# POLITECNICO DI TORINO

Master's Degree in Engineering and Management


Master's Degree Thesis

## Design and Implementation of an Object-Oriented Python Library for Project Management

**Supervisor:**
Prof. Demagistris Paolo Eugenio

**Candidate:**
Chiemerie Ezechukwu

Academic Year 2021/2022

# Abstract

When developing Project management backends, one often must plan for data types, ways to process the data and actions to take. This thesis explores how OOP methodologies benefit project management systems and software when the building blocks of the software or system are represented in terms of "self-sufficient" objects.

To improve the ease and speed with which applications are developed, there is the need for pre-developed libraries or framework. As such, a lot is abstracted from the library consumer, and they can come up with solutions quicker. A project management application is no exception to this and can benefit from a library that already implements or provides an interface for most project management objects and processes to be represented in the form of objects.

This thesis will produce a python library that models project management objects and processes that will be essential when developing a Python-based thesis project management application. For example, in a thesis setting, objects can be created to represent the project charter, the student involved, the professor and any other external stakeholders etc. These building blocks can then be aggregated, according to requirements, to form the core of the given project which should be self-sufficient (i.e., implement all the methods and attributes necessary). The benefits this will provide are many folds from appropriating the benefits of OOP to PM to reduced efforts in the development process due to code reusability to increased flexibility due to modularity and polymorphism.

# Table of Contents

# List of Figures

# Acronyms

**O2PM** Object oriented project management

**OOP** Object oriented programming

**PM** Project management

**WBS** Work breakdown structure

# Chapter 1

# Introduction

## 1.1 Object oriented project management

### 1.1.1 What is project management

Project management is the use of specific knowledge, skills, tools and techniques to deliver something of value to people [1].

Project management is the discipline of initiating, planning, executing, controlling and closing the work of a team to achieve specific goals and meet specific success criteria. A project is temporary in that it has a defined beginning and end in time, and therefore defined scope and resources [2].

### 1.1.2 What is a project

A project, on the other hand, is distinct in that it is not a routine action, but rather a specific group of operations aimed at achieving a single goal. As a result, a project team frequently consists of people who don't normally collaborate - perhaps from separate businesses and across several continents. Examples of projects can be the creation of software to improve a corporate process, the construction of a building or bridge, the relief effort following a natural disaster, expansion into a new geographical market.

### 1.1.3 Object oriented project management

Object oriented project management (O2PM) is based on concepts of Object oriented programming such as objects and attributes, classes and members. It consists of five major activities;

- finding classes and objects.
- identifying structures.
- identifying subjects.
- defining attributes.
- defining services.

Most importantly, O2PM is all about applying the object-oriented approach to project management [3].

Coming from the standpoint of O2PM, every aspect of a project is an object. In a typical project setting, the project manager constructs a project plan using a tool,

defines tasks, dates, and effort, and assigns them to team members; however there are possibly no adequate mechanisms to encapsulate work and define boundaries. This often will lead to accountability issues, challenges with effectively quantifying project milestones and producing inaccurate reports.

Many of these issues are addressed by O2PM. The object-oriented approach to project management is the focus of O2PM, and the key characteristics are as follows:

- **Encapsulation:** Team members can work well within defined bounds when work deliverables are encapsulated meaning that boundaries are well defined. Better control and, more crucially, accountability will result from this type of job encapsulation. It also aids in the easy identification of issues and their resolution.
- **Inheritance:** will specify the project's architecture, rules, standards, and processes, which must be "inherited" by all team members and their work deliverables.
- **Polymorphism:** Describes the concept that a singular object/interface can have several other implementations. For example, in a thesis project, the professor and student are different entities but both implement the same stakeholder class.
- **Communication:** This defines a known and standard medium through which each encapsulated work object communicates with each other.

## 1.2 Object oriented programming

### 1.2.1 What is Object oriented programming

Object oriented programming (OOP) is a programming paradigm that relies on the concept of classes and objects. It is used to structure a software program into simple, reusable pieces of code blueprints (called classes), which are used to create individual instances of objects. There are many object-oriented programming languages including JavaScript, C++, Java, and Python to name a few [4].

OOP allows programmers to handle software development as if they were dealing with actual real-world objects. People, in everyday life, have the knowledge and ability to do a variety of duties. Objects in OOP have fields to hold knowledge, state, and data, as well as the ability to execute numerous methods.

A class in OOP serves as a blueprint for creating more specific objects. They represent broad categories that share attributes. They enforce what attributes their instances can have but the values for any specific instance is set by that instance itself. For example, in project management, a stakeholder is a party who has an interest in a specific activity and can either affect or be affected by the said activity. Now there are several types of stakeholders but in the OOP world, they are all different implementation of the parent class/blueprint, stakeholder. In order words, any specific type of stakeholder, will be represented as an object, which itself is a specific example of the rather generic class, stakeholder and will have unique values to the properties defined in the class.

The above illustration visually depicts what has been explained so far. A Stakeholder class to contain all the properties a stakeholder must have and then from it are derived instances of a Stakeholder type object, Professor and Student to represent very specific stakeholders. Their attributes are isolated one from another and

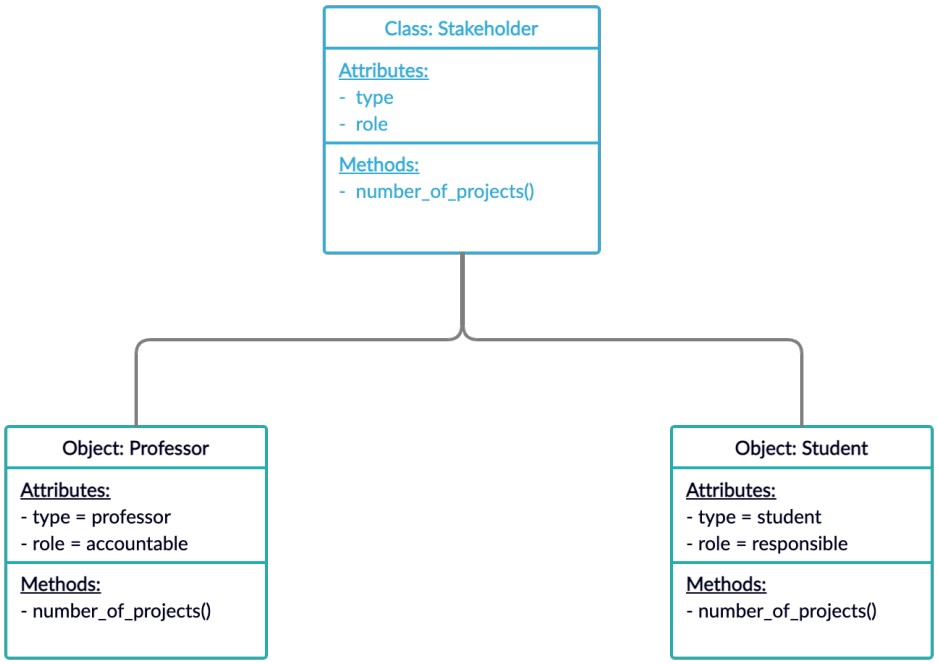


***Figure 1.1:*** *Class Stakeholder implemented in Professor and Student class*

can be modified without affecting the original class or other objects. This means that the singular blueprint, Stakeholder can be reused to represent any number of stakeholders. The "role" field in the image above is according to RACI model which describes a responsibility assignment matrix.

The benefits that OOP presents are manyfold, from modelling complex things as simple reproducible structures to implementing objects that can be reused across programs to modifying the behavior of specific objects through polymorphism to their ability to protect information due to encapsulation. It will be beneficial to explain some basic terminologies in OOP in the next section.

### 1.2.2 Building blocks of OOP

This section deals with elucidating the basic building blocks of an OOP program.

#### Classes

Classes are user-defined data types which is where the blueprint for the structure of attributes and methods are created. Individual objects can then be created or instantiated from this blueprint.

Classes have fields to hold attribute values and methods to model behavior. To illustrate, below is a code snippet in Python demonstrating how a generic class, `Cat` can be represented.

```python
from datetime import date

class Cat:
    def __init__(self, name, birthday: date):
        self.name = name
        self.birthday = birthday

    def get_age(self):
        # get the age using today's date and birthday
        return date.today() - self.birthday

    def meow(self):
        return "meow"
```

***Figure 1.2:*** *Cat class blueprint implementation*

The `Cat` class has attributes `name` and `birthday` and methods `get_age()` and `meow()`. With the knowledge that the class is only a blueprint for modelling a cat, a cat object can be instantiated from it to represent an individual real-world thing.

### Objects

OOP is all about objects. Objects are instances of a class created with specific data. In code snippet, `milana` is an instance of the `Cat` class

```
1  from datetime import date
2
3  class Cat:
4      def __init__(self, name, birthday: date):
5          self.name = name
6          self.birthday = birthday
7
8      def get_age(self):
9          # get the age using today's date and birthday
10         return date.today() - self.birthday
11
12     def meow(self):
13         return "meow"
14
15 # a new instance of the Cat class and an individual cat named Milana
16 milana = Cat(name="Milana", birthday=date(2019, 5, 15))
17
18 # another instance of the Cat class and another individual cat named Masha
19 masha = Cat(name="Masha", birthday=date(2020, 7, 24))
```

***Figure 1.3:*** *Cat class instance*

When the `Cat` class is called like on lines 16 and 19:

- A new object is created each time and stored in the variables `milana` and `masha` respectively.
- The constructor (`__init__`) method runs with the `name` and `birthday` arguments and assigns values.

### Attributes

The information that is stored is referred to as attributes. The Class template defines the attributes. Individual objects have data stored in the attributes field when they are instantiated.

The data in the object's attributes fields define the object's state. The `birthday` attribute could define the state of an object, allowing software to manage cats of various ages differently.

### Methods

The behaviors of objects are represented by methods. Methods carry out operations, such as returning information about an object or updating its data. The code for the method is specified in the class definition.

Objects can invoke the methods described in the class when they are instantiated. In the code snippet below, the `meow()` method is defined in the `Cat` class and the `meow()` method is called on either the `milana` or `masha` object on lines 21 and 22 respectively.

```
15  # a new instance of the Cat class and an individual cat named Milana
16  milana = Cat(name="Milana", birthday=date(2019, 5, 15))
17
18  # another instance of the Cat class and another individual cat named Masha
19  masha = Cat(name="Masha", birthday=date(2020, 7, 24))
20
21  milana.meow()  # returns "meow"
22  masha.meow()  # returns "meow"
```

***Figure 1.4:*** *class methods*

Methods are often employed to modify, update or delete data but they don't have to. As a matter of fact, the `meow()` method does not modify any data because meowing does not modify any of the attributes of the `Cat` class, `name` or `birthday`. Methods are how programmers keep functionality encapsulated within an object and promote reusability. When debugging, its reusability comes in handy. If an error occurs, there is only one location to look for it and correct it rather than numerous.

