PRIFYSGOL
BANGOR
UNIVERSITY

School of Computer Science
College of Physical & Applied Sciences

Report-card analysis of tabular flight data

Delvin Varghese

Submitted in partial satisfaction of the requirements for the
Degree of Master of Science
in Computer Science

Supervisor Dr. J. C. Roberts

September, 2015

Acknowledgements

I would like to thank Dr. Jonathan Roberts and Steve Russell for supervising me so brilliantly over the last year. Feedback was promptly given when required, and they were always at hand to help steer the project forward. Working with Data Exchange has been a great experience, and I am appreciative of their input and help in this project. Despite structural changes at Data Exchange, Steve was very much interested in the project and believed in its potential, so we owe a special gratitude to him. Special mention goes to my housemate Jason Norman, with whom I enjoyed many a lunch break watching The Office and eating an unhealthy amount of bacon and eggs, alongside a nice relaxing cup of filter coffee.

This project was funded by the ATM scheme. Access to Masters (ATM) is a pan-Wales higher level skills initiative led by Swansea University on behalf of the HE sector in Wales. It is part funded by the Welsh Government's European Social Fund (ESF) convergence programme for West Wales and the Valleys.

Statement of Originality

The work presented in this thesis/dissertation is entirely from the studies of the individual student, except where otherwise stated. Where derivations are presented and the origin of the work is either wholly or in part from other sources, then full reference is given to the original author. This work has not been presented previously for any degree, nor is it at present under consideration by any other degree awarding body.

Student:

Delvin Varghese

Statement of Availability

I hereby acknowledge the availability of any part of this thesis/dissertation for viewing, photocopying or incorporation into future studies, providing that full reference is given to the origins of any information contained herein. I further give permission for a copy of this work to be deposited with the Bangor University Institutional Digital Repository, and/or in any other repository authorised for use by Bangor University and where necessary have gained the required permissions for the use of third party material. I acknowledge that Bangor University may make the title and a summary of this thesis/dissertation freely available.

Student:

Delvin Varghese

Abstract

In this thesis, we detail our efforts of working with Data Exchange, a data analytics company specialising in flight datasets. The outcome of the thesis is a software that will be used by our company partner to generate health reports for datasets they receive from their clients. The received data is comprehensively analysed for missing values, syntactic, semantic and coverage errors; a Report Card is generated based on how well the dataset fares against the expected level of errors. This report card can then be shared with the client and raise concerns or commend the quality of the data. Previously generated report cards can be viewed later through a dedicated interface. Feedback from the client and usability studies have confirmed that the tool is an effective device in facilitating conversations about data health and how to improve it, particularly for sharing this information with audiences that are not tech-savvy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In a technology-driven world, data-saturation is at an all time high. Everything from salesmen to travel agencies generate data. With the abundance of data comes the need for tools and professionals equipped to decipher the data and derive meaningful analytics from it. It is no wonder then that Data Scientist has been called the sexiest job of the 21st century by Harvard Business Review [10]. This thesis deals with the data problem at a smaller scale, a UK travel agency data centre. Data Exchange is a small software company based in North Wales who provide analytics for travel agencies across the UK.

Data Exchange receives a large volume of data each month from the agencies they work with. However, before this data can be manipulated and used for further analysis, the data needs to be in an acceptable condition. Often, the datasets received are not in the correct format, and this causes problems for the clients' analytics software.

A solution is required that would enable them to identify data errors before a decision can be made whether to use the dataset for analysis, or to get in touch with the agency from where the dataset originated, and discuss improvements needed.

1.1 Hypothesis

When travel agencies send large quantities of data on a monthly basis, it would be efficient and productive to have a data quality analysing software

present that could check the quality and health of the incoming data. If such a software existed, it would make data import and further manipulation much easier and reduce the need for manual interference.

The hypothesis of this project could be stated succinctly as done below:

- *parse a raw dataset to identify important data attributes*
- *efficiently analyse what areas of the data needed improvement*
- *this analysis needs to be in a format easily shared with the client's data partners.*

If these three hypothesis are met, and meet the testable criteria attached to them, then the client's time can be diverted effectively to other tasks that require greater cognitive functioning.

1.2 Objectives

The author aims to achieve the following objectives though this project:

1. clear and intuitive user interface to enable client to check data quality without resorting to messy command line tools.
2. simple user interface to quickly assess data quality. Time taken to parse and check data should be minimal and enable client to focus on more important tasks.
3. availability of past results to check if improvements made on data. Results from an analysis of a dataset should be available in the future to compare future improvements against.
4. quick access to data quality on large quantities of data

1.3 Project Requirements

This section deals with the requirements of the system. The project has been carried out by Delvin Varghese (the author) working with the client, Data Exchange Wales under the supervision of Dr. Jonathan Roberts.

As described above, the purpose of this project is to produce a program that will visualize data quality errors in the client's flight datasets. It was decided to work in close association with the client in designing the application as our client was acutely aware of the data dimensions they wanted addressed in the solution, and possessed the knowledge and domain awareness required to help in the design process.

Mr. Stephen Russell was the primary representative for Data Exchange Wales (referred to as DX from here on), and as such, he will be the main user of this program. The user sees himself using this program on a monthly basis on incoming datasets, sent in by DX's many clients. The datasets are checked using this tool to check for errors as devised in the design specification, and a decision is made based on the error rating. If data quality is acceptable, then the client can do further processing with the data. If data quality is not acceptable, then the client is informed on which dimensions of data quality failed to meet the requirements and this information is relayed to DX's data providers so they can resend the data or send better quality data in the future. Mr. Russell has informed us that he will be the sole user of the program.

1.3.1 Application Context

Currently, the client manually scans the incoming data for errors and has stated that this approach is error prone. The client's patrons send their data to the client in the form of a csv file. The csv file contains the data attribute headers and data attribute contents in the form of strings. This csv file is then imported by the client into his MySQL database table. This table can then

be analysed for errors either manually, as done by the client at present, or through data quality analysis, as the project envisions it being done.

Once the client visually scans through the large table of data and if he detects any errors, he either corrects them or alerts the patron about the quality problem. At a future date, if the patron sends a modified version of the data, then the client has to analyse the old and the new data to comparatively deduce if there have been any changes. This is a very tedious process.

The datasets have a substantial number of columns (between 15-50), thus proving difficult to use cognitive processing alone to deduce errors. A systematic approach that processes all the dimensions and keeps track of the error rating would be much more efficient, and if this information is stored would be advantageous to compare improvements or decline in data quality.

1.3.2 Goals of the Project

Purpose of the system

The main goal of this project can be summarised thus: to provide a tool that visualizes errors (or lack thereof) in data provided by the client's customers. The client is sent data of different types each month by their customers that need to be imported into a central database. However, before importing, the client needs to check the "health" or the quality of the data. This is where the project comes in, providing an efficient and visual method of analysing the data quality. The data quality metrics are developed empirically, in association with the client, so that the client can assess the data dimensions of most relevance. The usefulness of the graphical visualization is assessed based on feedback from the client and peers of the researchers. This feedback is taken in the form of usability studies and responses open-ended questions on design criticism.

User description

The sole user of the solution will be the client representative, as described above. Having a single user is advantageous when designing a solution because of clarity in user demands. Mr. Russell is an experienced Computer Scientist, who expects to not only use this program but also programmatically change it in the future to include additional features. He will be heavily involved in program development with regards to oversight on project progress and constant feedback with regards to the requirements that are being addressed in the program.

A key attribute of this dissertation (which is stressed throughout this project) is the client's involvement in the design, development and testing process. Unlike traditional models of development, the agile methodology was embraced (more about this later) to enable the client to have a greater say in the program's development time-line. There were regular meetings with the client, in their office to show prototypes of each stage of program development, to get input and critical feedback.

1.4 Methodology

For this thesis, the Agile software development methodology was used. This methodology encourages iterative work progress, known as sprints and is an alternative to traditional sequential development [35]. This approach was used due to the short time frame of 3.5 months present in the design and development stages of the project. Short weekly cycles were used to iteratively add features to the program. This was followed by feedback from the client and/or the project supervisor before the next stage was undertaken.

The advantage of this methodology was that a working prototype of the program was present at every stage in the program development time-line, which was incrementally improved. The client could critique and comment

on an actual program and not on a third-person description given by the developer. Another advantage was that on a project like this, there is a non-exhaustive list of features to implement. Keeping rolling development stages meant that time could be effectively managed in implementing the necessary features, and the non-essential components could be incrementally added as time permits.

1.5 Structure of dissertation

Chapter 2 will describe a brief background on the material that is covered by the thesis. The reader is then directed to the Related Work chapter (Chapter 3) to further understand the current state of the field in Data Quality and Visualization of Data Quality.

The main body of the dissertation is divided into two sections: Data Quality analysis (Chapter 4) and Visual Depiction (Chapter 5). This is followed by a thorough critique of the research (Chapter 6) and finally, the conclusion (Chapter 7).

Chapter 2

Background

2.1 Overview

This chapter describes the conceptual framework needed to understand the project's aim and its attempt to solve the problems. Before a suitable solution can be devised for the project's aims and objectives, some examples from the information visualization field are shown to bolster the applicability of the solution proposed in this project. Visualizations that served as influences in the addressing of the tabular representation problem and the design of the project solution are also shown.

2.2 Problems in tabular representation

Tables are a useful structure to represent data collected from many everyday sources. For instance, see the simple table described in Table 2.1. Any non-simple visualization of this data will be overkill since the data and the meaning that is to be conveyed can be done so appropriately by presenting the content as-is. Even anomalies can be detected easily by the human eye. For instance, the lack of an *Occupation* value for the person known as Jane Austen will be picked up straight away by the human simply by glancing over the table. This error does not need to be highlighted to be picked up as the human eye performs Preattentive processing[39] to capture this anomaly

First Name	Surname	Age	Occupation	Location
John	Doe	18	Student	Worcester, UK
Jane	Austen	65		Cambridge, UK
Steve	Austin	43	Wrestler	Houston, USA
Lukas	Podolski	31	Footballer	Berlin, Germany

Table 2.1: A small table showing the simple nature of highlighting missing entries with small tables.

based on the contrast between the surrounding filled values and the single empty value.

Now consider a large data structure of thousands of rows. Very quickly one will realise that once the quantity of data exceeds the space available on the screen, either details or the summary level information is lost to the viewer. If the reader decides on detail, to detect the errors, only a select quantity of the data can be viewed on screen and thus the reader will not be able to see an overview of the total health or error quality of the data. If on the other hand the reader decides on a “zoomed-out” view, an overview of specific errors like missing values will be visible, but other errors like incorrect data, size exceptions etc. will not be inherently obvious.

This is where graphical visualization has a key part to play. In professional settings where time and resources are valuable, any time that is redundantly spent on data analysis should be reduced as much as possible. Visualization of high quantity datasets can speed up the process of gaining insights from raw data. This applies even when data quantity might be low but number of data dimensions may be high. In the above instance, if the data had more columns to represent full address, political views, religious views, health issues, number of vehicles, marital status etc., the variables in play quickly increase, and the need for visualization emerges.

In the context of the project partner, Data Exchange, high data volume and data variety are regular aspects of the data analysis process. As such, a solution is required to automate the process as much as possible, to save time spent on tedious data analysis tasks. As a company that works with travel agencies, the data they handle contains dozens of columns addressing

each aspect of a typical travel itinerary. This can range from fields that deal with flight information, to origin and destination information, to hotel and accommodation data. A robust visualization solution is needed to address the client's data analysis problem.

First Name	Surname	Age	Occupation	Location
John	Doe	18	Student	Worcester, UK
Jane	Austen	65		Cambridge, UK
Steve	Austin	43	Wrestler	Houston, USA
Lukas	Podolski	31	Footballer	Berlin, Germany
Vladmir	Putin	55	not set	Moscow, Russia
Keith	Green	unknown	not set	Nashville, USA
John	Lenn0x	71	Professor	Oxford, UK
Clive	Lewis	-5	Author	Oxford, United Kingdom

Table 2.2: When the number of rows and columns increases, and the types of errors present is magnified, they are harder to spot. Careful observation is needed to see the errors.

Consider an extension of the Table 2.1 in Table 2.2. The errors present in the former table are easily picked up. However, in the extended table there are a number of errors which need careful observation to be picked up:

- missing Occupation for Jane (missing error)
- Vladmir and Keith are missing Occupation field but not obvious because of placeholder text (actually a missing error but not recognised as such)
- Keith's Age attribute doesn't match the format of the column (format check error)
- John's surname contains a digit (format check error)
- Clive's age is a valid integer but a negative value (out of allowed range error)
- John and Clive live in the same Location but parsed as different because of inconsistent use of abbreviations.

The table is still a small data sample but nevertheless, gives a glimpse of the variety of data quality errors that are present in raw unprocessed data. Visualization techniques are needed to highlight these errors so that the "dirty data" can be "cleaned" and ready for further analysis.

