CE903

(Team Number: 14)

Software Requirements Specification

Document

for

FUNNINESS PREDICTOR OF A JOKE

Version: (1) **Date: (02/09/2023)**

Table of Contents

1. Introduction

The task of detecting humour in the text is challenging, as most previous work focuses on simply judging whether a sentence is funny. This paper proposes an algorithm for detecting humorous words based on contrastive representation learning and the BERT model. We will use a pre-trained model to predict the funny grades of a comedian's joke and then restrict the differences between pairs of jokes using contrastive learning. Finally, we develop a BERT model with mixed sequence embedding to generate tables about words and their corresponding improved humour when used in conjunction with different jokes.

1.1 Purpose

This document provides an overview of the overall functionality of the software architecture. It is intended for Academia, Engineers, Project Planners, Management Personnel, and Comedians who wish to understand this underlying mechanism.

1.2 Scope

We aim to develop a model that can accurately analyze the transcripts of stand-up comedies and determine how funny the jokes are based on laughter intensity and duration.

1.3 Overview

The overall specifications can be sub-sectioned into:

- Software development or project methodology adopted
- Requirements Analysis and Specification
- Feature Extraction
- Hardware requirements
- Software Requirements
- System architecture
- System Methodology
- Non-functional Requirements
- Testing

2. The Overall Description

Humour is an intriguing subject to study because it is hard to define, so the intensity and depth of understanding vary from person to person. The same person may find something comical one day but not the next, depending on the person's mood or what has transpired to him or her recently. These factors, amongst many others, make humour recognition difficult. Although most people need clarification about the steps involved in its recognition, a computational humour recognizer must consider all these steps to approach the same faculty as a human being.

Most researchers have concluded that jokes can be divided into two components, a setup and a punchline. The setup is the initial part that sets the mood of the joke and its after-effect. The punchline is the latter part which causes some form of conflict and can be detected computationally. Computational detection, although difficult, is possible.

As reported, humour occurs because of specific brain regions engaged in reward processing. Humour is that part of the brain that helps people cope with difficult situations and adds positive emotions to dilute otherwise hard-to-deal situations.

In order to differentiate between sarcasm and seriousness, this project aims to build and design a language model that can successfully distinguish funny text segments from humourless ones while maintaining a binary classification task. The project further touches upon topics including Natural Language Processing which would help develop skills such as designing, implementing, and fine-tuning a classifier.

2.1 Software development or project methodology adopted

In this project, we design a Text Analytics Model using the field of Natural Language Processing (NLP) and Python. We use them to develop concrete skills in designing, to implement, and fine-tuning a classifier in this domain. This is to be done by obtaining samples of current stand-up comedians and the jokes they tell, and the reaction it gets from an audience. We have to obtain the joke's transcript and measure the laughter's intensity and duration. We still need to determine with complete certainty how we will obtain the transcript and laughter. The collected data will be used to train the model and validate it. The end goal of this project is that if a joke has been delivered, it will give us a number between a particular range, telling us

how funny the joke is. The model we aim to use is an LLM (Large Language Model) and will most likely be a variation of the Fine-Tuning BERT found in Hugging face. We will take this model and develop it to our specifications. The input will be a sentence (this being the joke) and give us a number as the output (how funny it is).

2.2 Requirements Analysis and Specification

The dataset is to be extracted from joke delivery audios of comedians, and the transcripts of the jokes are to be taken. Audio is then used with the help of conversion tools and transcripts, which are directly used on Text Analytics Model to obtain the output. Additionally, this data is fed to an LLM (BERT Model) to predict the optimum funniness of a joke using keywords and matching them with various other jokes.

2.3 Feature Extraction

For the datasets, we will be extracting audio and transcripts from the video of the stand-up comedian. The features obtained are the intensity and duration of laughter, which are obtained from the transcripts. Then we match the joke with laughter intensity and duration. We predict the funniness of a joke. Furthermore, in the end, rate the joke on a specific scale. Optimal jokes will be with the highest grade.

2.4 Hardware requirements for Project

The following are the recommended hardware requirements for a Project.

