# 

**FIT 3161/ FIT COMPUTER SCIENCE PROJECT**

**Project Proposal**

**Project title: Improving Software Testing using Software Fault Prediction Methods when data is highly imbalanced**

**Team:**

FIT3161\_MA\_4

**Team Members:**

* Tah Wen Zhong
* Jason Toh Zhern Wee
* Ethan Hor Sheng Jian

**Project Supervisor:**

Golnoush Abaei

Word Count: 7944

**Table of contents**

[**1. Introduction**](#_v26dku8xrynu) **4**

[**2. Literature review**](#_x389czuixsut) **5**

[2.1 Introduction](#_p03iizxmxkuf) 5

[2.2 Prerequisite materials](#_i96p0y9ujdw8) 6

[2.2.1 Machine learning models](#_yv6jlbe6pkk4) 6

[2.2.2 K-fold Cross-validation](#_b4zgvzrerx6z) 6

[2.2.3 Software metrics](#_j93pbhtj0imz) 7

[2.2.4 Preprocessing techniques](#_8om45uzh1gr7) 7

[2.2.5 Undersampling](#_uyjuk3o27yu2) 8

[2.3 Related Works](#_lg9xanr6guek) 9

[2.4 Conclusion](#_q8cah0g3rtqn) 12

[**3. Project Management Plan**](#_ad1myphztm0q) **13**

[3.1 Project Overview](#_lwh02n2r7pwg) 13

[3.1.1 Milestones](#_lobr48fadoof) 13

[3.2 Project Scope](#_eloop9tygohb) 14

[3.2.1 Project deliverables](#_s0ghcb659aor) 14

[3.2.2 Product characteristics and requirements](#_6hu89bjfkoqy) 14

[3.2.3 Requirement Traceability Matrix](#_kmoct8xqx1vd) 15

[3.2.4 Product user acceptance criteria](#_enqyzwyq50yb) 16

[3.3 Project Organisation](#_4plj6e7lk89d) 17

[3.3.1 Process Model](#_uc2o8f1iinhm) 17

[3.3.2 Project Responsibilities](#_yr3t20sh5lau) 18

[3.4 Project Management Process](#_wxcye8a1r2pi) 20

[3.4.1 Risk Management](#_uykavlxs4fo) 20

[3.4.2 Monitoring and Controlling Mechanisms](#_euuld764xrfs) 20

[3.4.3 Communication and Reporting plan](#_uykavlxs4fo) 20

[3.4.4 Review and audit mechanism](#_uykavlxs4fo) 21

[3.5 Schedule and Resource Requirements](#_rijgi11wu19s) 23

[3.5.1 Schedule](#_lds8pxpimrq7) 23

[3.5.2 Resource Requirements](#_vbdjjmuwpbw0) 23

[**4. External Design**](#_zdhn0y4id58g) **25**

[4.1 User Interface](#_2mm22zqab4cy) 25

[4.2 External packages](#_q2fhuwuqp1ez) 28

[4.3 Datasets](#_xigkm1lcwtj5) 29

[4.4 Data Warehouses](#_mmmj7hkovxj7) 30

[4.5 Time performance](#_icqss417y9b8) 30

[4.6 Space performance](#_ollx3db55p7b) 30

[**5. Methodology**](#_k5njlh1sx8ir) **31**

[5.1 Overall description of the software](#_wana88n2qe18) 31

[5.2. Algorithm Explanation (High-level pseudocode)](#_ldaley8tpdaj) 33

[5.3 Pseudocode Methods](#_z6e48b85ayhf) 35

[5.4 Programming Language](#_3c7kl7w2pyvt) 38

[5.5 Data Management](#_owy587gwze27) 39

[5.6 Version Control](#_r6x43njlrmqj) 40

[5.7 Visualization Tools](#_k74drhurl144) 41

[**6. Testing Planning**](#_cxlirpt2ctcy) **42**

[6.1 Test coverage](#_npbnq3ihhub1) 42

[6.2 Test Methods](#_w9fprpfd039u) 42

[6.3 Sample test cases](#_52xnnhn88r02) 45

[**7. Conclusion**](#_p38gb674zd3t) **46**

[**Appendix A - Meeting Minutes**](#_fcg9mlueai90) **47**

[**Appendix B - Daily Status Update Form**](#_ng9elkc7jykr) **48**

[**Appendix C - Auditing checklist**](#_q51h5ai1aej7) **49**

[**Appendix D - Gantt chart**](#_qn06nfe9ereq) **50**

[**Appendix E - Work Breakdown Structure (WBS)**](#_ofjbnhtisql0) **51**

[**Appendix F - Requirement Traceability Matrix**](#_thq82hja06ds) **52**

[**Appendix G - Risk Register**](#_k2mzcjh0vqbx) **53**

[**References**](#_ihhzuvgckf0b) **54**

# 1. Introduction

Most software systems have security issues during their lifetime, therefore reinforcing the security of a software is an important part of the software development process (Zhang et al., 2015). When it comes to developing software, the testing phase will always play a big role in making sure that the software is defect free. As softwares become larger and more complex, the total cost of testing them becomes higher, making it hard for companies to allocate their expenses to testing when their budget is tight.

Software fault prediction models are prediction models that can help to optimize the amount of resources put into software testing. As Yucular et al. (2020) has stated, software developers can use these fault prediction models to help focus the testing on software modules that are more prone to defects. With this, the software testing phase will become more efficient in terms of cost and time taken.

The goal here is to create a software fault prediction model which is more efficient than the models found in recent research. Additionally, this software fault prediction model should have the ability to deal with the problem of data imbalance. As stated by Yen & Lee (2006), the datasets from most real-world applications are usually imbalanced, as there are usually less modules that are defect-prone compared to the ones that are not. By resolving the issue of data imbalance, the final output from our model will be a more meaningful one.

As of now, the right algorithm to develop for the goal to come to fruition has been identified. A prototype code module for the algorithm has already been created, and the project timeline has already been planned out. As for the risks of the project ,they will be dealt with as there are various ways to tackle them. With that being said, the development phase of the project will begin in the next semester.

# 

# 2. Literature review

## 2.1 Introduction

Software fault prediction models can easily find sections of software that have a higher probability of resulting in errors, therefore making use of models like these will help to save costs during the software testing phase (Pandey, Mishra, et al., 2021). As time has passed, there have been multiple renditions of software fault prediction models, with each rendition aiming to improve the efficiency of the previous version of the model. Aside from efficiency, there is also the problem of data imbalance when it comes to implementing a software fault prediction model, which needs to be dealt with as most real world datasets will contain data imbalance.

According to Boughorbel et al. (2017), data imbalance is a phenomenon that happens when the sample size in the data classes are unevenly distributed. While it is present in many real world datasets, it can also occur in software modules where only a small number of modules in a software are considered defects. Yap et al. (2013) explains that when a dataset is imbalanced, it becomes difficult to produce a good predictive model from it as there is a lack of information about the minority class. They also stated that this problem can potentially be tackled through techniques such as oversampling or undersampling the dataset, though both techniques have their own advantages and disadvantages when applied onto the dataset. Aside from resampling the data, Boughorbel et al. (2017) has also described an algorithmic strategy which involves cost-sensitive learning and boosting. This approach will attempt to learn more from the minority class by setting a high cost when the minority class is misclassified. These are just a few of the possible solutions to dataset imbalance.

In this study, we will look into the many different approaches that different researchers have taken to build an efficient software fault prediction model. Furthermore, we will also explore more solutions which deal with dataset imbalance that can be implemented in a software fault prediction model, as this is related to the scope of our project

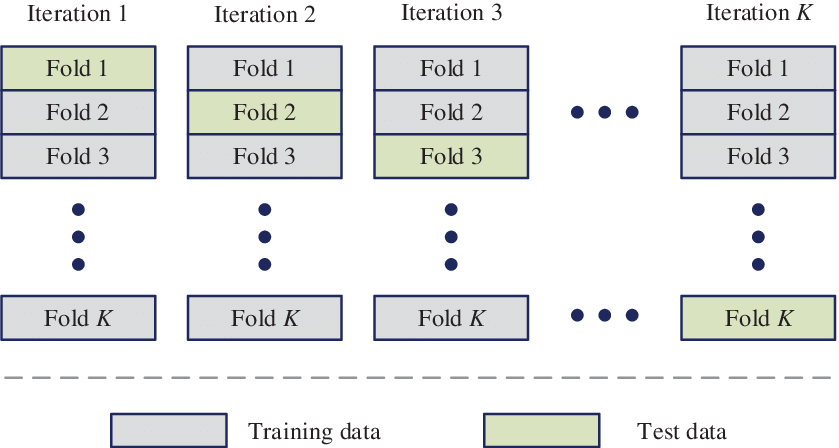
## 

## 2.2 Prerequisite materials

### 2.2.1 Machine learning models

In the context of machine learning and software testing, fault prediction is a classification task that aims to distinguish between fault prone and non-fault prone software modules defined with attributes such as static code attributes (Yucalar, Ozcift, et al., 2020). Basic machine learning models involve base classifiers such as the ones mentioned by Zhang et al. (2015), which are decision tree, random forest, Naive Bayes, Complement Naive Bayes etc. Though these machine learning models are sufficient, there are more effective methods to creating a model, such as ensemble methods. An ensemble algorithm can be categorized to be either homogeneous or heterogeneous. In general, homogenous types are more commonly practiced, but a heterogeneous type while being more complex will outperform most heterogeneous algorithms with the optimal ensemble combination (Haque, Noman et al., 2016). One example of a heterogeneous algorithm would be by Zhang et al. (2015), in which they devised their own ensemble method through building 6 models from 6 different base classifiers, before combining the prediction results of all 6 models to construct the main model. Additionally, Yucalar et al. (2020) studies revealed findings of the properties and effectiveness of many known ensemble predictors from literature of various types which include boosting, bagging and combination rule.

### 2.2.2 K-fold Cross-validation



**Fig. 1 Visualization of K-Fold cross validation (Li, n.d.)**

K-Fold Cross-validation is a resampling technique used to evaluate the skill of machine learning models. It is generally used in applied machine learning to compare and select a model for a given predictive modelling problem (Pandey, Mishra, et al., 2021). It accepts a single parameter k that indicates the number of groups of a given data sample should be split into. Its popularity comes from the fact that the algorithm is intuitively easy to understand and implement. In fault prediction models, K fold cross validation has seen its usage in evaluating the average performance of the classifiers that are used to build the models. The process involves selecting a parameter k, most notably 10, and it splits the dataset into 10 equal-sized folds where 9 folds will be used for training and the remaining fold will be used to test for effectiveness (Zhang et al., 2015). The 9 folds will then be subdivided into 9 folds and 1 fold until every fold is a test set.

Fig. 1 shows a visual representation of how the training and test sets are split. As shown, the number of folds and splits are determined by the k value.

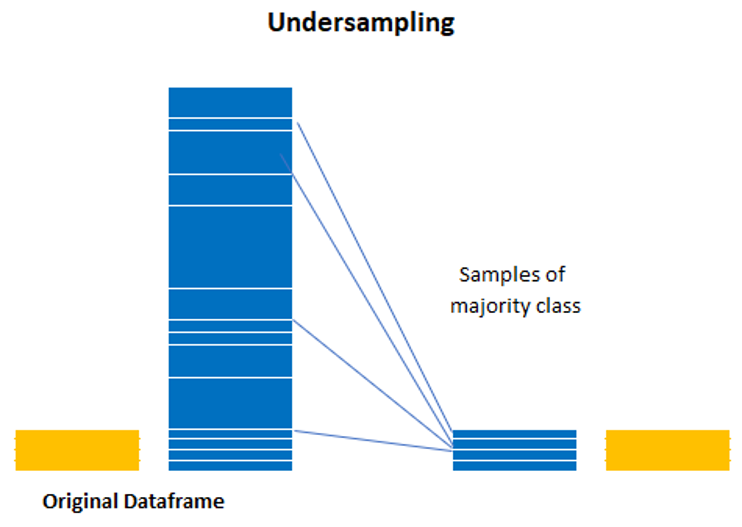
### 2.2.3 Software metrics

Software metrics are the measure of a software’s properties that are quantifiable which will be used to train predictors. Software fault prediction models are built through various software metrics which include static code metrics and process metrics (Tong, Liu, et al. 2018). In recent years, the performance of fault prediction software has been viewed to have a strong association with the selection of software metrics. As such, there has been much research on the relation of software metrics and software defects (Pandey, Mishra et at., 2021). Such research has led to claims such as one by practitioners Lessmann et al. (2008, as cited in Pandey et al, 2021) which states how the selection of software metrics have a major impact on the performance of software fault prediction models. This claim has been further supported by many recent studies. Tong et al. (2018) reveals how the majority of extracted software metrics are redundant and correlational, making a negative impact towards performance.