## 1.3 Principles of OOP

There are four main pillars of OOP;

- **Inheritance:** Child classes inherit data and behavior from their parent classes.
- **Encapsulation:** The process of enclosing information in an object and exposing only a necessary subset.
- **Abstraction:** exposing high-level public methods for accessing an object.
- **Polymorphism:** many methods can do the same task.

### Inheritance

Objects typically have a lot in common. They have some similar logic. However, they are not comparable. Inheritance allows to reuse the common logic and extract the unique logic into another separate class. It implies that a (child) class can be created by deriving from another (parent) class. The child class derives all the parent class's fields and methods (common part) and can implement its own (unique part).

Herding dogs, for example, have a one-of-a-kind capacity to herd animals. To put

it another way, all herding dogs are dogs, but not all dogs are herding dogs. This distinction can be illustrated by establishing a `HerdingDog` child class from a parent `Dog` class, and then adding the unique `herd()` behavior to it.

Inheritance has the advantage of allowing programs to build a generic parent class and then create more specific child classes as needed. This simplifies the overall development because, rather than duplicating the `Dog` class's structure repeatedly, child classes can have automatic access to their parent class's features.

```python
from datetime import date

class Dog:
    def __init__(self, name, birthday: date):
        self.name = name
        self.birthday = birthday

    def get_age(self):
        return self._calculate_age()

    def _calculate_age(self):
        # get the age using today's date and birthday
        return date.today() - self.birthday

    def bark(self):
        return "woof!"
```

***Figure 1.5:*** *Generic Dog class*

In the code snippet in Figure 1.6, the child class `HerdingDog` inherits the method bark from the parent class `Dog` in Figure 1.5 and adds a new method `herd()`.

```python
# Child class HerdingDog, inherits from parent class Dog
class HerdingDog(Dog):
    def __init__(self, name, birthday: date):
        super().__init__(name, birthday)

    def herd(self):
        # additional method for HerdingDog child class
        return "Stay together!"
```

***Figure 1.6:*** *Inheritance illustration*

The `bark()` method is not duplicated in the `HerdingDog` class; instead, it inherits the `bark()` method defined in the parent `Dog` class defined in Figure 1.5.

```python
# Child class HerdingDog, inherits from parent class Dog
class HerdingDog(Dog):
    def __init__(self, name, birthday: date):
        super().__init__(name, birthday)

    def herd(self):
        # additional method for HerdingDog child class
        return "Stay together!"

# instantiate a new HerdingDog object
jasper = HerdingDog("Jasper", date(2017, 12, 19))
jasper.bark()
```

***Figure 1.7:*** *Child class instance*

When the code in Figure 1.3 calls `jasper.bark()` method, the `bark()` method traverses up the chain to the parent where the method has been defined.

### Encapsulation

Encapsulation is the process of enclosing all critical information within an object and only presenting part of it to the outside world. Code within the class template defines attributes and behaviors.

Encapsulation hides the internal software code implementation inside a class and hides internal data of inside objects.

Encapsulation is achieved when each object keeps its state private, inside a class. Other objects don't have direct access to this state. Instead, they can only call a list of public methods.

As an example of encapsulation, consider an automobile. Public interfaces are how the automobile communicates with the outside world by employing blinkers to signal turns. Under the hood, on the other hand, the engine is hidden.

Security is improved via encapsulation. Private attributes and methods can be defined to prevent access from outside the class. Public methods and properties are used to obtain or alter data in an object.

Using the `Dog` class example, encapsulation helps here so that access to private information on other `Dog` instances is not possible. Consider the `get_age()` method in Figure 1.5, the calculation is hidden inside the `Dog` class and each instance of the class will use the `get_age()` method to access the `age` data. Since methods can also update an object's data, the developer controls what values can be changed through public methods.

### Abstraction

Abstraction can be thought of as a natural extension of encapsulation. It refers to the user's interaction with only a subset of an object's attributes and operations. To access a complex object, abstraction use simplified, high-level tools. When employing abstraction, each object should only disclose a high-level mechanism for interacting with it. Internal implementation details should be hidden behind this mechanism. Using the automobile example again, you don't have to know the details of the inner workings of the engine to drive it. The driver utilizes only a few tools, such as the gas pedal, brake, steering wheel, and blinker. The driver is not privy to the engineering. A lot of elements must function together under the hood to make an automobile run but revealing that knowledge to the driver would be a risky distraction.

In the `Dog` class example in Figure 1.5, the calculation of the age has been abstracted away into a private method `_calculate_age()` and a simple public method `get_age()` is exposed as a way to get the age information

The benefits of abstraction are summarized below:

- Simple, high level user interfaces.
- Complex code is hidden.
- Security.
- Easier software maintenance.
- Updates to code will rarely change abstraction.

### Polymorphism

Polymorphism refers to the design of items that have similar behavior. Objects can override shared parent behaviors with specific child behaviors through inheritance. Method overriding and method overloading are two ways that polymorphism allows the same method to perform various actions.

- **Method Overriding**
  In method overriding, the child class will provide its own implementation of a method already in the parent class effectively overriding it. If for the `Dog` class, there's the need to create another child class `TrackingDog` which for some reason barks differently, the `bark()` method established in the parent class will have to be overridden like in the code snippet in Figure 1.8 to produce a different bark sound.

- **Method Overloading**
  This kind of polymorphism happens at compile or code execution time. Methods may have the same name but behave differently based on the number of parameters passed to them.

```python
from datetime import date

class Dog:
    def __init__(self, name, birthday: date):
        self.name = name
        self.birthday = birthday

    def get_age(self):
        return self._calculate_age()

    def _calculate_age(self):
        # get the age using today's date and birthday
        return date.today() - self.birthday

    def bark(self):
        return "woof!"

# Child class TrackingDog, inherits from parent class Dog
class TrackingDog(Dog):
    def __init__(self, name, birthday: date):
        super().__init__(name, birthday)

    def track(self):
        # additional method for TrackingDog child class
        return "Searching...!"

    def bark(self):
        return "Found It!"

# instantiate a new TrackingDog object
buddy = TrackingDog("Buddy", date(2018, 8, 11))
buddy.bark()  # returns "Found It!"
```

***Figure 1.8:*** *Method overriding*

The benefits of Polymorphism are:

- Objects of different types can be passed through the same interface.
- Method overriding.
- Method overloading.

## 1.4 Libraries

A "library" is a collection of program parts that do common and/or specialized things that save the programmer from needing to "reinvent the wheel" when writing software. A class library is a pre-coded OOP template collection. It usually consists of functions to call and object classes you can instantiate [5].

Libraries encourage code reuse and decrease redundancy by providing implementation of repetitive jobs. Developing programs from the ground up can be a time-consuming and costly task. Class libraries include all necessary classes in a previously written coded format, making programming easier while also improving code quality. Customizing the class template is done in accordance with the programming requirements. To minimize programming language limitations, class libraries are regularly updated, and time verified for stability.

Programmers can decide to build everything from the ground up, but this is incredibly unsustainable and ill-advised for the reasons below;

- **Expertise in the domain covered by the library:** Authors are usually experts in the domain covered by the library. This will ensure that users of the library get the best implementation possible.
- **Stability:** Libraries have the benefit of being used by hundreds, if not thousands, of developers around the world. Most of the early issues have already been encountered and resolved by authors.
- **Knowledge:** By having to use code pre-written by others, developers gain knowledge in the process. Many well-known libraries are built by top developers and serve as excellent examples of solid coding and design.
- **Finances:** For commercial libraries, the equivalent of hundreds, if not thousands, of man days of work can be got for free thanks to open source.

# Chapter 2

# Implementation

The implementation of this thesis project focuses on the library to be developed with the Python programming language.

## 2.1 Tools and technologies used

### 2.1.1 Python

Python is a programming language that is commonly used to create websites (server-side) and applications, automate operations, and perform data analysis. Python is a general-purpose programming language, which means that it can be used to develop a wide range of applications and isn't specialized for any particular problem. Because of its versatility and beginner-friendliness, it has become one of the most widely used programming languages today.

Python was designed for readability and has some similarities to the English language

with influence from mathematics. Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly braces for this purpose.

Python is popular for a variety of reasons from its simple syntax that mimics natural language to its versatility to the large and incredibly active community that contributes to Python's pool of modules and libraries.

### 2.1.2 Git and GitHub (Version Control)

#### Git

From web developers to app developers, Git is useful to anyone who writes code or track changes to files. Git is the most used version control system. It keeps track of the changes you make to files so you can see what has been done and go back to previous versions if you need to. Git also facilitates cooperation by allowing several people's modifications to be merged into a single source. Initialize Git locally and your files and their history are stored on your computer. Git repositories contains all the project files and the entire revision history. It can be created by running the `git init` command in the location you wish to create this repository. This creates a `.git` subfolder, which contains all of the Git metadata for tracking changes.

#### GitHub

GitHub, on the other hand, is a Git hosting repository that makes it easier for developers to work together. Developers can work on the same code repository without conflicts using features like pull requests, code review, and issues (ticketing system).

Developers can work on separate branches of the same repository, commit code changes to store them, create a pull request to the main branch, debate and peer-review code changes before merging the pull request.

A repository is sometimes known as a repo. Repos are Git projects that contain files and directories associated with a development project, as well as the revision history of each file. Within a repository, new branches can be created, pull requests can be made, and merges can take place. The changelog displays the history of file changes in the repository.

Git and GitHub has been employed to store and version all code modifications for this project.

**Why use Git**

- **Simplifies the process of contributing to open source:** GitHub is used by nearly every open-source project to administer their project. If your project is open source, GitHub is free to use, and it contains a wiki and issue tracker that makes it simple to provide more detailed documentation and receive feedback. To contribute, simply fork a project, make your changes, and submit a pull request via the GitHub web interface.
- **Documentation:** You make it easy to get quality documentation by using GitHub. They include articles on practically every issue related to git that you can think of in their help section and tutorials.
- **Showcase your work:** Do you want to attract recruiters as a developer? The finest tool you can use for this is GitHub. Most firms now check at GitHub profiles while looking for fresh hires for their projects. Even if you did not attend a prestigious university or college, if your profile is available, you will have a better chance of being hired.