2.3 Utilizing graphical visualization

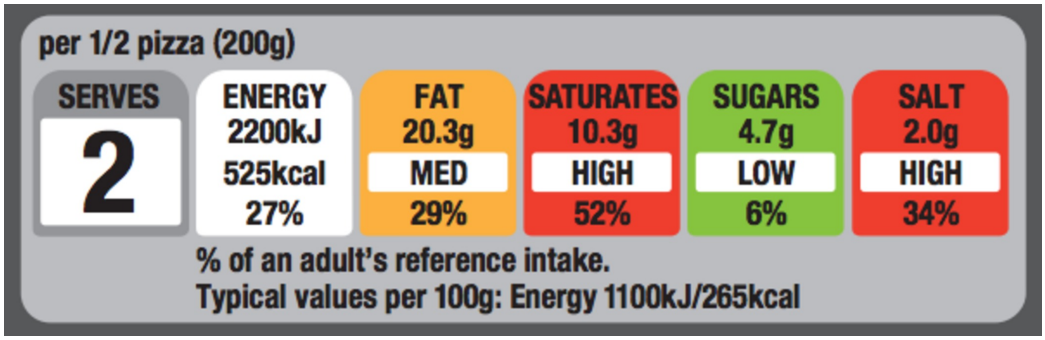

Figure 2.1: Food packaging label
Food packaging label showing important attributes about the food's nutritional contents in a single glance utilising visualization techniques like colour charts. Source:`http://theconversation.com/after-three-year-saga-health-star-rating-labels-finally-ready-to-go-25794`

There are a plethora of examples where visualization techniques have been used to bring out salient points in the data or to reinforce important points. For example, Figure 2.1 shows an example food packaging label, which shows important information about the nutritional content of the food product. Rather that simply listing the amount of fat, for instance, the amount listed is supplemented with a traffic colour system to visually show whether this is a healthy quantity. Additionally, this is reinforced by a lickert scale ranging from LOW to HIGH, displayed below the quantity.

Similarly, an Energy Performance Certificate, required by law for every self-contained property, is issued by builders and landlords to buyers and tenants of properties. An EPC provides information about the energy usage and Carbon Dioxide emissions of the property. It also has additional information about the potential energy ratings that could be achieved if recommendations are adhered to. Figure 2.2 shows an example of an EPC issued for a property in the UK.

The Energy efficiency component of the EPC utilises a red-green colour scale to visually represent undesirable (not energy efficient) to desirable (very energy efficient) ratings. The Environmental Impact aspect uses a colour

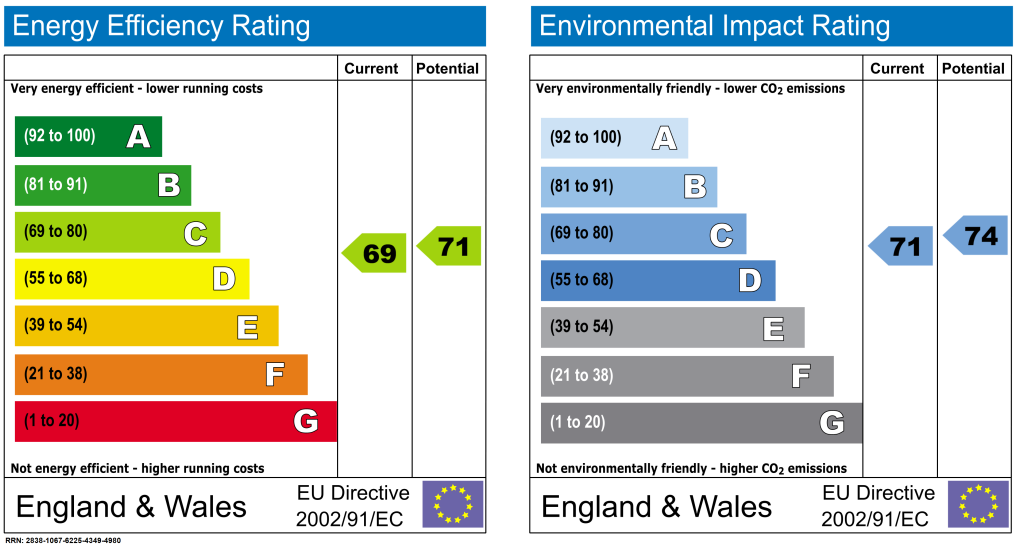

Figure 2.2: EPC Ratings.
An Energy Performance Certificate (EPC) is issued for every property in the UK and is another example of a summary visualization. Source: `http://energyapprise.co.uk/epc-bristol`

scale from dark grey to light blue to display undesirable (not environmentally friendly) to desirable (very environmentally friendly) ratings.

What is poignant is that the energy efficiency and CO_2 emissions scores are derived from a variety of scores that assess the total rating of the property, i.e. it is a score of scores. The final score is cumulative of the scores that check for insulation (roof, floor and wall), heating systems, double glazing etc. This is relevant to this project because data quality is assessed on a number of different factors, and when visualization of the errors takes place this project needs to show a total error rating or health of the data, in a similar vein to the EPC rating.

It would be appropriate to look at some visualization examples as seen on desktop systems or the web. One such instance is the visualization technique used by jeFit, a fitness tracking app available online and on various mobile platforms. Users enter the details of a workout they have done, including various details like name of exercise, amount of repetitions and weight on the bar (if a free-weights or machine-weights based exercise is done). At the end of the week, jeFit sends out an automated email with a weekly fitness summary. This is shown in Figure 2.3.

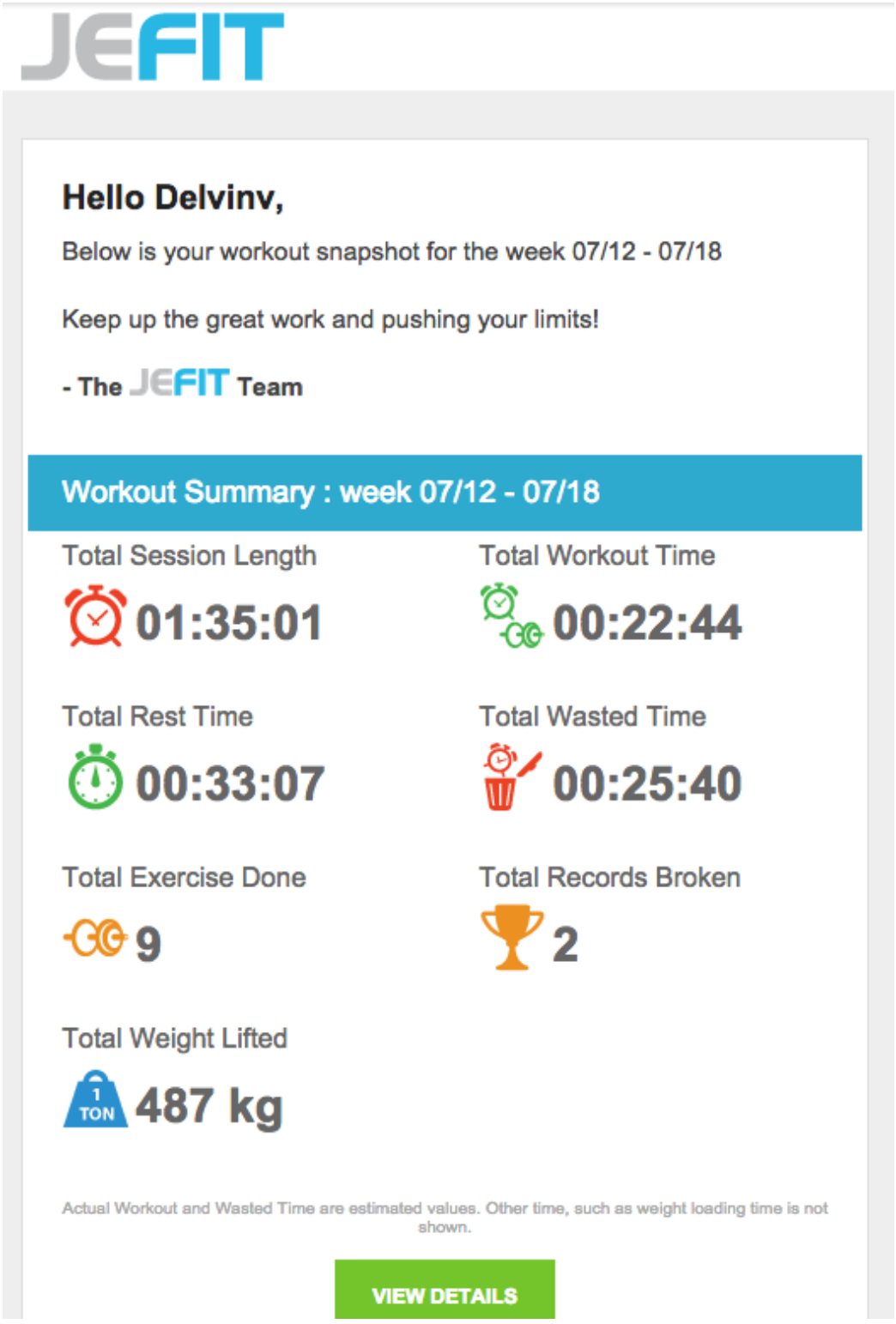

Figure 2.3: jeFit fitness tracker summary email
jeFit fitness tracker summary email. This shows a summary of the individual's workout statistics for the week displayed at the top. Source: `https://www.jefit.com/products/`

This summary visualization contains 7 different metrics calculated from all the exercises that have been tracked that week. The session lengths, rest times and total weights etc. are totalled, and an additional metric of number

of records broken is also given. Appropriate icons are also shown besides the metrics e.g. a clock on top of a bin is shown next to wasted time.

This can be compared against RescueTime's weekly summary email (Figure 2.4). RescueTime is a productivity tracker, that monitors how a user spends their time on a computer. A weekly email is sent at the end of the week showing how productively the time is spent. The visualization that is sent out contains a lot of information that is conveyed in a neat and concise manner. The summary is divided into 3 neat sections: a quick summary of total time spent and how productive that time was, a breakdown of the top applications and websites visited and the top category of use, and lastly how the user fared against their pre-set productivity goals.

RescueTime analyses a lot of different metrics to bring the summary visualization together. From a simple observation, a number of interesting metrics can be seen: number of total hours logged, percentage of time on productive and unproductive activities etc. Colours are used prominently to convey the message: deep blue to convey very productive scores adn a deep red to indicate very unproductive/distracting activities. Hues of these colours are used for activities that are on the spectrum between these two types.

A couple of other visualization examples that inspired this project are shown in Figures 2.5 and 2.6. Figure 2.5 shows the summary used in a budgeting software to show how the user spent their income in a typical month. Different categories of spending are shown in different colours and a textual summary is also given on the side. This is in sharp contrast to the visualization in Figure 2.6 . This is a complex visualization shown in-game to players of the game Call of Duty: Modern Warfare 3. More than a dozen different metrics are shown in a neat and ordered manners, with colours used sparingly for effect. From this particular visualization, we can deduce that the player with the username *rburnett* has played this game for 98 hours and is a Level 30 player. Their Kills-to-death ratio is also given to show how efficient the player is.

2.4 Summary

In this chapter the need for using graphical visualization techniques to show and explore non-simple data quantities has been shown. A number of case studies highlight how existing models have utilised the visualization space to do this efficiently, and they have served as an inspiration for this project. In the next chapter, current literature on the topics at hand are explored in depth, and serve as the building block for the design and implementation of the final solution.