Minimum hardware requirements for a Front-end web server for a dataset deployment of the Project:

Component	Minimum requirement
Processor	64-bit, four-core, 2.5 GHz minimum per core
RAM	8 GB for developer or evaluation use 16 GB for single server and multiple server farm installation for production use
Hard disk	80 GB

Minimum hardware requirements for the dataset deployment of the Project:

Component	Minimum requirement
Processor	64-bit, four-core, 2.5 GHz minimum per core
RAM	8 GB for developer or evaluation use 16 GB for single-server and multiple-server farm installation for production use
Hard disk	80 GB

2.5 Software Requirements

We will design the Large Language Model (BERT Model) using Google Collaboration Environment in Python Programming Language. The Python Libraries will be NumPy, Pandas, SciKit Learn, TensorFlow, Keras, and PyTorch. The dataset collected will be in audio files and transcripts.

2.6 System Architecture

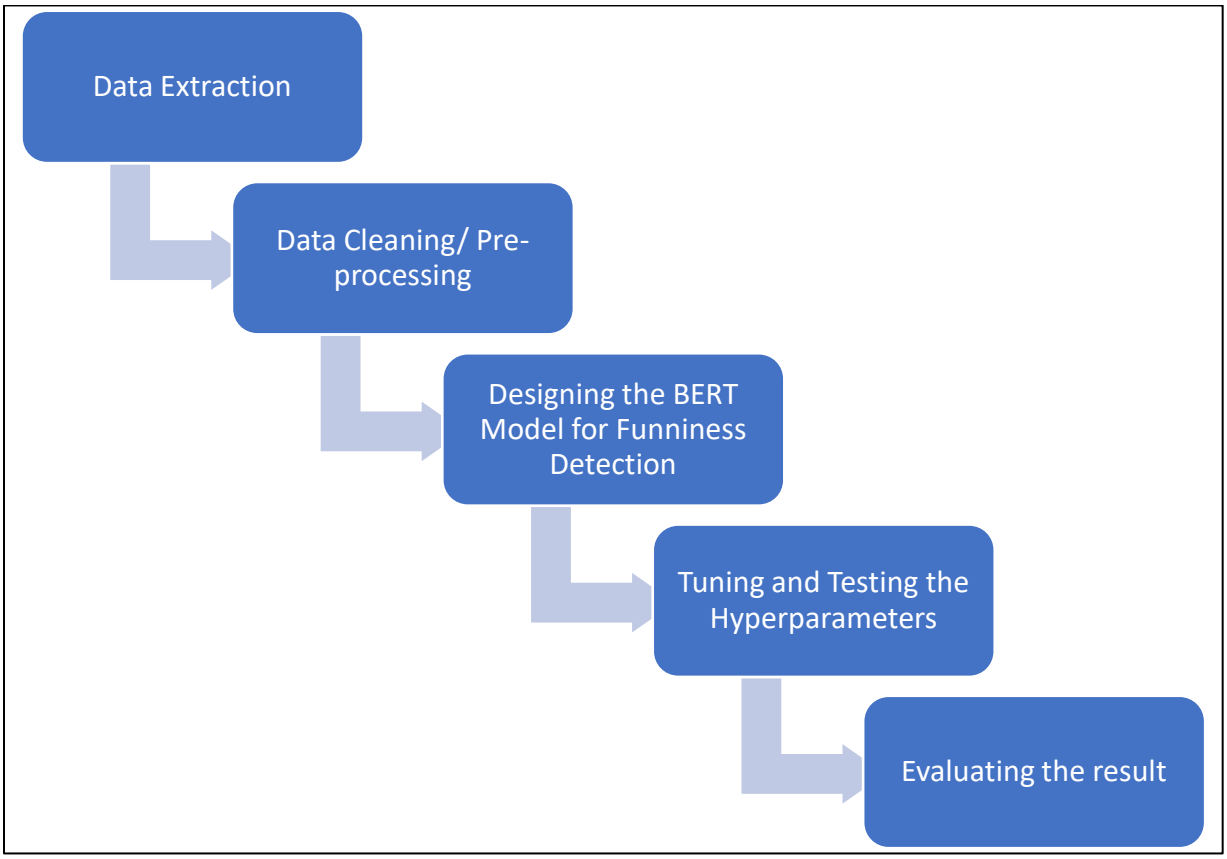


Figure1. System model

System Architecture is abstract, conceptualization-oriented, global, and focused on achieving the system's mission and life cycle concepts. It also focuses on high-level structure in system and system elements. It addresses the architectural principles, concepts, properties, and characteristics of the system of interest. It forms the typical structure, pattern, and requirements for classes or families of similar or related systems. The above figure explains our project processing in a step-by-step manner.

