2 Background research and related applications

This chapter presents the legal issues surrounding the release of medical micro-data and reviews the current anonymisation techniques that are being used to transform and publish this sensitive medical data. A review of the current tools which are used for similar use cases are also presented. This chapter concludes with a summary of the key findings from the research.

2.1 Data protection and information governance

To understand the current processes used in the NHS when dealing with sensitive medical data, 5 e-learning modules were completed. The titles and certificates for these courses are provided in Appendix A.1. A summary of the information obtained from these courses and additional research is provided in the following section.

2.1.1 UK Legislation and Guidance on the Data Protection Act

The data protection Act 1998 [1] is a piece of UK legislation that outlines the provision:

"for the regulation of the processing of information relating to individual, including, holding, use or disclosure of such information"

Section (1) of the act defines personal data, as data which relates to a *living* individual who can be identified:

(a) for those data, or

(b) from those data and other information which is in the possession of, or is likely to come into the possession of, the data controller.

Processing of the data is defined as "obtaining, recording or holding the information or data or carrying out any operation or set of operations on the information or data". Therefore, the processing of patient information as proposed by this project is clearly within the remit of data protection act.

The Act goes on in *Schedule 2 Section 4(3)* to outline the conditions relevant for processing any personal data and states:

> *"The data subject has given his consent to the processing."*

The obtaining of consent for the use of each record pragmatically speaking, is extremely difficult and therefore, any data output by the proposed solution must either be novel data that does not pertain to a real individual, or anonymised data such that it is not within the remit of the data protection act.

The information commissioner's office [2] is a non-departmental, independent public body that is responsible for the enforcement of the Data Protection Act. It has published documentation outlining good practice with regards to implementing the Data Protection Act. In regards to anonymous data, there is guidance that permits the dissemination of anonymized data. The ICO has a data anonymisation code of practice [3] that states:

> *"There is clear legal authority for the view that where an organization converts personal data into an anonymised form and discloses it, this will not amount to a disclosure of personal data."*

The document goes on to define the anonymisation as;

> *"anonymised means that it is not possible to identify an individual from the data itself or from that data in combination with other data, taking account of all the means that are reasonably likely to be used to identify them."*

In a separate report [4] the ICO acknowledges that it is very difficult to completely eliminate the risk of re-identification however, the aim should be to mitigate re-identification to such a degree that it is no longer significant i.e. the chances of re-identification are "extremely remote"

2.1.2 UK standards on anonymisation:

ISB 1523[1] is an anonymisation standard for publishing health and social care data. It is used by the NHS. The document gives pragmatic advice on the application of anonymisation techniques. It recommends the following techniques for anonymising data:

- **Aggregation:** This involves the summarisation of information. This could be the average or total of the dataset or a subset of the data. For example, if the age of three patients are to be published, instead of publishing their actual ages i.e. 10, 20 and 30, an aggregated average value of 20 is published.
- **Data Suppression:** Partial obfuscation. For example consider the postcode SW99 9AA, this could be supressed to as SW99 ***
- **k-anonymity**: This refers to a criterion where at least k records in the dataset have the same quasi-identifiers. For example, consider age of 30 and city of residence as London. For k = 3, k-anonymity will ensure that at least 3 records contain an age of 30 and the city of residence as London. These similar records are referred to as equivalence classes. This way, the records are not individually identifiable. This property can be achieved by generalising or supressing the data. A more detailed look at k-anonymity is provided in Section 2.2.1
- **Statistical disclosure control:** This involves the removal or aggregation of data which appears in low frequency for a small underlying population. For a smaller population size the risk for re-identification is greater and therefore, may need to be aggregated or rounded to remove information and minimise re-identification risk.

The degree to which these techniques are applied is determined by two risk levels, "weak" and "strong". For example, in the case of k-anonymity, weak k-anonymity specifies that each equivalence class must contain at least 3 records (i.e. k=3) with un-transformed variables that do not contain: date of birth, gender, ethnic category, postcode, event dates, employers and occupation. Strong k-

[1] The Information Standards Board (ISB) has been replaced by the Standardisation Committee for Care Information (SCCI) however, ISB 1523 is still relevant as it is listed on the new SCCI website at the time of writing this document.

anonymity is defined as having k = 5, where all variables but one is controlled through k-anonymity and the uncontrolled variable should not be postcode, date of birth or ethnic category.

The techniques that are ultimately applied to the original data depend on the risk of de-identification and the underlying population size, which determines whether statistical disclosure is applied. While the underlying population size is relatively trivial to determine, the risk of re-identification can be more ambiguous. The risk of re-identification is based on the motivation to re-identify the data. Therefore, an element of judgement must be applied when deciding which techniques to use.

2.1.3 International de-identification laws:

HIPPA is the Health Insurance Portability and Accountability Act [5]. It is a piece of legislation that outlines the rules for de-identification of medical data in the United States of America. The U.S. Department of Health and Human Services provide the following information with respect to de-identification:

For medical data to be used without patient consent it must conform one of the two HIPAA de-identification standards.

1. Expert Determination - A formal determination by a qualified expert
2. Safe Harbour method - The removal of specified individual identifiers as well as absence of actual knowledge that the remaining information could be used alone or in combination with other information to identify the individual.

After applying either of the two methods listed above, there is no restriction on the use or disclosure of de-identified health information. This is because after the protected identifiers have been removed, the data is no longer considered protected health information.

Since the aim of this project is to develop an automated solution that minimises the requirement for formal evaluation of datasets prior to publishing, the Safe Harbour method was investigated.

The safe harbour method states that the following 18 identifiers must be removed:

1	Names	10	Social security numbers
2	Geographical information smaller than state including postcodes	11	Internet protocols (IP) addresses
3	All dates directly related to an individual, including admission date, discharge date, death date and all ages over 89 years	12	Full face photographs and any comparable images
4	Telephone numbers	13	Biometric identifiers
5	Vehicle identifiers	14	Health plan beneficiary numbers
6	Fax numbers	15	Medical record numbers
7	Device identifiers and serial numbers	16	Account numbers
8	Email addresses	17	Any other unique identifying numbers
9	Web universal resource locators	18	Certificate / License numbers

Table 1 HIPAA identifiers

2.2 Anonymisation techniques

A summary of widely used anonymisation techniques and their limitations are presented in this section.

2.2.1 k-Anonymity

k-anonymity refers to a property of an anonymised dataset. Data is said to have a k-anonymity property if the information for each record in the dataset can not be distinguished from at least k-1 records in the same data set. An example of k-anonymity is provided below:

Consider the following data tables:

Name	Age	Gender	Postcode	Condition
John Smith	22	M	SW12 5BB	Cancer
Jane Doe	26	F	SW12 0DY	Influenza
Joe Bloggs	28	M	SW12 6FL	Asthma
Gina Baker	29	F	SW12 3ND	Influenza
Jim Jones	31	M	SW10 5VW	Diabetes
Jon Moore	34	M	SW10 8DW	Diabetes

Table 2 Original dataset

Name	Age	Gender	Postcode	Condition
pat001	[20,30]	M	SW12 ***	Cancer
pat002	[20,30]	F	SW12 ***	Influenza
pat003	[20,30]	M	SW12 ***	Asthma
pat004	[20,30]	F	SW12 ***	Influenza
pat005	[30,40]	M	SW10 ***	Diabetes
pat006	[30,40]	M	SW10 ***	Diabetes

Table 3 k-anonymised dataset

Let us assume that *Condition* is sensitive personal information and that *Name* is an identifier and *Age*, *Gender* and *Postcode* are quasi-identifiers (they can be linked together to identify a specific record). Table 2 shows original data extracted from a medical database. Table 3 is a data table that has been anonymized through the following transformations:

1. Pseudo anonymisation of identifying information - Name of the patient has been changed to a coded value.
2. Data generalization: Age has been generalized to a range.
3. Suppression of information: The last three digits of the postcode have been obscured.