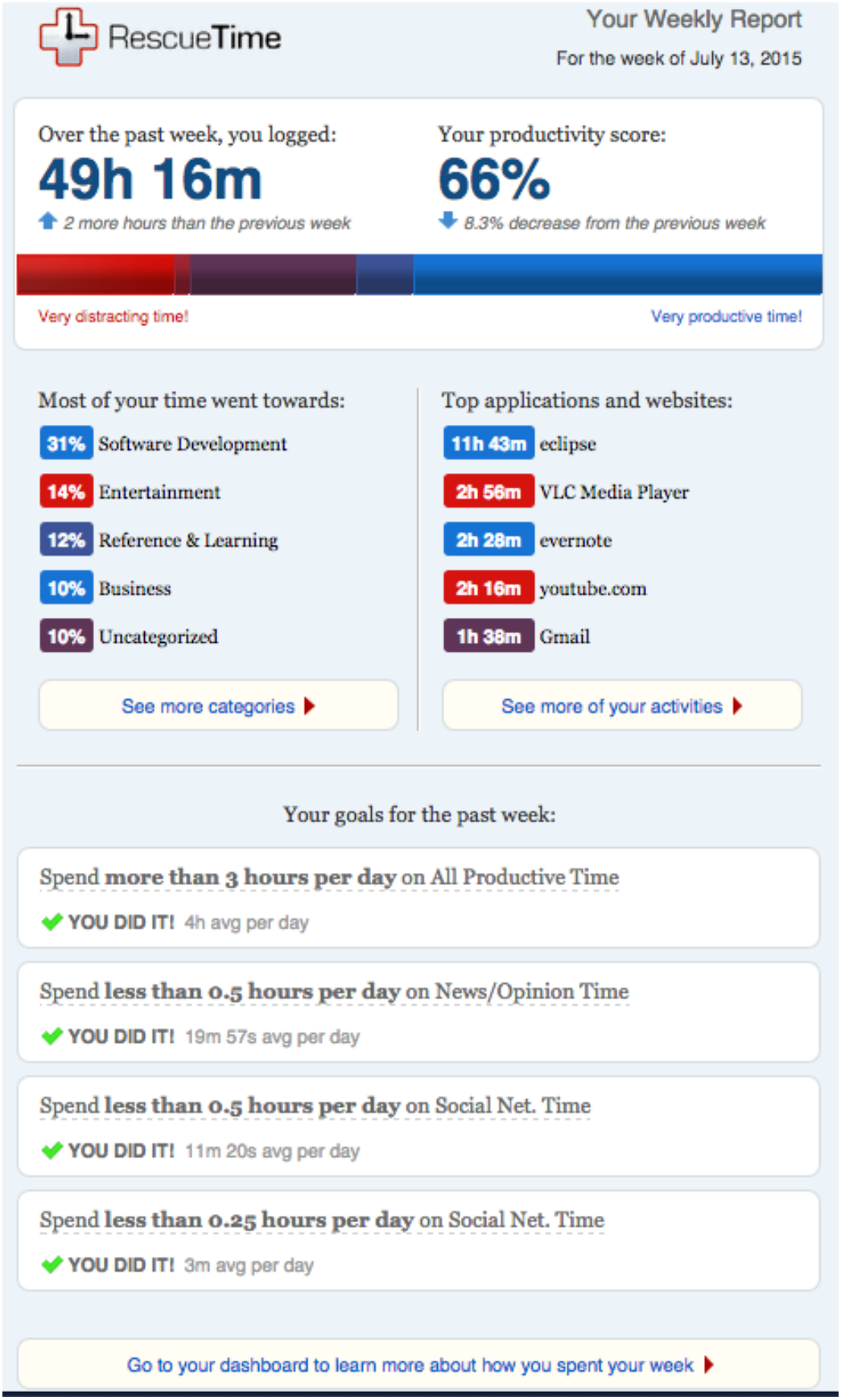

Figure 2.4: RescueTime productivity tracker summary email
RescueTime productivity tracker summary email sent on a weekly basis, and shows how time was spent on a computer. Colour and space are used to neatly convey information. Source: `https://www.rescuetime.com/dashboard`

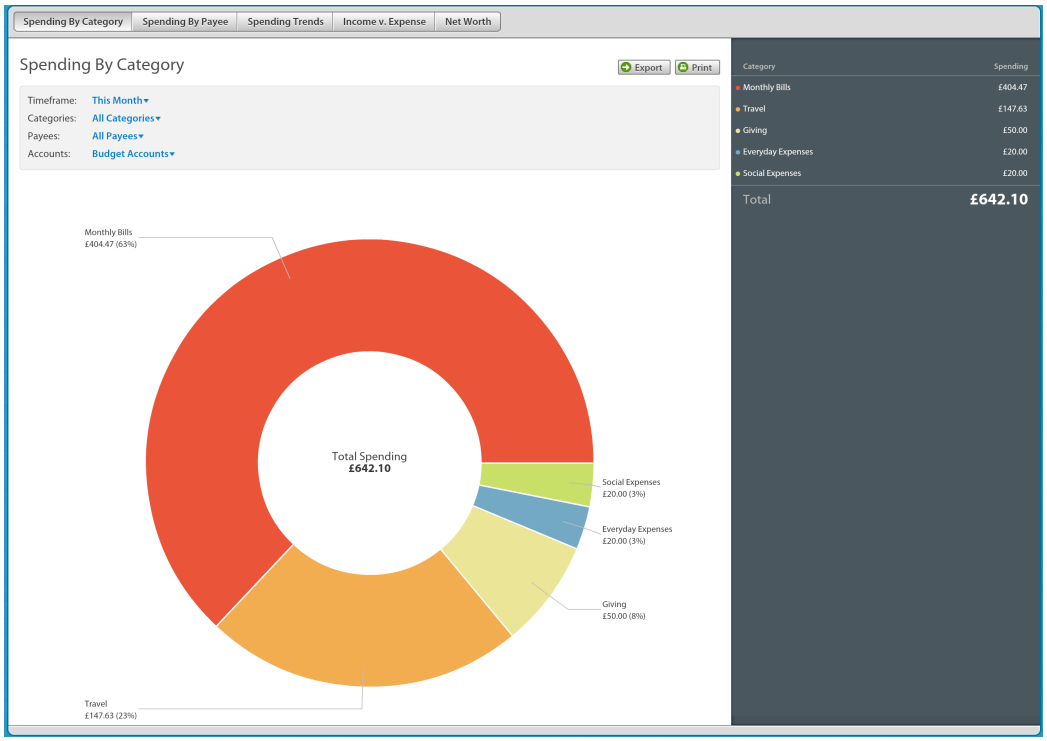

Figure 2.5: YNAB budgeting software summary page
You Need A Budget (YNAB) budgeting software summary page. Source: `https://www.youneedabudget.com/features`

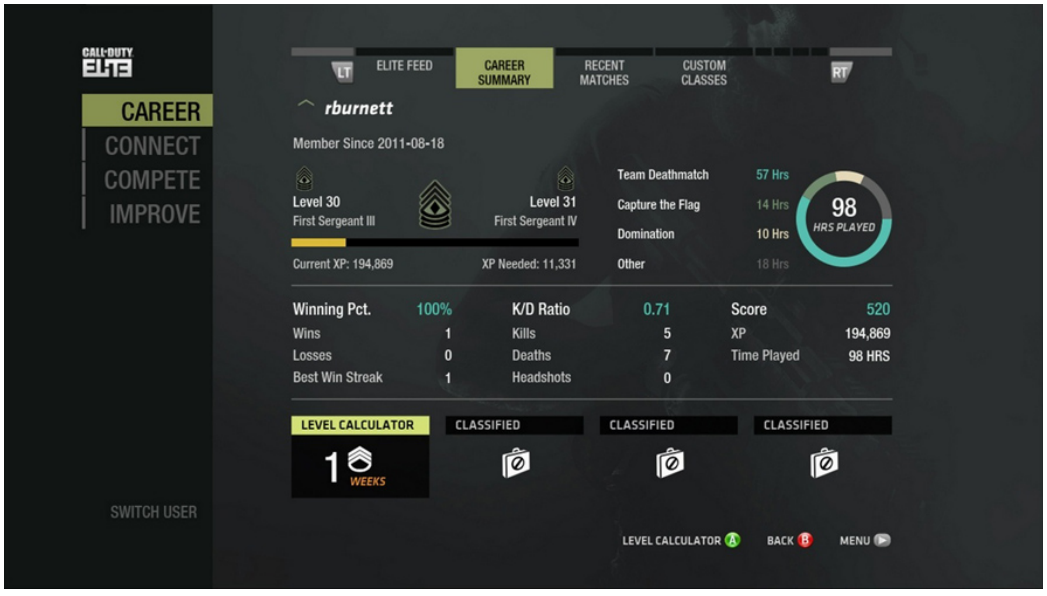

Figure 2.6: Call Of Duty MW3, FPS game summary screen
Call Of Duty MW3, is a First Person Shooter game. This screen is shown to the user as a summary of the player's career achievements and overall statistics. Source: `https://www.callofduty.com/mw3/game/overview`

Chapter 3

Related Work

3.1 Overview

This section looks at the existing literature and how the current project ties in with advances made in the field. There are two distinct aspects to this project that are unified in the final execution: data quality analysis and the visualization of the analysis. The former deals with decisions regarding which data quality dimensions to address when checking for errors and how to arbitrate between good and bad results, whereas the latter deals with visualising the error quality metrics devised as part of the analysis, so it is displayed accurately to the end user.

3.2 Data Quality Analysis

3.2.1 Need for Analysis

Operational databases can have from 60% to 90% of bad data [9] and research has shown that quality of input data detrimentally affects the quality of the results [33][28]. Thus we need to make data usable before we proceed with any operations [19]. There are many sources of data quality problems: human error, differing data models among data sources, change in classification resulting in incorrect categories to name a few. The author agrees with Kandel et al. that tightly integrating visualization into the iterative

process of data cleaning will help unearth data quality issues. Only once data is usable can we proceed to other tasks like deriving knowledge from the data and eventually make this knowledge accessible by utilising visualization techniques [42].

In their short article “Examining Data Quality” [37], Tayi and Ballou offer some insight into key considerations for data quality. They see the term “data quality” as best defined as “fitness for use”, implying that data quality is relative to the context and use of the data, especially when the same data has multiple uses. **Semantics** is highlighted as a key area of data quality, where the value may be correct but it can be easily interpreted. This is an aspect of data quality that will be handled in this dissertation: semantic errors in data. Data that is made available by the client is often far removed from the context in which it was generated since the client is a data acquisition and manages multiple agencies. Therefore the data needs to be checked on a syntactic as well as semantic level for data quality errors.

This approach has been taken based on a survey of current literature and in talks with the data consumer(client) as well as the researchers’ previous experience. Data quality is best expressed via multiple attributes or dimensions. This idea has been expressed by Ballou and Pazer in their important contribution to this discussion [2]. Having an overarching data management methodology is helpful in avoiding errors, as reported by [41]. Similarly, Thomas Redman exhorts those in charge of data output and data quality decisions to beware the consequences of poor quality data and how that affects operational, tactical and strategic decisions [32]. Redman presents some tangible effects of not having a proper data quality methodology in place: customer dissatisfaction, increased cost, ineffective decision making, and the reduced ability to make and execute strategy.

However, it is not possible to implement a data quality methodology at the data source, as the client is a data aggregator, collecting data sources from multiple data agencies that send data on a monthly basis. Therefore it is not feasible to implement a methodology of systematic high-quality data

production on the “assembly line” as it were. The project’s task will therefore be a process of assuming error-prone data is present, and finding ways to highlight these errors so that the client can present this error report to the agencies and encourage a better data output in future.

3.2.2 Addressing Data Quality

In their seminal article on Data Quality [40], Wang and Strong develop a framework [14] for dealing with data quality problems, especially with the data consumer in mind. They explore in detail what it means for a data to have “fitness for use” [37], looking at data from the consumer’s point of view as the consumer is the final judge on the data. They categorized three approaches in which the existing literature studied data quality: *intuitive*, *theoretical* and *empirical*. It was noted that the intuitive approach is undertaken based on the researchers’ experience or intuitive understanding about which attributes are “important”. A feature of this approach is that data quality attributes are often not collected from data consumers.

According to Wang et al. the second approach (theoretical) focusses on how data may become deficient during the data manufacturing process, however, there are very few examples of this approach being used. They note that despite the strengths that the two approaches bring to the table, their weakness is that they focus on the product in terms of development characteristics instead of use characteristics. The voice of the consumer is not heard. Therefore Wang et al. take an empirical approach to utilisation of data quality attributes. Attributes are selected based on their relative importance to the data consumer.

A similar study [7] by Chengalur-Smith et al. investigated the importance of attaching data quality information (a process they called data quality tagging) to make it easy for decision-makers to assess the level of data quality. Their results suggested that for simple tasks, the detailed information provided by data quality tagging is beneficial but for tasks with a higher complexity,

information overload can occur as the task itself has all the necessary quality information by nature of its complexity.

The research of Wang et al. was of particular interest because as part of this dissertation, as there will be a close working relationship with the data consumer and a data quality solution will be built for this consumer. The empirical approach will be used to dialogue with the consumer and identify data quality attributes that are most important to them.

Rahm and Do [30] classified problems in data quality that can be addressed by data cleaning and specified some approaches to take when importing data from sources using data transformation. They discovered that many existing tools only cover part of the problem and still require substantial manual effort or self-programming. This is exacerbated by other limitations like proprietary APIs and metadata representations that make it difficult to use multiple tools together. In their approach to “dirty data”, they laid out requirements that a data cleaning approach should satisfy:

1. detect and remove all major errors and inconsistencies both in individual data sources and when integrating multiple sources.
2. supported by tools to limit manual inspection and programming effort and be extensible to cover additional sources.
3. data cleaning should not be performed in isolation but together with schema-related data transformations based on comprehensive metadata.