2.7 System Methodology

In this section, we present the workflow of our proposed system to predict the funniness of jokes, as shown in Figure 1. More details about the workflow are in the following subsections.

Present appropriate system models:

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained Transformer-based deep learning model for natural language processing tasks such as text classification, question answering, and language inference. It was developed by Google AI Language and introduced in 2018. BERT uses a bidirectional training approach to understand the context of the words in a sentence. Like traditional models, it considers every word's left and proper context instead of just the left context. This makes BERT a powerful tool for NLP tasks and has set new state-of-the-art results on several benchmarking datasets. BERT can be fine-tuned for specific tasks with a small amount of supervised data, making it a highly flexible model for various NLP applications.
BERT models come in several sizes, determined by the number of layers and the model's hidden states. Some of the most popular BERT model sizes include:

BERT Base: 12 layers (transformer blocks), 768 hidden states (representations of the input), and 110 million parameters
BERT Large: 24 layers, 1024 hidden states, and 340 million parameters

In addition to these two popular model sizes, there are also several smaller BERT models available, such as BERT Mini and BERT Tiny, as well as several even larger models, including BERT X-Large and BERT XXL. The choice of model size will depend on the specific use case and the available computational resources. In

general, larger models will be more powerful but require more computational resources and may take longer to fine-tune.

BERT Large is used in some NLP tasks because it has a more extensive model capacity than BERT Base. This means that it has more parameters and a more profound architecture, allowing it to capture more complex relationships between words in a sentence and to perform better on some NLP tasks.

The larger model capacity also allows BERT Large to benefit from pre-training on a larger corpus of text data, which can help it learn a more general language representation. This makes it a good choice for NLP tasks requiring a broad understanding of the language, such as answering questions and text classification.

However, it is worth noting that BERT Large is also much more computationally expensive and may require more time and resources to fine-tune for a specific task. Whether to use BERT Large or BERT Base will depend on the specific requirements of the NLP task and the available computational resources. In some cases, BERT Base may be sufficient, and in other cases, BERT Large may be necessary to achieve the desired results.

General Language Understanding Evaluation
Stanford Q/A dataset SQuAD v1.1 and v2.0
Situation With Adversarial Generations

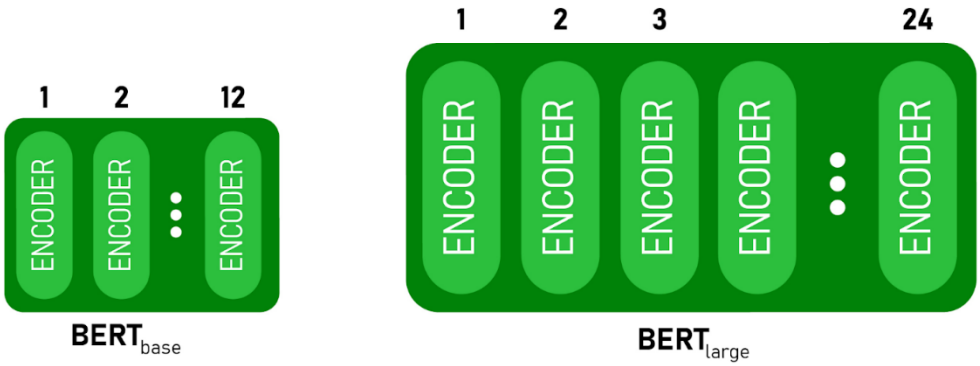


Figure 2: BASE/LARGE BERT Model

2.8 Non-functional Requirements

Non-functional Requirements define system attributes. They serve as constraints or restrictions on the system's design across the different backlogs.

1. Robust - Design focuses on improving the product's or process's fundamental function, thus facilitating flexible designs and concurrent methods. It is the most powerful method available to reduce product cost, improve quality, and simultaneously reduce development interval. Our model will be a robust system that works on a large dataset and produce predicted value.