### 2.2.4 Preprocessing techniques

A common problem in machine learning is the quality of datasets, there are several issues on quality which primarily includes the existence of duplicate data, missing data and noise. As such, data preprocessing is usually performed to address such issues. In a study by Rana et al. (2015), their proposed model revealed to have a significant improvement after preprocessing being performed. While these preprocessing techniques do not address the issue of imbalanced data, it has proven to provide drastic improvements to the overall performance for models. Another instance of this would be within the studies by Haque et al. (2016) proposed approach included several preprocessing techniques which are deletion of duplicated instances, replacement of missing values and data normalization. Data preprocessing when used appropriately shows undeniable benefits towards the improvements in machine learning models, but every technique can only address a single issue regarding quality.

### 2.2.5 Undersampling



**Fig. 2 Visualization of the undersampling technique (Fuchs, 2016)**

From our studies, we learned the existence techniques which directly address the imbalanced class issue, mainly data undersampling. Undersampling techniques are common and most effective for handling imbalanced datasets; it involves reducing the number of samples which have the majority class (Yen & Lee, 2006). The one-vs-all approach used by Haque et al. (2016) which is a form of undersampling that produces several datasets for each class. A study by Le et al. (2018) performs under-sampling using the IHT which is the likelihood of misclassifying the instance of data. By using this concept, they devised an undersampling approach which resamples the unbalanced dataset based on the IHT values. The higher the IHT, the more undersampling is done to balance the dataset.

Fig. 2 shows a graphical representation of the undersampling effects. We can observe that the samples from the majority class are reduced to balance the size of both classes.

## 

## 2.3 Related Works

From the research articles gathered, the findings of these articles have been categorised and summarized in the following tables. These findings will help us in pinpointing the general weaknesses of using such algorithms and what proposed solutions will help in resolving those issues when developing our fault prediction algorithm.

**Table 1: Compilation of research articles that are relevant to our topic**

| **Source** | **Technique** | | | **No. of features** | **Methods** | **Strength** | **Weakness/Future work** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sup** | **Semi** | **Unsup** |
| Zhang et al. (2015) | ✔ |  |  | 8 Procedural metrics  4 Object oriented metrics | * Preprocessing to tokenize the source code file * Tokens extracted in the preprocessing phase is used to build 6 classifiers * The 6 classifiers will be used to build a composer to determine if a source code file is vulnerable or not | * Prediction models that are built from text features perform more efficiently than models built from code metrics * VulPredictor approach has an impressive training time, faster than most ensemble predictors as well as certain base predictors | * Investigate the effectiveness of code smells to be used as features * Unlike most ensemble predictors, the approach is fixed which makes it less flexible than other predictors |
| Le et al. (2018) |  | ✔ |  | 7 Measurement metrics | * 5-fold-cross-validation * Four sections of the dataset are resampled through undersampling, and are then used to train the prediction model. Then, the trained prediction model is used to predict bankruptcy for the remaining 5th dataset | * Having the ability to manipulate the dataset through undersampling is a great tool. Through making use of a concept called Instance Hardness Threshold (IHT), the dataset is resampled by removing instances of data that are more likely to be misclassified. | * Since the algorithm used here is used for bankruptcy prediction, the algorithm we develop to resample the training data should be made compatible with datasets related to software metrics, as the approaches to dealing with bankruptcy prediction and software fault prediction are different. |
| Yucalar et at. (2020) | ✔ |  |  | 17 Procedural metrics  4 Object oriented metrics | * 10-fold-cross validation * Combining base predictors using ensemble predictions | * The ensemble prediction model outperforms the base prediction models by a large margin, obtaining better FM and AUC values. * Ensemble predictors are flexible, the base predictors can be added and removed to obtain different results.. | * The model performance should be compared between the ensemble predictions as well as the predictions from the base predictors it consists of. * The model is target towards achieving better performance for general cases, will require modifications to handle imbalance datasets |

**Table 1: Compilation of research articles that are relevant to our topic (continued)**

| **Source** | **Technique** | | | **No. of features** | **Methods** | **Strength** | **Weakness/Future work** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sup** | **Semi** | **Unsup** |
| Pandey et at. (2021) | ✔ |  |  | 9 Object oriented metrics | * A compilation of software prediction methods devised from other projects | * The article contains the results of many different projects regarding software predictions which becomes a good source for comparison when evaluating our project * The article includes common components used for software fault prediction as well as a detail explanation for each component | * Commonly used components are not necessarily the most effective ones |
| Boughorbel et al. (2017) | ✔ |  |  | 2 Measurement metrics | * In certain cases where the dataset is imbalanced but also small, performing undersampling would not be very desirable. * This article proposes a method which attempts to optimize Matthew’s Correlation Coefficient (MCC), a performance metric defined in terms of True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN), to deal with evaluating the performance of a machine learning algorithm on an imbalanced dataset. | * The performance metric created through the optimization of MCC is simple and efficient, therefore requiring less computational power compared to other performance metrics. | * A good classification model is needed to correctly construct the confusion matrix which is needed for MCC |
| Tong et al. (2018) | ✔ |  |  | 13 Procedural Metrics | * Uses a deep learning model called Stacked denoising autoencoders (SDAEs) to extract deep representations from the traditional software metrics * Uses two-stage ensemble (TSE) to deal with class imbalance and overfitting problems * 5 Fold Cross Validation | * The two-stage ensemble (TSE) uses ensemble classifiers as base learners and combines them using weighted average probabilities develops an effective and efficient method against overfitting and class imbalance compared to traditional ensemble methods | * Extend the method for cross-project defect prediction * Ensemble learning methods will have a risk of overfitting and suffer from class imbalance |

**Table 1: Compilation of research articles that are relevant to our topic (continued)**

| **Source** | **Technique** | | | **No. of features** | **Methods** | **Strength** | **Weakness/Future work** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sup** | **Semi** | **Unsup** |
| Ali Rana et al. (2015) | ✔ |  |  | 40 Procedural Metric | * Association mining is applied to get the focus itemsets * Naive Bayes classifier is developed and validated for the datasets * Model is validated for different number of bins to check stability of the preprocessing approach | * The preprocessing phase proposed in the article which segregates itemsets into Defective module and Non-Defective Modules will facilitate the later process much easier and faster as the data has been modified | * Investigate application of other binning techniques * Explore the impact of different number of bins for each variable |
| Yap et al. (2014) | ✔ |  |  | None | * Bagging   Randomly selects a datasets with N samples   * Obtain a learner from the resampled dataset * Use the predictive model to predict the cases * Combine all predicted models into aggregated model * Using a voting approach, the class that has been predicted most often is chosen * Boosting   Similar to Bagging, but instead apply the learner on the training dataset and compute the sum of the weighted errors of all training samples | * The article introduces two new ensemble methods - Bagging and Boosting * Bagging deals with variance and overfitting * Boosting deals with bias and underfitting * Both ensemble methods go hand-to-hand together and are useful to resample data when needed | * Future work   Simulation study should be carried out whereby data are generated and the different approaches are compared to obtain a conclusive decision on the best strategy to handle imbalance data   * Weakness   Time and computation is expensive as multiple learners have to be constructed in the process |
| Haque et al. (2016) | ✔ |  |  | None | * 10 fold cross validation * Genetic algorithm called GA-EoC uses the 10 fold cross validation on the set of training data to evaluate the performance of the of each candidate ensembles | * The article discusses how diverse set of base classifiers can be useful in identifying different characteristics of the training set and these classifiers are referred as heterogeneous ensembles | * Random sampling was used in the approach which can potentially cause skewness in the distribution as there is a chance it can omit a particular group of a sample |

**Table 1: Compilation of research articles that are relevant to our topic (continued)**

| **Source** | **Technique** | | | **No. of features** | **Methods** | **Strength** | **Weakness/Future work** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sup** | **Semi** | **Unsup** |
| Yen et al. (2006) |  |  | ✔ | 3 Performance Metric | * SBC (under-Sampling Based on Clustering) considers the ratio of the number of majority class to the number of minority class samples in the cluster and selects a suitable number of majority class for undersampling | * The article introduces a new technique for undersampling which is based on clustering * This further widens our techniques when choosing the appropriate sampling method * SBC was proven to not only has high classification accuracy but also fast execution time | * Selecting the appropriate ratio between the majority class and minority class can be difficult * An incorrect ratio can lead to a more biased sample and high sampling error |

## 2.4 Conclusion

From our studies, we were often reminded of the inevitability of imbalance datasets in fault prediction, well known for reducing the performance of machine learning algorithms. It is unavoidable due its frequency occurrence tied by the nature of fault proneness. In general, most standard machine learning algorithms often disregard the issue of class imbalance (Rana, Mian et at., 2015). This issue became a popular topic which led to many studies addressing this issue being introduced in recent years. This literature review presented many key concepts and findings on addressing the class imbalance issue within fault prediction as well as various fields which share this common issue. We were introduced to various machine learning algorithms with many studies showing the advantages of several ensemble algorithms, along with discussing the relation of software metrics with the performance. It has also shown us the properties of different preprocessing techniques which are revealed to have significant improvements towards addressing the class imbalance issue. From our findings, we were able to devise our own fault proneness method, incorporating many key components identified throughout our studies. As such, we believe our proposed method will show promising results and reveal valuable information towards the study of fault prediction with imbalanced datasets.

# 3. Project Management Plan

## 3.1 Project Overview

The objective of this project is to create a software that allows the user to input software datasets, then builds fault prediction models based on those datasets. After that, the prediction results of the models are shown. All of these steps will be done through a user interface. After a user inputs their datasets, they can then select their desired software metrics and classifiers for the prediction model to use. The user can create different models that use different software metrics and classifiers, so that they are able to see which model is best suited for their dataset. The prediction model with the best performance for the user’s dataset will also be displayed.

### 3.1.1 Milestones

Below are the major milestones for our project:

1. Implement preprocessing algorithms

We will implement the algorithms which handles the preprocessing on datasets such as undersampling, k-fold cross validation, filling missing values and normalisation.

1. Implement machine learning algorithms

In this stage, our selected machine learning algorithms will be implemented, this includes both base and ensemble prediction algorithms.

1. Algorithm testing

We will test our devised algorithms with our prepared test cases and provide a report.

1. User interface implementation

We will prepare the user interfaces for each stage in our software, this includes the configurations.

1. Software application

We will combine the components completed previously to form our final software.

1. Software testing

We will perform tests on our main software to ensure every component works as intended.

1. Documentation

We will write the user guide and document our evaluation on each model. Then we will compile all our documents and create our final report.

## 3.2 Project Scope

### 3.2.1 Project deliverables

Each project has its own set of deliverables that are produced throughout the course of planning for the project. The table below shows the list of project deliverables which includes various documents such as the risk register, the main software, the user manual etc.

**Table 2: Project Deliverables**

| **Deliverables** |
| --- |
| Project Prototype |
| Project Plan |
| Risk Register |
| Meeting minutes |
| Software design documents |
| Work breakdown structure |
| Gantt chart |
| Main software |
| User Manual |
| Final report |

### 3.2.2 Product characteristics and requirements

Product characteristics

When initiating the program, the main interface will be displayed with several sections, mainly for configuration purposes. The user will be able to upload datasets, configure the machine learning models to be built and set the file path if the user wants to save the evaluation result. When the user is done with the configurations, they will be brought to the next screen which allows them to select the software metrics to be used in the program. After this, the software will execute the model building and test evaluation process. Finally, the result screen will be displayed and the users will find a file containing the results saved in the file path set in the main interface.

Product requirements

**Table 3: The product’s functional requirements**

| **Functional requirements** |
| --- |
| The program can extract the software metrics within uploaded datasets correctly with appropriate variations during reading based on the file format |
| The program should ensure no duplicate and missing data is present after the data undergoes preprocessing |
| The program can undersample data correctly based on the criterias required |
| The program can identify the machine learning techniques selected as well as any configuration made by the user |
| The program can split test and training data appropriately based on the k value used during the k-fold cross validation |
| The program can build the prediction models appropriately and ensure each model were trained with the same training data |
| The program can performs all the required computation for every dataset and models selected within one run |
| The program can compile the results of every component to form the chart and result table |
| The program can store the results in the designated file path written by the user |

**Table 4: The product’s non-functional requirements**

| **Non-functional requirements** |
| --- |
| The program can divide the result screen content appropriately to ensure the data are readable |
| The helper tools within the program should contain a sufficient amount of information to guide users |

### 3.2.3 Requirement Traceability Matrix

To ensure our system fulfilled all the requirements a requirement traceability matrix was created. It is a table document which contains all the requirements for our project. It allows us to track whether every requirement is fulfilled along with the criterias to validate whether a requirement was achieved.