Figure 3.1 shows the classification Rahm et al. used for errors at the instance level. For data cleaning they looked at various phases like data analysis, defining transformation workflow and mapping rules, verifying the correctness and effectiveness of the transformation, executing the transformation and back-flow of cleaned data (replacing the dirty data with clean data in original sources to give legacy applications the improved data too.) It is to be noted that Rahm and Do recommend using a DBMS-based repository for the sake of consistency, flexibility and ease of reuse. Another contribution they made

Scope/Problem		Dirty Data	Reasons/Remarks
Attribute	Missing values	phone=9999-999999	unavailable values during data entry (dummy values or null)
	Misspellings	city="Liipzig"	usually typos, phonetic errors
	Cryptic values, Abbreviations	experience="B"; occupation="DB Prog."	
	Embedded values	name="J. Smith 12.02.70 New York"	multiple values entered in one attribute (e.g. in a free-form field)
	Misfielded values	city="Germany"	
Record	Violated attribute dependencies	city="Redmond", zip=77777	city and zip code should correspond
Record type	Word transpositions	$name_1$= "J. Smith", $name_2$="Miller P."	usually in a free-form field
	Duplicated records	emp_1=(name="John Smith",...); emp_2=(name="J. Smith",...)	same employee represented twice due to some data entry errors
	Contradicting records	emp_1=(name="John Smith", bdate=12.02.70); emp_2=(name="John Smith", bdate=12.12.70)	the same real world entity is described by different values
Source	Wrong references	emp=(name="John Smith", deptno=17)	referenced department (17) is defined but wrong

Figure 3.1: Some examples of single-source problems at instance level highlighted by Rahm et al.

was utilising data profiling to re-engineer metadata to address data quality problems. Figure 3.2 shows the table they provided with examples.

Problems	Metadata	Examples/Heuristics
Illegal values	cardinality	e.g., cardinality (gender) > 2 indicates problem
	max, min	max, min should not be outside of permissible range
	variance, deviation	variance, deviation of statistical values should not be higher than threshold
Misspellings	attribute values	sorting on values often brings misspelled values next to correct values
Missing values	null values	percentage/number of null values
	attribute values + default values	presence of default value may indicate real value is missing
Varying value representations	attribute values	comparing attribute value set of a column of one table against that of a column of another table
Duplicates	cardinality + uniqueness	attribute cardinality = # rows should hold
	attribute values	sorting values by number of occurrences; more than 1 occurrence indicates duplicates

Figure 3.2: Examples for the use of *re-engineered metadata* to address data quality problems [Rahm et al.]

Standardization is a preparatory step done before parsing, and involves converting attribute values to a consistent and uniform format to facilitate instance matching and integration. This is pertinent to this thesis because of the presence of different formats in key attribute fields provided by the client. An example of this is the presence of multiple international date formats. Standardizing them will be a necessary precursor as laid out by Rahm and Do.

3.2.3 Differentiating errors

Pipino, Lee and Wang [29] describe principles to help organisations develop usable data quality metrics in their article title "Data Quality Assessment".

They emphasize the need for a combination of subjective and objective data quality metrics.

Developing a single-valued aggregate measure of data quality would be subject to many deficiencies, but, if this is understood and the measure is interpreted accordingly, having such a measure could help companies assess their data quality status. This is especially the case if such a measure or heuristic is used that conveys the improvement (or lack thereof) in the data quality over time. A similar measure is envisioned as the outcome of this research

The work of Olivieria et al. [27], Müller et al. [26] and Kim [23] list data quality problems to check for and categorize the problems based on where they occur: (i) at the attribute/tuple level, (ii) at the relation (table) level, (iii) at level of multiple relations, and (iv) at level of multiple data sources. They can also be categorized as syntactical anomalies, semantic anomalies, and coverage anomalies, the phraseology used by Müller and Freytag. In this thesis, this taxonomy will be used to differentiate between different data quality problems and their detection and capture. These three types of errors can also be expressed simply (as done by Kim) as: (1) missing data, (2) not missing but wrong data, and (3) not wrong but unusable data.

Dividing data quality dimensions hierarchically in these three categories is subject to errors too, as semantic anomalies cannot truly be said to pertain to data as much as it does to information [38]. However, it is hard to isolate information quality as such from data quality, because Semantic ascribes meaning to data from the syntactic level. Similarly, the knowledge level only comes into play once the semantics have been established and it is interpreted correctly. This is especially true when the “data quality” might be what one expects it to be, but “data misinterpretation” affects the results regardless [25]. Therefore , the semantic and coverage anomalies need to be looked at and considered part of the valuation of data quality metrics.

3.3 Visualization of Data Quality

Visualising large datasets is no trivial task. Some attempts to visualize data have been through literally visualising every single data point by using non-standard visual attributes and interaction techniques[12]. Others have pursued pixel-based and dense pixel visualizations (like recursive pattern techniques, Hilbert curves, z-order curves etc.) [21], [22], [34]. However, when the visualization needs to use more than a single pixel point, data abstraction methods such as sampling and clustering are required to accurately summarise and reflect the data [8]. Shneiderman acknowledges as much when discussing dense pixel visualizations by stating that "meaningful aggregate visualizations show the greatest promise because they promote sense-making while keeping display complexity low".

Once it has been decided what metrics to use in the analysis of raw data, a way to visualize the results to the user needs to be found. One such way could be to visualize the raw data to see error patterns [19]. Previous research in data wrangling [13], [20], [31] has resulted in sophisticated tools that make interactive system for making data transformation possible. However, for generating a summary visualisation of the health of a dataset, the spreadsheet-like interface that these tools rely on are not sufficient. However, presenting the data in a tabular format is appropriate for visualizing an overview or a summary of the data. Using pixel-oriented visualization techniques [1], [21] could be the ideal solution for such a scenario.

Earlier in the section possible data quality dimensions to visualize were discussed. However, while uncertainty in the data is certainly visualized it is to be done in a simplistic manner. Uncertainty visualization techniques like Prediction, using glyphs etc.[11], [15], [36] are beyond the scope of the program due to the nature of the data: records and booking information of persons.

Once the visualization is generated after a sometimes time-consuming process of data cleaning and quality analysis, it is often not possible to replicate that image or visualization at a later time [6]. This is something that needs to be implemented for a successful data quality visualization tool, so that iterative improvements in the data can be compared by improvements in the visualization output too.

Tableau [24], an interactive data visualization software requires a special mention here. Its graphical history interface shown in Figure 3.3 visualizes states and labels that describes the history of actions performed within the program. This interface is a good representative of summary visualization.

Figure 3.3: Tableau's Graphical History Interface screen is a good representative of summary visualizations.

Depending on the size of the datasets expected to be parsed by the program, appropriate aggregate measures for visualizing the errors need to be considered. For instance, the sampling method can be used to visualize the errors in 20,000 rows when only a small space within a 640x420 pixel window is used for the report card window. The benefit of using such an aggregate measure is that the look and fool of the error visualization will remain consistent (otherwise a dataset of 100 rows would look thin and a 10,000 row dataset would be visualized using 10,000 lines, resulting in over-plotting and the user unable to decipher any meaningful information).

3.4 Summary

This chapter dealt with the previous research undertaken in the areas of data quality checking and methods of visualizing them. The approach used in this thesis has been built on the findings here, using empirical evidence from the client to measure data quality. The types of errors to check have been identified as syntactic, semantic and coverage errors, and these need to be identified separately. Finally, the identified errors are presented to the user in the form of a summarized report card, using aggregate measures where needed to show errors for medium to large datasets.

Chapter 4

Checking Data Quality

4.1 Overview

Based on the plethora of influential sources uncovered in a scan of the literature review, it was decided to design and implement a client-centric data quality analysis. Existing data characteristics are examined in detail and only then could a sufficiently robust error metric be generated. The error metric generation should utilise the client's current architecture when implementing the solution so that the user does not have any additional dependencies for running the program.

4.2 Design

This section will deal with the design process behind the data quality aspects of the thesis. The subsequent chapter on Visual Depiction builds on the design and implementation of the data quality aspects, hence it was paramount to undertake this section in distinction to that aspect of the thesis. Data Quality metrics generate statistics that highlight the characteristics of the data. Before this can happen, the data source needs to be identified and connected to so that data retrieval is possible. The desired error metrics and statistics are then chosen. It is only after the data is retrieved that it can be parsed and checked based on the metrics chosen in the previous steps. These steps can be listed as follows:

1. Identifying existing data characteristics
2. Designing error metrics and statistics
3. Parsing and checking data based on error metrics

Due to the bespoke nature of the project, the design stage was held in close collaboration with the client to ensure their prerequisites were listed and met as part of the implementation.

4.2.1 Identifying existing data characteristics

Essential data characteristics like data source, data storage type, data volume etc. were obtained so that the succeeding steps could utilise those in their design. The client revealed that currently they store their data in a MySQL table. MySQL is the most popular open-source RDBMS (relational database management system) in the world, and is supported by a number of programming languages as a database layer.

Once MySQL was installed for testing purposes, the sample dataset was imported into a test database for development purposes. The cross-platform portability of MySQL means that applications developed using MySQL as a backbone one one OS can be easily run on another, once the MySQL setup (including table names, username and password) are adequately catered for.

The client expects the datasets to contain between 5000-30000 rows. A small subset of an example dataset is reproduced here in Figure 4.1 to examine it further.

The primary attributes in the dataset that this project deals with are the ProviderID, FileType and RawData attributes. ProviderID holds the ID of the agency to whom this raw data in the RawData attribute belongs. FileType denotes the type of the data being stored. By iterating through all the rows of the dataset with a specific ProviderID, we can identify the data that uniquely belongs to the given Provider ID. This needs to be extracted otherwise the

ID	ProviderID	FileName	FileType	RawData
215741	17	data/Admin/Archive/17/ATC_airfareBookings_20140101.csv	AIRTRIP	actualFare,branchID,conjTickNum,countryCode,currencyCode,BusLei,eTkt,FlightNo,hotelTag,IATA,invDate,LowestLogicalFare,netInd,numSegs,OandD,Origin,Destin,pCarrier,pCarrierNum,resDate,segArriveDate,segCarrier,segClass,segDepartDate,segFare,segFareBasis,segFlightNum,surcharges,surcharges,taxes,tickNum,tourCode,tripClass,contracted,ancil,Exchange P/R,Exchange A/C,AirTrip/Refund,clientIdentifier
215742	17	data/Admin/Archive/17/ATC_airfareBookings_20140101.csv	AIRTRIP	30,6,,GB,GBP,not set,Y,1442,645488,91278040,01/01/2014,0,not set,1,not set,LHR,EDI,BA,125,01/01/2014,02/01/2014,BA,K,02/01/2014,30,,1442,,,0,4631677295,not set,Y,not set,not set,not set,not set,AirTrip,10001092
215743	17	data/Admin/Archive/17/ATC_airfareBookings_20140101.csv	AIRTRIP	698.2,6,,GB,GBP,not set,Y,5109,645488,91278040,01/01/2014,733.11,not set,1,not set,DXB,LHR,EK,176,01/01/2014,02/01/2014,EK,E,02/01/2014,698.2,,5109,,,13.2,4631677296,not set,Y,not set,not set,not set,not set,AirTrip,10001092

Figure 4.1: First three rows of the sample dataset sent by the client for development purposes.

implementation will deal with data from a variety of agencies at the same time, thus producing incorrect results.

According to the client, existing data belongs to one of four dataTypes (alternatively referred to as blockTypes). AIRTRIP is the blockType shown in the table sample. Other possible blockTypes include AIRREFUND (used to store ticket refund data), CAR(for agencies that deal with Car rental, the associated data is stored using this blockType) and HOTEL (for agencies that deal with Hotel room bookings etc., the data from such transactions is stored in this blockType). AIRTRIP is, however, the blockType that they encounter the most. Majority of their providers only provide AIRTRIP data and the other blockTypes are works in progress and need to be implemented in future versions. Due to these reasons AIRTRIP is the one blockType that will be tested extensively and considered when checking the software. The other blockTypes will still be catered for in the program.

The RawData attribute is a field containing comma separated values denoting each of the columns in the data being represented. Missing columns are skipped either using “not set” as a value or leaving a blank value between commas. Furthermore, the client has identified important attributes that are present in the RawType column. The attributes that are present in the RawType column are dependant on the blockType. Thus, a list of attributes

present in each blockType and their characteristcs have been provided to make parsing and classification easier.

4.2.2 Designing error metrics and statistics

To automate the errors/statistics generation process, the implementation should connect to the data source using a MySQL connector, parse the values contained in the RawData table and then check the parsed values for any errors. Using the information provided by the client on the important attributes that each blockType must identify, the user can select what attributes have been sent by each agency. This information is then stored as a mapping, so in the future this particular agency's attribute choices do not have to be manually entered.

The types of statistics that need to be generated for each dataset by the software are: statistics about the size of the dataset and about the health of the dataset. The size of the dataset are simply the number of rows in the raw data that are relevant to the selected ProviderID and blockType (fileType) combination. The health of the dataset is a more complex topic, which will be explored in detail as it is a concept devised to compare the quality of one dataset against another. The quality of the contents of each "cell" in the data row and the final quality of each row is assessed. Finally, an assessment of the overall quality of the dataset can be made using these statistics.

These error metrics and statistics are saved in a specific table in the MySQL database. This is to make the program extensible. The client requested that the error statistics that were generated not be restricted to the Visualization created by the implementation. By storing it in the database, the client can make their own visualization based on these error metrics and thus extend the functionality of the program in the future.

As the Related Work chapter points out, considerable amount of work has been done in the area of data quality and appropriating the health of a dataset

is done through implementing as many of the data quality metrics as possible to get a thorough picture of the quality of a dataset. However, due to the nature of the project, with data being in a unparsed state to begin with, and the unique requirements of the client, it was considered more appropriate to empirically determine the most useful data quality metrics. Operations on data, especially at the cell level are expensive when iterating through every single cell in a dataset, therefore it is best practice to identify the key error metrics to generate and work at optimizing them as much as possible.