- **Markdown:** Markdown allows you to write styled texts using a simple text editor. Everything on GitHub is written in Markdown, including the issue tracker, user comments, and everything else. With so many other computer languages to master in order to build up projects, having your content inputted in a format without having to learn yet another system is a huge benefit.
- **Repository** This has previously been noted, but it's worth repeating: GitHub is a repository. This means that it is possible for your work to be seen by the general audience. Furthermore, GitHub is one of the largest coding communities in the world right now, giving your project a lot of exposure.
- **Keep track of changes in your code over time:** It's difficult to maintain track of modifications when numerous people work on a project—who modified what, when, and where those files are stored. This problem is solved by GitHub, which keeps track of all the modifications that have been pushed to the repository. You can keep a version history of your code, much like you can with Microsoft Word or Google Drive, so that earlier versions aren't lost with each iteration.
- **Options for integration:** GitHub integrates with popular cloud platforms like Amazon and Google Cloud, as well as feedback tracking services like Code Climate, and can highlight syntax in over 200 different programming languages.

### 2.1.3 GitHub Actions

For continuous integration (CI), GitHub Actions has been employed.

GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. It allows for creating workflows that build and test your source code on events you configure

happening in your repository like pull requests or even merge pull requests to production [6].

GitHub Actions extends beyond DevOps by allowing you to execute workflows in response to other events in your repository. For example, anytime someone opens a new issue in your repository, you may execute a workflow to automatically add the required labels.

You can execute your workflows on GitHub's virtual machines running Linux, Windows, and macOS, or you can host your own self-hosted runners in your own data center or cloud infrastructure.

### The components of GitHub Actions

When an event occurs in your repository, such as when a pull request is opened or an issue is raised, you may set up a GitHub Actions workflow to be executed. Your workflow includes one or more jobs that can execute sequentially or concurrently. Each task contains one or more steps that either run a script you create or run an action, which is a reusable addition that can streamline your workflow.

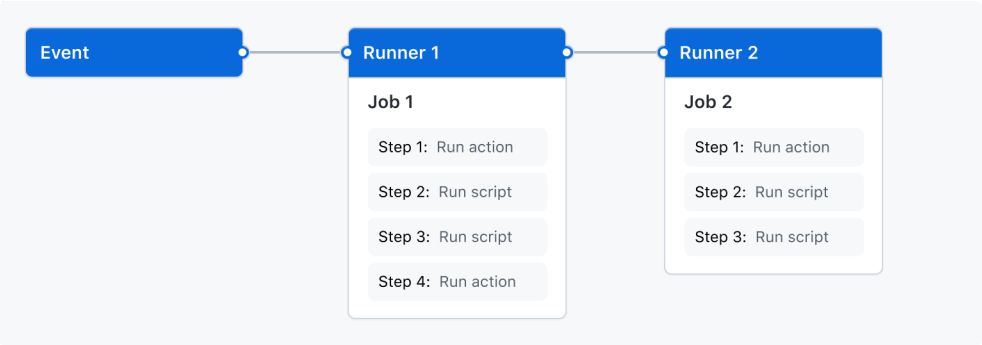


***Figure 2.1:*** *Components of GitHub Actions [6]*

### Workflows

A workflow is a programmable automated procedure that executes one or more tasks. Workflows are specified in a YAML file that is checked into your repository. They can be triggered by an event in your repository, manually, or on a set timetable.

A repository can contain numerous workflows, each of which can carry out a separate set of tasks. For example, you may have one workflow to build and test pull requests, another to deploy your application whenever a release is made, and still another to add a label if someone reports a new issue.

### Events

A workflow run is triggered by an event, which is a specific activity in a repository. For example, GitHub can be an activity when someone creates a pull request, opens an issue, or pushes a commit to a repository. A workflow can also be started manually on a timetable, or by triggering a REST API.

### Jobs

A job is a series of workflow stages that all run on the same runner. Each step consists of either a shell script that will be run or an action that will be performed. The steps are carried out in a sequential order and are interdependent. Because each step is run on the same runner, data can be shared from one step to the next. A step that builds your application, for example, could be followed by a step that tests the application that was built.

You can configure a job's dependencies with other jobs; by default, jobs execute in

parallel and have no dependencies. When a job becomes reliant on another job, it will not run until the dependent job has completed. You might have many build tasks for different architectures with no dependencies and a packaging job that is dependent on those jobs, for example. The build jobs will run in parallel, and the packaging job will begin once they have all completed successfully.

#### Actions

An action is a GitHub Actions platform custom application that performs a sophisticated but regularly repeated activity. To assist in limiting the amount of repetitive code in your workflow files, use an action. An action can grab your git repository from GitHub, configure your build environment's toolchain, or set up your cloud provider's credentials.

#### Runners

When your workflows are triggered, a runner is a server that executes them. Each runner is limited to one job at a time. To run your workflows, GitHub provides Ubuntu Linux, Microsoft Windows, and macOS runners; each workflow run takes place in a new, freshly provisioned virtual machine. You can host your own runners if you need a different operating system or a specific hardware setup.

## 2.2 Library Implementation

In this section, the implementation of the library is showcased and explained. The library will have 8 distinct project management objects each represented as a class in the library namely;

- `ProjectCharter` class
- `Stakeholder` class
- `Deliverable` class
- `ProjectGovernance` class
- `Risk` class
- `WBSItem` class
- `WBS` class
- `Project` class

### 2.2.1 The ProjectCharter class

A project charter is a formal, typically short document that describes your project in its entirety — including what the objectives are, how it will be carried out, and who the stakeholders are. It is a crucial ingredient in planning the project because it is used throughout the project lifecycle [7].

The `ProjectCharter` class presents a few methods and properties.

The class's methods and signatures are represented below in a very simplified interface to abstract away the implementation.

```
class ProjectCharter(project_title = None):
    # properties
    property executive_summary: str
    property project_stakeholders: list
    property project_title: str
```

```
    # methods
    method add_stakeholder(self, stakeholder)

    # static methods
    static is_class_stakeholder(obj) -> bool
```

The constructor is called anytime a new `ProjectCharter` instance is created. It takes an optional argument, `project_title` and goes ahead to initialize an empty list `_project_stakeholders` private property that will hold and track `Stakeholder` objects

The `executive_summary` property is a string field that holds summarized information about project.

There's a restriction to the `ProjectCharter` class in that only objects initialized from the `Stakeholder` class can be added to the `project_stakeholders` list. Adding a stakeholder must be done through the method `add_stakeholder`. A static method `is_class_stakeholder` is available solely for verifying objects before they are accepted to the stakeholder list. The static method simple returns a boolean `True` or `False` and is used in the method `add_stakeholder` like below

```
def add_stakeholder(self, stakeholder):
    assert self.is_class_stakeholder(stakeholder)
    if isinstance(stakeholder, (list, tuple)):
        for s in stakeholder:
            self.__finalize_add_stakeholder(s)
    else:
        self.__finalize_add_stakeholder(stakeholder)
```

```
@staticmethod
def is_class_stakeholder(obj) -> bool:
    """
    Checks that the stakeholder about to be added to the project
    is instance of the Stakeholder class
    """
    if isinstance(obj, list):
        return all(isinstance(s, Stakeholder) for s in obj)
    return isinstance(obj, Stakeholder)
```

### 2.2.2 The Stakeholder class

This class provides a template for modelling stakeholder objects. Its methods, properties and signatures are below in a simplified interface.

```
class Stakeholder(first_name, last_name, email):
    # methods
    method add_to_involved_projects(self, value)
    method get_full_name(self) -> str
    method get_number_of_projects(self) -> int

    # properties
    property projects_involved: List[Project]
    property responsibility: str
    property role
    property stakeholder_title
    property stakeholder_type
```

The method `add_to_involved_projects` is designed to be used only by the `__finalize_add_stakeholder()` private method defined in the `ProjectCharter` class. It is simply used to add new `ProjectCharter` instances to the list of projects the stakeholder in question is involved with.

`get_number_of_projects()` simply returns an integer depicting the number of projects that stakeholder instance is involved with at any point in time.

The property `projects_involved` returns a list of objects of the `ProjectCharter` charter class readily available should the library consumer require it.

For the `role` property of the various stakeholders, a simple enum following the RACI responsibility models is defined.

```
class RoleEnum(Enum):
    """
    * R = Responsible (Those who do the work to complete the task)
    * A = Accountable (also approver or final approving authority)
    * C = Consulted (Those whose opinions are sought)
    * I = Informed (Those who are kept up-to-date on progress)
    """

    RESPONSIBLE = "R"
    ACCOUNTABLE = "A"
    CONSULTED = "C"
    INFORMED = "I"
    UNSET = "UNSET"
```

### 2.2.3 The Deliverable class

Deliverables are very important to the success of any project and this library provides an implementation of a deliverable model. The methods and properties it provides are showcased in the simple interface below

```
class Deliverable(doc_title):
    # methods
    method mark_as_accepted(self)
```

```
    method mark_as_rejected(self)
    method set_reviewer(self, value)

    # properties
    property accepted: bool
    property reviewer
    property acceptance_criteria
    property description
    property expected_result
```

The `mark_as_accepted` and `mark_as_rejected` methods serve to set the `accepted` property to `True` or `False` depending on whether the deliverable has been accepted or not. `set_reviewer` sets the `reviewer` property to the "responsible" who is accountable for the deliverable. `acceptance_criteria` property serves to define what determines when a deliverable can be accepted or not.

### 2.2.4 The ProjectGovernance class

Project Governance is the set of rules, procedures and policies that determine how projects are managed and overseen.

These rules and procedures define how decisions are made during projects. As part of the oversight process, project governance also determines the metrics by which project success is measured.

The methods and properties this class implements are in the simplified interface below.

```
class ProjectGovernance(project):
    # methods
    method get_all_stakeholders(self) -> list
```

```
    method sign_agreement(self, signee: Stakeholder)

    # properties
    property not_signed_by: list
    property signed_by: list
    property signed_by_all: bool
```

The `get_all_stakeholders` method simply returns a list of stakeholder objects included in the current project governance object.

`sign_agreement` simply exposes a method used to sign the project governance by the stakeholder object passed in the argument.

There are a few properties in this class as well. `not_signed_by` returns a list of stakeholder objects included in the project governance object but has yet to consent the terms that will govern the project execution. `signed_by` also returns a list of stakeholder objects and is the exact opposite of the former. `signed_by_all` is a boolean type that simple returns `True` if all the involved stakeholders have signed and `False` otherwise.

### 2.2.5 The Risk class

Risk is any unexpected event that can affect your project — for better or for worse [8]. People, processes, technology, and resources can all be impacted by risk. Risks are not the same as issues, which is a crucial distinction to recognize. Issues are something you know you'll have to deal with and may even know when they'll happen, such as a planned vacation for a team member or a large rise in product demand during the holidays. Risks are occurrences that can happen at any time and that you may not be able to predict — for example, a sudden outbreak of flu in your office or a critical product component being out of stock.