The requirements traceability matrix can be found in Appendix F.

### 3.2.4 Product user acceptance criteria

**Table 5: User stories and their acceptance criterias**

| **User Story** | **Acceptance Criteria** |
| --- | --- |
| As a user, I would like to be able to configure the machine learning methods and software metrics used to create the prediction models. | The program allows the user to select their desired machine learning methods as well as configure their individual properties, such as selecting the software metrics used for building the models. |
| As a user, I would like to view the evaluation results of the prediction models in various formats | The program’s result screen should be able to display the results in different forms, such as graphs and tables. |
| As a user, I would like to be able to select multiple models and datasets to compare their results to one another | The program should be able to accept multiple datasets uploaded by the user, followed by building all models selected by the users to allow comparisons to be done. |
| As a user, I would like for the program to be able to process datasets from commonly used file formats. | The program can correctly interpret and process uploaded datasets of varying file types. |
| As a user, I would like to be able to download the evaluation results into a separate file within our desired file path. | The program should have an input text field which will be used as the file path for storing the results of our program, a button would also be added for the user to save the file. |
| As a user, I would like to know the progress of the model building process and model testing process at any given time. | The program should be able to display a screen which shows the progress status of the model building and testing process. (Waiting, Ongoing, Completed) |

## 

## 3.3 Project Organisation

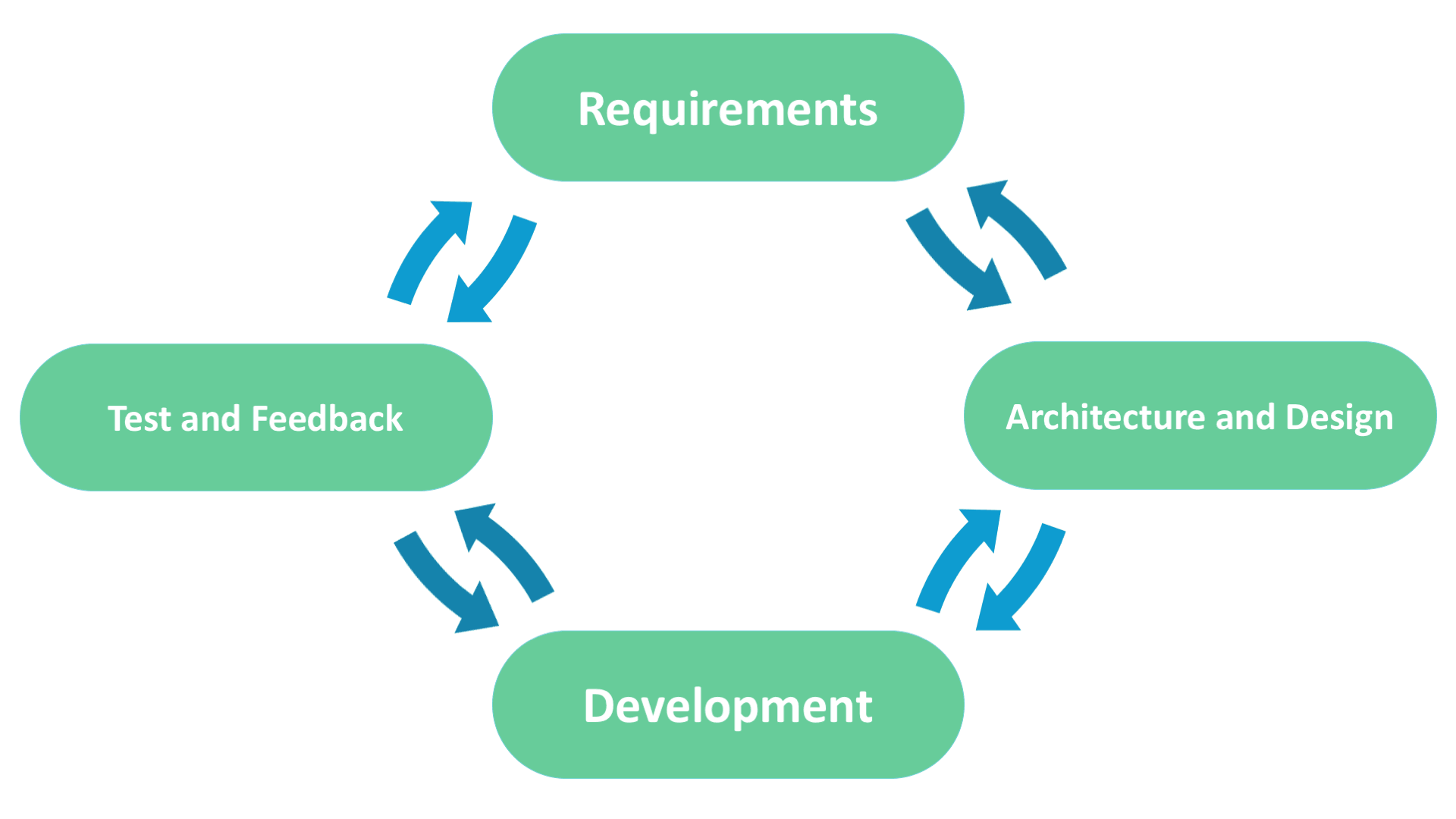
### 3.3.1 Process Model

There exists various project management life cycle methodologies and each carries their own sets of pros and cons. There is no one life cycle methodologies that is suitable for all types of projects and selecting the right lifecycle methodology is essential as the lifecycle model varies based on the suitability of the project.

Our team firmly believes agile methodology was the most suited for our project as methodology itself is flexible which allows our team to adapt to changing requirements or incorporate any new modifications during the whole project life cycle.

Another reason for this selection was due to the fact that our team did not have a clear project scope and requirements initially. In order to rectify this our team had to constantly research related articles as well as having multiple consultations with the project supervisor in order to get a clearer picture of our project. This partly contributed to our decision of selecting agile methodology because we are able to respond to new changes quickly and effectively.

One perk of agile methodology is that it ensures the project delivered is of high quality. Due to the nature of it being iterative and incremental, testing becomes much more frequent with regular check-ups to ensure that everything is working as expected. Furthermore, any issues that may arise from the project can be identified easily and resolved at the earliest.



**Fig. 3 The Agile Software Development Life Cycle Model (Existek, 2017)**

Fig. 3 shows the 4 stages along with the flow direction from one stage to the next.

### 3.3.2 Project Responsibilities

**Table 6: Project responsibilities**

| **Project functions/activities** | **Task type** | **Assigned member** | **Percentage** |
| --- | --- | --- | --- |
| Implementation of undersampling algorithm | Programming | Ethan Hor Sheng Jian | 65% |
| Tah Wen Zhong | 25% |
| Jason Toh Zhern Wee | 10% |
| Implementation of normalisation algorithm | Programming | Ethan Hor Sheng Jian | 20% |
| Tah Wen Zhong | 50% |
| Jason Toh Zhern Wee | 30% |
| Implementation of algorithm for dealing with missing data | Programming | Ethan Hor Sheng Jian | 40% |
| Tah Wen Zhong | 30% |
| Jason Toh Zhern Wee | 30% |
| Implementation of k-fold cross validation algorithm | Programming | Ethan Hor Sheng Jian | 10% |
| Tah Wen Zhong | 60% |
| Jason Toh Zhern Wee | 30% |
| Implement base prediction algorithms | Programming | Ethan Hor Sheng Jian | 30% |
| Tah Wen Zhong | 40% |
| Jason Toh Zhern Wee | 30% |
| Implement ensemble prediction algorithms | Programming | Ethan Hor Sheng Jian | 30% |
| Tah Wen Zhong | 40% |
| Jason Toh Zhern Wee | 30% |
| Algorithm testing | Testing | Ethan Hor Sheng Jian | 30% |
| Tah Wen Zhong | 10% |
| Jason Toh Zhern Wee | 60% |
| User interface implementation | Programming/  User experience | Ethan Hor Sheng Jian | 25% |
| Tah Wen Zhong | 50% |
| Jason Toh Zhern Wee | 25% |

**Table 6: Project responsibilities (continued)**

| **Project functions/activities** | **Task type** | **Assigned member** | **Percentage** |
| --- | --- | --- | --- |
| Software application | Programming | Ethan Hor Sheng Jian | 25% |
| Tah Wen Zhong | 50% |
| Jason Toh Zhern Wee | 25% |
| Software testing | Testing/  Quality assurance | Ethan Hor Sheng Jian | 30% |
| Tah Wen Zhong | 20% |
| Jason Toh Zhern Wee | 50% |
| User guide | Documentation | Ethan Hor Sheng Jian | 50% |
| Tah Wen Zhong | 25% |
| Jason Toh Zhern Wee | 25% |
| Model evaluation | Documentation | Ethan Hor Sheng Jian | 20% |
| Tah Wen Zhong | 40% |
| Jason Toh Zhern Wee | 40% |
| Final report | Documentation | Ethan Hor Sheng Jian | 40% |
| Tah Wen Zhong | 30% |
| Jason Toh Zhern Wee | 30% |

## 

## 3.4 Project Management Process

### 3.4.1 Risk Management

The goal of risk management is to prepare for the potential risks, by coming up with methods to mitigate their negative effects or maximize positive effects based on the type of potential risk. We identified the potential risks through brainstorming, and recorded the identified risk through a risk register as well as relevant information of each risk.

Brainstorming was the primary technique used to identify our project risks. As we have already identified the components of our project, we just needed to think of the potential risks for each component whilst adhering to the rules of brainstorming. When deciding on the probability, impact and potential responses for each risk, we played the Planning Poker game to help ease the process of making decisions. As for analysing the risks, we decided on the rank of each risk through the usage of a Probability/Impact matrix.

As we work on our project, more potential risks may manifest. So, we will add these risks with the same techniques and update the status of risks during our weekly meetings. With that, we can mitigate any problem occurring during the project through a predetermined course of action.

The risk register can be found in Appendix G.

### 3.4.2 Monitoring and Controlling Mechanisms

Throughout the project, we will carry out monitoring and controlling activities to make sure that the progress made is on par with the timeline created for the project. These activities (usually in the form of meetings) will also be used to keep each team member updated on each other's status. It will also be a good place for the members of the team to highlight any issues they face whilst doing the project, so that other members can help tackle the issue as well. With that being said, these monitoring and controlling activities will be carried out on a weekly basis, as well as on days where a situation arises where a meeting is needed to discuss the problems regarding the situation.

### 3.4.3 Communication and Reporting plan

For our project to be successful, we require effective communication between each team member. We have decided to use platforms such as Discord, Whatsapp, and Zoom to maintain contact with each other. Additionally, we will make use of forms to note down meeting minutes (one is created for each meeting), as well as daily updates for each member of the team. The samples of those can be found in Appendix A, B. A table that represents the various communication processes which will be carried out throughout the timeline of the project can be found below.

**Table 7: Various types of communication processes**

| **Communication** | **Method** | **Goal** | **Frequency** | **Update/**  **Deliverables** |
| --- | --- | --- | --- | --- |
| Daily standup | Meeting | Discussion to keep everyone updated on individual tasks | Daily  (excluding weekends) | Meeting minutes, Standup sheet |
| Daily status update | Status report | Notify the progress status of each member | Daily | Daily status report |
| Progress update log | Gantt chart | Updates the progress made on each deliverable for the project | Weekly | Gantt chart |
| Sprint Planning | Meeting | Discussion to decide on what is to be done for the upcoming sprint | Bi-weekly (Monday) | Meeting minutes, Scrum board |
| Sprint Review | Meeting | Discussion to reflect on what has been achieved for the current sprint | Bi-weekly  (Friday on week before the next sprint planning) | Risk Register, Meeting minutes |
| End Meeting | Meeting | Discussion to reflect on what has been done after the project timeline has reached its end | Once | Meeting minutes |

### 3.4.4 Review and audit mechanism

Code review

Every two weeks or so, a code review session will be hosted, with each team member participating in it. The main objective of these code review sessions are for team members to help each other improve themselves in terms of coding skills. This is done by having each team member share about a particular task in the project (related to coding) from the past two weeks that they have found to be interesting or challenging. After that, the other members can then provide their own input on the task that has been shared, giving feedback on ways to further improve on the implementation for the code related to that task. The structure of these sessions are not fixed, as team members are also free to share new tips and tricks that they have picked up as they continue to develop the software for the project.