The data table on the right can now be said to have a k-anonymity property of 2 (k = 2). This means that each record can not be distinguished from at least k-1 (one in this case) other record(s). As the highlighted equivalence classes in Table 3 show, this is the case. Pat001 has exactly the same quasi-identifiers as Pat003.

2.2.2 Limitations of k-anonymity

For many high-dimensional cases, preservation of even k = 2 anonymity resulted in a significant loss of information, such that the data "may not be acceptable from a data mining point of view", with loss of information being significantly high for a dimensionality of >20 [6].

While k-anonymity provides a straightforward and simple method for anonymizing data, several studies have concluded that it does not adequately protect again attribute disclosure. For example, [6] lists two types of attacks. Examples of these attacks are given below:

1. Homogeneity attack: Consider the table above, Jon Moore's (pat006) relative knows he lives in SW10 and is between 30 and 40. Therefore, he can conclude that Jon Moore has diabetes, this represents a disclosure of sensitive personal information.
2. Background attack: Suppose John Smith's (pat001) friend knows he lives in SW12 and does not have Asthma. John Smith's friend can now assume with a high degree of confidence that John Smith has cancer.

To overcome sensitive attribute disclosure identified by this attack, attribute specific transformations such as l-diversity [6] and t-closeness [8] are proposed. These methods further reduce the re-identification and attribute disclosure risks.

2.3 Limitations of syntactic anonymisation methods

The methods that have been discussed up to now relate to syntactic anonymisation methods. The aim of these methods is to modify the source data to achieve a desired property (i.e. k-anonymity). Legislation stipulates that if the published data can be combined with other available information to re-identify the original record, the record has not been adequately anonymised. The syntactic approach requires an assumption of the information that an adversary who wants to re-identify dataset has.

It becomes evident that by assuming an adversary has access to greater information, it is possible to show that previous anonymisation techniques are inadequate. Assuming that an adversary has access to greater amounts of information results in the application of more transformations to the data-set. These transformations reduce the re-identification risks associated with the data set, but also adversely affect the utility of the data. Furthermore, taken to it's logical conclusion, privacy can only be guaranteed by accounting for an adversary that has almost all information. Subsequently, the amount of transformations required to keep the data anonymous would result in a data set which has very low utility, i.e. very little or nothing meaningful can be learnt from it. Therefore, what is required is a method of anonymisation that is independent of the knowledge of an adversary. This can be achieved by state of the art methods such as differential privacy. [9]

Data shuffling techniques

Shuffling or permutation of values is a simple data masking technique whereby fields in a table are shuffled such that they are linked arbitrarily to other values. A very simply example of this would be randomly shuffling all the values in a column of a database table and repeating for all columns. However, where strong functional dependencies exist between the attributes of the table, a 'what-if' analysis can be used to piece data back together. Consider a basic example shown in Table 4.

Age	Job title	Gender
75	Snr. Electrical Engineer	M
24	Student	M
40	Retired	M
17	Junior developer	M

Table 4 Shuffled Table

In Table 4 Shuffled TableTable 4 is can be said that age is a determinant of job title and even after shuffling it is possible to identify the age of the persons mentioned in the table above. Re-identification of the data above is trivial, no privacy has been gained from the transformation, i.e. loss of utility with no benefit to privacy. While real life examples are rarely this trivial, it is recommended that where strong functional dependencies exist, random shuffling may not be suitable [8].

Shuffling can be a useful technique as it retains the univariate characteristics of the data. For example, if the age column in a database was shuffled, it would still retain the same average, frequency distribution and standard deviation. However, multivariate characteristics such as co-variance would be lost unless they were shuffled in the same manner.

2.4 Noise addition

Another method for the anonymisation of sensitive attributes can be through noise addition. In it's simplest form, this can be achieved through the addition of Gaussian uncorrelated noise, also referred to as Gaussian white noise. A formal definition of the technique is given in Equation 1.

$$\boldsymbol{Z = X + \varepsilon} \qquad \textbf{(Eqn 1)}$$

Where Z is the new value with added noise, X is the original value and ε is a Radom variable (noise component) with a distribution of $\varepsilon \sim N(0, \sigma^2)$. The additional of uncorrelated noise does not preserve correlation coefficients between attributes. To maintain correlation coefficients, correlated white noise can be added. However, this can be significantly complex to implement.

2.5 Review of existing solutions:

ARX de-identifier [9], sdcMicro [10] and CAT: Cornell Anonymisation Toolkit [11] were identified as toolkits that provided implementations of k-anonymity as well as other more advanced techniques including (ε, δ)-differential privacy. All three of these software packages also provide measures for data utility and re-identification risks following transformation of the data. However, sdcMicro and CAT were difficult to use. In the case of CAT, configuration files and generalisation hierarchies were to be defined manually, this would be potentially difficult for the intended users of the proposed software for this project. ARX de-identifier provided a clean and easy to use interface however, it lacked data generation features such as swapping out real names for novel names and attribute shuffling. However, ARX provides a clean API for anonymising data to k-anonymity.

2.6 Conclusions from literature review.

The Data Protection Act and guidance from the public body in charge of enforcing the Data Protection Act (ICO) clearly state that there is a legal authority for the release of anonymised patient records.

HIPAA [5] provides a full list of identifiers that must be anonymised prior to the release of patient data. This list will be used in the project to classify attributes as identifiers. Furthermore, if the scope of the proposed software was ever to change, compliance of HIPAA rules would allow for the software to be used in the United States.

k-anonymity and other syntactic anonymisation techniques have limitations. However, the use of k-anonymity is explicitly prescribed in the NHS anonymisation standard [14]. Therefore, it should be included in the proposed application. To comply with “weak” and “strong” anonymisation, the proposed application should allow the user to specify the level of k-anonymity to be achieved.

Semantic privacy models such as ($\boldsymbol{\varepsilon}$, δ)-differential privacy represent the cutting edge techniques for data anonymisation and overcome some of the limitation associated with semantic anonymisation models such as k-anonymity. However, their application is an active area of research and given the delivery constraints of this project, it will not be implemented in this iteration. However, it should be considered as a candidate for future development.

Simple data shuffling can be an effective method for masking data. Where strong functional dependencies between attributes exist, these dependencies should be maintained, as independent shuffling of strongly dependent attributes reduces utility and potentially offers no additional privacy.

Transformation of variables through the use of additive uncorrelated Gaussian noise is a simple process. The application of this uncorrelated noise does not maintain correlation coefficients between attributes. However, for the purposes of representative mock data it is adequate.

The guidance document from the European Commission [8] states that the common mistakes with the application of shuffling or noise addition techniques is that they are considered *enough* in isolation. It is recommended that they should be used in conjunction with each other as well as with additional techniques such as the removal or swapping of other obvious identifying attributes. Therefore, the proposed application should allow users the option to use these techniques in combination.

While mock data generators and anonymisation software exists in isolation, there is no software that combines the functionality of both. The proposed PEACH generator builds on the existing software by combining both sets of functionalities of data generation and data anonymisation. It will be able to generate novel data sets by randomly sampling from pools of mock data for selected attributes. The remaining attributes will be transformed through techniques such as shuffling and noise injection. Furthermore, the long term vision for PEACH generator is automatically detect database structure and identify lookup tables, and intelligently sample from them. While these features are not in the remit of this project, this is the long term aspiration of the toolkit as specified by the client. Currently, none of the tools reviewed provide such functionality. However,

ARX de-identifier [9] provides an API for k-anonymity and this API will be used in the project to minimise duplication of effort.

3 Requirements and Analysis

3.1 Problem Statement

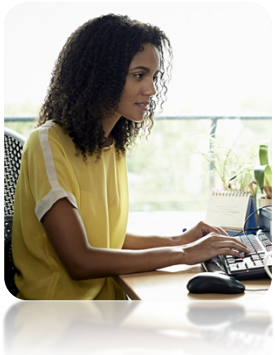
A detailed problem statement is given in Chapter 1. In summary, the purpose of this project is to develop a holistic toolkit that can take a sensitive input dataset and generate representative mock data from it. The toolkit should also be able to anonymise data. The output data from the system should present minimal risk of re-identification and can therefore be released to researchers and developers.