The `Risk` class for this library has an interface as showcased below

```
class Risk(risk_name, probability, impact)
    # methods
    method assign_risk_owner(self, value)
    method get_risk_score(self)
    method set_counter_measure(self, counter_measure)

    # properties
    property counter_measure
    property description
    property impact
    property probability
    property risk_owner
```

`assign_risk_owner` is simply the method for assigning the risk owner. `get_risk_score` is a value calculated based on the `probability` and `impact` properties. An Enum class `RiskScore` is defined to organize the various risk score points and is the return type of the method `get_risk_score`.

```
class RiskScore(Enum):
    VERY_HIGH = range(80, 101) # 80% - 100%
    HIGH = range(50, 80) # 50% - 79%
    MODERATE = range(30, 50) # 30% - 49%
    LOW = range(0, 30) # 0 - 29%

    @classmethod
    def get_risk_score(cls, value):
        for val in cls:
            if value in val.value:
                return val
```

The `set_counter_measure` method is responsible for setting the counter measure to be adopted for mitigating the particular risk object in question.

There are a possible of allowed values for the return value of the `set_counter_measure` method. It is defined in the `RiskCounterMeasure` Enum class

```
class RiskCounterMeasure(Enum):
    REDUCE = "REDUCE"
    PREVENT = "PREVENT"
    ACCEPT = "ACCEPT"
    TRANSFER = "TRANSFER"

    @classmethod
    def list_allowed_values(cls):
        return list(map(lambda c: c.value, cls))
```

### 2.2.6 The WBS class

The Work breakdown structure (WBS) class provides an interface for organizing objects from the `WBSItem` class. The `WBSItem` class's properties are displayed below

```
class WBSItem(work_item_name, duration, lag):
    # properties
    property duration
    property lag
    property objective
    property percent_complete
    property resource
    property status
```

Several `WBSItem` objects are aggregated under a list type property to make up part of the `WBS` class.

Here is a look at the `WBS` class simplified interface

```
class WBS(name, start_date):
    # methods
    method add_wbs_item(self, *args: WBSItem)
    method get_critical_path(self)
    method link_nodes(self, *args: Tuple[WBSItem, WBSItem])
    method update_all(self)

    # properties
    property duration
    property finish_date
    property percent_complete
    property start_date
    property wbs_items: List[WBSItem]
```

The method `add_wbs_item` takes as many arguments as are passed to it which themselves must be objects made from the `WBSItem`'s class. The method adds the individual `WBSItem` objects to the `WBS` object into the property `wbs_items`

It is important to mention that the `WBS` class makes use of an external python package (library), `criticalpath` [9]. It uses the CPM method to calculate the critical path of a network of tasks, assuming the graph in question is acyclic (i.e. has no closed loops)

The method `get_critical_path` calculates the critical path of the graph utilizing the `criticalpath` package. The method returns a visual string representation of the critical path of the graph in question.

`link_nodes` takes as many tuple type arguments of `WBSItem` objects to link to one another

An example usage from a test case is shown below.

```
def test_WBS_link_nodes(wbs: WBS, wbs_item_factory):

    ...

    node_pairs = (
        (node1, node2),
        (node2, node3),
        (node3, node6),
        (node6, node7),
        (node2, node4),
        (node4, node6),
        (node2, node5),
        (node4, node5),
        (node5, node7),
    )
    wbs.link_nodes(*node_pairs)
    wbs.duration.days # 29
    wbs.get_critical_path() # "node1 -> node2 -> node4 -> node6 -> node7"
```

The graph that produced the `node_pairs` in the example above will look like in the image in Figure 2.2.

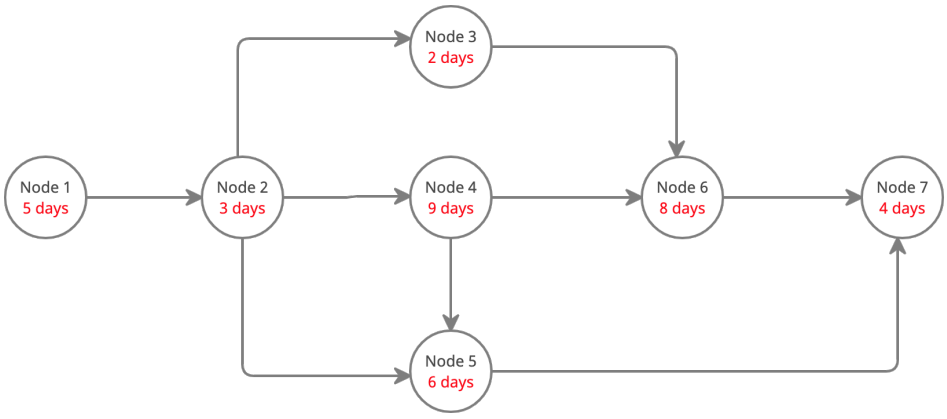


***Figure 2.2:*** *Sample activity node diagram*

Each node pair has to be represented in order to ensure accurate results when accessing the `.duration.days` property or even while calling the `.get_critical_path()`. Repitions are ignored. For Figure 2.2, the relevant node pairs are

- node1, node2
- node2, node3
- node3, node6
- node6, node7
- node2, node4
- node4, node6
- node2, node5
- node4, node5
- node5, node7

### 2.2.7 The Project class

The project class serves as a demonstration as to how the other class objects can be managed under a common umbrella. It provides an interface for interacting with other objects and performing operations on them.

```
class Project(project_charter):
    # methods
    method create_deliverable(self, deliverable_name)
    method add_deliverable(self, deliverable: Deliverable)
    method accept_deliverable(self, deliverable: Deliverable)
```

```
    method reject_deliverable(self, deliverable: Deliverable)
    method add_new_project_risk(self, risk)

    # properties
    property deliverables
    property project_risks
    property project_charter
    property stakeholders
    property project_governance
```

`.create_deliverable()` method take an argument, `deliverable_name` and returns an object of type `Deliverable`. It also calls the method `.add_deliverable()` with the newly created `Deliverable` object as the only argument.

`.accept_deliverable()` and `.reject_deliverable()` are light wrappers around the Deliverable objects and serve to mark the deliverable objects as accepted or rejected using the methods `.mark_as_accepted()` and `.mark_as_rejected()` which the deliverable objects provides themselves.

```
class Project:

    ...

    def accept_deliverable(self, deliverable: Deliverable):
        assert deliverable in self._deliverables, "Deliverable does not exist
            within the current project"
        deliverable.mark_as_accepted()

    def reject_deliverable(self, deliverable: Deliverable):
        assert deliverable in self._deliverables, "Deliverable does not exist
            within the current project"
        deliverable.mark_as_rejected()
```

# Chapter 3

# Testing

## 3.1 What is Testing

In general, testing is finding out how well something works. Software testing is the act of examining the artifacts and the behavior of the software under test by validation and verification [10].

There are several types of tests that can be carried out on a software but for the sake of this thesis project, all emphasis will be on unit test.

## 3.2 What is unit testing

Unit testing is a type of software testing where individual units or components of a software are tested. The goal is to ensure that each unit of software code works as intended. Unit test is carried out by developers throughout the development

(coding) phase of an application. Unit tests are used to isolate a part of code and ensure that it is correct. Any singular function, method, process, module, or object might be considered a unit.

Unit testing ensures that all code meets quality standards before it's deployed. This ensures a reliable engineering environment where quality is paramount. Over the course of the product development life cycle, unit testing saves time and money, and helps developers write better code, more efficiently. It offers a wide variety of advantages but a few has been listed below:

- **Bugs are found easily and quicker:** Code that is covered by tests is more dependable than code that is not. If a future modification breaks something in the code, developers will be able to quickly pinpoint the source of the problem rather than having to sift through a large codebase.
- **Unit testing provides documentation:** Unit tests are a type of living product documentation. Developers can use unit tests to acquire a basic overview of the logic of a module and the system as a whole in order to figure out what functionality is offered by one module vs another. Unit test cases are indications that provide information about whether a software component is being used correctly or incorrectly. As a result, these examples serve as excellent documentation for these indicators.
- **Modularity:** Thanks to the modular nature of the unit testing, parts of a project can be tested without waiting for others to be completed.

## 3.3 Testing the library

Each module and each function down to each line of code was tested yielding a 100% test coverage. Pytest, a popular Python testing library, was used to accomplish the

test. A sample test case is included.

```
from project import Project
from unittest.mock import Mock
from project.governance import ProjectGovernance


def test_Project(project: Project):
    assert str(project) == "Fake Project"
    assert isinstance(project.project_charter, Mock)
    assert hasattr(project.project_charter, "project_stakeholders")
    assert hasattr(project.project_charter, "project_title")
    assert isinstance(project.project_governance, ProjectGovernance)
    assert project.deliverables == []
    assert project.project_risks == []
```

The test cases are enforced through a series of `assert` statements

These tests were configured to run automatically on GitHub using Github Actions whenever there's a new push to the repository. The configuration is in a `.yaml` file format.

```yaml
name: Test

on:
  push:
  pull_request:
    branches:
      - master

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - name: Check Out
        uses: actions/checkout@v2

      - name: Get Branch Name
        run: echo "BRANCH=$(echo ${GITHUB_REF##*/})" >> $GITHUB_ENV

      - name: Install requirements, run tests with make and write coverage
        run: make testall

      - name: Get Coverage for badge
        run: echo "COVERAGE=$(cat coverage.json | jq .totals.percent_covered)%"
            >> $GITHUB_ENV

      - name: Create Badge
        uses: schneegans/dynamic-badges-action@v1.1.0
        with:
          auth: ${{ secrets.GIST_SECRET }}
          gistID: a1817190b63b0cd8f551cb3ec2ab6524
          filename: ${{ github.event.repository.name }}_${{ env.BRANCH }}
              _coverage.json
          label: Code Coverage
          message: ${{ env.COVERAGE }}
          color: green
          namedLogo: GitHub Actions
```

The source code and tests in their entirety are in the Appendix A and Appendix B section respectively

# Chapter 4

# Conclusion

## 4.1 Outcomes

The thesis's outcome is stated in light of the thesis's goal and scope, which were established at the beginning.

The main purpose of this thesis was to develop a project management library. It was designed to provide the basic functionality that will be needed when developing a thesis management web application but can be also be adapted for other purposes. Extensive unit tests have also been written to ensure that when future functionalities are implemented, old ones don't break. These tests have been configured to run automatedly in Github hosted linux runners. The result of this test is attached to a badge on the README.md on the repository to give library users a quick insight about the health of the source code.

## 4.2 Future Improvements

- Right now, it's just a library and haven't been used in any MVP yet. Utilizing and integrating this library in real world applications will lead to improvement as some bugs that might have slipped through the unit testing procedure might eventually be discovered and fixed.
- Integrate a static tool checker like mypy. Python is a dynamically typed language meaning that variables that point to objects can be assigned different data types during their lifetime. Integrating mypy will reduce errors stemming from wrong data type assignment.
- Loosely couple modules to promote independent use. Some modules require other modules. For instance, the `governance` module requires a `Stakeholder` object from the `project_stakeholder` module. This creates interdependence making using one and not the other impossible. It will be beneficial to conduct an analysis and implement design patterns to solve the concern.