Based on discussions with the client, and working towards the aim of generating a summary visualization, a handful of key error metrics were chosen that were most pertinent to the client. Using these metrics, methods were devised to generate a heuristic, which could then be visualized in the visual depiction side of the program. The key types of errors that need to be checked for are:

1. Syntactic errors
2. Semantic errors
3. Coverage errors

Syntactic errors, as the name suggests, is to do primarily with errors at the syntax level. These are checks to see if the data entered in a cell matches the description of the cell. They answer questions like “Is the data in the right format?”, “Is the entry missing or null?”.

Semantics is the study of meaning. Semantic errors then check the meaning of a row and check if it makes sense. For instance, if a row contains duplicate information to a previous row in a primary key column, then that row is inconsistent with the rules of the table.

Coverage errors are statistics generated on special cases. These are not universal rules but rules that apply to particular datasets, and are provided as a way to make the error checker extensible, readily for future extensions. An instance of a coverage error could be the number of unique ticket numbers in a table compared to the number of rows in the table. Not all tables will

Check Name	Check Type	Description
Missing value	Syntactic error	error if cell value is missing or null
Format	Syntactic error	error if format of the cell does not match the specified attribute format
Length	Syntactic error	error if length of the cell exceeds or is below the specified size
Integrity	Syntactic error	error if cell value is above or below the allowable range of values
Duplicate	Semantic error	error if the column(s) are repeated on multiple rows
Contradictions	Semantic error	error if cell value is lesser than the value of the specified column's value
Semi-empty	Semantic error	error if more than 50% of the columns in the row are empty
Unique Tick. Num	Coverage error	calculates number of unique ticket numbers as a ratio of the number of rows in the dataset (applicable to AIRTRIP blockType only)
Total Sum	Coverage error	calculates the total sum of all values for this column
Segment Excess	Coverage error	error if a particular ticket is repeated more than its allowable max entries (applicable to AIRTRIP blockType only)

Table 4.1: Each error checked in the program are listed in the table. Errors are either at the syntactic, semantic or coverage level.

have a unique identifier like a ticket number, so this error metric will not be appropriate for such tables.

The errors that have been identified as most pertinent for the client's datasets are shown below in Table 4.1

4.2.3 Parsing and checking data based on error metrics

This step involves parsing the string contained in the RawData attribute of the dataset and extracting the individual columns contained therein. Based on the blockType of the row, the necessary column headers and their presence (or lack of) in the string is saved as a mapping, which can be loaded later when the same provider uses the interface. As described above, since the string is a comma separated value, the parser can parse each of the columns present inside the string and then present the parsed data to the

error checking component of the implementation. The error checking stage entails iterating through all the columns of each raw in the dataset and finding if pre-configured data violations are flouted. If errors are presented, they are logged in the MySQL table. The visualization component of the program will use these generated logs to build the visual depiction.

4.2.4 Architecture considerations

In the three key stages of data handling in the workflow above, the choice of programming language and which architecture to implement was not discussed to allow for consideration of the options available. The storage of existing data is in an MySQL database, and MySQL is heavily integrated into the client's production environment. However despite this project dealing mainly with a data analysis, and not munging or wrangling data, one could argue that the project is not confined to the MySQL RDBMS storage paradigm. This is not the case, as the client requested that extensibility be taken into consideration. The client not only wanted a usable data analysis and visualization tool, which this project aimed to deliver, but hoped to utilise the error metrics that were generated as part of this process by the tool.

In consultation with the client, it was found that it would be in the best interest for both parties to continue the use of MySQL as a storage structure for the program development so that the said error metrics could be retrieved by the client as needed. Other alternatives like storing data to disk or using data structures like JSON and XML were thus not considered. Switching to an alternative system would have introduced unnecessary processes into the workflow for the client.

A web visualisation interface was considered and a prototype was built using HTML5, CSS3 and JavaScript. However, as expected, the speed of the interface was considerably slow, especially when parsing datasets with more than a few thousand rows. A web visualization would have been ideal, but it wasn't

a requirement for the client. The client was equally happy with a desktop based program.

Java was selected as an ideal programming language because of its easy connectivity with MySQL databases using the JDBC driver. A strong reason for choosing Java was the presence of the Processing.org graphical design API which is a powerful drawing software and very suitable for the task of rendering the data quality visualization (which is described in detail in the next chapter). The author and the client had previous experience using the Processing.org API so it was a straightforward first choice. Alternatives abound, for instance Java has the built in Graphics2D drawing tool, but compared to Processing.org it is limited.

4.2.5 Other design considerations

Miscellaneous considerations that were listed as desirable outcomes or envisioned as part of program design are as follows:

- **Program execution speed**: the client stated that while data processing was a slow task, they would desire a program that could analyse 1000 rows in 1-2 seconds. It was agreed that it was a reasonable time-frame to aim for.
- **Budget**: there will be no costs incurred as part of development.
- **Regular development input**: it was desirable for both the author and the client to have regular meetings to ensure program development was in accordance with stated aims. This was beneficial to the development since an Agile methodology was pursued and regular meetings provided the ideal avenue for showing successive prototypes.
- **Computing resources**: as the program will be run from the client's powerful desktop computer, memory wasn't a consideration. Initially when a web based visualization was proposed, internet bandwidth was one of the considerations but this was scrapped when an offline desktop program was pursued.

4.3 Implementation

Once the product development workflow had been designed and ratified by the client, the chosen architecture was implemented using Java as the programming language of choice, MySQL databases as the data storage structure and Processing.org was used as the visualization language (this will be dealt with in the Visual Depiction chapter).

Mac OSX, the development machine of choice, doesn't ship with its own copy of MySQL. mysql stable version 5.6.25 was installed on the computer using the Homebrew package manager. Java Development Kit (JDK) version 1.8.0.25 was already configured on the computer, with Eclipse Luna being used as an IDE for coding.

4.3.1 Attempt at web-based visualization

During the planning stages of the thesis, before interacting with the client, a web based visualization was considered as a novel way to present the health of a dataset. Due to the author's previous experience working in Javascript and web development languages, a few prototypes were developed.

Using HTML5, CSS, and Javascript (and JS associated libraries like jQuery and FontAwesome), a website was built to emulate the look and feel of the report card's syntactic grid feature. Javascript has numerous canvas drawing libraries, of which Raphael.js was chosen for its ease of use.

Figure 4.2 shows an early visualization website using just basic web technologies to render a HTML table containing raw data (CSV file) that was imported. Each cell's contents are checked and coloured based on the primitive type and whether it was a missing value.

However, this was felt to be a very primitive interface and therefore the prototype shown in Figure 4.3 was developed. This incorporated

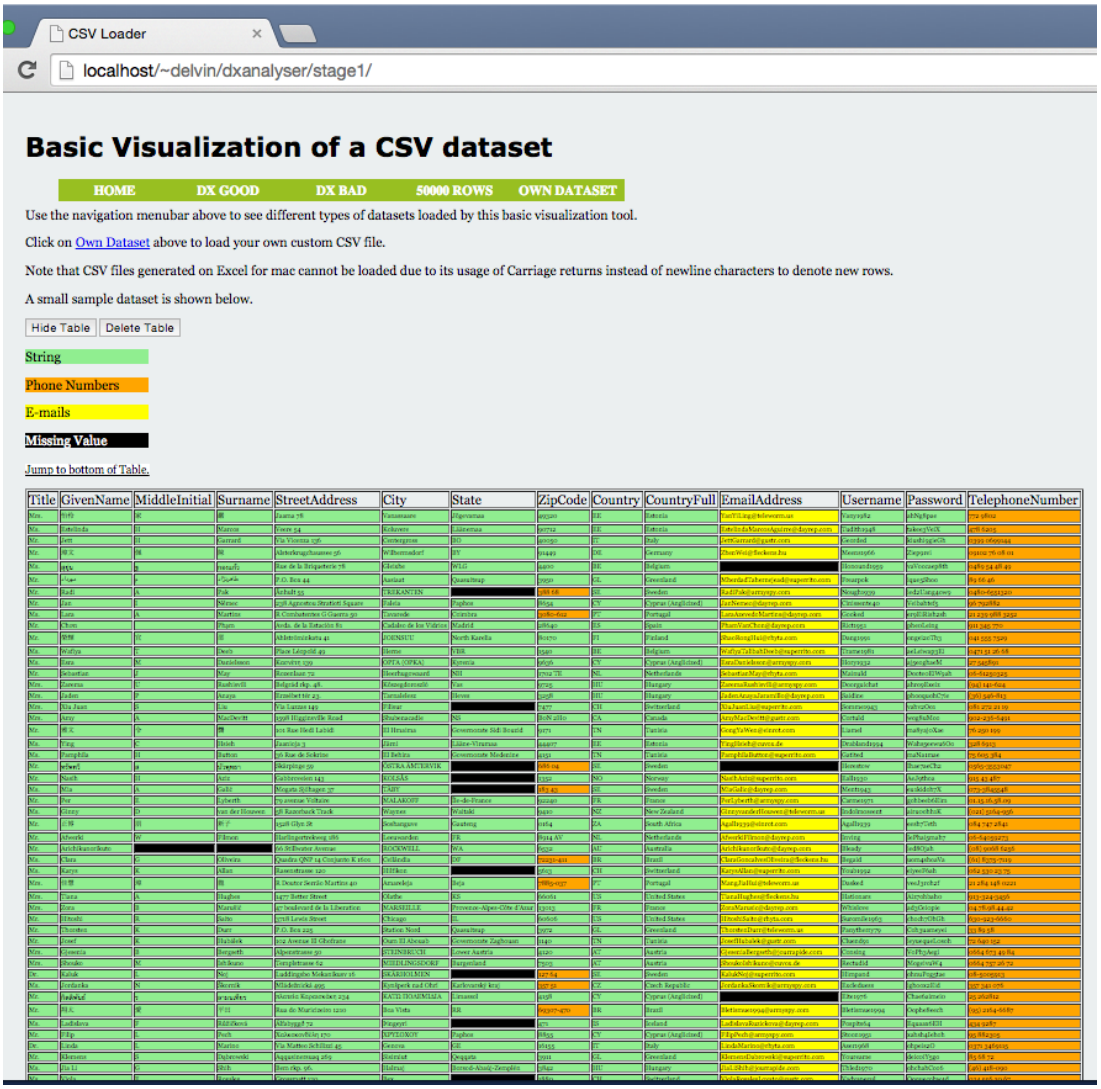

Figure 4.2: Web visualization Prototype 1, developed at a very early stage in development process. This was implemented using a simple HTML5 table + Javascript coding abilities.

the aforementioned Raphael.js drawing library to render a grid based visualization, the cells of the grid being coloured according to the cell's contents.

Working with web-based visualizations for even a simple error check like the ones done here made it apparent that efficiency would be a major issue. The second prototype took 8 seconds to read a 10,000 row dataset and do a couple of simple error checks on them and then rendering the cells on screen. If, as the project's plan has always been, a number of non-trivial error checks are done on the dataset contents, a considerable amount of time would be taken. This would not be ideal for the client, working in a business environment, as time is a major factor in data analysis.