3.2 Requirements Gathering

The requirements for this project were elicited through weekly meetings with the client. During these meetings prototypes were presented to the client to iteratively refine the requirements. The requirements from the client were supplemented by the research presented in the previous chapter. As the client would not be the sole user of the software, persona's were created to identify additional relevant requirements.

3.2.1 Personas

Personas are created to model the typical user of the software that is being created. They are useful in the software development process as they allow designers to consider a specific person. Below are the personas of the main users of the proposed tool.



Laura is a systems administrator working at a large hospital in London. She is facilitating the release of medical micro-data for research purposes. The data she is releasing will be analysed and used for data analytics and machine learning. She is familiar with the basics of statistical disclosure control and needs an easy to use application, where she can upload data and anonymize it. She needs

the data to be complaint to legislation and the guidance provided by the hospital, so that she can release it with minimal delay.

Gavin is a doctor working at a Large Hospital in London. He is a surgeon who has a busy schedule. Part of the work he does in the hospital involves identifying opportunities to apply technology to improve patient care. Gavin is in contact with a group of students at a local university who are helping him create medical applications for the hospital trust he is working for. For data driven applications such as dashboards and automated meeting scheduling applications, Gavin wants to give students representative data, so they can develop software applications rapidly. However, the release of real medical data is often a time consuming process. Gavin wants a tool that will help him quickly transform real patient data into representative mock data for use in basic software applications.

3.2.2 Functional and Non-Functional Requirements

The information from the research, client meetings and analysis of personas were formalised into the functional and non-functional requirements listed below. These requirements where then analysed using the MoSCoW priority framework. MoSCoW is an acronym that outlines four levels of prioritisation. *Must* requirements are essential requirements that must be fulfilled to successfully complete the project. *Should* requirements may be as important as *Must* requirements but they are not critical for the completion of the project. *Could* requirements should only be completed if there is sufficient time. Completion of these requirements is desirable by not necessary or essential. *Wont* requirements are those that have been agreed as having the least utility. *Wont* requirements can either be dropped from consideration or included as part of future iterations.

Table 5 Functional and Non-Functional Requirements

ID	Requirements	Category	Type	MoSCoW
R.1	The system shall allow the user to specify the type of output that is required.	Task selection	Functional	Must
R.2	The system shall accept Comma Separated Values files as input data.	Data input	Functional	Must
R.3	The system shall allow the user to preview the uploaded data.	Data visualization	Functional	Must
R.4	The system shall allow the user to specify the privacy level and data type of each data field.	Configuration option	Functional	Must
R.5	The system shall allow the user to preview the noise to be applied to the data.	Data visualization	Functional	Must
R.6	The system shall allow the user the ability to modify the amount of noise applied.	Configuration option	Functional	Must
R.7	The system shall allow for the obfuscation of identifying information.	Data generation	Functional	Must
R.8	The system shall allow the user to shuffle identifying information	Data generation	Functional	Must
R.9	The system shall allow for novel data to replace identifying information.	Configuration option	Functional	Must
R.10	The system shall allow the user to specify a save location for the output file.	Data output	Functional	Must
R.11	The system shall implement the k-anonymity anonymisation model.	Anonymisation	Functional	Must
R.12	The system shall allow the user to specify the level of k-anonymity applied to the data.	Anonymisation	Functional	Must
R.13	The system shall allow the user to select the generalization hierarchy to be applied to the data.	Anonymisation	Functional	Must
R.14	The system shall output the data in comma separated format.	Data output	Functional	Must
R.15	The system shall transform all data locally (i.e. not over the internet)	Implementation	Functional	Must
R.16	The system shall automatically detect the presence of empty columns and remove them	Data input	Functional	Should
R.17	The system shall allow for the creation of new generalization hierarchies.	Anonymisation	Functional	Could
R.18	The system shall automatically detect the variable type	Configuration option	Functional	Could
R.19	The system shall allow the user to specify dependencies between fields.	Configuration option	Functional	Could
R.20	The system shall automatically detect semantic functional dependencies	Configuration option	Functional	Could
R.21	The system shall be compatible with XML file formats	Connectivity	Functional	Wont
R.22	The system shall handle the anonymisation of unstructured free-text information	Configuration option	Functional	Wont
R.23	The system shall be easy to use	Configuration	Non functional	Must
R.24	The system shall be easy to extend	Maintainability	Non functional	Must
R.25	The system shall be styled to the PEACH style guide	Styling	Non functional	Could

3.2.3 Use Cases

Use cases describe a series of interactions between the users and a system, in order to realise a user goal. In the context of this project, two actors were identified. The data controller, the medical professional. These two actors correspond to the first and second personas (outlined in section 3.2.1) respectively. A summary of the use cases identified are presented in Table 6

ID	Use Case	Primary Actor	Secondary Actor
UC1	Upload data	All	None
UC2	Preview data and modify settings	All	None
UC3	Specify and preview data transformations	Medical professional	None
UC4	Specify data anonymisation settings	Data controller	None
UC5	Export transformed data	All	None

Table 6 Summary of Use Cases

3.2.4 Mock-ups

Analysis of the requirements and use cases resulted in the creation of a set of mock-ups. The purposes of creating the mock-ups was to ensure the realization of the the requirements and pre-emptively identifying any issues. The use cases that correspond with each mock-up have been listed in parenthesis. The mock-ups relate only to *must have* and *should have* requirements, as the *could have* and *want* requirements required further investigation for feasibility.

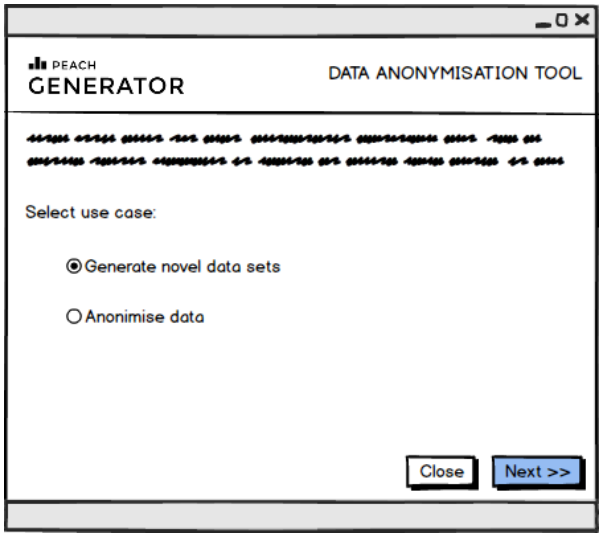


Figure 1 Select use case view

If the user selects the Generate novel dataset option, the following screen will be presented

The user can import data, initially CSV format will be supported (UC 1)

The user can preview the imported data in the scroll pane on the left hand side, to ensure the data has been uploaded correctly (UC 2)

The user can specify the type of data and what privacy level must be applied to each field using drop down menu boxes (UC 2)

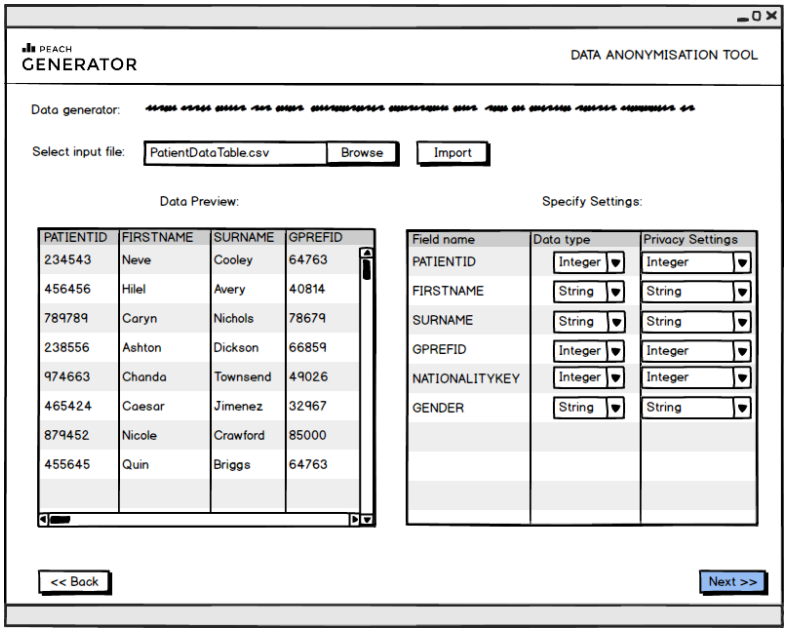


Figure 2 Data generation: import and settings view

The user can select what type of operation they wish to perform.