2. Organisational - The project should be completed on time and meet the deadline. The code should be debugged. The standards should be maintained while creating the Project.

3. Performance - This states that the created model should be designed and built with an acceptable standard of performance as a minimum requirement.

4. Scalability - It refers to increasing or decreasing resources as needed to meet a project's higher or lower demands. Vertical (scale-up) scalability increases the capacity of hardware or software by adding resources to a physical system, such as adding processing power to a server to make it faster.

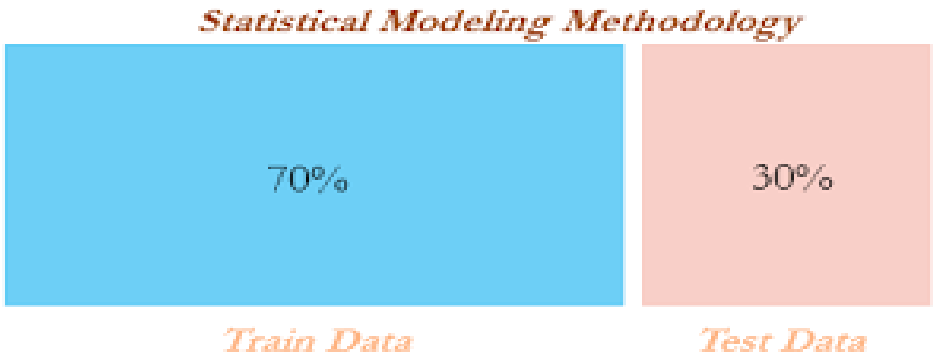
5. Usability - The specific activities the requirements describe should reflect both a range of user goals that the system must support and business goals for creating the new system. For commercial purposes, comedians can use our model to detect the funniness of their jokes.

6. Reliability - Software is considered reliable if it consistently performs tasks without failure. Software Reliability Requirements are the requirements that a software vendor has to meet in order to be able to sell software.

7. Implementation - This is the final phase of a project. After completing all the planning, coding, and testing parts, the model needs to be implemented.

2.9 Testing

We will split our dataset into Train (70%) and Test (30%).



Testing in a White Box

We will be using White Box Testing for our code testing; this is a software examination technique wherein we test the underlying structure of the product, its design, and the codes to cross-check the input-output flow, which would help in improving design, usability, and security. The term "White Box Testing" can also refer to Clear box testing (the code is visible to the testers), Code-based testing (tests are done based on specific pieces of source code instead of just running tests.

White box testing is a method used to test the functionality of software code. This process involves running the code under normal conditions and ensuring it behaves as intended. The usage specification is as follows –

- Security flaws within an organization can lead to internal damage, creating vulnerabilities that attackers can exploit.
- There are specific Paths in the coding process that can be difficult to follow or better structured. This can lead to problems down the line.
- Inputs specified by the user take the path through the program.
- What are the expected results of this action?
- Conditional loops are helpful for various purposes, including when there is uncertainty or when multiple variables need to be considered.

- Individualized testing of each statement, object, or function is essential to ensuring accurate results.
  Individualized testing of each statement, object, or function is essential to ensuring accurate results.

We will also create white box model test cases after the completion of the code.

After completing the training data, we will use the test data, tune the hyperparameters on different values, and evaluate the values of different parameters.

3. Project Planning

The project is planned with discipline, addressing the defined stages, designated resources, and listing the time frames. A work breakdown structure is created to identify the tasks at each stage of the project and can be seen in Figure 3. Work Breakdown Structure below.