# References


[1] Project Management Institute. What is project management? https://www.pmi.org/about/learn-about-pmi/what-is-project-management.

[2] Harelimana JB. *Basic Concepts of Project Management*. Austin Publishing Group, 2017.

[3] K.R. Chandrashekar. Object-oriented project management (o2pm) objectizing work. *Project Management Institute*, 2010.

[4] Erin Doherty. What is object-oriented programming? oop explained in depth. https://www.educative.io/blog/object-oriented-programming.

[5] Steven Parker. What exactly is a library in programming? https://teamtreehouse.com/community/what-exactly-is-a-library-in-programming.

[6] GitHub Docs. Understanding github actions. https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions.

[7] Wrike. What is a project charter in project management? https://www.wrike.com/project-management-guide/faq/what-is-a-project-charter-in-project-management/.

[8] Wrike. What is risk in project management? https://www.wrike.com/project-management-guide/faq/what-is-risk-in-project-management/.

[9] Chris Spencer. criticalpath 0.1.5. https://pypi.org/project/criticalpath/.

[10] Wikipedia. Software testing. https://en.wikipedia.org/wiki/Software_testing.

# Appendix A

# Source code

## A.1 src/project/__init__.py

```
from src.project.project_charter import ProjectCharter

from .deliverables import Deliverable
from .governance import ProjectGovernance


class Project:
    def __init__(self, project_charter) -> None:
        self.project_charter: ProjectCharter = project_charter
        self.stakeholders = project_charter.project_stakeholders
        self._deliverables = []
        self._project_risks = []
        self.project_governance = ProjectGovernance(self)

    @property
    def deliverables(self):
        return self._deliverables
```

```
@property
def project_risks(self):
    return self._project_risks

def add_new_project_risk(self, risk):
    assert risk not in self._project_risks, "This Risk already exists within
        the current project"
    self._project_risks.append(risk)

def add_deliverable(self, value: Deliverable):
    assert value not in self._deliverables, "This Deliverable already exists
        within the project"
    self._deliverables.append(value)

def create_deliverable(self, deliverable_name):
    deliverable = Deliverable(deliverable_name)
    self.add_deliverable(deliverable)
    return deliverable

def accept_deliverable(self, deliverable: Deliverable):
    assert deliverable in self._deliverables, "Deliverable does not exist
        within the current project"
    deliverable.mark_as_accepted()

def reject_deliverable(self, deliverable: Deliverable):
    assert deliverable in self._deliverables, "Deliverable does not exist
        within the current project"
    deliverable.mark_as_rejected()

def __repr__(self) -> str:
    return self.project_charter.project_title
```

## A.2 src/project/deliverables/__init__.py

```python
from src.utils import AbstractBaseDocument


class Deliverable(AbstractBaseDocument):
    def __init__(self, doc_title) -> None:
        super().__init__(doc_title)
        self._description = None
        self._acceptance_criteria = None
        self._reviewer = None
        self._expected_result = None
        self._accepted = None

    @property
    def accepted(self):
        if not isinstance(self._accepted, bool):
            return "N/A"
        return self._accepted

    @property
    def description(self):
        return self._description

    @description.setter
    def description(self, value: str):
        self._description = value

    @property
    def acceptance_criteria(self):
        return self._acceptance_criteria

    @acceptance_criteria.setter
    def acceptance_criteria(self, value: str):
        self._acceptance_criteria = value

    @property
```

```
    def expected_result(self):
        return self._expected_result

    @expected_result.setter
    def expected_result(self, value: str):
        self._expected_result = value

    @property
    def reviewer(self):
        return self._reviewer

    def set_reviewer(self, value):
        """
        If the reviewer is internal to the project,
        value should be an instance of the Stakeholder class
        """
        self._reviewer = value

    def mark_as_accepted(self):
        self._accepted = True

    def mark_as_rejected(self):
        self._accepted = False
```

## A.3 src/project/governance/__init__.py

```
from ..project_charter import ProjectCharter
from ..project_stakeholder import Stakeholder


class ProjectGovernance:
    TERMS_N_CONDITIONS = None

    def __init__(self, project) -> None:
        self._project = project
        self._signed_by = []
        self._project_charter: ProjectCharter = project.project_charter
        self._not_signed_by: list = self._project_charter.project_stakeholders[:]

    @property
    def signed_by(self) -> list:
        return self._signed_by

    @property
    def not_signed_by(self) -> list:
        return self._not_signed_by

    @property
    def signed_by_all(self) -> bool:
        return not bool(len(self._not_signed_by))

    def sign_agreement(self, signee: Stakeholder):
        assert isinstance(signee, Stakeholder), "Expected type Stakeholder, found
            %s" % type(signee).__name__

        assert signee in self.get_all_stakeholders(), (
            "%s can't sign as they are external to the project" % signee.
                get_full_name()
        )

        assert signee not in self._signed_by, "This document has already been
```

```
            signed by %s." % signee.get_full_name()
        self._signed_by.append(signee)
        self._not_signed_by.remove(signee)

    def get_all_stakeholders(self) -> list:
        return self._project_charter.project_stakeholders
```

## A.4 src/project/project_charter/__init__.py

```python
from typing import List, Optional, Union

from src.utils import AbstractBaseDocument

from ..project_stakeholder import Stakeholder


class ProjectCharter(AbstractBaseDocument):
    """
    A project is defined by a project charter.
    Once a project charter exists, it is assumed that a project is underway
    """

    def __init__(self, project_title: Optional[str] = None) -> None:
        super().__init__("Project Charter")
        self._project_title = project_title
        self._project_stakeholders: List[Stakeholder] = []
        self._executive_summary = None

    def __repr__(self) -> str:
        return f"{self.__class__.__name__}({self._project_title})"

    @property
    def project_title(self):
        return self._project_title

    @project_title.setter
    def project_title(self, value: str):
        self._project_title = value

    @property
    def executive_summary(self):
        return self._executive_summary

    @executive_summary.setter
```

```
    def executive_summary(self, value: str):
        self._executive_summary = value

    @property
    def project_stakeholders(self) -> List[Stakeholder]:
        return self._project_stakeholders

    def add_stakeholder(self, stakeholder: Union[Stakeholder, List[Stakeholder]])
        :
        assert self.is_class_stakeholder(stakeholder), (
            "Only Stakeholder instances can be added, not %s." % type(stakeholder)
                .__name__
        )
        if isinstance(stakeholder, (list, tuple)):
            for s in stakeholder:
                self.__finalize_add_stakeholder(s)
        else:
            self.__finalize_add_stakeholder(stakeholder)

    def __finalize_add_stakeholder(self, stakeholder: Stakeholder):
        assert stakeholder not in self._project_stakeholders, (
            'This Stakeholder "%s" already exists within the charter' %
                stakeholder.get_full_name()
        )
        self._project_stakeholders.append(stakeholder)
        stakeholder.add_to_involved_projects(self)

    @staticmethod
    def is_class_stakeholder(obj) -> bool:
        """
        Checks that the stakeholder about to be added to the project
        is instance of the Stakeholder class
        """
        if isinstance(obj, list):
            return all(isinstance(s, Stakeholder) for s in obj)
        return isinstance(obj, Stakeholder)
```

## A.5 src/project/project_stakeholder/helpers.py

```
from enum import Enum


class StakeholderTypeEnum(Enum):
    """Add more stakeholder types to this class as needed"""

    PROFESSOR = "professor"
    STUDENT = "student"
    UNKNOWN = "unknown"

    @classmethod
    def list_allowed_values(cls):
        return list(map(lambda c: c.value, cls))


class RoleEnum(Enum):
    """
    This class is meant to model the responsibility roles in the RACI matrix
        model

    * R = Responsible
    Those who do the work to complete the task. There is at least one role with a
         participation type of responsible.

    * A = Accountable (also approver or final approving authority)
    The one who ensures the prerequisites of the task are met and who delegates
        the work.
    In other words, an accountable must sign off (approve) work that responsible
        provides.
    There must be only one accountable specified for each task or deliverable.

    * C = Consulted
    Those whose opinions are sought, typically subject-matter experts; and with
        whom there is two-way communication.
```

```
    * I = Informed
    Those who are kept up-to-date on progress, often only on completion of the
        task or deliverable.
    """

    RESPONSIBLE = "R"
    ACCOUNTABLE = "A"
    CONSULTED = "C"
    INFORMED = "I"
    UNSET = "UNSET"

    @classmethod
    def list_allowed_values(cls):
        return list(map(lambda c: c.value, cls))
```

## A.6 src/project/project_stakeholder/__init__.py

```
from src.utils import AbstractBaseStakeholder

from .helpers import RoleEnum, StakeholderTypeEnum


class Stakeholder(AbstractBaseStakeholder):
    def __init__(self, first_name, last_name, email) -> None:
        super().__init__(first_name, last_name, email)
        self._stakeholder_type: StakeholderTypeEnum = StakeholderTypeEnum.UNKNOWN
        self._responsibility = None
        self._stakeholder_title = None
        self._role: RoleEnum = RoleEnum.UNSET
        self._projects_involved = []

    def __repr__(self) -> str:
        return f"{self.__class__.__name__}({self.first_name})"

    @property
    def stakeholder_type(self):
        return self._stakeholder_type.value

    @stakeholder_type.setter
    def stakeholder_type(self, value: str):
        allowed_values = StakeholderTypeEnum.list_allowed_values()
        assert value.lower() in allowed_values, 'Couldn\'t set property "
            stakeholder_type". "%s" is not in %s' % (
            value,
            allowed_values,
        )
        self._stakeholder_type = StakeholderTypeEnum(value.lower())

    @property
    def role(self):
        return self._role.value
```

```
    @role.setter
    def role(self, value: str):
        allowed_values = RoleEnum.list_allowed_values()
        assert value.upper() in allowed_values, 'Couldn\'t set property "role". "%
            s" is not in %s' % (
            value,
            allowed_values,
        )
        self._role = RoleEnum(value.upper())

    @property
    def responsibility(self):
        """A brief description of the stakeholder's designated activities"""
        if self._responsibility is None:
            raise AttributeError("This attribute is not set. call the setter to
                set it")
        return self._responsibility

    @responsibility.setter
    def responsibility(self, value: str):
        self._responsibility = value

    @property
    def stakeholder_title(self):
        if self._stakeholder_title is None:
            raise AttributeError("This attribute is not set. call the setter to
                set it")
        return self._stakeholder_title

    @stakeholder_title.setter
    def stakeholder_title(self, value: str):
        self._stakeholder_title = value

    def get_full_name(self) -> str:
        return f"{self.first_name} {self.last_name}"

    @property
    def projects_involved(self):
```

```
        """A list of projects the current stakeholder is involved with"""
        return self._projects_involved

    def add_to_involved_projects(self, value):
        """
        This method should only be called by the `.__finalize_add_stakeholder()`
        method in the ProjectCharter class
        """
        self._projects_involved.append(value)

    def get_number_of_projects(self) -> int:
        return len(self._projects_involved)
```