Auditing

Since each sprint lasts for two weeks, each member is required to perform auditing on their code once per week. This process is done to keep track of the current condition of the code. Depending on how long their code is, this process may take up to a day to finish. To help smoothen the process, a checklist will be made for each team member to use when they are auditing their codes. These checklists will then be taken into account when planning for the next sprint. A sample of the auditing checklist can be found in Appendix C.

Version control

To manage version control, we have decided to make use of GitHub. A repository will be created there to keep track of the changes made to our program. The process is as follows:

* For each sprint, each team member will have their own branch created for the tasks that they are responsible for.
* Team members will continue to work on their own branches for the rest of the sprint. If a team member needs help on a task, the other members can conveniently switch to that branch to help out.
* When a sprint ends, each branch will then be merged back to the main branch. A meeting will be held to ensure that the merging process goes smoothly.

## 

## 3.5 Schedule and Resource Requirements

### 3.5.1 Schedule

**Deadline of the project:** Unknown

**Rough duration of the project**: 4 months

A Gantt chart as well as a Work Breakdown Structure (WBS) has been created to outline the timeline of the project. Both of these can be found in Appendix D, E.

We aim to achieve the following milestones in the current order:

1. Implement preprocessing algorithms
2. Implement machine learning algorithms
3. Algorithm testing
4. User interface implementation
5. Software application
6. Software testing
7. Documentation

### 3.5.2 Resource Requirements

Numbers and types of personnel required

The number of personnel for this project is 3 as we are a group of 3 students. Jason is the project manager of the team and focuses on managing the project and is in charge of overseeing the duties of the other team members as well as assisting in any technical development of the project. Ethan is responsible for quality assurance which involves creating test harnesses and ensuring all expected areas of the project are up to appropriate standards. Tah is the technical lead and oversees all technical development of the project and is the core backbone in implementing the fault prediction software.

Computer time

Each member of the group needs to spend an approximate 40 hours a week which may include weekends or public holidays.

Software requirement

The main software we will be utilizing in this project will be Jupyter Notebook. Jupyter Notebook incorporates a ton of libraries which are designed for creating large scale data science projects. Another reason for using Jupyter Notebook is that all of the members have familiarity with Python and have utmost confidence in programming with the said language.

The libraries we will be using are Scikit Learn, NumPy, SciPy, Matplotlib, Seaborn. These libraries are appropriate for our project as it revolves around developing machine learning algorithms which are related to our project topic.

For any front end features in our project, we decided to utilize Visual Studio Code as our main software. Visual Studio Code is a renowned IDE for building and debugging modern and web applications. It has an integrated terminal which allows fast traversal between each file as well as inbuilt Git version control.

Hardware requirement

* Laptop - HP Pavilion Laptop 15-ck0xx
* CPU - Intel Core i7-8550U Processor
* GPU - Intel(R) UHD Graphics 620
* RAM - 8GB RAM
* Storage - 512GB HDD + 128GB SSD

Maintenance requirement

* Corrective maintenance: Any faults or bugs occurred must be reported and fixed within the same time frame to avoid any problems from arising in the future
* Preventive maintenance: Regular checkup and maintenance at fixed intervals on the software and hardware to ensure all parts are working at optimal condition
* Risk-based maintenance: Assets which carries the biggest risk should be prioritised first for maintenance if they were to fail
* Condition-based maintenance: Evaluate the condition of the asset to determine if needed a maintenance or not

# 

# 4. External Design

## 4.1 User Interface

The software developed will allow users to upload their own desired datasets and allows several aspects, mainly the prediction settings to be configured based on the user’s liking. There are two main interfaces which are displayed for users to configure the several aspects as well as a result screen which showcases the outcome of the prediction.

## 

**Fig. 4 Home menu UI**

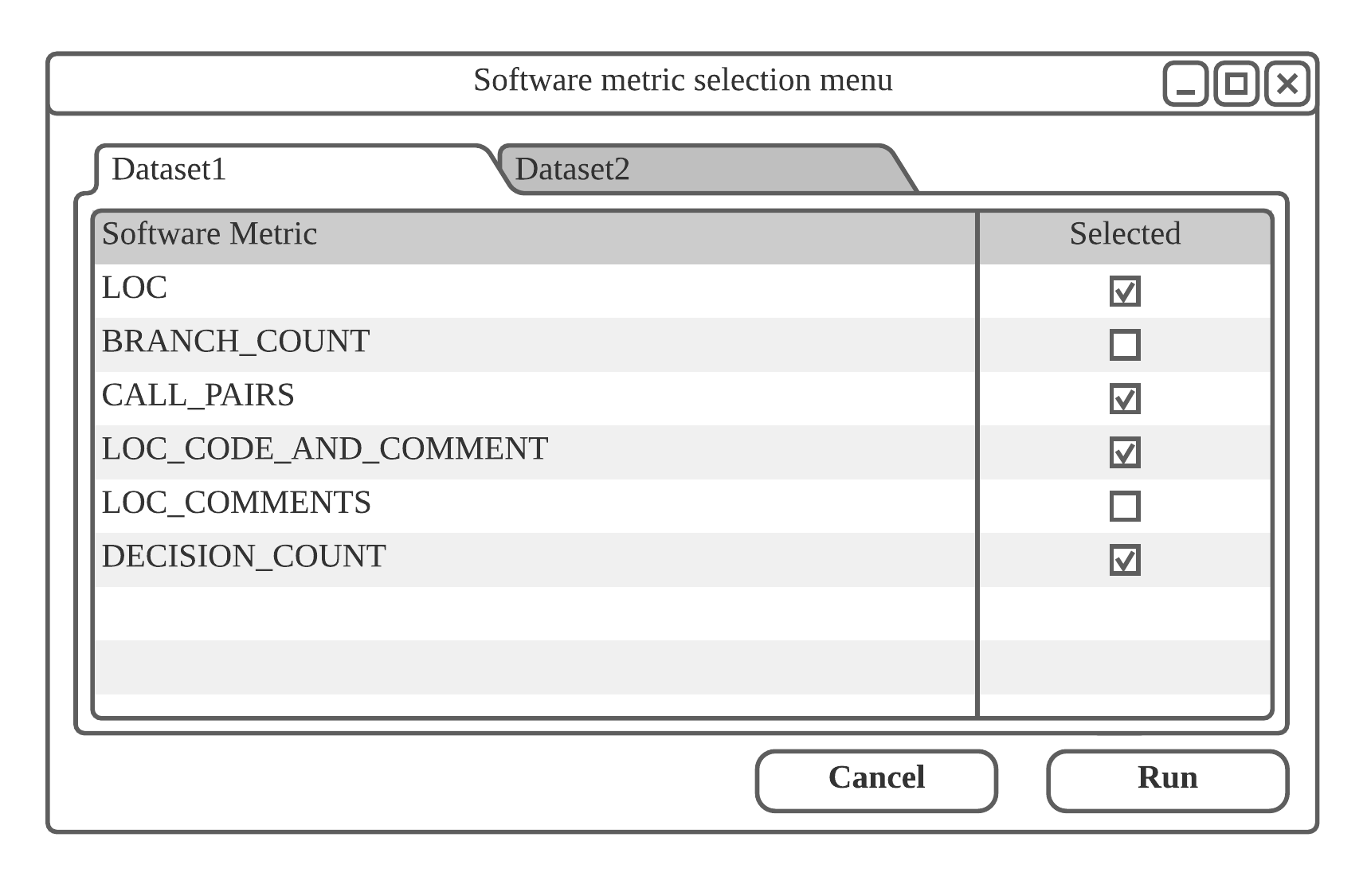
The application’s home menu will be designed as shown in Fig. 4. The users will be able to configure several aspects of the application, these configurations are available under the two settings shown, Prediction settings and Training settings:

1. Prediction settings

The user can select the base predictors that will be used by the ensemble predictor, this is under the Base tab where base predictors can be selected based on the user’s preferences. However, only one option can be selected for the ensemble predictor under the Ensemble tab. A help button will be included which will open another tab which explains the overall process and usage of every predictor.

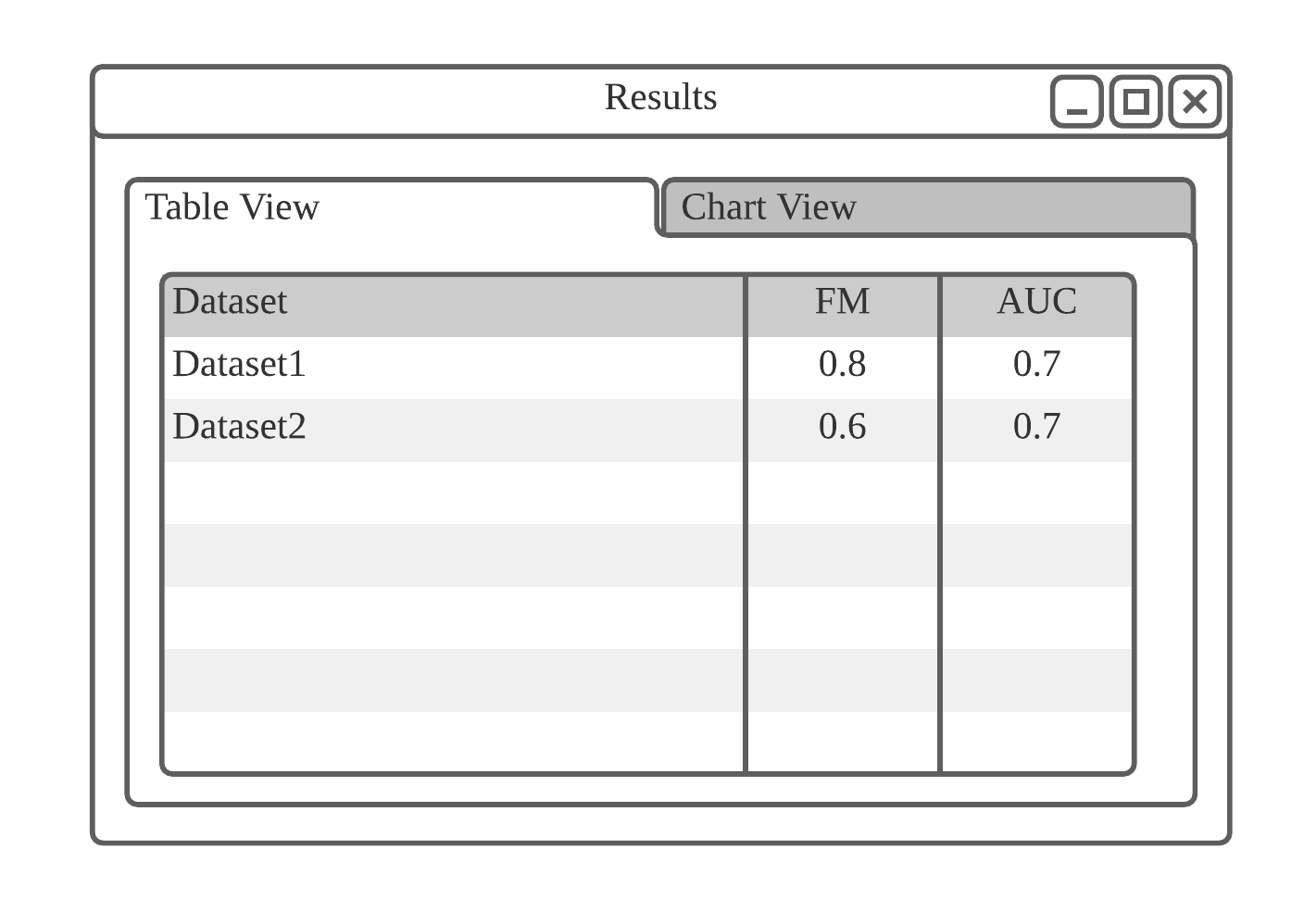
1. Training settings

The user will be able to configure the training/test split ratio for the k-fold cross validation. Multiple datasets can be uploaded and removed freely by the user, these datasets will be performed separately. The file path as well as filename can be modified by the user which determines the location and name of the file which will contain the results produced by the application.



**Fig. 5 Software metric selection menu**

After the user starts the program, the application will first read the datasets uploaded by the users and list the software metrics found in the datasets. Fig. 5 shows the interface design. The user can select the software metrics they want to include (default all will be selected) before executing the prediction process.



**Fig. 6 Result screen (Table view)**



**Fig. 7 Result screen (Chart view)**

After the prediction process, the result screen will be displayed as shown in Fig. 6 and Fig. 7. The result screen includes a chart view which includes a bar chart which shows the performance of the ensemble predictor as well as the base predictors it consists of. The table view allows users to view data in tabular form.

## 

## 4.2 External packages

Although we plan to develop our software in the Python language, the libraries that are installed by default on most devices are insufficient. Therefore, we plan to download additional external packages for Python which are related to data science. Below is a table listing all the packages being downloaded.