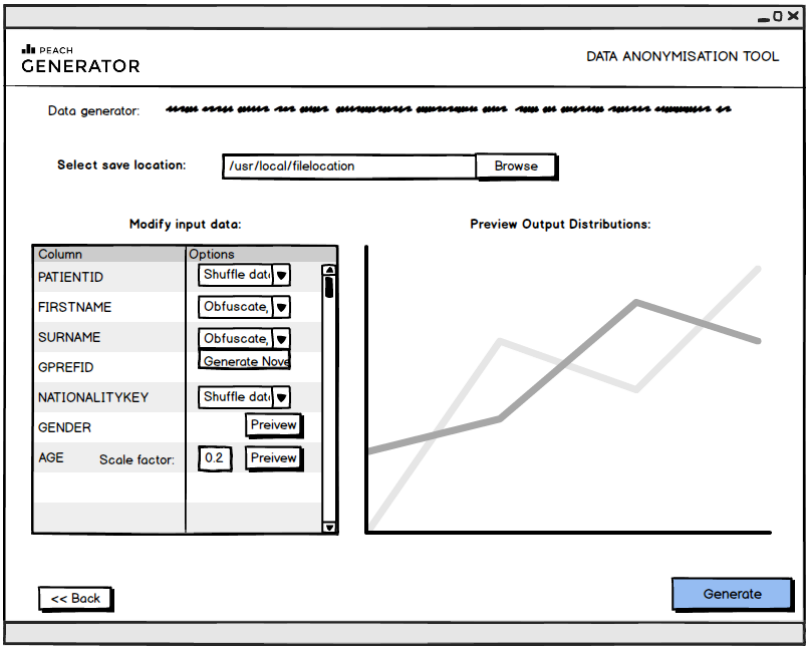


Figure 3 Data generation: Preview transformations and export view

After the upload view the user will be presented with the view on the left

This view allows the user to specify a save location (UC 5)

The user can preview the noise injected into the system along with the original dataset (UC 3)

The user can specify how to deal with identifying fields (UC 3)

The user can generate the new dataset (UC 5)

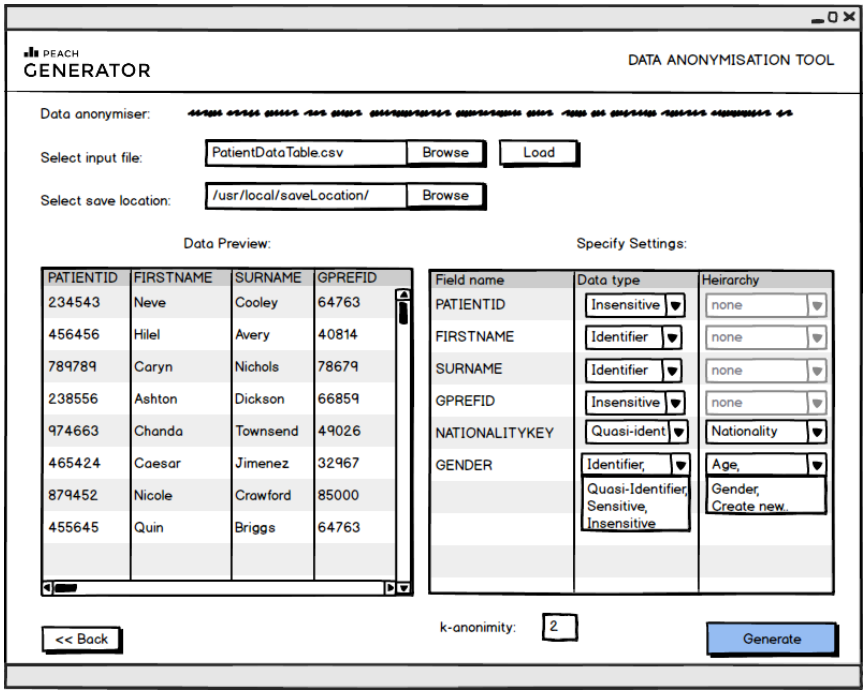


Figure 4 Anonymisation: import and anonymisation settings view

If the user selected the Anonymise dataset option from the selection view, the view on the left will appear.

Again, the user import data (UC 1)

The user can preview the imported data in the scroll pane on the left hand side, to ensure the data has been uploaded correctly and manage the anonymisation settings on the right (UC 2 and 4)

The user can specify the save location generate the new dataset (UC 5)

4 Design and Implementation

4.1 Methodology

Implementation of the system was done through stepwise refinement. After understanding the high architecture, a process flow chart was created. The flow chart contained a high level description of the tasks that needed to be completed. This aided in the creation of a class diagram. Once the key classes were identified an architectural design pattern was selected. The next stage was to identify implementation details such as the tools, data models and algorithms to be used. Finally, mock data was needed to develop the system. The process for creating this data is presented at the end of this chapter.

4.2 High level architecture

The proposed application will serve as a middleware system that will anonymise datasets as well as generate novel datasets. The resulting datasets will be used as input for future research and software development tasks. A high level architecture of the system is provided below:

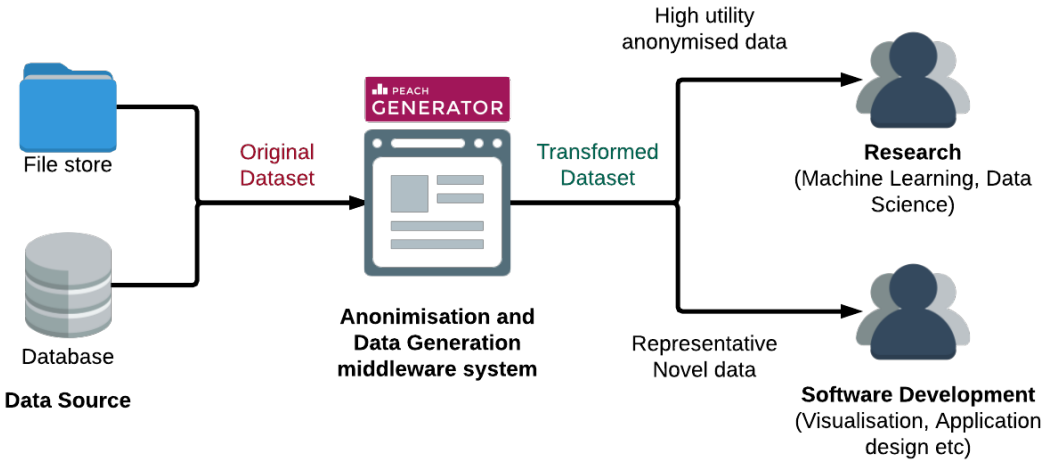


Figure 5 High level architecture

4.3 Process flow chart

A process flow chart created to formalise the approach of the system and identify major components. The flow chart is shown in Figure 6.