## A.7 src/project/risks/helpers.py

```
from enum import Enum


class RiskProbabilty(Enum):
    """
    Characterizes the likelihood that a particular risk arises during a
        particular project

    Some arbitrary values are defined here and the upper bounds chosen

    * RARE = 0% < x   5%
    * UNLIKELY = 5% < x   29%
    * POSSIBLE = 30% < x   50%
    * LIKELY = 50% < x   80%
    * CERTAIN = 80% < x   100%
    """

    RARE = 5
    UNLIKELY = 30
    MODERATE = 50
    LIKELY = 80
    CERTAIN = 100


class RiskImpact(Enum):
    """Defines how a particular risk can affect the progress of a project"""

    HIGH = 100
    MEDIUM = 50
    LOW = 10


class RiskScore(Enum):
    VERY_HIGH = range(80, 101) # 80% - 100%
    HIGH = range(50, 80) # 50% - 79%
```

```
    MODERATE = range(30, 50) # 30% - 49%
    LOW = range(0, 30) # 0 - 29%

    @classmethod
    def get_risk_score(cls, value):
        for val in cls:
            if value in val.value:
                return val


class RiskCounterMeasure(Enum):
    REDUCE = "REDUCE"
    PREVENT = "PREVENT"
    ACCEPT = "ACCEPT"
    TRANSFER = "TRANSFER"

    @classmethod
    def list_allowed_values(cls):
        return list(map(lambda c: c.value, cls))
```

## A.8 src/project/risks/__init__.py

```
import re

from .helpers import RiskCounterMeasure, RiskImpact, RiskProbabilty, RiskScore


class RiskMetaClass(type):
    pattern = r"^(rare|unlikely|moderate|likely|certain)_probability_(high|medium
        |low)_impact$"

    def __getattr__(cls, attr):
        if not re.match(cls.pattern, attr):
            raise AttributeError("'%s' object has no attribute '%s'" % (cls.
                __name__, attr))

        return lambda risk_name: cls.__get_risk_class(attr, risk_name)

    def __get_risk_class(cls, attr: str, risk_name: str):
        probability, _, impact, _ = attr.split("_")
        probability_value = RiskProbabilty[probability.upper()].value
        impact_value = RiskImpact[impact.upper()].value
        return Risk(risk_name, probability_value, impact_value)


class Risk(metaclass=RiskMetaClass):
    def __init__(self, risk_name: str, probability: int, impact: int):
        """
        :param name: Name of the Risk instance
        :type name: str
        :param probability: A value between 0 and 100 where each point denotes a
            percentage point
        :type probability: int
        :param impact: A value between 0 and 100 where each point denotes a
            percentage point
        :type impact: int
        """
```

```
        self.risk_name = risk_name
        self.impact = impact
        self.probability = probability
        self._risk_owner = None
        self._description = None
        self._counter_measure = None

    def get_risk_score(self):
        return RiskScore.get_risk_score(int(self.probability / 100 * self.impact))

    @property
    def risk_owner(self):
        return self._risk_owner

    @risk_owner.setter
    def risk_owner(self, value):
        self.assign_risk_owner(value)

    def assign_risk_owner(self, value):
        self._risk_owner = value

    @property
    def description(self) -> str:
        return self._description

    @description.setter
    def description(self, value):
        self._description = value

    @property
    def probability(self) -> int:
        return self._probability

    @probability.setter
    def probability(self, value: int):
        """
        :param value: An integer between 0 and 100 where each point denotes a
            percentage point
```

```
        :type value: int
        """
        assert value in range(101), "Please pass an integer between 0 and 100"
        self._probability = value

    @property
    def impact(self) -> int:
        return self._impact

    @impact.setter
    def impact(self, value: int):
        """
        :param value: An integer between 0 and 100 where each point denotes a
            percentage point
        :type value: int
        """
        assert value in range(101), "Please pass a number between 0 and 1"
        self._impact = value

    @property
    def counter_measure(self):
        return self._counter_measure

    @counter_measure.setter
    def counter_measure(self, value):
        self.set_counter_measure(value)

    def set_counter_measure(self, counter_measure: str):
        """counter_measure is a value to be chosen from the RiskCounterMeasure
            Enum"""
        allowed_values = RiskCounterMeasure.list_allowed_values()
        assert counter_measure.upper() in allowed_values, "%s is not in %s" % (
            counter_measure,
            allowed_values,
        )
        self._counter_measure = RiskCounterMeasure(counter_measure.upper())
```

## A.9 src/project/wbs/__init__.py

```
from datetime import datetime
from typing import List, Tuple

from src.utils import Node

from ..wbs_item import WBSItem


class WBS:
    _check_update_called = False

    def __init__(self, name, start_date: datetime = datetime.now()) -> None:
        self.name = name
        self._wbs_items: List[WBSItem] = []
        self.start_date: datetime = start_date
        self.parent_node = Node(self.name)

    @property
    def percent_complete(self):
        assert self._wbs_items, "The WBS Items list is empty. Populate it to get
            the value of this attribute"
        return sum(x.percent_complete for x in self._wbs_items) // len(self.
            _wbs_items)

    @property
    def start_date(self):
        return self._start_date

    @start_date.setter
    def start_date(self, value: datetime):
        assert isinstance(value, datetime), "start_date takes a datetime object,
            not %s." % type(value).__name__
        self._start_date = value

    @property
```

```
def finish_date(self):
    """This value depends on duration which depends on wbs_items not being
        empty"""
    return self.start_date + self.duration if self.duration else "N/A"

@property
def duration(self):
    self.update_all()
    return self.parent_node.duration

@property
def wbs_items(self):
    return self._wbs_items

def update_all(self):
    """
    Updates timing calculations for all children nodes.
    Sets the es, ls, ef, lf information and updates duration
    """
    if not self._check_update_called and self.wbs_items:
        self._check_update_called = True
        self.parent_node.update_all()

def get_critical_path(self):
    """calculate the critical path using CPM"""
    self.update_all()
    critical_path = self.parent_node.get_critical_path()
    return " -> ".join(str(x) for x in critical_path) if critical_path else
        None

def add_wbs_item(self, *args: WBSItem):
    for wbs_item in args:
        assert isinstance(wbs_item, WBSItem), "Only WBSItem instances can be
            added, not %s." % (
            type(wbs_item).__name__,
        )
        if wbs_item in self._wbs_items:
            continue
```

```
            self._wbs_items.append(wbs_item)
            self.parent_node.add(wbs_item.node_instance)

    def link_nodes(self, *args: Tuple[WBSItem, WBSItem]):
        """
        links the WBSItems in a directed graph
        """
        initial = self.parent_node

        for pair in args:
            assert all(isinstance(x, WBSItem) for x in pair), "Only tuples of
                WBSItem instances are allowed"
            origin, dest = pair

            try:
                initial = initial.link(str(origin), str(dest))
            except KeyError as ke:
                raise KeyError(
                    f"The WBSItem {ke} does not exist within this WBS. "
                    "Did you forget to add it? add it using the method `.
                        add_wbs_item()`"
                )

    def __repr__(self) -> str:
        return self.name
```

## A.10 src/project/wbs_item/__init__.py

```
from datetime import timedelta

from src.utils import Node


class WBSItem:
    def __init__(self, work_item_name: str, duration: timedelta, lag: timedelta =
         timedelta(days=0)) -> None:
        self.work_item_name: str = work_item_name
        self.duration: timedelta = duration
        self.lag: timedelta = lag
        self._percent_complete: int = 0
        self._status = None
        self._objective = None
        self._resource = None

    @property
    def duration(self):
        return self.__duration

    @duration.setter
    def duration(self, value: timedelta):
        assert not (
            hasattr(self, "duration") or hasattr(self, "__duration")
        ), "This attribute is immutable once the object is created"
        assert isinstance(value, timedelta), "Only timedelta objects can be set,
            not %s." % type(value).__name__

        self.__duration = value

    @duration.deleter
    def duration(self):
        raise Exception("This attribute is immutable")

    @property
```

```
def lag(self):
    return self._lag

@lag.setter
def lag(self, value: timedelta):
    assert isinstance(value, timedelta), "Only timedelta objects can be set,
        not %s." % type(value).__name__
    self._lag = value

@property
def percent_complete(self):
    return self._percent_complete

@percent_complete.setter
def percent_complete(self, value: int):
    assert value in range(0, 101), "Enter a integer between 0 and 100"
    self._percent_complete = value

@property
def status(self):
    return self._status

@status.setter
def status(self, value):
    self._status = value

@property
def objective(self):
    return self._objective

@objective.setter
def objective(self, value):
    self._objective = value

@property
def resource(self):
    return self._resource
```

```
    @resource.setter
    def resource(self, value):
        self._resource = value

    @property
    def node_instance(self):
        return Node(name=self.work_item_name, duration=self.__duration, lag=self.
            _lag)

    def __repr__(self) -> str:
        return self.work_item_name
```

## A.11 src/utils/__init__.py

```
# flake8: noqa

from .generic_document import AbstractBaseDocument
from .node import Node
from .stakeholder import AbstractBaseStakeholder
```

## A.12 src/utils/generic_document.py

```
from abc import ABC
from datetime import datetime, timezone



class AbstractBaseDocument(ABC):
    def __init__(self, doc_title) -> None:
        super().__init__()
        self.doc_title = doc_title
        self.time_created = datetime.now(tz=timezone.utc)
```

## A.13 src/utils/stakeholder.py

```python
from abc import ABC, abstractmethod


class AbstractBaseStakeholder(ABC):
    def __init__(self, first_name, last_name, email) -> None:
        super().__init__()
        self.first_name = first_name
        self.last_name = last_name
        self.email = email

    @abstractmethod
    def get_full_name(self):
        """an instance method to get the stakeholders full name"""
```

# Appendix B

# Unit test code

## B.1 tests/factories.py

```
import random
from datetime import timedelta

import factory

from project.project_stakeholder import Stakeholder
from project.wbs_item import WBSItem

from .faker import faker


class StakeholderFactory(factory.Factory):
    class Meta:
        model = Stakeholder

    first_name = factory.LazyAttribute(lambda _: faker.unique.first_name())
    last_name = factory.LazyAttribute(lambda _: faker.unique.last_name())
```

```
    email = factory.LazyAttribute(lambda self: "{}{}@example.com".format(self.
        first_name, self.last_name).lower())


class WBSItemFactory(factory.Factory):
    class Meta:
        model = WBSItem

    work_item_name = factory.LazyAttribute(lambda _: faker.unique.text().split("
        ")[0])
    duration = factory.LazyAttribute(lambda _: timedelta(days=random.randint(0,
        365)))

    @factory.post_generation
    def add_percent_complete(self, create, extracted, **kwargs):
        self.percent_complete = random.randint(1, 100)
```

