch of the main development branch. As a result, they can work on their new feature without affecting the main development branch.

2 When a developer wants to fix a bug in a software project, they would create a new branch of the main development branch. As a result, they can work on fixing the bug without affecting the main development branch.

3 When a developer wants to experiment with a new idea for a software project, they would create a new branch of the main development branch. This allows them to work on their new idea without affecting the main development branch.

6 Question 6: Working with Git

6a) Here, the task was to *"design a method in a class which generates random numbers. The method accepts the following variables as input (int amount, int minRange, int maxRange)"*.

```python
import random

class Class:

    #Generate a given amount of random numbers between a range of numbers.
    def random_number_generator(amount, minRange, maxRange):

        assert isinstance(amount, int) #Precondition 1 - amount shold be an integer.
        assert minRange < maxRange #Precondition 1 - minRange should be less than maxRange.

        random_numbers = []
        for i in range(amount):
            random_numbers.append(random.randint(minRange, maxRange))

        assert len(random_numbers) == amount #Postcondition 1 - there should only be an expected amount of random numbers.
        assert min(random_numbers) >= minRange and max(random_numbers) <= maxRange

        return random_numbers

x = Class.random_number_generator(10, 1, 100)
print (x)
```

✓

```
[83, 77, 71, 23, 48, 92, 10, 67, 35, 4]
```

Figure 7: First code

Then it was required to host my results of Question 6a in my local repository.

```
benja@username MINGW64 ~/OneDrive/Skrivebord/random (master)
$ cd random

benja@username MINGW64 ~/OneDrive/Skrivebord/random/random (main)
$ git init
Reinitialized existing Git repository in C:/Users/benja/OneDrive/Skrivebord/rand
om/random/.git/

benja@username MINGW64 ~/OneDrive/Skrivebord/random/random (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        First.ipynb

nothing added to commit but untracked files present (use "git add" to track)
```

```
benja@username MINGW64 ~/OneDrive/Skrivebord/random/random (main)
$ git add .
warning: in the working copy of 'First.ipynb', LF will be replaced by CRLF the n
ext time Git touches it

benja@username MINGW64 ~/OneDrive/Skrivebord/random/random (main)
$ git commit -m "Adding"
[main (root-commit) ab40e40] Adding
 1 file changed, 71 insertions(+)
 create mode 100644 First.ipynb

benja@username MINGW64 ~/OneDrive/Skrivebord/random/random (main)
$ git status
On branch main
Your branch is based on 'origin/main', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

nothing to commit, working tree clean
```

```
benja@username MINGW64 ~/OneDrive/Skrivebord/random/random (main)
$ git remote add origin git@github.com:benjaahagen/random.git
error: remote origin already exists.

benja@username MINGW64 ~/OneDrive/Skrivebord/random/random (main)
$ git push
Enter passphrase for key '/c/Users/benja/.ssh/id_ed25519':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 899 bytes | 899.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:benjaahagen/random.git
 * [new branch]      main -> main
```

6.1 Updating

Update the project 3 times by adding appropriate commit message and with some minor changes in the script each time.

```
benja@username MINGW64 ~/OneDrive/Skrivebord/random/random (main)
$ git commit -m "Added precondition 2"
[main d80b69d] Added precondition 2
 1 file changed, 8 insertions(+), 2 deletions(-)

benja@username MINGW64 ~/OneDrive/Skrivebord/random/random (main)
$ git push
Enter passphrase for key '/c/Users/benja/.ssh/id_ed25519':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 482 bytes | 482.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:benjaahagen/random.git
   ab40e40..d80b69d  main -> main
```

Figure 8: Added precondition 2

```python
import random

class Class:

    #Generate a given amount of random numbers between a range of numbers.
    def random_number_generator(amount, minRange, maxRange):

        assert isinstance(amount, int) #Precondition 1 - amount shold be an integer.
        assert minRange >= 0 #Precondition 2 - consider only positive numbers.
        assert minRange < maxRange #Precondition 3 - minRange should be less than maxRange.

        random_numbers = []
        for i in range(amount):
            random_numbers.append(random.randint(minRange, maxRange))

        assert len(random_numbers) == amount #Postcondition 1 - there should only be an expected amount of random numbers.
        assert min(random_numbers) >= minRange and max(random_numbers) <= maxRange

        return random_numbers

x = Class.random_number_generator(10, 1, 100)
print (x)
```