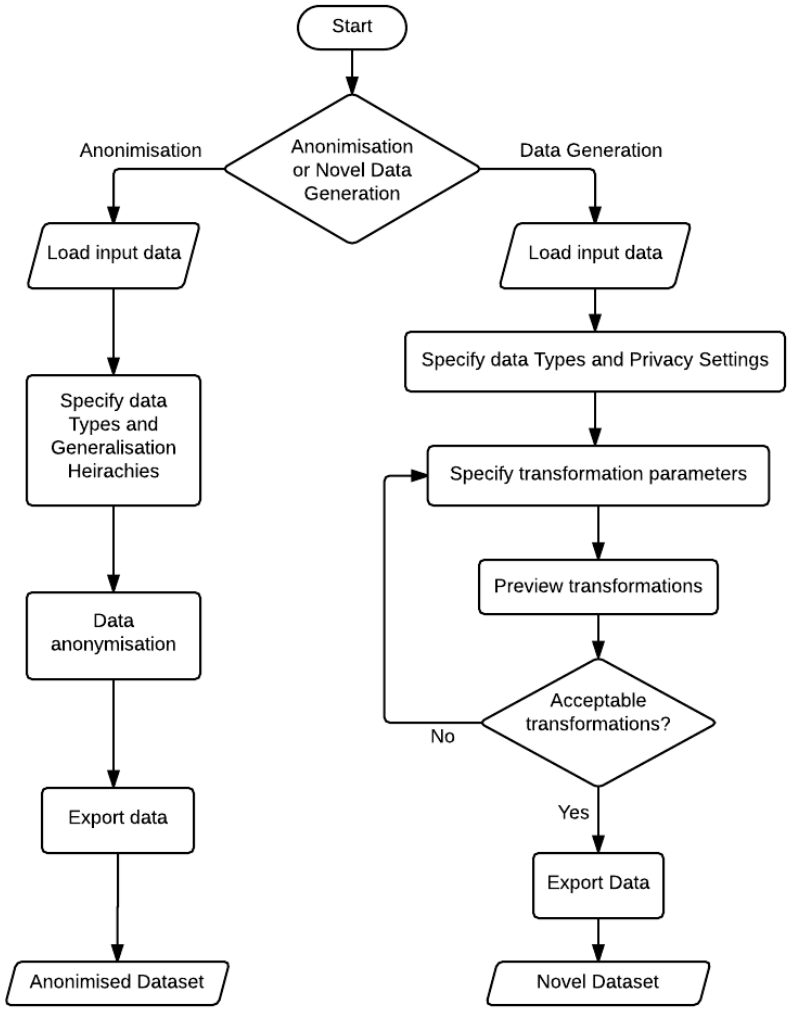


Figure 6 Process Flowchart

The system will have two streams that correspond with the user goals of the two actors i.e. Anonymisation or mock data generation. The two streams share very similar process flows, with an import, modification of settings and an output stage. However, as shown in Figure 4, the anonymisation import, settings and export functionality is provided on a single form. However, in the case for the data generation case, they are two views. Therefore, while the processes were similar they were kept separate.

4.4 Class Diagram

The components of the flow chart were analysed to identify candidate classes. These classes were then modelled using a class diagram. The class diagram is a static structural representation of the system. It outlines how the major components of the system are related. In continuation of the stepwise refinement, the internal elements of the classes and their return type were also modelled. For the sake of clarity, several common methods and instance variables have been aggregated in the class diagram. They are explained below:

- **ElementGUI** element: These include standard JavaFX components such as Button, Gridpane, RadioButton etc.
- **File** dataFiles: These include input and output files
- **Setters** and **Getters**: Methods used for setting or getting instance variables.

POJOs were utilised to encapsulate anonymisation settings and plot settings, these are simple classes that contained only getters and setters. These classes have been omitted from the class diagram.

A non-functional requirement for the project (R.24) states that the system must be easy to extend. Chapter 2 identified limitations in the techniques that have been selected for anonymisation. Therefore, the creation of an architecture that allowed for easily extending the design was a priority. The shuffler, noise generator and anonymiser classes would implement the anonymisation techniques outlined in Chapter 2. However, it is noted that state of the art methods were significantly more sophisticated. Therefore, these classes were designed to interfaces to facilitate the implementation of more complex versions of these classes without the requirement for significant refactoring of the codebase.

The class diagram for the application is shown in Figure 7

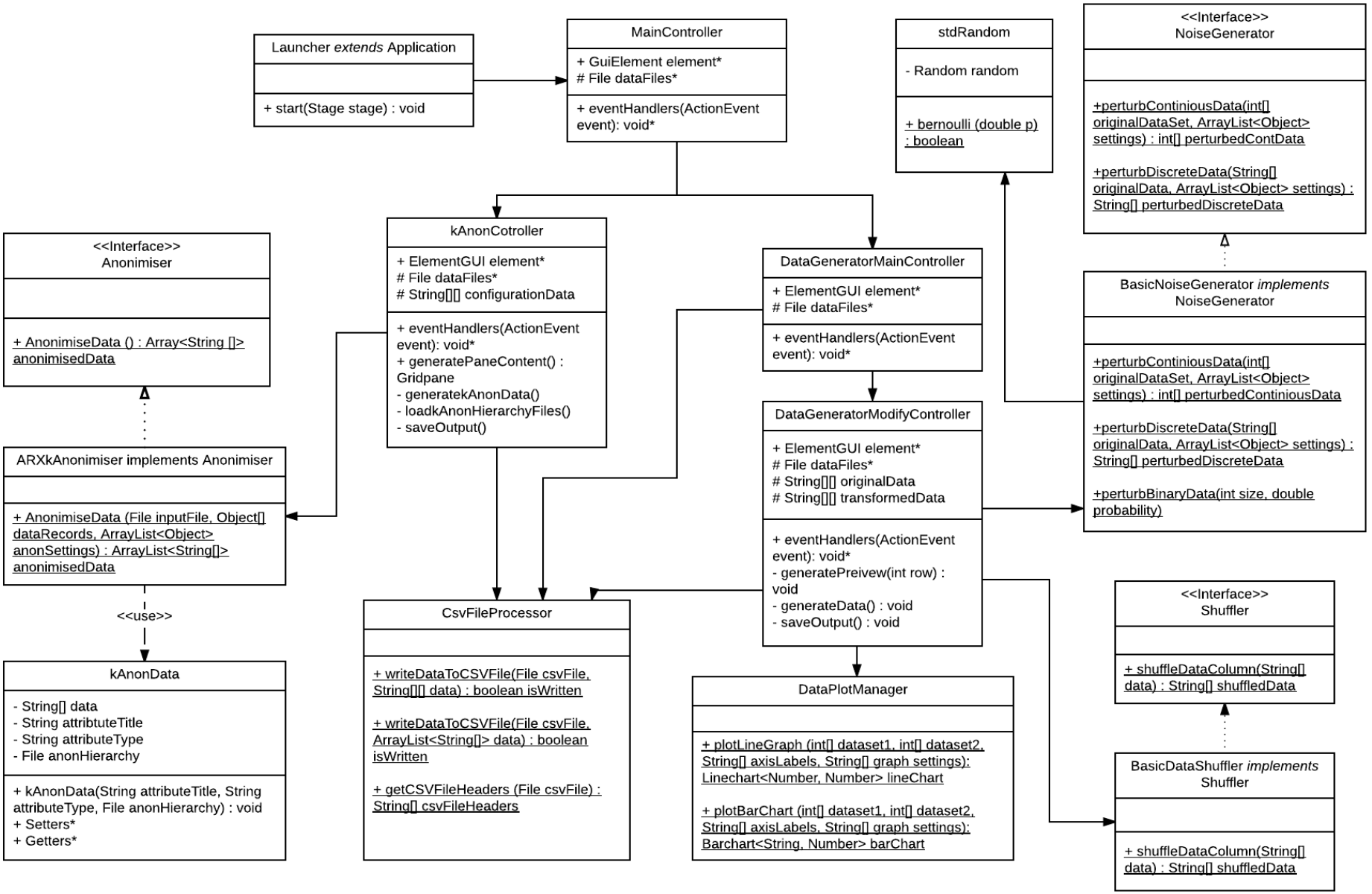


Figure 7 Class diagram

4.5 Architecture Design Pattern: Model-View-Presenter

The application used a Model-View-Presenter architectural design pattern. A representation of this architecture is shown in Figure 9. Model-View-Presenter is designed to facilitate the use of automated testing. It seeks to improve the separation of concerns to an even greater extent than the MVC approach. In this framework, the presenter layer contains all business logic. Once an event is detected by the view layer, information is passed to the presentation layer through event handlers or data-binding. The presenter layer is able to execute the business logic and make the relevant changes to the model. The presenter layer then updates the view. As all the business logic is executed in the presenter layer it eases the implementation of unit testing. JavaFX uses FXML files to define GUI elements, these are represent as XML in Figure 9. The implementation of the architecture can be seen in the source code structure shown in Figure 8.