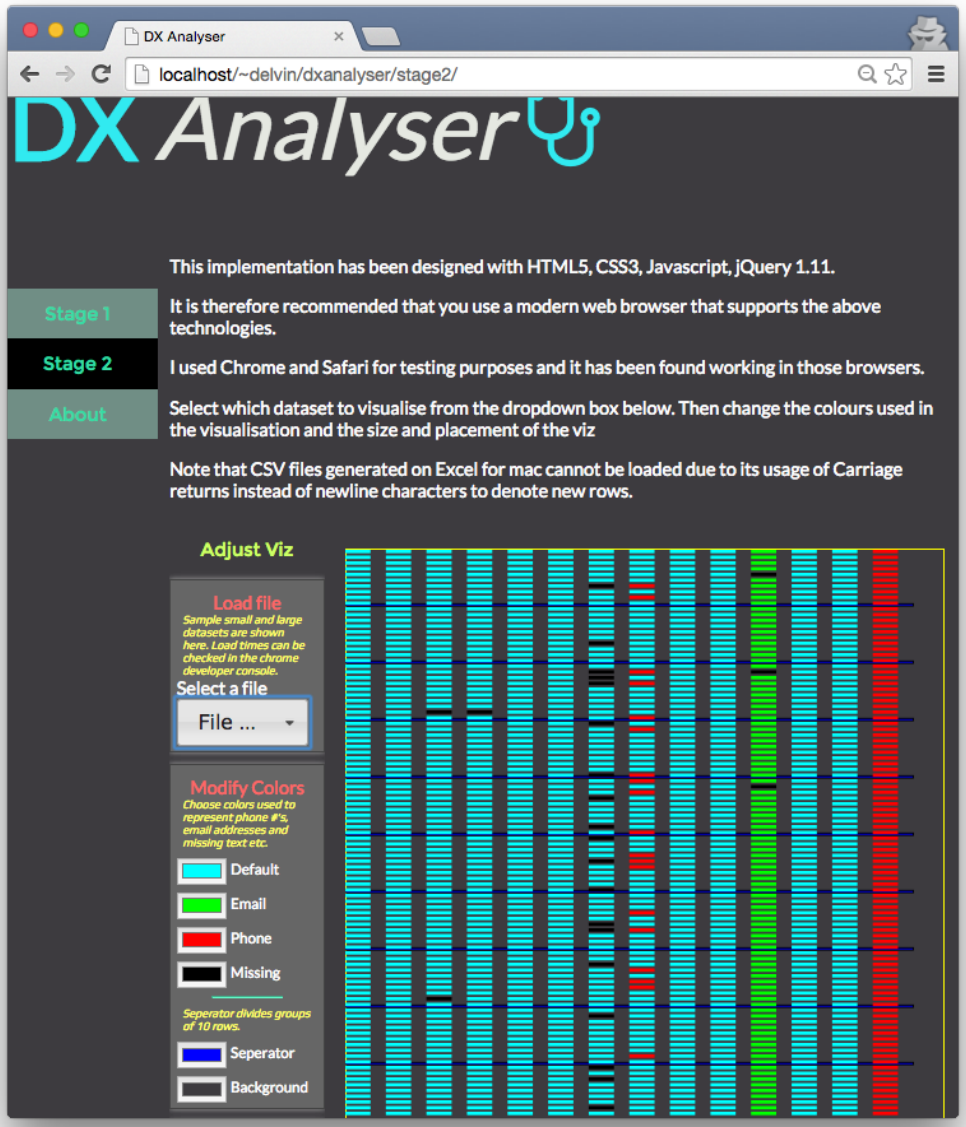

Figure 4.3: Web visualization Prototype 2 utilising jQuery and Raphael canvas libraries for a more sophisticated visualization. The problem of program execution speed was encountered when using canvas to render large datasets.

Therefore it was decided to work on a desktop based environment, where the browser speed was not a bottleneck.

4.3.2 Initial dataset import

Since the whole project revolves around the dataset provided as the source, the dataset was analysed to identify the best possible way to interpret it's contents. A software called Sequel Pro (available on Mac OSX) was invaluable

in providing easy graphical access to the MySQL tables required for this project. The client's initial sample dataset which was briefly mentioned in the previous section was provided as a Comma Separated Value (CSV) file. A table was created in MySQL to emulate the database structure on the client's MySQL database. The structure of the table is shown below in SQL notation:

```
CREATE TABLE ‘newrawdata‘ (
  ‘ID‘ int(11) NOT NULL,
  ‘ProviderID‘ tinyint(3) DEFAULT NULL,
  ‘FileName‘ varchar(100) DEFAULT NULL,
  ‘FileType‘ varchar(10) DEFAULT NULL,
  ‘RawData‘ text,
  PRIMARY KEY (‘ID‘),
  UNIQUE KEY ‘ID_UNIQUE‘ (‘ID‘)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

A tabular description of the table is given in Table 4.2.

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	-
ProviderID	tinyint(3)	YES	-	NULL	-
FileName	varchar(100)	YES	-	NULL	-
FileType	varchar(10)	YES	-	NULL	-
RawData	text	YES	-	NULL	-

Table 4.2: Table description of *newrawdata*

Once the table has been created, a SQL script was used to import the *wings.csv* dataset into the table. The *LOAD DATA INFILE* command is provided in MySQL for such a scenario, where the delimiters used in the dataset and the presence of speech marks can also be set as an option to enable optimal parsing.

```
LOAD DATA INFILE "data/tables/wingsNew.csv" INTO TABLE newrawdata
COLUMNS TERMINATED BY ’,’OPTIONALLY ENCLOSED BY ’"’ESCAPED BY ’"’
LINES TERMINATED BY ’\n’ IGNORE 1 LINES;
```

Once the data has been successfully loaded, thus emulating the dataset as it would be present in the client's environment, this can be confirmed by using the MySQL command line interface and submitting the query to see if the data is there.

```
SELECT * FROM newrawdata;
```

The output (shown in Figure 4.4) confirms the data's presence.

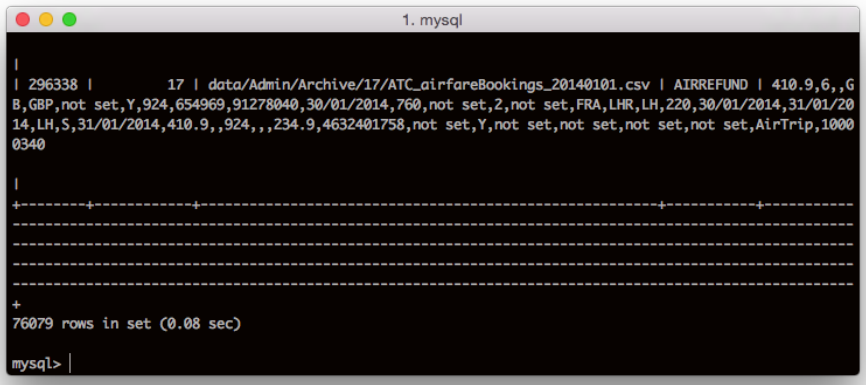

Figure 4.4: Querying *newrawdata* using the *mysql* command line interface to check whether data import was successful. The presence of rows here confirms this.

4.3.3 Description of important attributes

The importance of using blockTypes to filter the data extracted from the dataset was briefly alluded to in an earlier chapter. Each blockType is a different category of data that is provided by the agencies that the client works with. Therefore to facilitate efficient error checking, each blockType's key attributes were described and subsequently stored in a MySQL table, so that the characteristics of each attribute of each blockType could be retrieved on the fly. The attributes names and characteristics were provided by the client. They were imported into a table called *attributeDescriptor*. The create table syntax is shown below:

```
CREATE TABLE ‘attributeDescriptor‘ (
```

```
  `attributeID` int(4) unsigned NOT NULL AUTO_INCREMENT,
  `attributeName` varchar(20) DEFAULT NULL,
  `blockType` varchar(15) DEFAULT NULL,
  `size` int(3) DEFAULT NULL,
  `format` varchar(10) DEFAULT NULL,
  `missingCheck` tinyint(1) NOT NULL DEFAULT '1',
  `minSize` int(1) unsigned zerofill NOT NULL DEFAULT '0',
  `minVal` varchar(11) NOT NULL DEFAULT 'NA',
  `maxVal` varchar(11) NOT NULL DEFAULT 'NA',
  `primaryKey` tinyint(1) DEFAULT '0',
  `contradictionCheck` varchar(15) DEFAULT 'NA',
  `calculateSum` tinyint(1) DEFAULT '0',
  PRIMARY KEY (`attributeID`)
) ENGINE=InnoDB AUTO_INCREMENT=82 DEFAULT CHARSET=utf8;
```

The columns in the table along with a brief description of them is shown in Table 4.3. Note that not all the error checks are done through the *attributeDescriptor* table, since it is much more efficient to do them programmatically during data parsing than through a redundant check along with the other error checks. This includes Unique Ticket num. and the Segment Excess checks.

Once the structure has been defined, the contents of the *attributeDesriptor* table can be explained. This is shown below in Table 4.4

Explanatory notes

attributeDescriptor (Table 4.4) is the MySQL table that governs the attributes used in the parsing of raw data. The advantage of using a centralised structure such as this is that in the future changes made to the attributes can be made from a single interface. The spread of attribute types that are covered by this table is quite large. There are four different blockTypes and their attributes described here. Another aspect is that not all data attributes have all their

Field	Associated check	Description
attributeID	-	an auto-incremented integer ID
attribute Name	-	name of the attribute
blockType	-	which blockType this attribute belongs to
size	Length	the maximum allowed size for the contents of this attribute
format	Format	describes the type, i.e. String, Integer, Date, Currency etc. so it can be parsed accordingly
missing Check	Missing value	if this value is missing or not
minSize	Length	the minimum acceptable size of attribute contents
minVal	Integrity	the minimum in the range of values permitted. For instance, integers might begin at 0
maxVal	Integrity	the maximum in the range of values permitted.
primaryKey	Duplicate	whether to do a duplicate check on contents of this attribute
contradiction Check	Contradictions	the column against which to compare the values of this column
calculate Sum	Total Sum	boolean to check if a sum should be generated from all values in this attribute

Table 4.3: Description of *attributeDescriptor* table. This table contains the attributes used in the parsing of raw data, and is a centralised place for managing rules for parsing and error checking.

attribute ID	attributeName	blockType	size	format	missing Check	min Size	minVal	maxVal	primary Key	contradiction Check	calculate Sum
1	actualFare	AIRTRIP	9	Currency	1	0	5	NA	0	NA	1
2	hotelTag	AIRTRIP	10	Alpha	1	0	NA	NA	0	NA	0
3	IATA	AIRTRIP	8	Integer	1	7	NA	NA	0	NA	0
4	invDate	AIRTRIP	12	Date	1	0	01/01/14	NA	0	segDepart Date	0
5	numSegs	AIRTRIP	2	Integer	1	0	1	8	0	NA	0
6	Origin	AIRTRIP	3	String	1	0	NA	NA	1	NA	0
7	Destin	AIRTRIP	3	String	1	0	NA	NA	1	NA	0
8	pCarrier	AIRTRIP	3	Alpha	1	0	NA	NA	0	NA	0
9	pCarrierNum	AIRTRIP	3	Integer	1	0	NA	NA	0	NA	0
10	segArriveDate	AIRTRIP	12	Date	1	0	01/01/14	NA	0	NA	0
11	segCarrier	AIRTRIP	3	Alpha	1	0	NA	NA	0	NA	0
12	segClass	AIRTRIP	1	String	1	0	NA	NA	0	NA	0
13	segDepartDate	AIRTRIP	12	Date	1	0	01/01/14	NA	0	segArrive Date	0
14	segFare	AIRTRIP	9	Currency	1	0	NA	NA	0	actualFare	1
15	segFareBasis	AIRTRIP	10	Alpha	1	0	NA	NA	0	NA	0
16	segFlightNum	AIRTRIP	4	Alpha	1	0	NA	NA	0	NA	0
17	tickNum	AIRTRIP	10	Integer	1	0	NA	NA	1	NA	0
18	tourCode	AIRTRIP	9	Alpha	1	0	NA	NA	0	NA	0
19	tripClass	AIRTRIP	1	String	1	0	NA	NA	0	NA	0
20	currencyCode	AIRTRIP	3	String	1	0	NA	NA	0	NA	0
21	actualFare	AIRREFUND	9	Currency	1	0	5	NA	0	NA	1
22	branchID	AIRREFUND	10	Alpha	1	0	NA	NA	0	NA	0
23	countryCode	AIRREFUND	3	String	1	0	NA	NA	0	NA	0
24	currencyCode	AIRREFUND	3	String	1	0	NA	NA	0	NA	0
25	hotelTag	AIRREFUND	10	Alpha	1	0	NA	NA	0	NA	0
26	IATA	AIRREFUND	8	Integer	1	7	NA	NA	0	NA	0
27	numSegs	AIRREFUND	2	Integer	1	0	1	8	0	NA	0
28	OandD	AIRREFUND	2	String	1	0	NA	NA	0	NA	0
29	Origin	AIRREFUND	3	String	1	0	NA	NA	1	NA	0
30	Destin	AIRREFUND	3	String	1	0	NA	NA	1	NA	0
31	pCarrier	AIRREFUND	3	String	1	0	NA	NA	0	NA	0
32	pCarrierNum	AIRREFUND	3	Integer	1	0	NA	NA	0	NA	0
33	resDate	AIRREFUND	10	Date	1	0	01/01/14	NA	0	NA	0
34	segArriveDate	AIRREFUND	12	Date	1	0	01/01/14	NA	0	NA	0
35	segCarrier	AIRREFUND	3	String	1	0	NA	NA	0	NA	0
36	segClass	AIRREFUND	1	String	1	0	NA	NA	0	NA	0
37	segDepartDate	AIRREFUND	12	Date	1	0	01/01/14	NA	0	NA	0
38	segFare	AIRREFUND	9	Currency	1	0	NA	NA	0	NA	1
39	segFareBasis	AIRREFUND	10	String	1	0	NA	NA	0	NA	0
40	segFlightNum	AIRREFUND	4	Integer	1	0	NA	NA	0	NA	0
41	surcharges	AIRREFUND	9	Currency	1	0	NA	NA	0	NA	1
42	surcharges	AIRREFUND	9	Currency	1	0	NA	NA	0	NA	1
43	taxes	AIRREFUND	9	Currency	1	0	NA	NA	0	NA	1
44	tourCode	AIRREFUND	3	String	1	0	NA	NA	0	NA	0
45	tripClass	AIRREFUND	3	String	1	0	NA	NA	0	NA	0
46	contracted	AIRREFUND	3	Alpha	0	0	NA	NA	0	NA	0
47	ancil	AIRREFUND	9	Currency	1	0	NA	NA	0	NA	1
48	actualFare	CAR	9	Currency	1	0	5	NA	0	NA	1
49	airportCode	CAR	3	String	1	0	NA	NA	0	NA	0
50	branchID	CAR	10	Alpha	1	0	NA	NA	0	NA	0
51	chainCode	CAR	10	Alpha	1	0	NA	NA	0	NA	0
52	city	CAR	10	String	1	0	NA	NA	0	NA	0
53	citycode	CAR	10	String	1	0	NA	NA	0	NA	0
54	countryCode	CAR	3	String	1	0	NA	NA	0	NA	0
55	currencyCode	CAR	3	String	1	0	NA	NA	0	NA	0
56	IATA	CAR	8	Integer	1	7	NA	NA	0	NA	0
57	invDate	CAR	10	Date	1	0	01/01/14	NA	0	NA	0
58	durationDays	CAR	2	Integer	1	0	NA	NA	0	NA	0
59	numDays	CAR	2	Integer	1	0	NA	NA	0	NA	0
60	rentDate	CAR	10	Date	1	0	01/01/14	NA	0	NA	0
61	resDate	CAR	10	Date	1	0	01/01/14	NA	0	NA	0
62	rtnDate	CAR	10	Date	1	0	01/01/14	NA	0	NA	0
63	typeCode	CAR	9	String	0	0	NA	NA	0	NA	0
64	zipCode	CAR	10	String	1	0	NA	NA	0	NA	0
65	HotelName	HOTEL	15	String	1	0	NA	NA	0	NA	0
66	actualFare	HOTEL	10	Currency	1	0	5	NA	0	NA	1
67	chainCode	HOTEL	10	Alpha	1	0	NA	NA	0	NA	0
68	city	HOTEL	10	String	1	0	NA	NA	0	NA	0
69	citycode	HOTEL	10	String	1	0	NA	NA	0	NA	0
70	countryCode	HOTEL	3	String	1	0	NA	NA	0	NA	0
71	currencyCode	HOTEL	3	String	1	0	NA	NA	0	NA	0
72	hotelTag	HOTEL	10	Alpha	1	0	NA	NA	0	NA	0
73	IATA	HOTEL	8	Integer	1	7	NA	NA	0	NA	0
74	inDate	HOTEL	10	Date	1	0	01/01/14	NA	0	NA	0
75	invDate	HOTEL	10	Date	1	0	01/01/14	NA	0	NA	0
76	numDays	HOTEL	2	Integer	1	0	NA	NA	0	NA	0
77	numGuest	HOTEL	2	Integer	1	0	NA	NA	0	NA	0
78	outDate	HOTEL	10	Date	1	0	01/01/14	NA	0	NA	0
79	quantity	HOTEL	2	Integer	1	0	NA	NA	0	NA	0
80	typeCode	HOTEL	10	Alpha	0	0	NA	NA	0	NA	0
81	zipCode	HOTEL	10	Alpha	1	0	NA	NA	0	NA	0

Table 4.4: Important attributes in each blockType and their characteristics as present in Table *attributeDescriptor* are shown here.

corresponding characteristics. Where this is the case “NA” or “0” is used to denote that particular characteristic irrelevant.

Data attributes in different formats are also shown here as a quick glance at the **format** will reveal. There are certain underlying assumptions that need to be highlighted. For instance, the minimum value (minVal) and maximum value (maxVal) for date formats are dates, whereas for attributes with Integer format, the corresponding minimum and maximum values will be Integers. The predicament that is raised here is that it is quite easy to check if an integer is between two numbers by doing arithmetic comparison, but not so for Date formats. Therefore a custom method needs to be implemented or the Date functionality of the programming language needs to be utilised to make comparisons, something that is quite easy to do in Java with its built in Calendar functions.

Since MySQL does not support true or false as boolean values, 1 and 0 are used as their replacements. This is shown in the calculateSum and primaryKey columns for instance, where a “1” indicates that the particular check in question needs to be undertaken and a “0” implies that this check should not take place.

The contradictionCheck column is the only one of its type. The presence of a value apart from “NA” implies that the attribute represented by that row has a corresponding attribute to compare against for a contradiction. For example, **invDate** (attributeID) 4 in AIRTRIP has a contradictionCheck of **segDepartDate**. When the program reads this row, it will subsequently issue a check on each row in the original dataset to compare the values of invDate and segDepartDate. If invDate is after segDepartDate then a contradiction error has occurred. The value of the attributeName cell cannot be greater than the same row’s contradictionCheck attribute cell.

4.3.4 Saving agency mappings

For each blockType, a corresponding table was created in MySQL to save the parsing settings for each agency's datasets. The raw data consists of just one string which contains numerous sub-strings, each one representing a different attribute of the row. The client selects the substring which corresponds to each attributeName for tha blockType, and the program saves the index of that substring. This index will reflect the position of the attribute in all subsequent data from this agency for this particular blockType.

Since each blockType has its distinct attributes and characteristics, each blockType requires its own agency mappings table. Inside the table, the providerID of the current agency is stored along with a timestamp and other characteristics about the datasets that they provide. Next time the agency with this particular providerID sends a dataset and it is loaded, the program automatically detects the providerID present in the dataset and loads the list of mappings that share this providerID. The client can then choose an appropriate mapping which contains the right indexing for the attributes for this blockType.

MySQL Table for AIRTRIP agency mappings