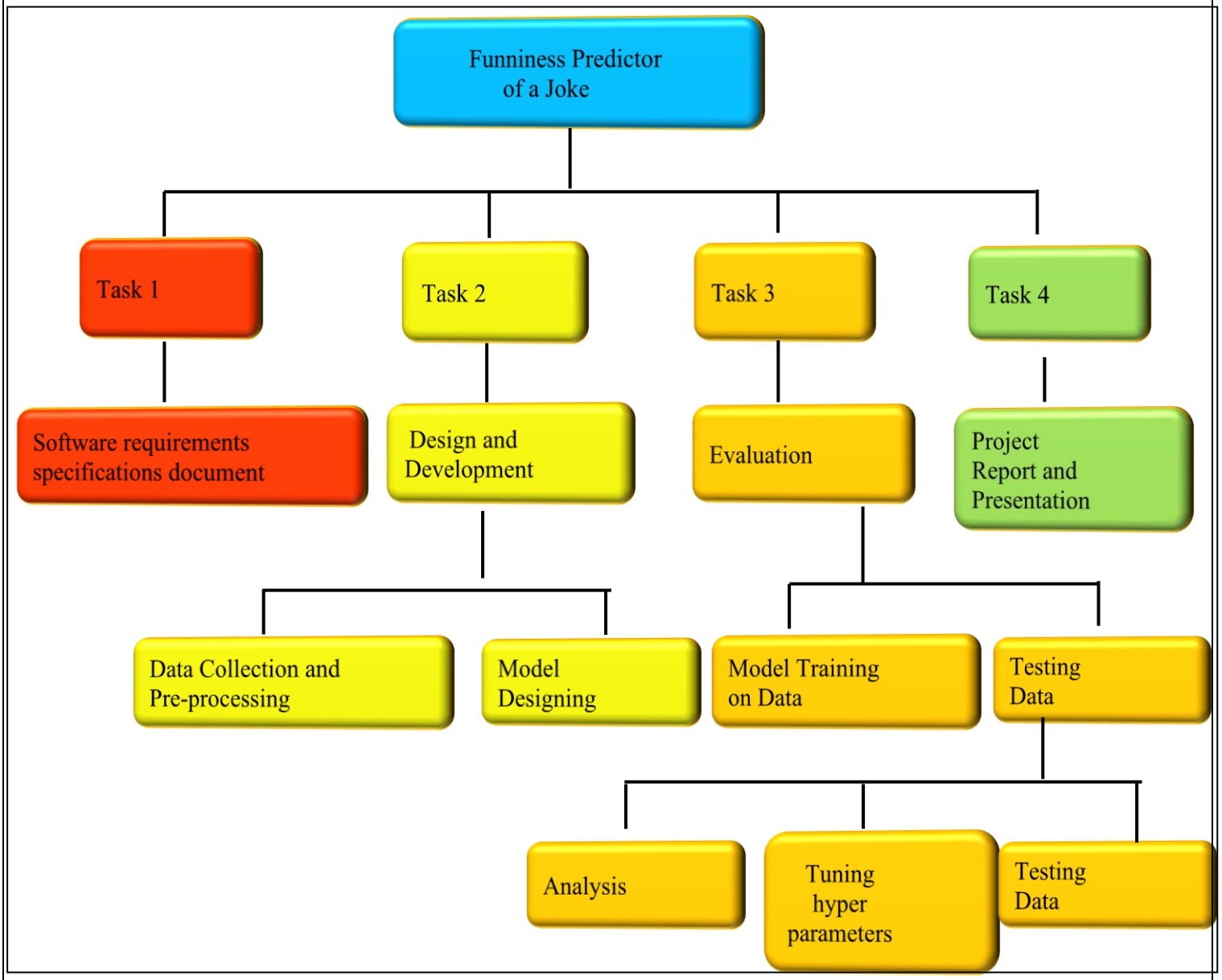


Figure 3: Work Breakdown Structure

This project has scope for iterations and adjustments to the requirements. To control this type of workflow, Agile methodology has been used for the management process. This method is implemented in the Jira tool for organizing the activities and tracking the tasks within the project. An epic for the project in Backlog is generated, and the story type for the issues for each sub-task is created. Based on the tasks divided and the structure, each story type issue has a sprint created for the task (in Figure 4. Sprints in Jira tool).