## B.2 tests/faker.py

```
from faker import Faker


faker = Faker()
```

## B.3 tests/test_project/test_deliverables.py

```
from project.deliverables import Deliverable
from project.project_stakeholder import Stakeholder


def test_Deliverable(deliverable: Deliverable, monkeypatch):
    """
    :param deliverable: comes from conftest.py at the root
    :type deliverable: Deliverable
    :param monkeypatch: pytest fixture for patching on the fly
    """
    assert deliverable.doc_title == "Fake Deliverable"
    assert deliverable.accepted == "N/A"
    assert deliverable.description is None
    assert deliverable.acceptance_criteria is None
    assert deliverable.expected_result is None
    assert deliverable.reviewer is None

    monkeypatch.setattr(deliverable, "description", "Fake Description", raising=
        True)
    assert deliverable.description == "Fake Description"

    monkeypatch.setattr(deliverable, "acceptance_criteria", "Fake Criteria",
        raising=True)
    assert deliverable.acceptance_criteria == "Fake Criteria"

    monkeypatch.setattr(deliverable, "expected_result", "Fake Result", raising=
        True)
    assert deliverable.expected_result == "Fake Result"


def test_can_set_deliverable_reviewer(deliverable: Deliverable, stakeholder:
    Stakeholder):
    deliverable.set_reviewer(stakeholder)
    assert deliverable.reviewer is stakeholder
```

```
def test_can_accept_deliverable(deliverable: Deliverable):
    deliverable.mark_as_accepted()
    assert deliverable.accepted is True


def test_can_reject_deliverable(deliverable: Deliverable):
    deliverable.mark_as_rejected()
    assert deliverable.accepted is False
```

## B.4 tests/test_project/test_governance.py

```
from unittest.mock import Mock

import pytest

from project.governance import ProjectGovernance


@pytest.fixture
def governance(stakeholder_factory, monkeypatch):
    stakeholders = stakeholder_factory.create_batch(5)

    mock_project_charter = Mock(spec=["project_stakeholders"])
    monkeypatch.setattr(mock_project_charter, "project_stakeholders",
        stakeholders, raising=True)

    mock_project = Mock(spec=["project_charter"])
    monkeypatch.setattr(mock_project, "project_charter", mock_project_charter,
        raising=True)

    governance = ProjectGovernance(project=mock_project)

    yield governance


def test_ProjectGovernance(governance, monkeypatch):
    stakeholders = governance.get_all_stakeholders()

    assert stakeholders == governance.not_signed_by

    terms = "Some terms and conditions"
    monkeypatch.setattr(governance, "TERMS_N_CONDITIONS", terms, raising=True)
    assert governance.TERMS_N_CONDITIONS == terms

    assert not governance.signed_by
    assert not governance.signed_by_all
```

```
    for stakeholder in stakeholders:
        governance.sign_agreement(stakeholder)

    assert governance.signed_by_all
    assert governance.signed_by == stakeholders


def test_external_stakeholder_cannot_sign(governance, stakeholder):
    with pytest.raises(AssertionError):
        governance.sign_agreement(stakeholder)


def test_stakeholder_cannot_sign_twice(governance):
    stakeholders = governance.get_all_stakeholders()
    governance.sign_agreement(stakeholders[0])

    with pytest.raises(AssertionError):
        governance.sign_agreement(stakeholders[0])

    assert len(governance.signed_by) == len(stakeholders) - len(governance.
        not_signed_by)


def test_only_stakeholder_instances_allowed_to_sign(governance):
    not_stakeholder_instance = Mock()

    with pytest.raises(AssertionError):
        governance.sign_agreement(not_stakeholder_instance)
```

## B.5 tests/test_project/test_project_charter.py

```
from typing import List
from unittest.mock import Mock, call, patch

import pytest

from project.project_charter import ProjectCharter
from project.project_stakeholder import Stakeholder


@pytest.fixture
def charter():
    yield ProjectCharter(project_title="Fake Project")


def test_ProjectCharter(charter: ProjectCharter):
    assert charter.doc_title == "Project Charter"
    assert charter.project_title == "Fake Project"
    assert not charter.project_stakeholders
    assert not charter.executive_summary
    assert str(charter) == "ProjectCharter(Fake Project)"


def test_ProjectCharter_set_project_title(charter: ProjectCharter, monkeypatch):
    monkeypatch.setattr(charter, "project_title", "New Fake Project", raising=
        True)
    assert charter.project_title == "New Fake Project"


def test_ProjectCharter_set_executive_summary(charter: ProjectCharter,
    monkeypatch):
    monkeypatch.setattr(charter, "executive_summary", "Fake Summary", raising=
        True)
    assert charter.executive_summary == "Fake Summary"
```

```
def test_ProjectCharter_is_class_stakeholder(charter: ProjectCharter, stakeholder
    , stakeholder_factory):
    assert charter.is_class_stakeholder(stakeholder) is True

    stakeholders: List[Stakeholder] = stakeholder_factory.build_batch(5)
    assert charter.is_class_stakeholder(stakeholders) is True

    not_stakeholder = Mock()
    assert charter.is_class_stakeholder(not_stakeholder) is False

    altered_stakeholders = stakeholders.append(not_stakeholder)
    assert charter.is_class_stakeholder(altered_stakeholders) is False


def test_ProjectCharter__finalize_add_stakeholder(charter: ProjectCharter,
    stakeholder: Stakeholder):
    charter._ProjectCharter__finalize_add_stakeholder(stakeholder)
    assert stakeholder in charter.project_stakeholders
    assert charter in stakeholder.projects_involved


def test_ProjectCharter__finalize_add_stakeholder_cannot_add_stakeholder_twice(
    charter: ProjectCharter, stakeholder: Stakeholder
):
    charter._ProjectCharter__finalize_add_stakeholder(stakeholder)

    with pytest.raises(AssertionError):
        charter._ProjectCharter__finalize_add_stakeholder(stakeholder)


@patch("project.project_charter.ProjectCharter.is_class_stakeholder")
def test_ProjectCharter_add_stakeholder(mocked_is_class_stakeholder, charter:
    ProjectCharter, stakeholder_factory):
    assert charter.is_class_stakeholder is mocked_is_class_stakeholder
    stakeholders: List[Stakeholder] = stakeholder_factory.build_batch(5)

    mocked_is_class_stakeholder.return_value = False
    with pytest.raises(AssertionError):
```

```
    charter.add_stakeholder(stakeholders[0])

mocked_is_class_stakeholder.return_value = True
charter._ProjectCharter__finalize_add_stakeholder = Mock()
charter.add_stakeholder(stakeholders[0])
charter._ProjectCharter__finalize_add_stakeholder.assert_called_with(
    stakeholders[0])

charter.add_stakeholder(stakeholders)
expected_calls = (call(stakeholder) for stakeholder in stakeholders)
charter._ProjectCharter__finalize_add_stakeholder.assert_has_calls(
    expected_calls)
```

## B.6 tests/test_project/test_project_stakeholder.py

```
from unittest.mock import Mock

import pytest

from project.project_stakeholder import Stakeholder
from tests.faker import faker

FIRST_NAME = faker.first_name()
LAST_NAME = faker.last_name()
EMAIL = "{}{}@example.com".format(FIRST_NAME, LAST_NAME).lower()
RESPONSIBILITY = faker.text()
TITLE = faker.job()


@pytest.fixture
def stakeholder(stakeholder_factory):
    yield stakeholder_factory.build(first_name=FIRST_NAME, last_name=LAST_NAME)


def test_Stakeholder(stakeholder: Stakeholder):
    assert isinstance(stakeholder, Stakeholder)
    assert stakeholder.first_name == FIRST_NAME
    assert stakeholder.last_name == LAST_NAME
    assert stakeholder.email == EMAIL
    assert not stakeholder.projects_involved
    assert stakeholder.stakeholder_type == "unknown"
    assert stakeholder.role == "UNSET"
    assert str(stakeholder) == f"Stakeholder({stakeholder.first_name})"
    assert stakeholder.get_full_name() == f"{stakeholder.first_name} {stakeholder
        .last_name}"


@pytest.mark.parametrize("attr", ["responsibility", "stakeholder_title"])
def test_Stakeholder_attr_raises_when_not_set(attr, stakeholder):
    with pytest.raises(AttributeError):
```

```
        getattr(stakeholder, attr)


@pytest.mark.parametrize("attr, value", [("responsibility", RESPONSIBILITY), ("
    stakeholder_title", TITLE)])
def test_Stakeholder_attr_not_raises_when_set(attr, value, stakeholder):
    setattr(stakeholder, attr, value)
    assert getattr(stakeholder, attr) == value


@pytest.mark.parametrize(
    "attr, value",
    [("stakeholder_type", "not_allowed_value"), ("role", "not_allowed_value")],
)
def test_Stakeholder_cannot_set_attr_not_in_allowed_values(attr, value,
    stakeholder):
    with pytest.raises(AssertionError):
        setattr(stakeholder, attr, value)


@pytest.mark.parametrize("allowed_value", ["professor", "student", "unknown"])
def test_Stakeholder_can_set_type_in_allowed_values(allowed_value, stakeholder):
    setattr(stakeholder, "stakeholder_type", allowed_value)
    assert getattr(stakeholder, "stakeholder_type") == allowed_value


@pytest.mark.parametrize("allowed_value", ["R", "A", "C", "I"])
def test_Stakeholder_can_set_role_in_allowed_values(allowed_value, stakeholder):
    setattr(stakeholder, "role", allowed_value)
    assert getattr(stakeholder, "role") == allowed_value


def test_add_project_to_stakeholder(stakeholder: Stakeholder):
    fake_project1 = Mock()
    fake_project2 = Mock()
    stakeholder.add_to_involved_projects(fake_project1)
    assert stakeholder.get_number_of_projects() == 1
    stakeholder.add_to_involved_projects(fake_project2)
```

```
    assert stakeholder.get_number_of_projects() == 2
```