```
CREATE TABLE ‘attributeMappings‘ (
  ‘ProviderID‘ int(5) NOT NULL,
  ‘savedTime‘ varchar(25) NOT NULL DEFAULT ’0’,
  ‘rowHeader‘ int(3) NOT NULL DEFAULT ’0’,
  ‘dateFormat‘ varchar(11) NOT NULL DEFAULT ’0’,
  ‘actualFare‘ varchar(10) DEFAULT ’0’,
  ‘currencyCode‘ varchar(11) DEFAULT ’0’,
  ‘hotelTag‘ varchar(10) DEFAULT ’0’,
  ‘IATA‘ varchar(10) DEFAULT ’0’,
  ‘invDate‘ varchar(10) DEFAULT ’0’,
  ‘numSegs‘ varchar(10) DEFAULT ’0’,
```

```
  ‘Origin‘ varchar(10) DEFAULT ’0’,
  ‘Destin‘ varchar(10) DEFAULT ’0’,
  ‘pCarrier‘ varchar(10) DEFAULT ’0’,
  ‘pCarrierNum‘ varchar(10) DEFAULT ’0’,
  ‘segArriveDate‘ varchar(10) DEFAULT ’0’,
  ‘segCarrier‘ varchar(10) DEFAULT ’0’,
  ‘segClass‘ varchar(10) DEFAULT ’0’,
  ‘segDepartDate‘ varchar(10) DEFAULT ’0’,
  ‘segFare‘ varchar(10) DEFAULT ’0’,
  ‘segFareBasis‘ varchar(10) DEFAULT ’0’,
  ‘segFlightNum‘ varchar(10) DEFAULT ’0’,
  ‘tickNum‘ varchar(10) DEFAULT ’0’,
  ‘tourCode‘ varchar(10) DEFAULT ’0’,
  ‘tripClass‘ varchar(10) DEFAULT ’0’,
  PRIMARY KEY (‘ProviderID‘,‘savedTime‘)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

MySQL Table for AIRREFUND agency mappings

```
CREATE TABLE ‘airrefundMappings‘ (
  ‘ProviderID‘ int(5) NOT NULL,
  ‘savedTime‘ varchar(25) NOT NULL DEFAULT ’0’,
  ‘rowHeader‘ int(3) NOT NULL DEFAULT ’0’,
  ‘dateFormat‘ varchar(10) NOT NULL DEFAULT ’0’,
  ‘actualFare‘ varchar(10) DEFAULT ’0’,
  ‘branchID‘ varchar(10) DEFAULT ’0’,
  ‘countryCode‘ varchar(10) DEFAULT ’0’,
  ‘currencyCode‘ varchar(10) DEFAULT ’0’,
  ‘hotelTag‘ varchar(10) DEFAULT ’0’,
  ‘IATA‘ varchar(10) DEFAULT ’0’,
  ‘numSegs‘ varchar(10) DEFAULT ’0’,
  ‘OandD‘ varchar(10) DEFAULT ’0’,
  ‘Origin‘ varchar(10) DEFAULT ’0’,
```

```
  ‘Destin‘ varchar(10) DEFAULT ’0’,
  ‘pCarrier‘ varchar(10) DEFAULT ’0’,
  ‘pCarrierNum‘ varchar(10) DEFAULT ’0’,
  ‘resDate‘ varchar(10) DEFAULT ’0’,
  ‘segArriveDate‘ varchar(10) DEFAULT ’0’,
  ‘segCarrier‘ varchar(10) DEFAULT ’0’,
  ‘segClass‘ varchar(10) DEFAULT ’0’,
  ‘segDepartDate‘ varchar(10) DEFAULT ’0’,
  ‘segFare‘ varchar(10) DEFAULT ’0’,
  ‘segFareBasis‘ varchar(10) DEFAULT ’0’,
  ‘segFlightNum‘ varchar(10) DEFAULT ’0’,
  ‘flightSurcharges‘ varchar(10) DEFAULT ’0’,
  ‘segSurcharges‘ varchar(10) DEFAULT ’0’,
  ‘taxes‘ varchar(10) DEFAULT ’0’,
  ‘tourCode‘ varchar(10) DEFAULT ’0’,
  ‘tripClass‘ varchar(10) DEFAULT ’0’,
  ‘contracted‘ varchar(10) DEFAULT ’0’,
  ‘ancil‘ varchar(10) DEFAULT ’0’,
  PRIMARY KEY (‘ProviderID‘,‘savedTime‘)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

MySQL Table for CAR agency mappings

```
CREATE TABLE ‘carMappings‘ (
  ‘ProviderID‘ int(5) NOT NULL,
  ‘savedTime‘ varchar(25) NOT NULL DEFAULT ’0’,
  ‘rowHeader‘ int(11) DEFAULT NULL,
  ‘dateFormat‘ varchar(11) DEFAULT NULL,
  ‘actualFare‘ varchar(10) DEFAULT ’0’,
  ‘airportCode‘ varchar(10) DEFAULT ’0’,
  ‘branchID‘ varchar(10) DEFAULT ’0’,
  ‘chainCode‘ varchar(10) DEFAULT ’0’,
  ‘city‘ varchar(10) DEFAULT ’0’,
```

```
  `citycode` varchar(10) DEFAULT '0',
  `countryCode` varchar(10) DEFAULT '0',
  `currencyCode` varchar(10) DEFAULT '0',
  `IATA` varchar(10) DEFAULT '0',
  `invDate` varchar(10) DEFAULT '0',
  `durationDays` varchar(10) DEFAULT '0',
  `numDays` varchar(10) DEFAULT '0',
  `rentDate` varchar(10) DEFAULT '0',
  `resDate` varchar(10) DEFAULT '0',
  `rtnDate` varchar(10) DEFAULT '0',
  `typeCode` varchar(10) DEFAULT '0',
  `zipCode` varchar(10) DEFAULT '0',
  PRIMARY KEY (`ProviderID`,`savedTime`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