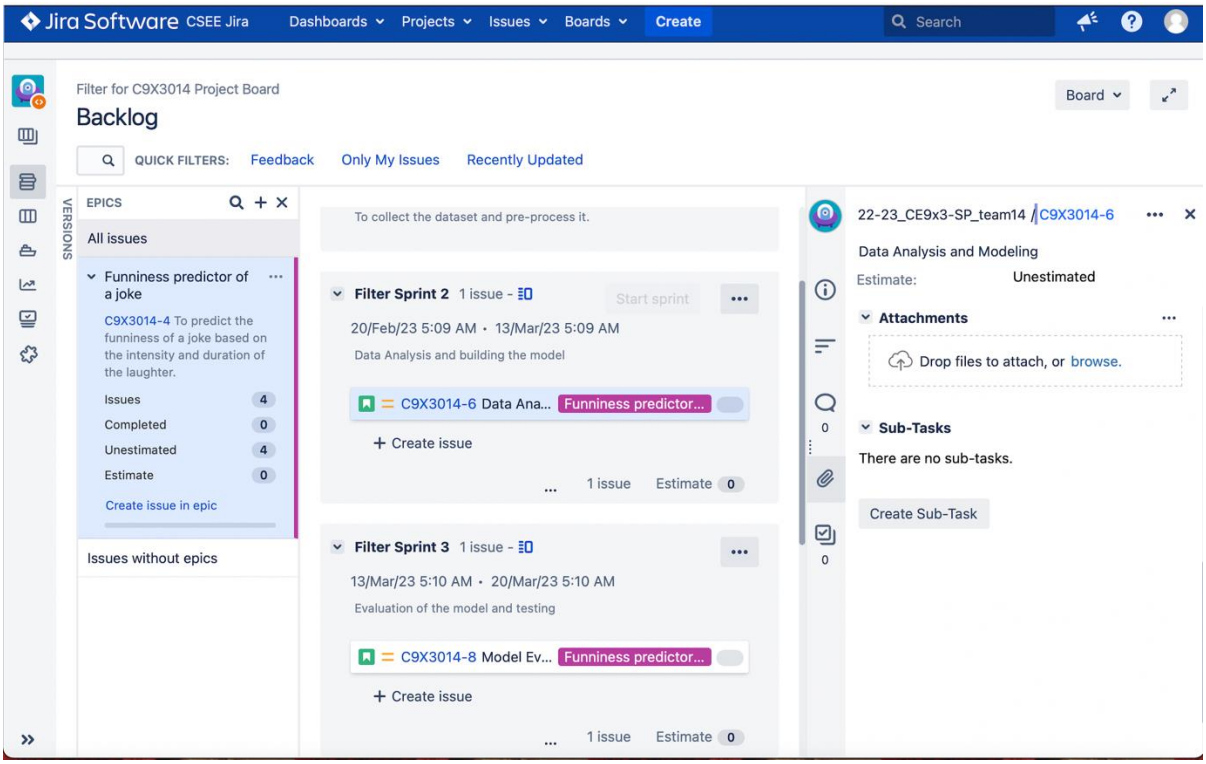


Figure 4. Sprints in Jira

While the project workflow process is managed in the Jira tool, the software developed will be controlled in the GitHub repository. Any changes and modifications to the codes are handled in this repository and will be tracked by the number of commits. An example of a team repository is shown in Figure 5. GitHub.