## B.7 tests/test_project/test_project.py

```
from unittest.mock import Mock

import pytest

from project import Project
from project.deliverables import Deliverable
from project.governance import ProjectGovernance


@pytest.fixture
def project(stakeholder_factory, monkeypatch):
    stakeholders = stakeholder_factory.create_batch(5)

    mock_project_charter = Mock(spec=["project_stakeholders", "project_title"])
    monkeypatch.setattr(mock_project_charter, "project_stakeholders",
        stakeholders, raising=True)
    monkeypatch.setattr(mock_project_charter, "project_title", "Fake Project",
        raising=True)

    project = Project(project_charter=mock_project_charter)

    yield project


def test_Project(project: Project):
    assert str(project) == "Fake Project"
    assert isinstance(project.project_charter, Mock)
    assert hasattr(project.project_charter, "project_stakeholders")
    assert hasattr(project.project_charter, "project_title")
    assert isinstance(project.project_governance, ProjectGovernance)
    assert project.deliverables == []
    assert project.project_risks == []


def test_Project_create_deliverable(project: Project):
```

```
    deliverable = project.create_deliverable(deliverable_name="Fake deliverable")

    assert isinstance(deliverable, Deliverable)
    assert len(project.deliverables) == 1
    assert project.deliverables[0] == deliverable


def test_Project_accept_reject_deliverable(project: Project):
    deliverable = project.create_deliverable(deliverable_name="Fake deliverable")
    project.accept_deliverable(deliverable=deliverable)
    assert deliverable.accepted is True

    project.reject_deliverable(deliverable=deliverable)
    assert deliverable.accepted is False


def test_Project_add_new_project_risk(project: Project):
    risk = Mock()
    assert project.project_risks == []
    project.add_new_project_risk(risk=risk)
    assert project.project_risks == [risk]
```

## B.8 tests/test_project/test_risks.py

```
import pytest

from src.project.risks import Risk
from src.project.risks.helpers import RiskCounterMeasure, RiskImpact,
    RiskProbabilty, RiskScore
from tests.faker import faker


@pytest.fixture
def risk():
    yield Risk(risk_name="Fake Risk Name", probability=50, impact=100)


def test_create_Risk_object_directly(monkeypatch, stakeholder, risk):
    assert risk.risk_name == "Fake Risk Name"
    assert risk.impact == 100
    assert risk.probability == 50
    assert risk.risk_owner is None
    assert risk.description is None
    assert risk.counter_measure is None
    assert risk.get_risk_score() == RiskScore.HIGH
    assert isinstance(risk.get_risk_score(), RiskScore)

    monkeypatch.setattr(risk, "risk_owner", stakeholder, raising=True)
    assert risk.risk_owner == stakeholder

    monkeypatch.setattr(risk, "description", "Fake Description", raising=True)
    assert risk.description == "Fake Description"


def test_cannot_set_unallowed_counter_measure(risk):
    with pytest.raises(AssertionError):
        setattr(risk, "counter_measure", "Fake counter measure")
    assert risk.counter_measure is None
```

```
@pytest.mark.parametrize("counter_measure", ["ReDuce", "prevenT", "aCCepT", "
    transfer"])
def test_can_set_allowed_counter_measures_and_value_is_case_insensitive(
    counter_measure, risk):
    setattr(risk, "counter_measure", counter_measure)
    assert isinstance(risk.counter_measure, RiskCounterMeasure)
    assert risk.counter_measure == RiskCounterMeasure(counter_measure.upper())


@pytest.mark.parametrize("wrong_classmethod", ["not_a_real_classmethod", "
    maybe_probability_high_impact"])
def test_access_to_dynamic_classmethod_not_matched_by_regex_raises(
    wrong_classmethod):
    with pytest.raises(AttributeError):
        getattr(Risk, wrong_classmethod)


@pytest.mark.parametrize(
    "class_method",
    [
        "rare_probability_high_impact",
        "rare_probability_medium_impact",
        "rare_probability_low_impact",
        "unlikely_probability_high_impact",
        "unlikely_probability_medium_impact",
        "unlikely_probability_low_impact",
        "moderate_probability_high_impact",
        "moderate_probability_medium_impact",
        "moderate_probability_low_impact",
        "likely_probability_high_impact",
        "likely_probability_medium_impact",
        "likely_probability_low_impact",
        "certain_probability_high_impact",
        "certain_probability_medium_impact",
        "certain_probability_low_impact",
    ],
)
```

```
def test_access_to_dynamic_classmethod_matched_by_regex_will_not_raise(
    class_method):
    name = faker.unique.name()
    instance = getattr(Risk, class_method)(risk_name=name)

    assert instance.risk_name == name
    assert isinstance(instance.get_risk_score(), RiskScore)

    probability, _, impact, _ = class_method.split("_")
    probability_value = RiskProbabilty[probability.upper()].value
    impact_value = RiskImpact[impact.upper()].value

    assert instance.get_risk_score() == RiskScore.get_risk_score(int(
        probability_value / 100 * impact_value))
```

## B.9 tests/test_project/test_wbs_item.py

```
from datetime import timedelta

import pytest

from project.wbs_item import Node, WBSItem

NAME = "Fake WBSItem"
DURATION = timedelta(days=56)


@pytest.fixture
def wbs_item(wbs_item_factory):
    yield wbs_item_factory.build(work_item_name=NAME, duration=DURATION)


def test_WBSItem(wbs_item: WBSItem, monkeypatch):
    assert str(wbs_item) == NAME == wbs_item.work_item_name
    assert wbs_item.work_item_name == NAME
    assert wbs_item.duration == DURATION
    assert wbs_item.lag == timedelta(days=0)
    assert wbs_item.percent_complete != 0
    assert wbs_item.percent_complete in range(1, 101)
    assert wbs_item.status is None
    assert wbs_item.objective is None
    assert wbs_item.resource is None
    assert isinstance(wbs_item.node_instance, Node)

    monkeypatch.setattr(wbs_item, "percent_complete", 50, raising=True)
    assert wbs_item.percent_complete == 50

    monkeypatch.setattr(wbs_item, "status", "IN_PROGRESS", raising=True)
    assert wbs_item.status == "IN_PROGRESS"

    monkeypatch.setattr(wbs_item, "objective", "Fake objective", raising=True)
    assert wbs_item.objective == "Fake objective"
```

```
    monkeypatch.setattr(wbs_item, "resource", "Fake resource", raising=True)
    assert wbs_item.resource == "Fake resource"


def test_WBSItem_duration_cannot_be_deleted_or_modified(wbs_item: WBSItem):
    with pytest.raises(AssertionError):
        setattr(wbs_item, "duration", timedelta(days=4))

    with pytest.raises(Exception):
        delattr(wbs_item, "duration")
```

## B.10 tests/test_project/test_wbs.py

```
from datetime import datetime, timedelta
from typing import List

import pytest

from project.wbs import WBS, Node
from project.wbs_item import WBSItem

WBS_NAME = "Fake WBS"
FIXED_START_DATE = datetime(2022, 1, 14, 11, 26, 3)


@pytest.fixture
def wbs():
    yield WBS(name=WBS_NAME, start_date=FIXED_START_DATE)


def test_WBS(wbs: WBS):
    assert str(wbs) == WBS_NAME == wbs.name
    assert wbs.wbs_items == []
    assert wbs.start_date == FIXED_START_DATE
    assert wbs.finish_date == "N/A"
    assert wbs.get_critical_path() is None
    assert isinstance(wbs.parent_node, Node)

    with pytest.raises(AssertionError):
        """
        Accessing this atribute before adding WBSItems should raise an assertion
            error
        """
        assert wbs.percent_complete


def test_WBS_add_items(wbs: WBS, wbs_item_factory):
    number_of_items = 100
```

```
    wbs_items: List[WBSItem] = wbs_item_factory.build_batch(number_of_items)
    wbs.add_wbs_item(*wbs_items)
    assert wbs.percent_complete == sum(x.percent_complete for x in wbs_items) //
        number_of_items
    assert wbs.wbs_items == wbs_items
    assert len(wbs.wbs_items) == number_of_items


def test_WBS_link_nodes(wbs: WBS, wbs_item_factory):
    node1 = wbs_item_factory.build(work_item_name="node1", duration=timedelta(
        days=5))
    node2 = wbs_item_factory.build(work_item_name="node2", duration=timedelta(
        days=3))
    node3 = wbs_item_factory.build(work_item_name="node3", duration=timedelta(
        days=2))
    node4 = wbs_item_factory.build(work_item_name="node4", duration=timedelta(
        days=9))
    node5 = wbs_item_factory.build(work_item_name="node5", duration=timedelta(
        days=6))
    node6 = wbs_item_factory.build(work_item_name="node6", duration=timedelta(
        days=8))
    node7 = wbs_item_factory.build(work_item_name="node7", duration=timedelta(
        days=4))

    # must call `.add_wbs_item()` first and add all items
    wbs.add_wbs_item(node1, node2, node3, node4, node5, node6, node7)
    assert len(wbs.wbs_items) == 7

    wbs.add_wbs_item(node1, node2) # duplicate not added
    assert len(wbs.wbs_items) == 7

    # assuming the graph is this below

    #           /                 node3node6
    #          /              /     \
    # node1---node2-------node4---    node7
    #           \         \        /
    #            \__________node5_____/
```

```
# The corresponding node pairs is this below
node_pairs = (
    (node1, node2),
    (node2, node3),
    (node3, node6),
    (node6, node7),
    (node2, node4),
    (node4, node6),
    (node2, node5),
    (node4, node5),
    (node5, node7),
)
wbs.link_nodes(*node_pairs)
assert wbs.duration.days == 29
assert wbs.get_critical_path() == "node1 -> node2 -> node4 -> node6 -> node7"

# test cannot call `.link_nodes() with a node not added to wbs_items
node8 = wbs_item_factory.build(work_item_name="node8", duration=timedelta(
    days=4))
node9 = wbs_item_factory.build(work_item_name="node9", duration=timedelta(
    days=44))
with pytest.raises(KeyError):
    wbs.link_nodes((node8, node9))
```

## B.11 tests/test_utils/test_generic_document.py

```
from datetime import datetime

from utils.generic_document import AbstractBaseDocument


def test_AbstractBaseStakeholder():
    document = AbstractBaseDocument(doc_title="fake_doc_title")

    assert document.doc_title == "fake_doc_title"
    assert isinstance(document.time_created, datetime)
```

## B.12 tests/test_utils/test_stakeholder.py

```
import pytest

from utils.stakeholder import AbstractBaseStakeholder


def test_AbstractBaseStakeholder_raises_when_not_subclassed():
    with pytest.raises(TypeError):
        AbstractBaseStakeholder(
            first_name="fake_first_name",
            last_name="fake_last_name",
            email="fake_email",
        )


def test_AbstractBaseStakeholder():
    class Stakeholder(AbstractBaseStakeholder):
        def get_full_name(self):
            return "%s %s" % (self.first_name, self.last_name)

    stakeholder = Stakeholder(
        first_name="fake_first_name",
        last_name="fake_last_name",
        email="fake_email",
    )

    assert stakeholder.first_name == "fake_first_name"
    assert stakeholder.last_name == "fake_last_name"
    assert stakeholder.email == "fake_email"
    assert stakeholder.get_full_name() == "fake_first_name fake_last_name"
```