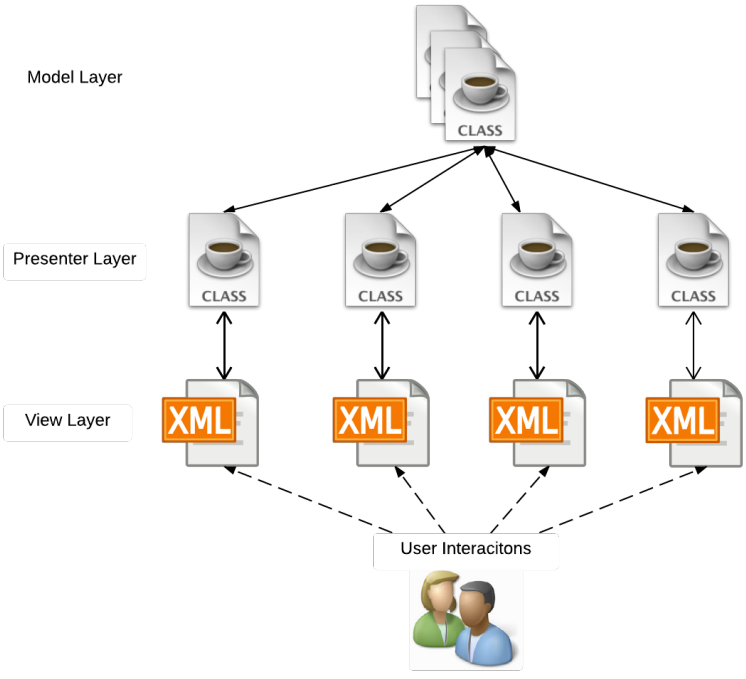


Figure 9 Model-View-Presenter Architecture

```
src
├── Presenter
│   ├── Launcher.java
│   ├── MainController.java
│   ├── NoiseInjectorMainController.java
│   ├── NoiseInjectorModifyController.java
│   └── kAnonController.java
├── external
│   └── utility
│       ├── ARXBaseClass.java
│       ├── AliasMethod.java
│       └── StdRandom.java
├── model
│   ├── ARXkAnonymiser.java
│   ├── BarGraphSettings.java
│   ├── DataIdentifier.java
│   ├── DataShuffler.java
│   ├── LineGraphSettings.java
│   ├── NoiseGenerator.java
│   └── kAnonData.java
├── tests
│   ├── Presenter
│   │   ├── MainControllerTest.java
│   │   └── NoiseInjectorMainControllerTest.java
│   ├── model
│   │   ├── ARXkAnonymiserTest.java
│   │   ├── BarGraphSettingsTest.java
│   │   ├── DataShufflerTest.java
│   │   ├── LineGraphSettingsTest.java
│   │   ├── NoiseGeneratorTest.java
│   │   └── kAnonDataTest.java
│   └── utility
│       ├── CsvFileProcessorTest.java
│       └── DataPlotManagerTest.java
├── utility
│   ├── CsvFileProcessor.java
│   └── DataPlotManager.java
└── view
    ├── kAnonDataInput.fxml
    ├── mainGUI.fxml
    ├── noiseInjectorGUImain.fxml
    └── noiseInjectorGUImodify.fxml
```

Figure 8 Project Structure

4.6 Data Model:

The input data is in the form of a table. To effectively model the data in memory, in-memory databases HS [15] and JavaDB [16] were investigated. However, the extraction, modification and

re-insertion of data using these databases required a large overhead relative to the functionality that was needed. Therefore, in memory database were not used. Following this, standard CSV file handlers, SuperCSV [17] and OpenCSV [18] libraries were considered for use. After investigating these libraries, it was concluded that the functionality required for the application could be achieved more intuitively and easily without the use of these libraries. Ultimately, it was decided that simple 2-dimensional arrays were sufficient for the data storage. A utility class; *CsvFileProcessor* was created for manipulating these arrays.

4.7 Implementation of anonymisation techniques

When selecting which anonymisation techniques to implement, it was noted that the overall aim of this project is to create a proof of concept application that serves as the foundation for future iterations. Implementation of state of the methods would have required a significantly longer project duration. Therefore, pragmatism heavily influenced the final implementation decisions.

4.7.1 k-Anonymity

As per the findings in chapter 2, k-anonymity is implemented using the ARX de-identifier API. In order to anonymise an attribute, it must first be marked as a quasi-identifier by the user. Quasi-identifiers can be used in combination to re-identify individual records. However, if several other records with the same Quasi-identifiers exist (and other identifying attributes have been obscured) individual record re-identification can not occur. If this is not the case, they must be generalised so that several records share the same value for these attributes thereby allowing the entire dataset to achieve a k-anonymity property. In order to carry out these generalisation, a generalisation hierarchy is required. For an attribute such as age, one hierarchy could be [10-20], [10-40], [10-80] and >80. If the original quasi-identifying attribute was age which had a value of 19, the k-anonymity algorithm attempts to locate other records with 19 as the age. If k-1 records can be found where age is 19 and all other quasi-identifiers are the same, then the algorithm has no need to generalise this variable, provided that k-1 other sets for the other ages in the dataset exist. If k-1 records can not be found the algorithm must generalise this attribute to [10-20] to increase the chances of matching the attributes of this record to k-1 other records. It is evident the probability of matching increases as generalisation occurs because once generalised all records between the

ages of 10-20 can be matched. If further generalisation is required, the algorithm will refer the next value in the hierarchy, in this case it would be [10-40]. As the system must only fulfil proof of concept requirements, it currently contains pre-defined hierarchies for only age and gender.

4.7.2 Random sampling of discrete distributions

Generating randomized categorical values requires random sampling from discrete distributions. There are several ways to achieve this. The technique implemented in this application was the Alias method which is one of the most efficient methods for sampling from a discrete distribution. With regards to time complexity, after O(n log n) pre-processing time, random discrete variables can be drawn from the distribution in O(1) time. An open source Java implementation of this method was obtained and tested prior to implementation.

4.7.3 Noise addition to continuous variable

As mentioned in Chapter 2, the additional of uncorrelated Gaussian noise is a simple method for applying noise to a continuous distribution. Since the requirements stipulate that users should be able to specify the amount of added noise, the noise to be added can be scaled by a factor specified by the user. Figure 10 shows the implementation used in this project.

```
for(int i=0;i<originalDataSet.length;i++){

    /* apply additive Gaussian noise */
    randNoise = rand.nextGaussian()*variance*scalingFactor;
    perturbData[i] = (int)((double) originalDataSet[i] + randNoise);

    /* ensure min value within acceptable range */
    if (perturbData[i] < MIN_THRESHOLD){
        perturbData[i] = MIN_VALUE;
    }
}
```

Figure 10 Implementation of additive noise

4.7.4 Data Shuffling

Data shuffling can be achieved through the application of the Fisher-Yates shuffle algorithm. The Fisher-Yates algorithm randomly permeates a finite set. While there exist more sophisticated, non random methods for shuffling that minimise information loss, the use case in this instance is the generation of mock data and therefore, information loss optimisation is of a lesser concern. *Collections.Shuffle* is a standard Java library method which can be used to randomly permeate a

finite set. This method implements the modern version of the Fisher-Yates algorithm which optimises time complexity to O(n) from $O(n^2)$, which would be the time complexity if a naïve solution was implemented. *Collections.shuffle* was used to implement data shuffling.