**Table 8: Descriptions of external packages**

| **Package name** | **Details** |
| --- | --- |
| **Scikit Learn** | A machine learning library that contains various classification, regression and clustering techniques and is designed for Python Programming Language |
| **NumPy** | A library that supports mathematical functions that operates large multidimensional arrays and matrices |
| **SciPy** | A library that is used for scientific computing and technical computing and contains modules for optimization, linear algebra as well as image processing |
| **Matplotlib** | A plotting library that creates static, animated and interactive visualizations in python |
| **Seaborn** | A plotting library that provides a high-level interface for drawing attractive and informative statistical graphics. |

## 

## 4.3 Datasets

There is a large variety of datasets that are utilized in software fault prediction. These datasets are from two publicly available data repositories, which are the NASA (Shepherd et al., 2014) and Promise (Menzies, 2004) repositories. We gathered the datasets and identified its properties as shown in the tables below:

**Table 9: Properties of datasets from NASA (Shepherd et al., 2014) repository**

| **Dataset** | **Procedural Metric** | **Object oriented Metric** | **Misc.**  **Metric** | **Metric count** | **Defective** | **Not defective** | **Module count** | **Degree of imbalance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CM1.arff | 25 | 11 | 2 | 38 | 12.84% | 87.16% | 327 | Moderate |
| JM1.arff | 17 | 4 | 0 | 21 | 20.88% | 79.12% | 7720 | Mild |
| KC1.arff | 17 | 4 | 0 | 21 | 25.3% | 74.7% | 1162 | Mild |
| KC3.arff | 25 | 13 | 2 | 40 | 18.56% | 81.44% | 194 | Moderate |
| KC4.arff  (Raw) | 24 | 14 | 2 | 40 | 48.8% | 51.2% | 125 | Normal |
| MC1.arff | 24 | 13 | 2 | 39 | 1.84% | 98.16% | 1952 | Moderate |
| MC2.arff | 24 | 13 | 2 | 39 | 35.48% | 64.52% | 124 | Mild |
| MW1.arff | 24 | 11 | 2 | 37 | 10% | 90% | 250 | Moderate |
| PC1.arff | 24 | 11 | 2 | 37 | 8.1% | 91.9% | 679 | Moderate |
| PC2.arff | 23 | 11 | 2 | 36 | 2.22% | 97.78% | 722 | Moderate |
| PC3.arff | 24 | 11 | 2 | 37 | 12.35% | 87.65% | 1053 | Moderate |
| PC4.arff | 24 | 11 | 2 | 37 | 13.86% | 86.14% | 1270 | Moderate |
| PC5.arff | 23 | 13 | 2 | 38 | 27.04% | 72.96% | 1694 | Mild |

**Table 10: Properties of datasets from Promise (Menzies, 2004) repository**

| **Dataset** | **Procedural Metric** | **Object oriented Metric** | **Misc.**  **Metric** | **Metric count** | **Defective** | **Not defective** | **Module count** | **Degree of imbalance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| kc1.arff | 17 | 4 | 0 | 21 | 15.46% | 84.54% | 2109 | Moderate |
| cm1.arff | 17 | 4 | 0 | 21 | 9.84% | 90.16% | 498 | Moderate |
| kc2.arff | 17 | 4 | 0 | 21 | 20.5% | 79.5% | 522 | Mild |
| jm1.arff | 17 | 4 | 0 | 21 | 19.35% | 80.65% | 10885 | Moderate |
| pc1.arff | 17 | 4 | 0 | 21 | 19.35% | 80.65% | 1109 | Moderate |

## 4.4 Data Warehouses

As we incorporate the use of GitHub for our collaborative work, our main algorithms and program will be stored in a cloud-based repository that will be branched to each member’s devices. As such, our project will not require a large amount of storage space, so a cloud storage such as Google Drive will suffice. We will be mainly storing the documentations made throughout the project as well as the datasets used for testing.

## 4.5 Time performance

Since the application allows the prediction methods to be configurable, the response time will be dependent on the number of prediction models being produced as well as the size of the datasets. For a general case, the processing speed will be relatively quick with the component which requires the heaviest computation being the building of models. However, assuming that the prediction configuration was tuned the maximum, the completion time will drastically decrease due to the increase in computational demands.

## 4.6 Space performance

The space required for our software will mainly be used to store the results of our tests, evaluating the performance of built prediction models for the provided datasets. As such, our program has low demand for space so a simple cloud-base storage such as Google Drive would be sufficient.

# 5. Methodology

## 5.1 Overall description of the software

**Concept**

In our research, we were able to identify and select several techniques to be used for our algorithm. The machine learning algorithms we selected are based on their effectiveness towards handling imbalanced datasets and their performance we learned in our research. In theory, there is a combination of techniques and machine learning models which would be the best method for finding fault proneness in a system when given a highly imbalanced dataset. With that in mind, we want a software that allows us to evaluate the performance of models from various combinations and configurations.

Our software will incorporate all our selected techniques involving processing the dataset and will be capable of building multiple models, allowing the performance of each model to be compared with one another. Furthermore, multiple datasets and models can be used in a single computation to assist observation and performance analysis. Lastly, ensemble models have configurable parts, so our program will allow users to input their desired settings.

List of techniques used:

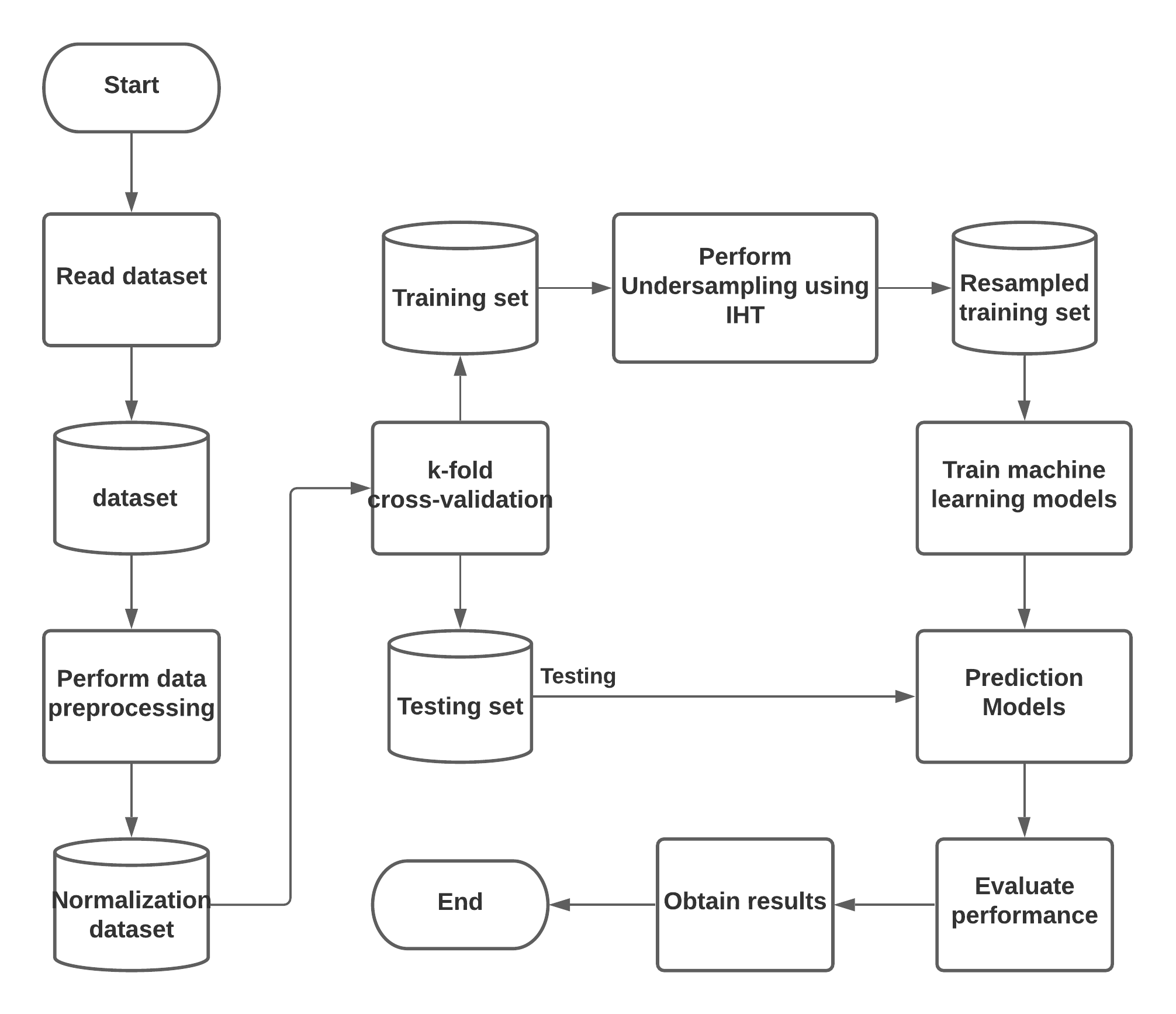
1. Missing data handling
2. Normalisation
3. Undersampling using IHT techniques
4. K-fold Cross-validation

List of machine learning models (base)

1. Multi-Layer Perceptron
2. Logistic Regression
3. Naïve Bayes
4. Complement Naïve Bayes
5. Decision Tree

List of machine learning models (ensemble)

1. Rotation Forest
2. Voting
3. Random Forest



**Fig. 8 Flow diagram**

Fig. 8 shows the workflow for the prediction process. As shown, the program will first read the dataset and perform various preprocessing on it. The train/test splits are then obtained through the usage of k-fold cross-validation. Next, the models are builts and are each evaluated. Lastly, the program will gather the results and display it to the user.

**Limitations**

There are a few assumptions and limitations for our system, which are listed in the tables below:

**Table 11: List of limitations**

| **Limitation** |
| --- |
| The system does not include a tokenization algorithm, so it is built to process datasets and will not be able to extract the software metrics from the raw data of a system. |
| The system has no features for editing the datasets |
| The machine learning models available are limited to the ones implemented in the system |
| The system can only read datasets that follow the standard dataframe structure, where all attributes have appropriate data types. |

## 5.2. Algorithm Explanation (High-level pseudocode)

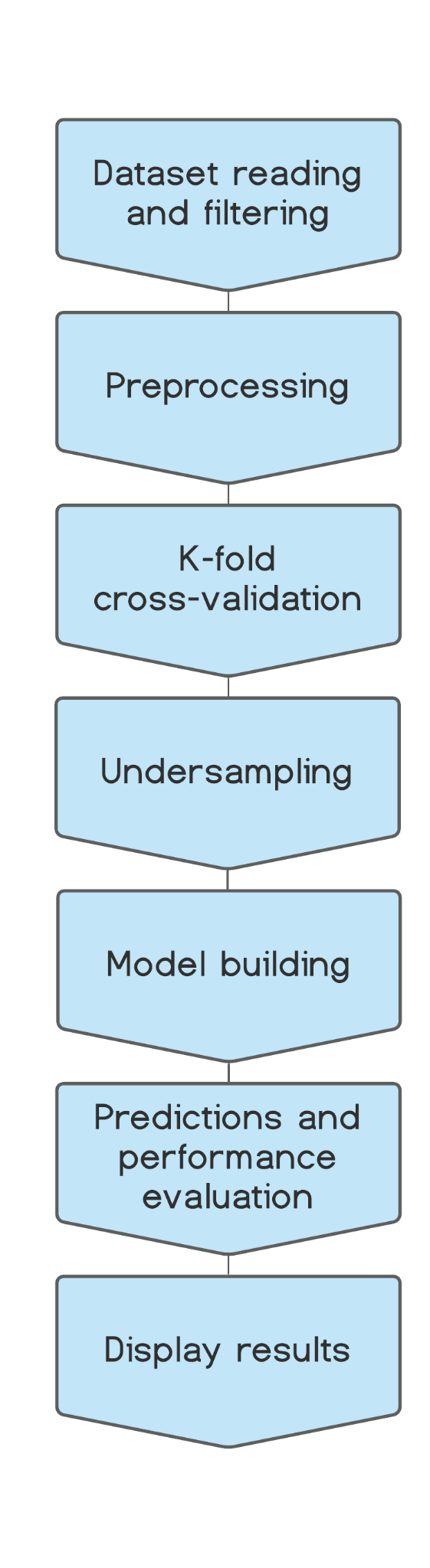
Below are the pseudocode for the main algorithms within our system, ordered based on the process steps. The readFile and filterMetrics functions are used for extracting data from the dataset. The readFile will take in the name of the dataset and search and extract the file within the directory. The filterMetrics is used to filter the metrics to be used based on the user’s selection. Next will be the preprocess function which handles the missing data appropriately based on criterias set, followed by normalising the data. Following this will be the algorithm which performs k-fold cross-validation to obtain the training/test splits based on the k value input. After this, the undersampling function will be used, the amount of data undersampled will be based on the IHT value calculated within the same algorithm. The evaluate function will calculate the AUC and F1 score based on the test set and prediction made by a model. The last algorithm will build the models selected by users, this model will be trained with the training set obtained from k-fold cross validation and evaluated using the evaluate function. The results will be gathered and shown to the user after the algorithm completes.