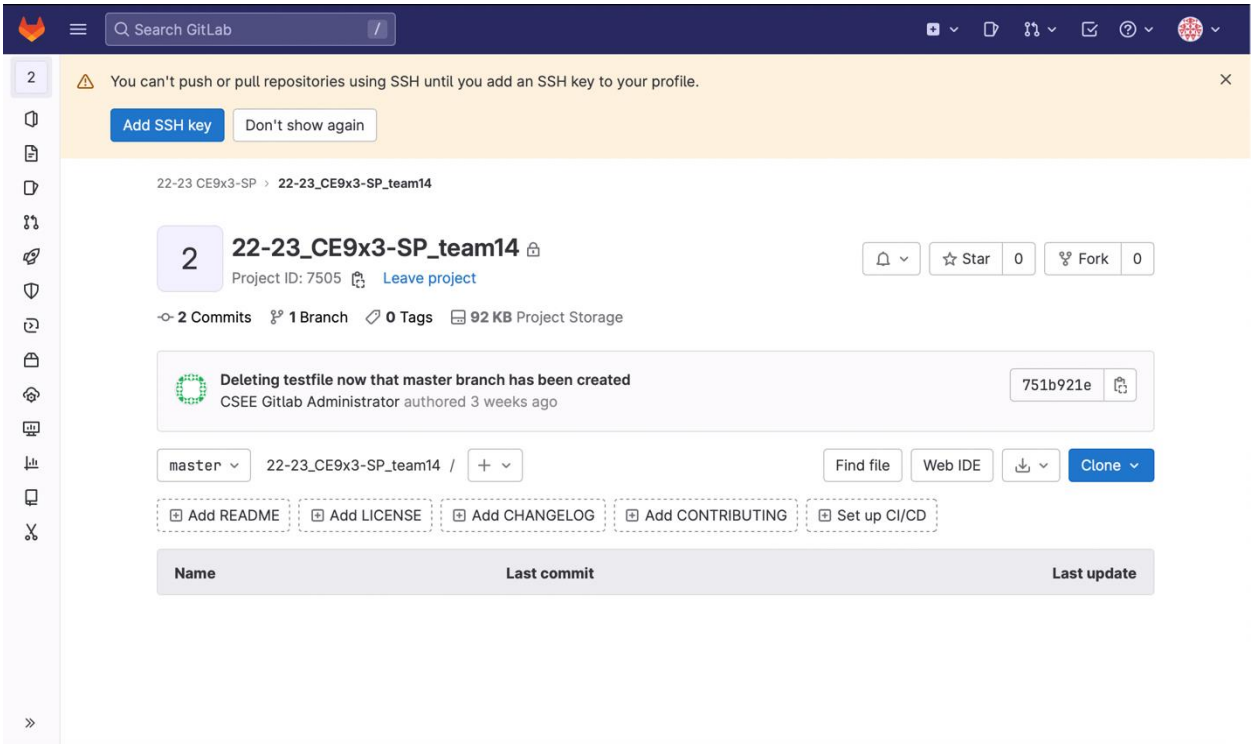


Figure 5: GitHub

The planning of the Project with stipulated timeframes, the tasks, and the duration is plotted on the Gantt chart in Microsoft's Excel sheet (Figure 6 Project Plan-Gantt chart). Each dedicated team is assigned individual responsibility for the activities/tasks shown in the table.