4.7.5 Data-Swapping

The proof of concept application currently only supports the swapping of names. A list of unisex names is stored within the project folder. These names are used to replace all the records in an attribute when the user selects the *generate novel* option in the program. The list of names is loaded into memory and randomly sampled.

4.8 Automatic detection of continuous and categorical values

In order aid usability, the system automatically attempts to detect whether a variable is continuous or categorical. This is based on a rule based system where the system performs a check to to make sure the value is numerical. If the value is not numerical it is classified as categorical. If the value is numerical, the frequencies for all the values of the attribute in the dataset are computed. If the number of unique values is less than 10%, the system recommends a categorical label, otherwise the variable is tagged as continuous. The implementation of this algorithm is shown below:

```
if(org.apache.commons.lang.StringUtils.isNumeric(colData[1])){
    /* Check data type */
    HashMap<String,Integer> inputData = DataProperty.getEnumFrequency(colData);
    if(inputData.size() > 0.1*colData.length){
        return "continious";
    } else {
        return "categorical";
    }
} else {
    return "categorical";
}
```

Figure 11 Automatic detection of categorical values

4.9 Tools and Libraries:

The application was designed using the JavaFX graphics library. For testing, unit tests were run using JUnit. For automated GUI testing, TestFX was used. Git and Github were used as version control and repository hosting services respectively. The application was developed using the IntelliJ IDEA IDE.

ARX API was used to implement the k-anonymity privacy model. ARX de-identifier [11] is an open source software used for privacy preserving micro-data publishing. It has an API that allows anonymisation to be added to any Java applications. ARX provides functionality for the syntactic anonymisation techniques mentioned in Chapter 2.

4.10 Synthetic data modelling

For the purposes of this project, the development team was given access to a radiology dataset. This dataset was a subset of a database dump of the production radiology database. The files were provided in CSV format. Fields in each table were analysed to identify the most sensitive table with respect to personal and sensitive information. The table selected was the demographics table labelled *DATADEMOG*. This table was cleaned up to remove superfluous attributed. The table was them modelled in a MySQL database. The creation of the database in MySQL was done to allow for additional fields and relations to be added during the development of the toolkit. Where the *DATADEMOG* table contained references to external tables, simplified versions of these tables were created. The final schema for the test database is as show:

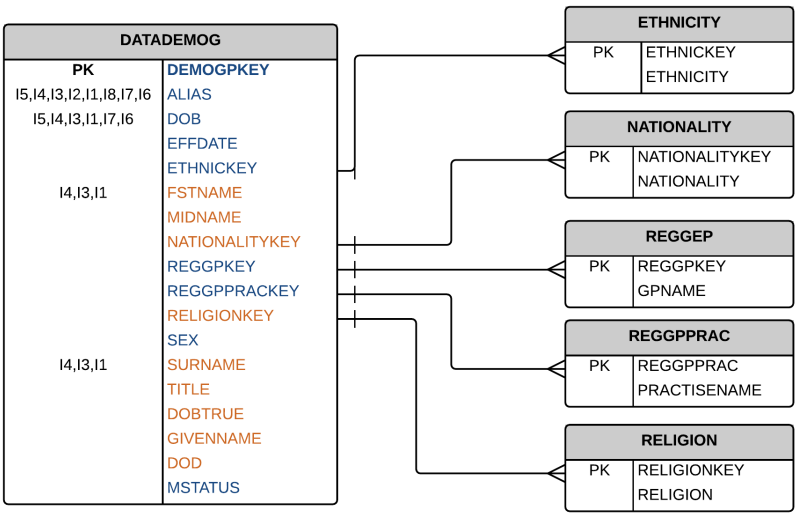


Figure 12 Mock database schema

This database was populated using an online data generator, 100 records were created for the DATADEMOG table. This table was then exported to Csv. The exported file served as input for the application during development.

5 Testing and system evaluation

This section describes the overall strategy for the testing and then proceeds to take a more detailed look at each of the methods used. This includes a description of the testing method, examples and then the outcome of the testing.

5.1 Testing Strategy

5.1.1 Unit testing

The elements in this project can be partitioned into two sets, the Graphical elements and the non-graphical elements. Code associated with non-GUI elements included the classes present in the *Model* and *Utility* packages. These classes were called by the presenter to manipulate the data model. For these elements, Test Driven Development was followed. However, for Graphical elements a strict Test Driven Development was not followed. The Graphical layer was designed to have a presenter layer just beneath it as shown in Figure 9. Therefore, the functionality of the GUI was reduced to trivial mapping of controls and could be excluded from Test Driven Development.

5.1.2 Functional testing of anonymisation and data generation classes

Additional Black box functional testing was carried on out on the classes associated with k-anonymity (ARKkAnonimiser) and discrete data-sampling (AliasTable) as these classes were obtained externally.

The anonymisation functionality was tested by creating a k-anonymised dataset using the application, and then testing to ensure that the correct number of equivalence records were present in the data set.

For random categorical data sampling, an array of probabilities was passed into the AliasTable class. Probability distribution functions for the original and randomly sampled plots were compared. By generating a large number of random samples, the difference between the samples could be evaluated against a small tolerance, to evaluate whether the probability densities converged.

5.1.3 Acceptance testing

The purpose of acceptance testing is the ensure that the system created has met the requirements specification. This testing was carried out manually on the finished application. The test cases comprised of all the functional requirements outlined in Table 5.

5.2 Examples of unit testing

5.2.1 Non-GUI elements - Test Driven Development

Test Driven Development was used when developing non-GUI classes and methods. Test Driven Development is a software development process, where the emphasis is on testing the code as development progresses, rather than at the end of the development cycle. Test Driven Development was implemented by adhering the to following steps:

1. Write a test case for a new segment of code that must be written
2. Run the test and ensure the test fails
3. Write a small segment of code to fulfil the test, and only the test.
4. Re-run all tests to ensure any changes to code base have not caused un-intended side effects
5. Repeat until the end of development.

JUnit was used to conduct the testing. An example of a typical test is provided in Figure 13

```java
/**
 * Test checks if data is correctly written to a new file
 * @throws Exception
 */
@Test
public void testWriteAllDataToCSVFile() throws Exception {

    /* Create new output file */
    File testOutputFile = new File("TestOutputFile.csv");

    /* Test method */
    boolean writeResult = CsvFileProcessor.writeAllDatatoCSVFile(testOutputFile,testData);

    /* Check file exists */
    Assert.assertTrue(testOutputFile.isFile());

    /* Check file header and data entered correctly */
    Scanner sc = new Scanner(testOutputFile);
    String strHeader = sc.next();
    String strData = sc.next();

    sc = new Scanner(strHeader);
    sc.useDelimiter(",");

    /* Test headers were correctly written to file */
    Assert.assertTrue(sc.next().equals("heading1"));
    Assert.assertTrue(sc.next().equals("heading2"));

    /* Check method returns true i.e. successful write */
    Assert.assertTrue(writeResult);

    /* remove test file and scanner */
    testOutputFile.delete();
    sc.close();
}
```

Figure 13 Unit test example

Mock objects were created to break dependencies and test individual units of functionality. The @Before JUnit annotation was used to instantiate the mock objects before each test. An example is shown in Figure 14.

```
/* create mock data sets */
protected int[] dataset1 = {1,2,3,4};
protected int[] dataset2 = {5,6,7,8};
String[] xAxisTitles = {"t1","t2","t3","t4"};

@Before
public void intialiseMockObject(){

    barGraphSettings = new BarGraphSettings(
            dataset1,
            dataset2,
            xAxisTitles,
            "TestTitle",
            "TestDataTitle1",
            "TestDataTitle2",
            "TestxAxisTitle",
            "TestyAxisTitle"
    );
}
```

Figure 14 Data mocking example

5.2.2 GUI elements - Unit testing

TestFX was used in conjunction with JUnit to conduct automated GUI tests, an example of a test shown in Figure 15.

```
/**
 * Test radio buttons on main page
 * @throws Exception
 */
@Test
public void testRadioButtonStatus() throws Exception {
    RadioButton generateNovelData = find("#rbkAnon");
    RadioButton anonData = find("#rbNoiseInjector");

    /* Check generate-data and only novel data button is selected */
    click(generateNovelData);
    Assert.assertTrue(generateNovelData.isSelected());
    Assert.assertFalse(anonData.isSelected());

    /* Check Anon-data and only anon data button is selected */
    click(anonData);
    Assert.assertFalse(generateNovelData.isSelected());
    Assert.assertTrue(anonData.isSelected());
}
```

Figure 15 TestFX GUI test example

As shown in Figure 15, TestFX allows for automated simulation of user interaction. In Figure 15 the RadioButton click events are simulated. This test ensures that RadioButton objects were behaving in the correct manner, i.e. multiple incompatible selections were not made.