| **Algorithm**    **function** readFile(filename)  read(filename)  get software\_metrics  add software\_metrics in dataset  **for** line **in** dataset **do**  get attributes and label  convert label to boolean  store [attributes,label] in dataset  **end for**  **return** dataset  **end function**    **function** filterMetrics(dataset, selection\_list):  get software\_metric\_list  **for** software\_metric **in** software\_metric\_list **do**  **If** software\_metrics inselection\_list == False **then**  remove software metric and associated column in dataset  **return** dataset  **end function**  **function** preprocess(dataset)  handle missing data  perform normalization  **return** dataset  **end function**    **function** k-fold\_cross-validation(k,dataset)  perform k-fold cross validation with k folds  get training\_test\_splits  **return** training\_test\_splits  **end function**    **function** undersampling(dataset)  calculate threshold  **for** data **in** dataset **do**  calculate IHT value  **if** IHT value higher > threshold **then**  remove data from dataset  **return** dataset  **end function**  **function** evaluate(test, predictions)  calculate AUC score  add AUC score to result  calculate F1 score  add F1 score to result  **return** result  **end function**  **function** main(training\_test\_splits,decision\_list)  extract selected\_models and model\_configurations from decision\_list  **for** model **in** selected\_models  **If** model type **is** ensemble **do**  get model configurations  set model to follow configurations  **for** train\_set,test\_set **in** training\_test\_splits  train model with train\_set  get model predictions with test\_set  add results from evaluate(test\_set,predictions) to result\_list  **else do**  **for** train\_set,test\_set **in** training\_test\_splits  train model with train\_set  get model predictions with test\_set  add results from evaluate(test\_set,predictions) to result\_list  **end if**  **return** result\_list  **end function** |
| --- |

## 

## 5.3 Pseudocode Methods

The following methods describe the process of the main algorithm within the program for a single dataset. The actual program will accept multiple dataset uploaded so it will iterate through each dataset and perform the following steps for each dataset.



**Fig 9. Step process diagram**

The step process diagram in Fig 9. shows the process sequence of our main algorithm. The processes done in each step are explained below: Starting from dataset reading and filtering step and ending with displaying the results.

Step 1: Dataset reading and filtering

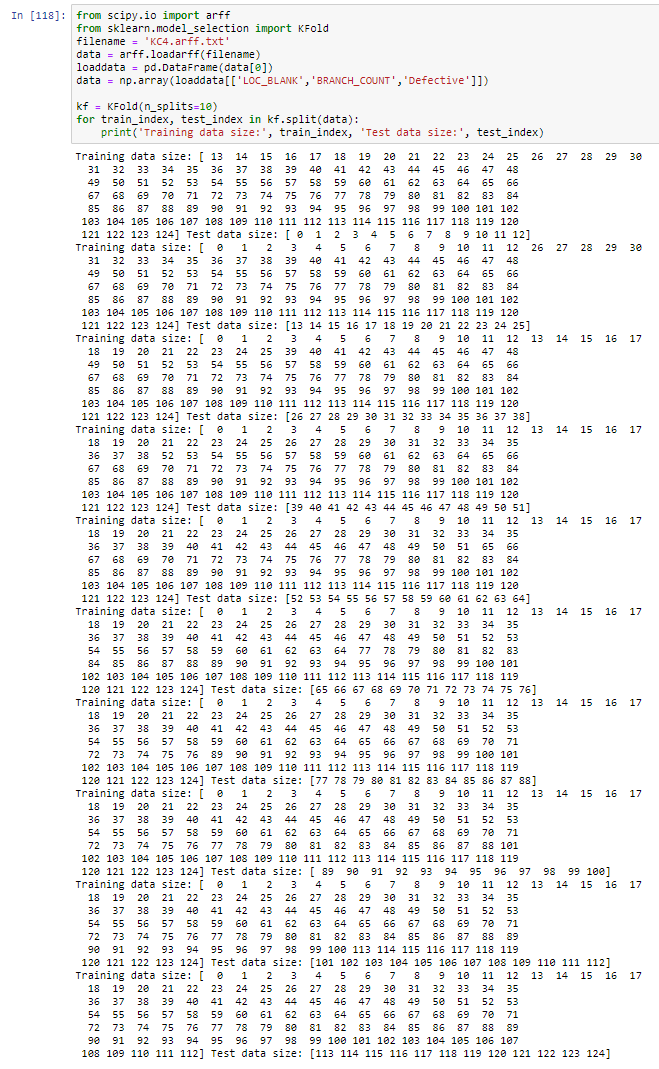
The algorithm begins by reading the dataset uploaded by the user. The algorithm will extract the attributes, column names and labels accordingly. After the dataset has been read, the user will be able to select the software metrics they would like to include through an interface and the program will filter the software metrics that were excluded from the selections.

Step 2: Preprocessing

The preprocesses done to the dataset include handling missing data and normalization. When missing data is found, deletion or imputation will be performed based on criterias set. After that, the data will be normalized.

Step 3: K-fold cross validation

K-fold cross validation is performed to obtain k number of training/test data splits, where each split will have k-1 training data. The number of folds k will be determined by the number of instances contained in the dataset.



**Fig. 10 K-fold cross validation using KFold**

Fig. 10 shows a demonstration of the K-fold cross validation step. This was done in Jupyter notebook using the K-Folds cross-validation from Sklearn. The dataset used for this was KC4 from the NASA repository.

Step 4: Undersampling

Step 4 will be performed on each train/test split. So this step will repeat m times where m is the number of train/test splits from step 3. This step performs undersampling on the training set that will be used to build the models in step 5. The process of undersampling the data will be based on a threshold IHT value which is set. If the IHT value for a certain datal exceeds that threshold, the data will be removed.

Step 5: Model building

After the undersampling is performed the training set will be ready to use for building our models. Step 5 and 6 will repeat (n\*m) times where n is the number of models selected by the user and m is the number of train/test splits from step 3. Our algorithm will include 5 base models as well as 3 ensemble models. Based on the model type the building step will differ.

For the base model algorithms, the method is fixed so there are no configurable components. So the model will be built, trained with the training data provided.

For the ensemble model algorithms, the way the models will be built are based on the user’s configurations. Example, the users can select the number of decision trees to be generated for a Random Forest model. The models will be trained using the training data and how its built will vary based on the configurations set.

## 

**Fig. 11 Code demonstrating the Random Forest classifier training**

Fig. 11 is a code snippet which demonstrates how the models will be built, for this demonstration we will show how the Random Forest model will be trained. Here we will use train\_test\_split instead of KFold to produce the training/testing data for a simple demonstration.

Step 6: Predictions and performance evaluation

After the model is built, the model will then be used to make predictions on the test data. The predictions will then be compared against the actual results. This comparison will allow the AUC and F1-score values to be retrieved and stored in the list of results.

## 

**Fig. 12 Code demonstrating the evaluation process**

Fig. 12 shows a demonstration of how the model will be evaluated. It is just a simple demonstration showing how the evaluation metrics such as accuracy of the model can be produced, our selected performance metrics are AUC and F-score which will be performed in a similar manner.

Step 7: Display results

After the loops between step 4 to 6 have ended the results will be gathered and displayed to the user. The results will be shown in tabular and chart form.

## 5.4 Programming Language

****

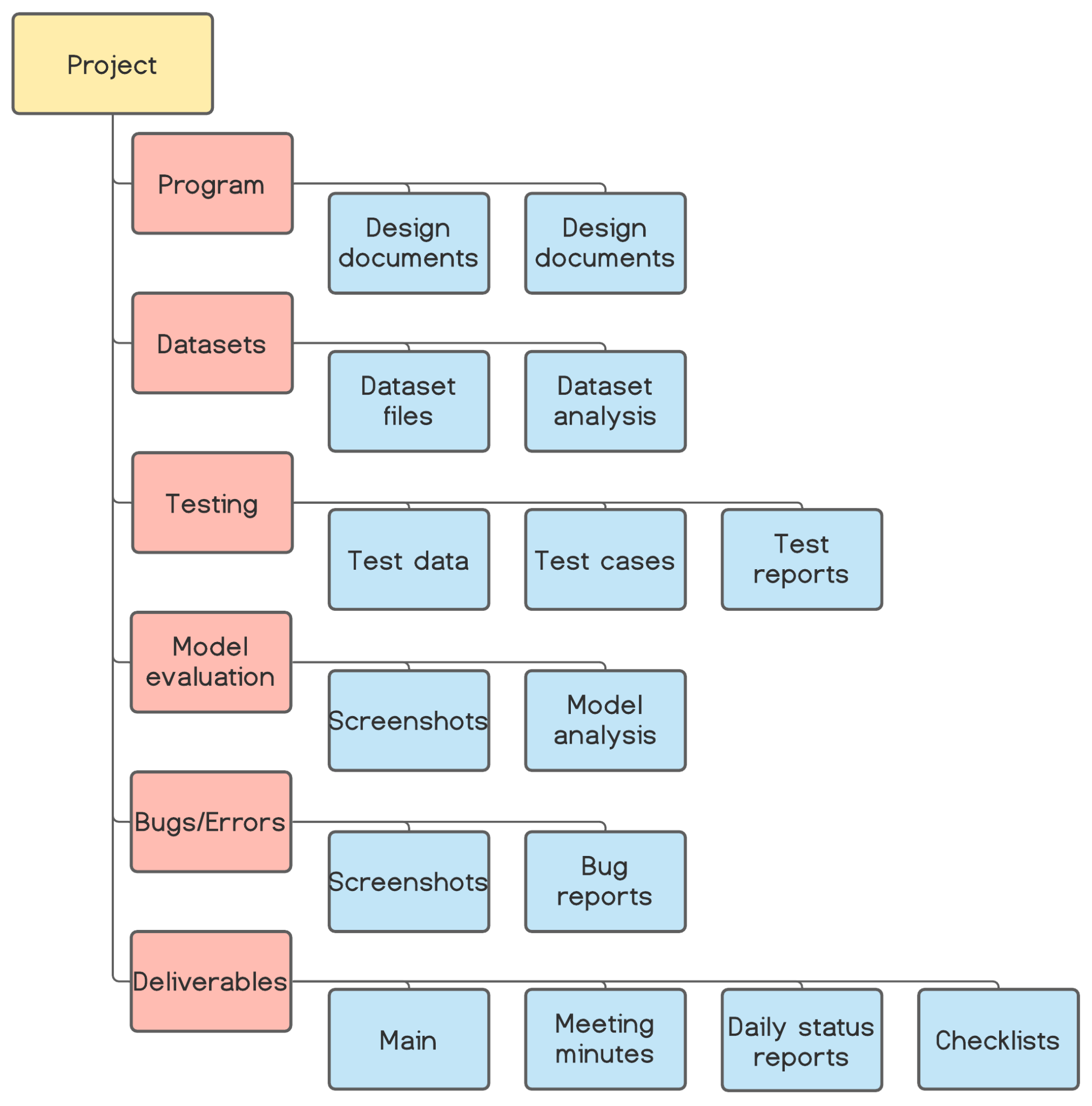
**Python 3.7.3/3.6.9**

Our algorithms and final product will be developed using Python. All software that is used in our project is compatible with Python, mainly Jupyter notebook. Python offers all the essential software machine learning libraries for our project, mainly scikit-learn, NumPy and SciPy. Furthermore, the majority of machine learning functions available in public sources are usually written in Python. Lastly, every team member is proficient in programming with Python, making this the ideal choice.

## 

## 5.5 Data Management

As mentioned previously, the data we will mainly be storing are the results, datasets and documentations throughout the project.



**Fig. 13 Graphical representation of the file structure**

To keep the data well organized, we will be following the file structure as shown in Fig. 13. The folders are based on the core components in our project, each having subfolders to keep files organized.

## 

## 5.6 Version Control

****

**GitHub**

**Description**

GitHub is a platform that allows individuals to collaboratively work on software development. It provides version control features, giving users the ability to update their code, whilst having access to previous versions of the code they developed. It is also a convenient software to use as the projects are stored in their cloud server, meaning that we can access the project from any device so long as we can access github.