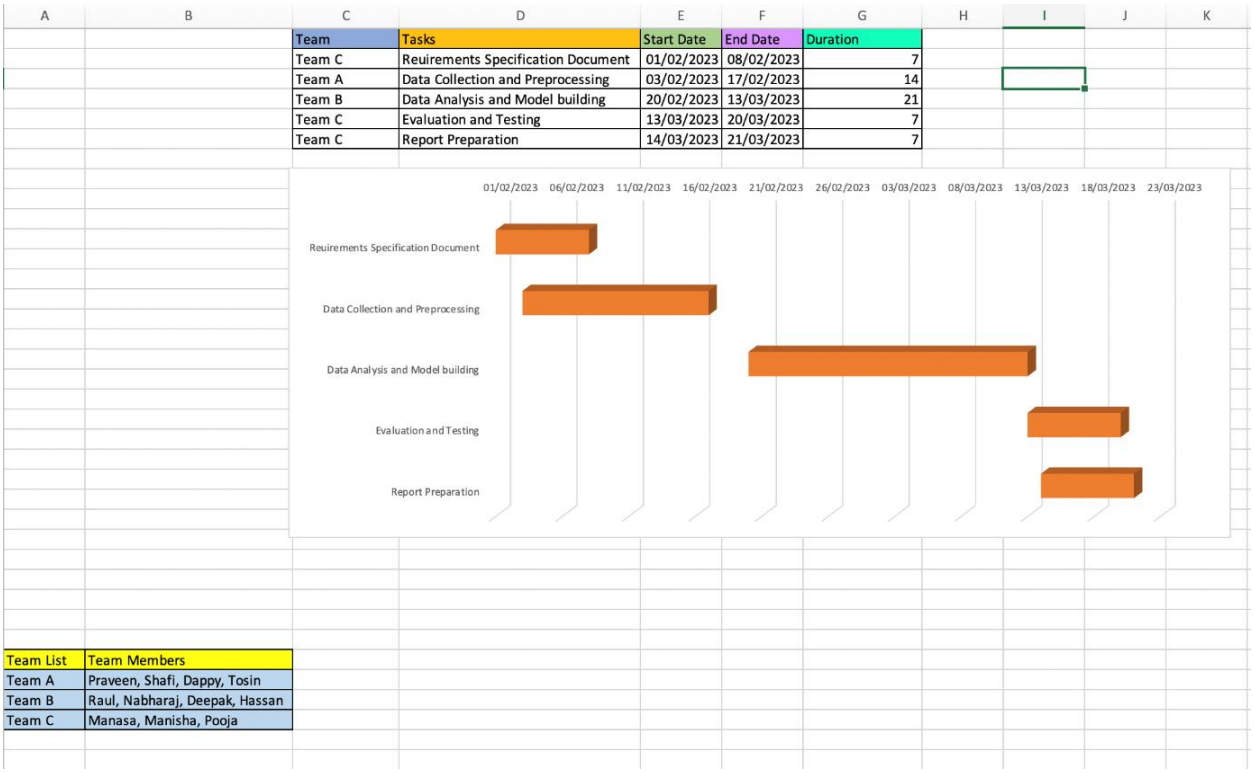


Figure 6: Project Plan- Gantt Chart

4. References

1 Y. Du and Z. Tian, "Funny words detection via Contrastive Representations and Pre-trained Language Model," *IEEE Xplore*, Oct. 01, 2021. https://ieeexplore.ieee.org/document/9725106 (accessed Feb. 08, 2023).

2 Scaled Agile, "Nonfunctional Requirements – Scaled Agile Framework," *Scaledagileframework.com*, 2018. https://www.scaledagileframework.com/nonfunctional-requirements/

3 "Reliability Requirements and Specifications," *www.weibull.com*. https://www.weibull.com/hotwire/issue80/relbasics80.htm

4 J. Nielsen, "Usability 101: Introduction to usability," *Nielsen Norman Group*, Jan. 03, 2012. https://www.nngroup.com/articles/usability-101-introduction-to-usability/

5 A. Papworth, "Non Functional Requirements | Quick Guide For The Business Analyst In 2022," *businessanalystmentor.com*, 2008. https://businessanalystmentor.com/non-functional-requirements/

6 "Scaled Agile Support," *support.scaledagile.com*. https://www.scaledagile.com/about/about-us/permissions-faq (accessed Feb. 08, 2023).

7 "System Architecture - SEBoK," *sebokwiki.org*. https://sebokwiki.org/wiki/System_Architecture (accessed Feb. 08, 2023).

8 "White Box Testing: A Complete Guide with Techniques, Examples, & Tools," *Softwaretestinghelp.com*, Feb. 02, 2015. https://www.softwaretestinghelp.com/white-box-testing-techniques-with-example/

9 D. Radev *et al.*, "Humor in Collective Discourse: Unsupervised Funniness Detection in the New Yorker Cartoon Caption Contest," *arXiv:1506.08126 [cs, stat]*, Jun. 2015, Accessed: Feb. 08, 2023. [Online]. Available: https://arxiv.org/abs/1506.08126

10 "Build software better, together," *GitHub*. https://github.com/topics/humor-detection (accessed Feb. 08, 2023).

11 Tutorials Point, "SDLC Waterfall Model," *www.tutorialspoint.com*, 2019. https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

12 "What Are the 4 Pillars of Agile? | Wrike Agile Guide," *www.wrike.com*. https://www.wrike.com/agile-guide/faq/what-are-pillars-of-agile/

13 R. Bandakkanavar, "Software Requirements Specification document with example," *Krazytech*, Jun. 10, 2021. https://www.google.com/amp/s/krazytech.com/projects/sample-software-requirements-specificationsrs-report-airline-database/amp (accessed Feb. 08, 2023).

14 "Types of Software Testing You Should Know," *TestingXperts*, Feb. 19, 2020. https://www.testingxperts.com/blog/types-of-software-testing (accessed Feb. 08, 2023).

15 F. Shatnawi, M. Abdullah, and M. Hammad, "MLEngineer at SemEval-2020 Task 7: BERT-Flair Based Humor Detection Model (BFHumor)," *ACLWeb*, Dec. 01, 2020. https://aclanthology.org/2020.semeval-1.136/ (accessed Feb. 08, 2023).

16 CE903 Module, Lecture 2