5.3 Summary of Results

5.3.1 Unit testing code coverage

80 unit tests were created and run. Code coverage was calculated for non-GUI components using the default code coverage plugin provided in the IDE. The final code coverage was 100% of classes, 100% methods with 88%-line coverage.

5.3.2 k-Anonymity anonymisation process

The dataset used for the functional test is shown in the Appendix A. 2. A k-anonymity value of 3 was used for the test. The output from the test was sorted and equivalence classes were highlighted. A summary table of the Equivalence classes and the number of values contained in the are presented in Table 7.

Equivalence class ID	Values in Class	Equivalence class ID	Values in Class
1	6	10	5
2	6	11	6
3	6	12	6
4	7	13	6
5	10	14	4
6	6	15	8
7	7	16	5
8	4	17	4
9	4	Total	100

Table 7 k-Anonymity test results

100 records were tested. As shown in Table 7, 17 equivalence classes were generated. All of these classes contained at least 4 identical records with respect to quasi-identifiers (Age and Gender) that were specified in the test. The input and transformed files are shown in the Appendix A.2. It can be concluded that the ARX k-anonymity library is transforming the data in the expected manner.

5.3.3 Categorical data random sampling

A test probability distribution was passed into the AliasTable method. Then n samples were generated from the AliasTable class. The maximum differences between the probability density values between the original and randomly sampled data were calculated. They are reported in Table 8 Alias table method test

Sample size	Max difference
100	0.13
1000	0.012
10000	0.009
100000	0.00018

Table 8 Alias table method test

As expected, for small samples the difference was relatively large however, as the number of samples drawn increases the probability densities become almost identical. This conforms the correct behaviour of the AliasTable class. The plots for the results are presented in the Appendix A.2

5.3.4 Acceptance testing results

The final application was reviewed with respect to the original requirements outlined in Table 5. It was concluded that 100% (15/15) of the *Must* priority features were implemented, 100% (1/1) of the *Should* priority features were implemented and 25% (1/4) of the *Could* features were implemented. This results in 86% (19/22) of the original functional requirements being fulfilled. A summary of the features that passed the manual acceptance tests are shown in the Appendix A.2.

6 Conclusion and Project Evaluation

6.1 Project Achievements

This chapter concludes the project by revisiting the aims and objectives of the project and assessing whether each of the original goals were met. An evaluation of the project assesses the process that was implement to deliver the project. Finally, future works and concluding remarks are presented.

6.1.1 Project Achievements

"The goal of this project is to create an easy to use, proof of concept software which acts as a holistic toolkit for generating novel patient data and anonymizing existing patient data."

A proof of concept anonymisation toolkit was successfully created. The toolkit can be regarded as holistic as it fulfils both the requirements of mock data generation and anonymisation, this is in contrast to current tools which implement only one of these features.

"Generate representative mock data from existing datasets. The output from this generator could be used on simple projects i.e. dashboards, visualisation tools."

The application generates representative data by adding noise from the underlying distribution. Users have the option to specify how much noise they wish to apply. For categorical data novel values are sampled from the distribution in the data. Shuffling and swapping techniques keep the data representative while minimising re-identification risks.

"Generate data that is anonymised and retains an adequate balance between privacy and data utility. The anonymisation feature of the tool should be used to generate datasets that are suitable for use in data analytics and machine learning type projects."

The application that was produced utilised existing libraries which optimise for data transforms which result in the highest utility available, while maintaining low re-identification risks. Therefore, the output data should have the potential to be suitable for tasks that require lower levels of information loss. However, whether the data is suitable depends on the privacy levels specified by the user.

6.1.2 Personal Achievements

"Improve software engineering skills through the design and execution of the application."

Multiple software engineering processes were used to analyse the problem and propose a solution. Especially with regards the development of the architecture. These tools can be used again in the future and therefore, the authors ability of plan and implement software has improved.

"Learn how to use to use the JavaFX graphics library."

The author feels confident in creating new applications using the JavaFX graphics library.

"Implement and Test Driven Development software development process."

Test driven development was used extensively during the implementation of this project. The author is now familiar with implementing Test Driven Development.

6.2 Critical Evaluation

As mentioned in section 5.3.4, 3 of the 22 requirements identified in Table 5 could not be implement. These were requirements R.17, R.19 and R.20. R.17 states that the user should be allowed to upload generalisation hierarchies. This feature could not be implemented because of project delivery constraints. This was also the case for R.19. However, for the purposes of a proof of concept software, these omissions were acceptable and hence it was given a *Could* prioritisation at the requirements phase. R.20 states automatic detection of functional dependencies. Following research into this feature, it was concluded that the implementation of this feature was unfeasible within the project timeline as it pertains to a problem which is under active research. As the project achieved 100% of the *Must* requirements it can be considered a success

6.2.1 Data utility and data re-identification metrics not evaluated.

For datasets that required anonymisation, k-anonymity was achieved through the application of generalisation hierarchies. This represented a smaller change to the original data than the combination of noise addition, shuffling, obfuscation and data swapping that was performed on the data when trying to generate novel data. Therefore, it can be concluded that the anonymised

data had higher utility whereas the novel data had lower re-identification risks. However, the absolute value of the utility and re-identification risk following transformation for both data sets was not explicitly calculated. Calculation of these metrics requires significant investigation into information theory as there is no consensus on which metrics should be used to measure these values.

6.2.2 Review of Implementation

The application was designed almost exactly to the class diagram and mock-ups shown in chapters 4 and 3 respectively. Once the planning elements were in place, the project was relatively well defined. This helped greatly during the development phase.

With respect to technology design decisions, the application was developed in Java as several useful anonymisation libraries such as ARX de-identifier were available for this language. This greatly benefited the development of the application as it facilitated code re-use.

6.3 Future work

6.3.1 Implement additional anonymisation models that extend k-anonymity, techniques such as l-diversity and t-closeness

Additional anonymisation models can be implemented by using the ARX API that was used in this project. It supports the functionality to do so. These additional models further minimise the re-identification risk of anonymised data sets and should be investigated.

6.3.2 Implement semantic privacy models such as differential privacy

Semantic privacy techniques such as differential privacy are the current state of the art technique for anonymisation. They offer the potential for a good trade off between high utility data and low re-identification risks.

6.3.3 Styling to fit in the rest of the platform

A PEACH design guide was created in order for all the tools developed in the PEACH suite to have a homogenous look. The design guide should be applied to this tool in the future.

6.4 Concluding Remarks

PEACH was essentially a greenfield project. As a result, much time was spent deciding which direction to take the platform down. There were also issues with access to data at the beginning of the project and these were not resolved until the middle of the project. This led to project mobilisation taking significant longer than anticipated and ultimately resulted in a very challenging delivery schedule. In hindsight arrangements such as the procurement of data and other 'mission critical' tasks should be resolved prior to the commencement of a project.

Fortunately, future iterations of this project should not suffer from these shortcomings. Significant work has been undertaken to define the scope and direction of the PEACH platform and the suite of projects that fall under it. Furthermore, the data sharing agreement is in place for a subset of data that is stored by UCLH. For these reasons, project mobilisation should be easier in the future.