**Alternatives**

GitLab

**Reason**

GitHub hosting service provides all the required features we want for our project such as access control and collaboration features. It offers branching capabilities along with standard features such as repository sharing, backups and version history. Unlike other platforms, GitHub has both a web application and desktop application. The web application eases access for viewing and control of repositories accessible in many devices, whereas the desktop application provides an interface which eliminates the need for users to manually input command lines. Furthermore, it also provides organizational and visual tools which eases tracking as well as control.

## 

## 5.7 Visualization Tools

Lucidchart is our main tool for visualising the components and processes within our program, most figures such the ones shown in the User Interface section external design were also made through Lucidchart. It offers many features and shapes which makes it user friendly and better visuals. As such for our risk register, we used Google Sheets as it has a sharing and collaborating feature, allowing any users with access to view and edit from any device. Lastly, we used ProjectLibre for our Gantt charts. Realising Gantt charts is only one of several features which aids the management of schedule and plans, making it an ideal choice.

**Table 12: Description of visualization tools**

| **Software** | **Description** |
| --- | --- |
| **Lucidchart** | A web-based collaborative platform that allows users to create charts and diagrams. Its library contains various templates and content which improve the visuals of any traditional diagrams. |
| **Google Sheets** | An online spreadsheet program that allows individuals to collaborate together on creating and customizing spreadsheets. Google Sheets will mainly be used to write and update our sprint cards weekly. It helps to keep track of the upcoming tasks and ongoing tasks at hand |
| **ProjectLibre** | An open-source project management software that helps with the management and organisation of projects. It offer tools that allow users to create items widely used in project managements such as Gantt chart and Work Breakdown Structure |

# 

# 6. Testing Planning

## 6.1 Test coverage

**Implementation of pre-processing algorithms**

To test out the effectiveness of the pre-processing algorithms, we will have fixed test cases and test datasets that we pass into the algorithm. After the test dataset goes through the pre-processing algorithms, we will check if the results of the processed dataset align with the expected results specified in the test cases.

**Implementation of machine learning algorithms**

Our software will allow the user to select and configure the machine learning algorithms they want to use. Therefore, our main aim will be to test the machine learning algorithms based on the specified configurations we give to them.

**Implementation of user interface**

For the user interface, we will make sure to test each functionality that has been added to it. This includes things such as the download function for the performance results and the selection and configuration screen for the prediction models.

## 6.2 Test Methods

Each section will have different steps for its testing methods

**Implementation of pre-processing algorithms**

1. Create a test dataset that is imbalanced to act as a test case, then set up the expected test case results (the expected look of the dataset after pre-processing)
2. Pass the test dataset through the pre-processing algorithms
3. After the test dataset has fully gone through pre-processing, compare it to the dataset in the expected test case results section.
4. If the test dataset is around 95% similar to the one in the expected test case results, the test case passes. Anything lower than that will fail the test case.

**Implementation of machine learning algorithms**

1. Create a test dataset, then set up a threshold AUC value for each machine learning algorithm when using that test dataset
2. Use k-fold cross validation to split the test dataset into training data and test data.
3. Train each machine learning algorithm with the training data, then test it against the test data.
4. If the AUC value for that machine learning algorithm is within the acceptable threshold AUC value in the test case, the test case passes

To evaluate the performance of each machine learning algorithm, we will make use of both the AUC and F1-score evaluation metrics. Below is the explanation of the both the metrics as well the process of testing:

**AUC (Area Under The Curve)**

AUC stands for area under the ROC curve and it is the measure of the ability of the classifier to distinguish between classes. AUC has seen its application in many published research papers. In order to evaluate the performance of the proposed model, AUC measures the area under the ROC curve which describes the trade off between the True Positive Rate and False Positive Rate (Tong, Liu, Wang tl al, 2018). A study by Tuong Le (2018) shows that AUC was used as an evaluation metric for the proposed Cluster-based Boosting Algorithm where it achieves 86.4% AUC, outperforming other algorithms like the GMBoost Algorithm when handling imbalanced dataset for bankruptcy prediction. AUC is particularly useful for imbalance datasets due to the fact that the majority class can be referred to as the negative outcome whereas the minority class can be referred to as the positive outcome.

**F-score**

F-score is defined as the balance between precision and recall and is widely used as a measure to evaluate classification problems.It typically involves four possible outcomes (True positive, False positive, False negative and True negative). F1-score is particularly useful when the precision and recall of the model is paramount. There exists a tradeoff between precision and recall where increasing precision involves sacrificing recall or vice versa (Zhang et al, 2015). Difficulties can occur when evaluating models solely based on precision or recall as both metrics are important when handling imbalanced data as there is a majority class and minority class (Zhang et al, 2015).

**The Testing Process**

1. Since software defect prediction has only two possible prediction results, we can construct a confusion matrix for it.

**Table 13: Sample Confusion Matrix**

|  | Predicted: Non-defect | Predicted:  Defect |
| --- | --- | --- |
| Actual: Non-defect | TN | FP |
| Actual: Defect | FN | TP |

1. Using the values from the confusion matrix, we can then derive a few values from the following formulas as shown by Yucular et al. (2020):

* Probability of Detection (PD) = (1)
* Probability of False Alarms (PF) = (2)
* Accuracy (ACC) = (3)
* Precision (PR) = (4)
* F-score = (5)

1. After deriving these equations, we can then plot a Receiver operating characteristic (ROC) curve, which helps to calculate the benefit of using the machine learning model
2. Once the ROC curve is constructed, the area under the curve (AUC) can be calculated for each machine learning algorithm. Based on the calculated AUC value, we can check whether the machine learning algorithm in question is processing the data correctly or not

**Implementation of user interface**

To test the user interface, we will come up with test cases through whitebox testing methods. We will think of instances of test cases which will cause the functionalities of the user interface to fail, as well as instances which will have the functionalities work as usual.

## 

## 6.3 Sample test cases

When creating test cases, we decided to make use of the public datasets provided by the NASA repository as they are mostly related to software defect prediction. A summary of these datasets can be found at Table 9. There will be tests carried out for three categories, which are:

1. Implementation of pre-processing algorithms
2. Implementation of machine learning algorithms
3. Implementation of user interface

**Table 14: Test cases for the pre-processing algorithms**

| Test Case Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| --- | --- | --- | --- | --- |
| Pre-processing correctly balances data | JM1.arff from the NASA repository | 95% or more similarity when compared to expected outcome | - | - |

**Table 15: Test cases for the machine learning algorithms**

| Test Case Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| --- | --- | --- | --- | --- |
| Performance evaluation of machine learning algorithms | KC1.arff from the NASA repository | AUC >= 0.9 | - | - |

**Table 16: Test cases for the user interface**

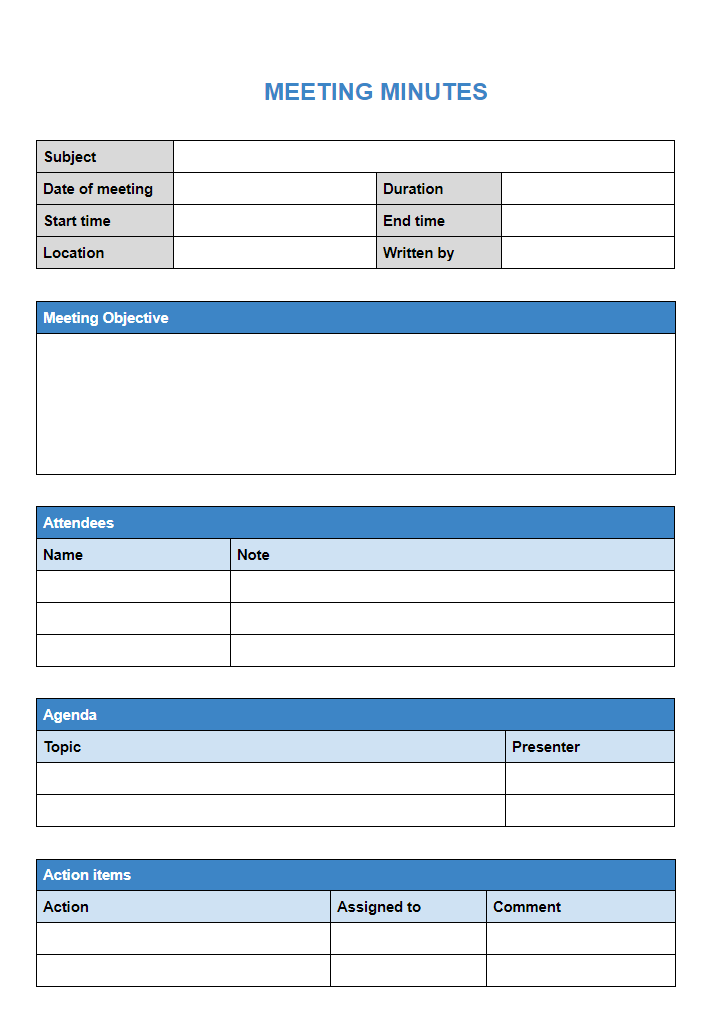
| Test Case Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| --- | --- | --- | --- | --- |
| Checking the functionality of the download function | A sample evaluation result generated from the software | Download function correctly saves the result to the right directory | - | - |
| Selection of software metrics are being processed correctly | Any random selection of software metrics | Only the selected software metrics are being used | - | - |

# 

# 7. Conclusion

As previously mentioned, our objective is to devise a method involving supervised learning for finding fault proneness within a system. Additionally, our method should specialize in handling data that are highly imbalanced. From our research, we were able to identify the components for building a fault prediction software as well as various concepts and techniques to address the dataset imbalance issue. As such, our main program will incorporate all the identified items. Essentially, our main program will allow us to test various combinations of methods and provide an evaluation to determine the performance level of the combined methods. We are certain that we can find several effective methods through testing various combinations of tools and techniques. In conclusion, our devised method will be highly effective as it will be based on the combination which has the best performance within our program. Furthermore, our findings will be a great contribution towards studies on fault prediction.

