1 Design

1.1 Methodology

When Designing software there are a number of approaches that can be taken. Most of these approaches break development down into a number of different phases and describe the order in which they should be completed.

Most methodologies were designed for teams of engineers, as this project was completed by one individual many components of the methodologies are redundant. However there are some principles and techniques which can be utilised by an individual developer and aid in organisation of a project.

1.1.1 Waterfall

The waterfall methodology is a linear approach to development. Each stage of development is completed all at once for the whole piece of software. This means that once a stage is completed there is no need to return to it and the development process flows in one direction like a a waterfall.

The waterfall approach is good because the whole project is rigorously planned and documented before coding begins. This allows major difficulties in projects to be dealt with before significant time has been invested. It also allows projects to be easily passed between developers as a new developer can look at the plan of the project. In less linear approaches there may be plans for future parts of software which are never documented.

The downfall of the waterfall approach comes when a project is not fixed or there is a a change in requirements, there is no stage in which plans can be modified as that would involve going backwards in development.

1.1.2 Spiral

The spiral approach

1.1.3 Agile

The agile approach is a methodology which focuses on iterative development. Deliverables are identified and designed individually, there is also a much greater focus on executable code than documentation than in waterfall

Many parts of the agile methodology are not needed in this project such the parts that deal with the customer and cooperation between developers as this is a solo project without a customer.

The most important part of agile is its ability to deal with changing scope through process. Also since testing is done at each stage it will allow the project to be easily shortened or extended based on time constraints, as after every sprint a working piece of software has been developed.

1.1.4 Chosen Approach

The chosen approach for this project was agile. The reduced amount of documentation will speed up development. Agile development also allows the scope of the project to change over time. This allowed the scope of the project to start off small and increase if time allowed it.

For example, once the PC algorithm had been implemented, if time allowed, the project could easily be extended to include the FCI algorithm.

Within the Agile methodology there are a number of popular frameworks. However, these teams are designed for teams of software engineers with clients. Since this is a solo project without a client many parts of these frameworks will not be applicable. Therefore, the applicable parts of various frameworks were chosen and compiled into a framework that will work for this project.

1.2 Requirements

Requirements were elicited at the beginning of a sprint. A deliverable was chosen and the requirements were found for that deliverable.

1.2.1 Epics

Epics describe a very high level interaction between a user and the software. They describe the general functionality of a deliverable without discussing it's implementation. Epics were the first stage of requirement elicitation as they they capture the functionality that the requirements must capture.

1.2.2 User Stories

Each epic is then broken down into user stories. The stories describe describe what a user must be able to do with the software during the epic. They are fairly low level and form a set of required functionality of the software.

1.2.3 Tasks

From the stories can be formed. Theses tasks describe the software that must be implemented to fulfil the requirements.
Each task has associated stories so that all software can be traced back to an epic.

1.3 Abstract Design

1.4 Sprints

1.4.1 Tasks

1.4.2 Concrete Architecture

1.4.3 Unit Testing

Test driven development (TDD) was one of the main features of the development framework. In this development, tests were created before the actual software was written. This ensured that all software that was written would be functional. It also ensures that any refactoring performed would not compromise the functionality of software.

All software was tested using the python unittest framework. In this framework, list of tests can be compiled into test cases and cases compiled into a test suite. Test cases test an individual feature of software, so can be run individually when a feature is being developed or maintained. Test Suites can then test the software as a whole.

1.5 Tools

A number of existing tools were used throughout the project to aid in development.

1.5.1 Version Control

An important part of software development is Version Control (VC). VC allows changes to be made to software without risk of losing previous iterations. At any point software can be reverted to a previous version if needed. VC also helps with agile development as deliverables can be easily identified in a VC system.

Git is a standard VC system used in development.

1.5.2 Python Libraries

Various external python libraries were used to facilitate development.

Pandas is a library containing DataFrames which allow easy manipulation of data. This includes labelling variables and calculating contingency tables which is important in the χ^2 test of independence.

NetworkX was used for graphical models, the Graph and Digraph were used and extended to store various graphs used in the algorithms. The built in functionality of these graphs made extension to new types such as a PDAG simple.