✓ 0.4s

```
[54, 35, 85, 68, 57, 45, 34, 7, 4, 34]
```

Figure 9: Second code

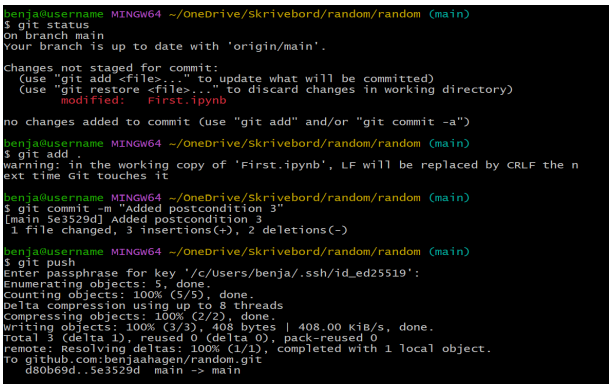

Figure 10: Added post-condition 3

```python
import random

class Class:

    #Generate a given amount of random numbers between a range of numbers.
    def random_number_generator(amount, minRange, maxRange):

        assert isinstance(amount, int) #Precondition 1 - amount shold be an integer.
        assert minRange >= 0 #Precondition 2 - consider only positive numbers.
        assert minRange < maxRange #Precondition 3 - minRange should be less than maxRange.

        random_numbers = []
        for i in range(amount):
            random_numbers.append(random.randint(minRange, maxRange))

        assert len(random_numbers) == amount #Postcondition 1 - there should only be an expected amount of random numbers.
        assert min(random_numbers) >= minRange #Postcondition 2 - all the random numbers should be greater than or equal to minRange.
        assert max(random_numbers) <= maxRange #Postcondition 3 - all the random numbers should be less than or equal to maxRange.

        return random_numbers

x = Class.random_number_generator(10, 1, 100)
print (x)
```

✓ 0.5s

```
[6, 71, 66, 92, 19, 73, 34, 61, 9, 5]
```

Figure 11: Third code

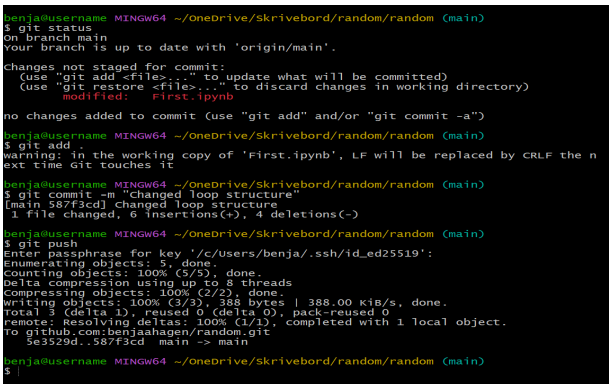

Figure 12: Changed loop structure

```python
import random

class Class:

    #Generate a given amount of random numbers between a range of numbers.
    def random_number_generator(amount, minRange, maxRange):

        assert isinstance(amount, int) #Precondition 1 - amount shold be an integer.
        assert minRange >= 0 #Precondition 2 - consider only positive numbers.
        assert minRange < maxRange #Precondition 3 - minRange should be less than maxRange.

        random_numbers = []
        i = 0
        while i < amount:
            random_numbers.append(random.randint(minRange, maxRange))
            i += 1

        assert len(random_numbers) == amount #Postcondition 1 - there should only be an expected amount of random numbers.
        assert min(random_numbers) >= minRange #Postcondition 2 - all the random numbers should be greater than or equal to minRange
        assert max(random_numbers) <= maxRange #Postcondition 3 - all the random numbers should be less than or equal to maxRange.

        return random_numbers

x = Class.random_number_generator(10, 1, 100)
print (x)
```

✓ 0.6s

```
[29, 15, 60, 84, 67, 59, 64, 48, 10, 91]
```

Figure 13: Final code

Here is a link to my GitHub repository.

References

[Black,] Black, R. What is system testing? - definition from whatis.com. https://www.techtarget.com/searchsoftwarequality/definition/system-testing.

[geeksforgeeks, a] geeksforgeeks. Centralized vs distributed version control: Which one should we choose? - geeksforgeeks. https://www.geeksforgeeks.org/centralized-vs-distributed-version-control-which-one-should-we-choose/?ref=lbp.

[geeksforgeeks, b] geeksforgeeks. Difference between component and unit testing - geeksforgeeks. https://www.geeksforgeeks.org/difference-between-component-and-unit-testing/.

[geeksforgeeks, c] geeksforgeeks. Difference between v-model and spiral model - geeksforgeeks. https://www.geeksforgeeks.org/difference-between-v-model-and-spiral-model/.

[geeksforgeeks, d] geeksforgeeks. Version control systems - geeksforgeeks. https://www.geeksforgeeks.org/version-control-systems/.

[McCormick,] McCormick, M. Waterfall vs. agile methodology. http://www.mccormickpcs.com/images/Waterfall_vs_Agile_Methodology.pdf.

[Schiestl,] Schiestl, B. Code branching definition | what is a branch (version control)? | perforce. https://www.perforce.com/blog/vcs/branching-definition-what-branch.

[SDLC, a] SDLC. Sdlc - agile model. https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm#.

[SDLC, b] SDLC. Sdlc - waterfall model. https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm.

[SmartBear,] SmartBear. What is unit testing? | smartbear. https://smartbear.com/learn/automated-testing/what-is-unit-testing/.

[Sommerville, 2016] Sommerville, I. (2016). *Software Engineering*. Pearson Education.