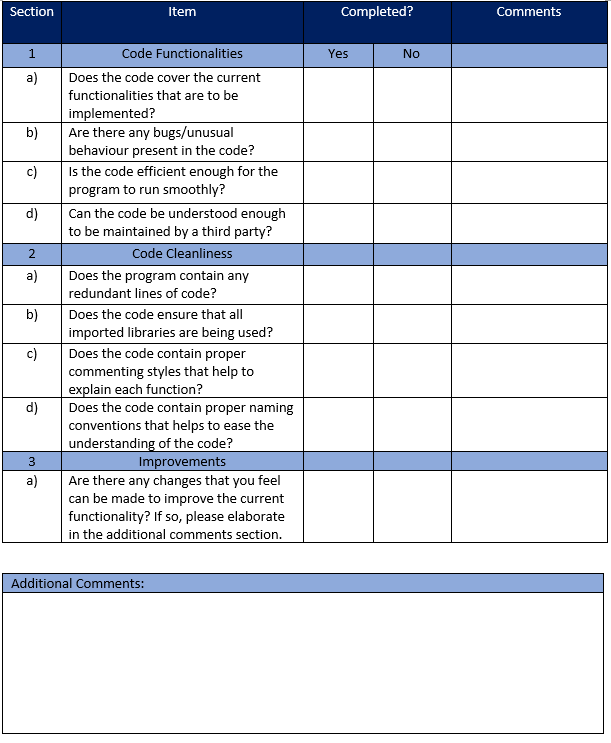
# Appendix A - Meeting Minutes



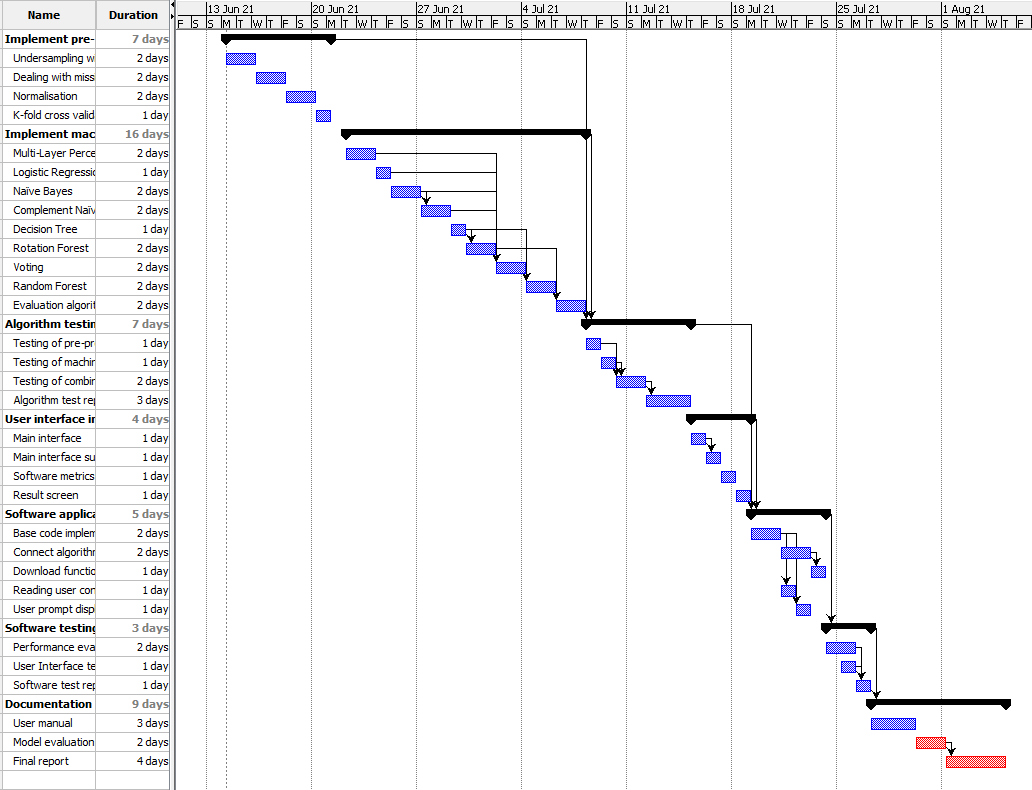
# Appendix B - Daily Status Update Form

# 

# Appendix C - Auditing checklist



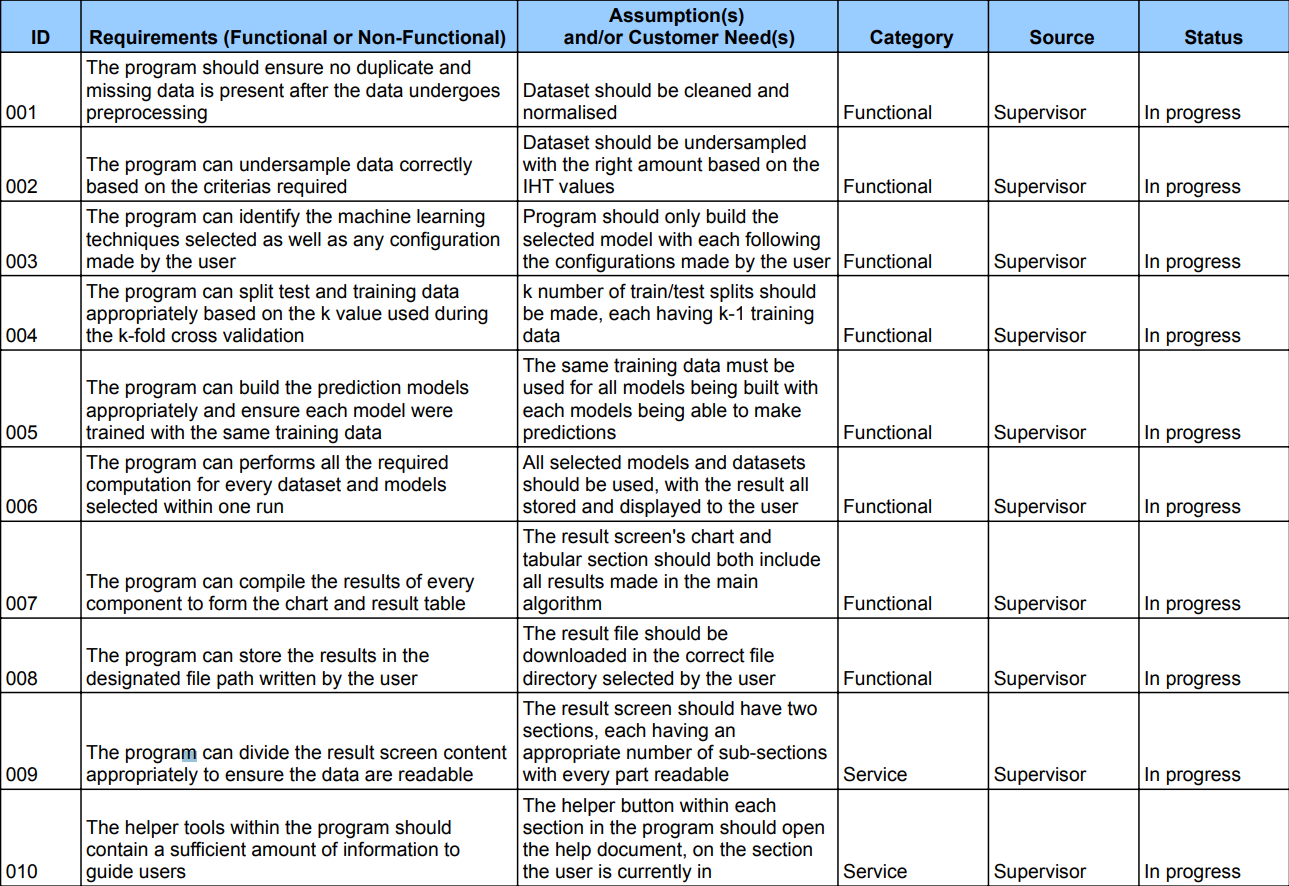
# Appendix D - Gantt chart



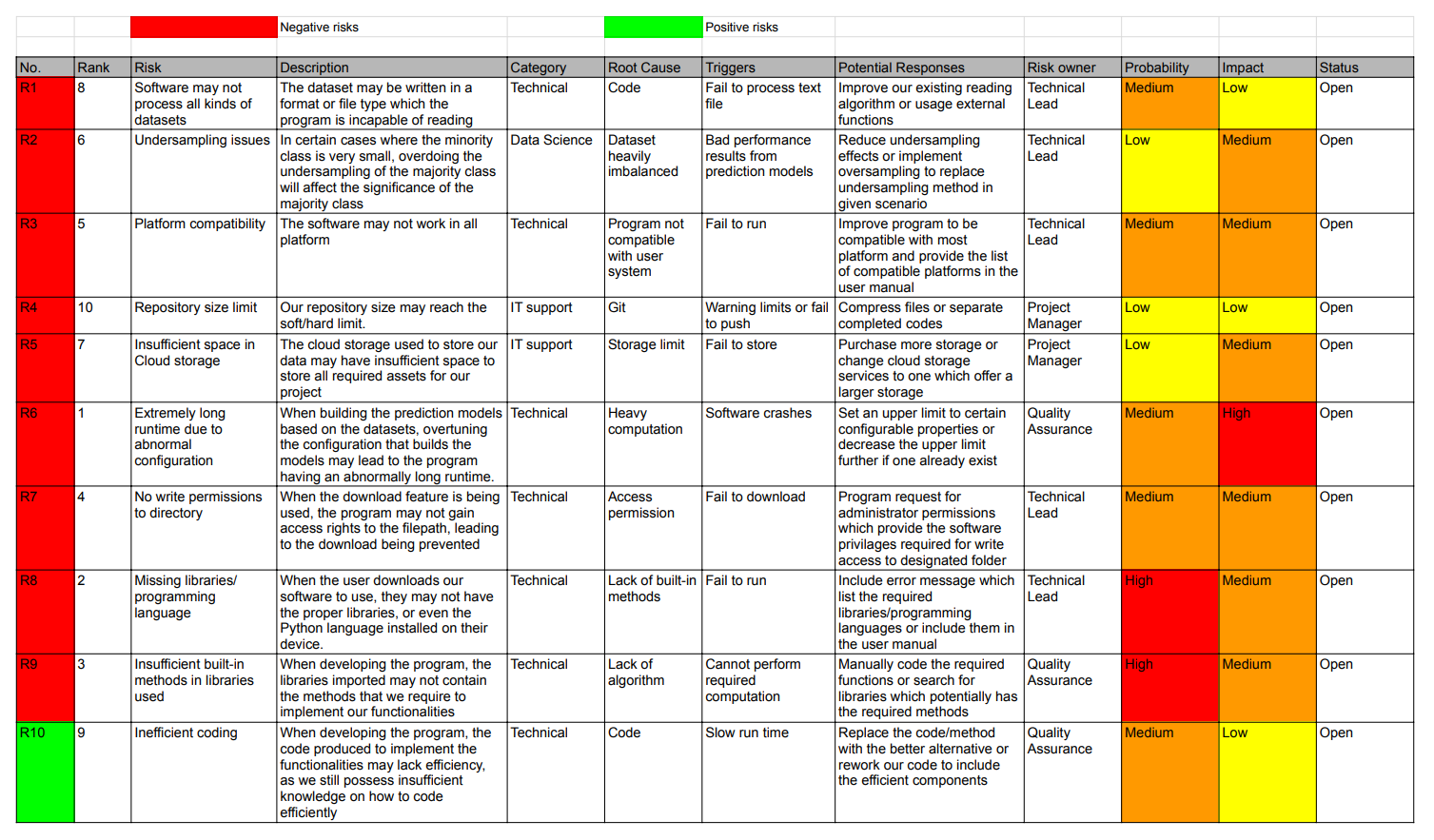
# Appendix E - Work Breakdown Structure (WBS)



# Appendix F - Requirement Traceability Matrix



# Appendix G - Risk Register



# References

ABB (2021). *4 types of maintenance strategy, which one to choose?.* Retrieved May 18,

2021 from <https://new.abb.com/medium-voltage/service/maintenance/feature-articles/4-types-of-maintenance-strategy-which-one-to-choose>

Rana, Z. A., Mian, M. A., & Shamail, S. (2015). Improving Recall of software defect

prediction models using association mining. *Knowledge-Based Systems*, *90*, 1-13.

Boughorbel, S., Jarray, F., & El-Anbari, M. (2017). Optimal classifier for imbalanced data

using Matthews Correlation Coefficient metric. *PloS one*, *12*(6), e0177678.

Existek (2017). *The Agile Software Development Life Cycle Model (Fig 3).* Retrieved May

20, 2021 from <https://existek.com/blog/sdlc-models/>

Fuchs, M. A. (2016). *Visualization of the undersampling technique (Fig 2).* Retrieved May

20, 2021 from <https://michael-fuchs-python.netlify.app/2020/01/16/dealing-with-imbalanced-classes/>

Haque, M. N., Noman, N., Beretta, R., & Moscato, P. (2016). Heterogeneous ensemble

combination search using genetic algorithms for class imbalanced data classification. *PloS one*, *11*(1), e0146116.

Le, T., Le Son, H., Vo, M. T., Lee, M. Y., & Baik, S. W. (2018). A cluster-based boosting

algorithm for bankruptcy prediction in a highly imbalanced dataset. *Symmetry*, *10*(7), 250.

Li, M. (n.d.). *Visualization of K-Fold cross validation (Fig 1).* Retrieved May

20, 2021 from <https://www.researchgate.net/figure/K-fold-cross-validation-method_fig2_331209203>

Menzies, T. (2004). *Promise Software Engineering Repository.* Retrieved May 15, 2021 from

<http://promise.site.uottawa.ca/SERepository/datasets-page.html>

Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2021). Machine learning based methods for

software fault prediction: A survey. *Expert Systems with Applications*, 114595.

Shepherd et al. (2014). *NASADefectDataset.* Retrieved May 15, 2021 from

<https://github.com/klainfo/NASADefectDataset/tree/master/CleanedData/MDP/D>''

Tong, H., Liu, B., & Wang, S. (2018). Software defect prediction using stacked denoising

autoencoders and two-stage ensemble learning. *Information and Software Technology*, *96*, 94-111.

Yap, B. W., Abd Rani, K., Abd Rahman, H. A., Fong, S., Khairudin, Z., & Abdullah, N. N.

(2014). An application of oversampling, undersampling, bagging and boosting in handling imbalanced datasets. In *Proceedings of the first international conference on advanced data and information engineering (DaEng-2013)* (pp. 13-22). Springer, Singapore.

Yen, S. J., & Lee, Y. S. (2006). Under-sampling approaches for improving prediction of the

minority class in an imbalanced dataset. In *Intelligent Control and Automation* (pp. 731-740). Springer, Berlin, Heidelberg.

Yucalar, F., Ozcift, A., Borandag, E., & Kilinc, D. (2020). Multiple-classifiers in software

quality engineering: Combining predictors to improve software fault prediction ability. *Engineering Science and Technology, an International Journal*, *23*(4), 938-950.

Zhang, Y., Lo, D., Xia, X., Xu, B., Sun, J., & Li, S. (2015, December). Combining software

metrics and text features for vulnerable file prediction. In *2015 20th International Conference on Engineering of Complex Computer Systems (ICECCS)* (pp. 40-49). IEEE.