MySQL Table for HOTEL agency mappings

```
CREATE TABLE `hotelMappings` (
  `ProviderID` int(5) NOT NULL,
  `savedTime` varchar(25) NOT NULL DEFAULT '0',
  `rowHeader` int(3) NOT NULL DEFAULT '0',
  `dateFormat` varchar(10) NOT NULL DEFAULT '0',
  `HotelName` varchar(10) DEFAULT '0',
  `actualFare` varchar(10) DEFAULT '0',
  `chainCode` varchar(10) DEFAULT '0',
  `city` varchar(10) DEFAULT '0',
  `citycode` varchar(10) DEFAULT '0',
  `countryCode` varchar(10) DEFAULT '0',
  `currencyCode` varchar(10) DEFAULT '0',
  `hotelTag` varchar(10) DEFAULT '0',
  `IATA` varchar(10) DEFAULT '0',
  `inDate` varchar(10) DEFAULT '0',
  `invDate` varchar(10) DEFAULT '0',
```

```
  ‘numDays‘ varchar(10) DEFAULT ’0’,
  ‘numGuest‘ varchar(10) DEFAULT ’0’,
  ‘outDate‘ varchar(10) DEFAULT ’0’,
  ‘quantity‘ varchar(10) DEFAULT ’0’,
  ‘typeCode‘ varchar(10) DEFAULT ’0’,
  ‘zipCode‘ varchar(10) DEFAULT ’0’,
  PRIMARY KEY (‘ProviderID‘,‘savedTime‘)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

4.3.5 User Interface creation

The user interface available to the client is where the error checking meets the visual depiction. The Graphical User Interface has been designed in such a way that the user is directed through the initial process of data source identification and parsing to the final step of generating and viewing generated visualisations in a logical and intuitive manner.

This section will deal mainly with those aspects of the GUI that pertain to data parsing and data quality analysis. The next chapter’s Implementation section will deal with the visualization components of the GUI.

WindowBuilder

For GUI design, an eclipse plugin called *WindowBuilder* was used. WindowBuilder is a powerful bi-directional Java GUI designer, and makes it easy to design Swing components using a WYSIWYG visual designer. This was essential to the development in many ways as it saves a lot of time that is otherwise spent writing code to draw components to the screen.

WindowBuilder is fast and the generated code does not require the plugin to be installed to run, and it can also reverse engineer hand-written GUI code.

All these features made it an ideal choice to use alongside Eclipse IDE to design the program's GUI.

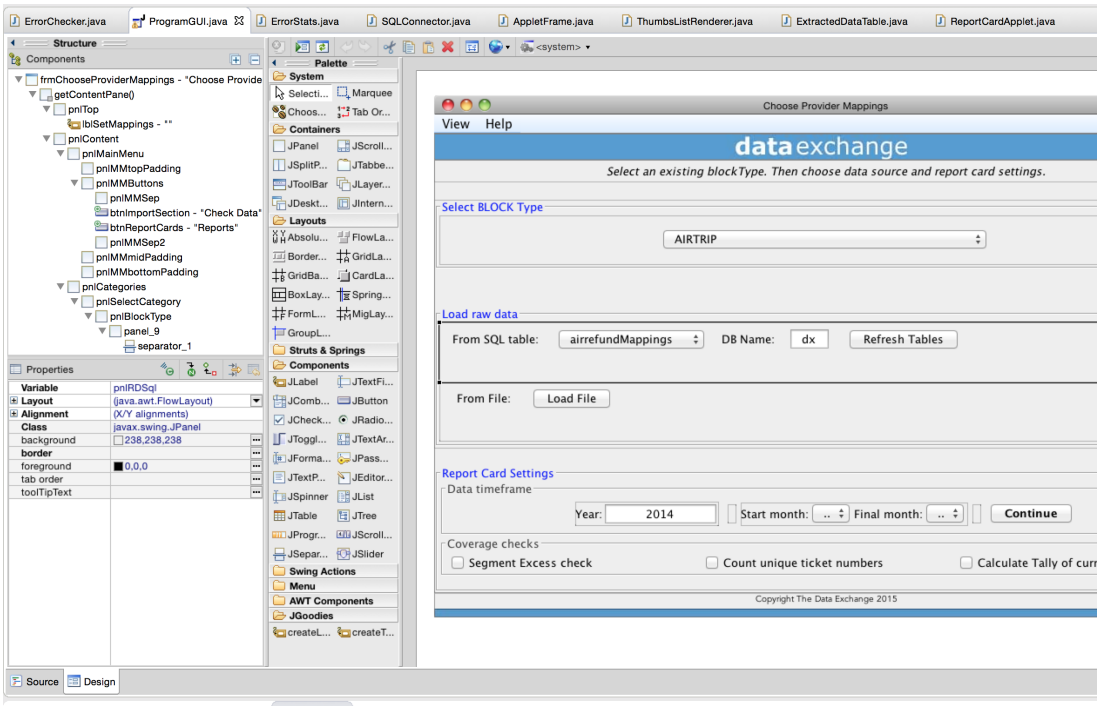

Figure 4.5: Using the eclipse WindowBuilder plugin to design the GUI. Drag and drop controls enable easy GUI creation.

Program start

When the program is started the user is presented with a simple screen where they can choose either to check a new dataset or load report cards from previous checks. In this section, we will deal with only the first option and related operations.

All program operations except for the generation of the report card occur within this single window, maintaining a unified interface. The client's logo presented in the banner and colour theme maintained in the footer helps give an appearance of familiarity through design since the header and footer remain in place regardless of the processes. The Menu bar at the top likewise remains at the top of every screen in the program window. The components that change take place happen in the panel that occupies the space between the header and the footer.

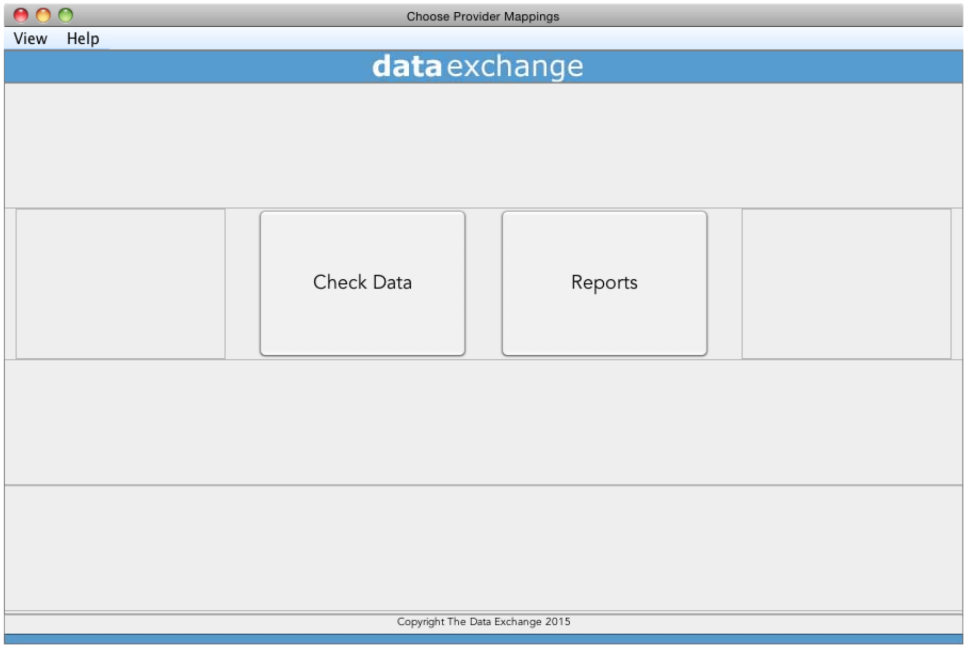

Figure 4.6: Start screen presented to the user when program is started.

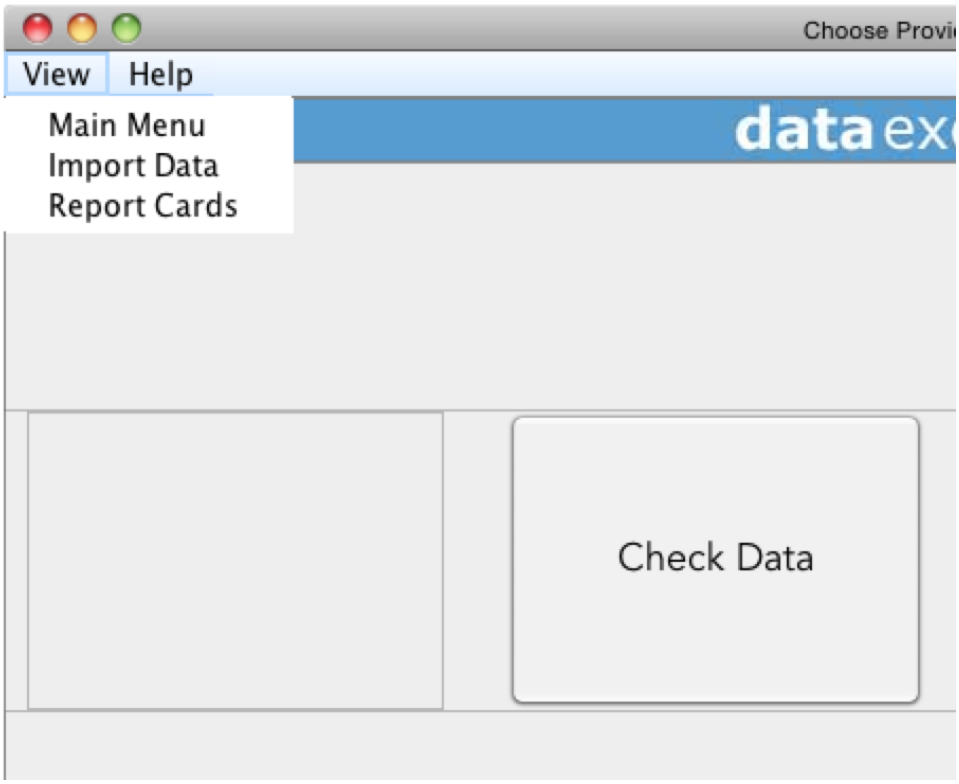

Figure 4.7: The View menu in the Menubar for easy navigation between program features

Setting preliminary settings

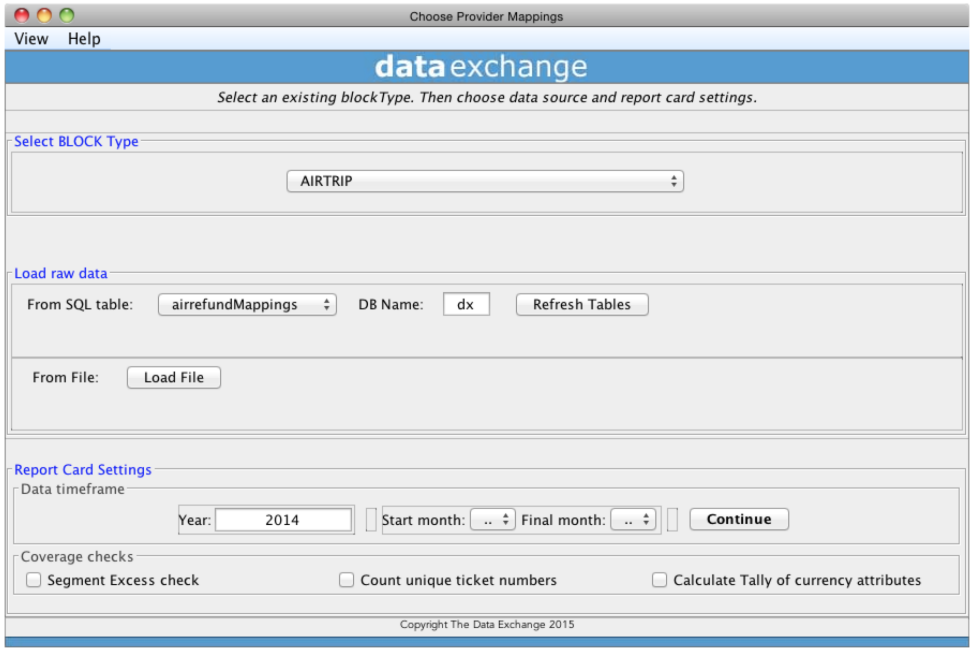

Figure 4.8: Screen presented when user clicks on the start screen's Check Data button

When the user clicks on the "Check Data" button, the GUI shows a series of settings that need to be set before data parsing can occur. Initially the user must select the blockType of the dataset that they wish to parse. Then, the data source needs to be selected, either from disk (CSV and Excel files are supported) or from one of the MySQL tables that are present on the user's database.

Finally, the user must select the time-frame of the data being parsed and whether special coverage checks need to be done on the data. These settings are send to the report card generator which decides the output of the visualization and storage of the report card based on these settings.

Saving and loading agency mappings

Once the preliminary settings are decided by the user and they Continue to the next screen, they are met with the mappings screen. The screen was built using the WindowBuilder designer with components being dragged from a palette onto the frame window. This is shown in Figure 4.10.

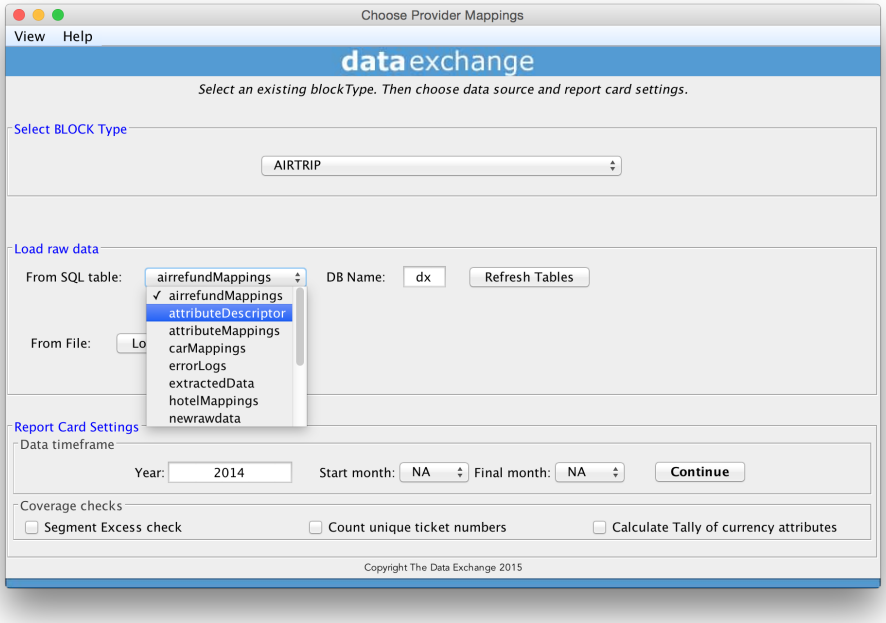

Figure 4.9: A list of tables currently present in the MySQL database are loaded into the combobox shown in the figure. This enables the user to select the source of the raw data.

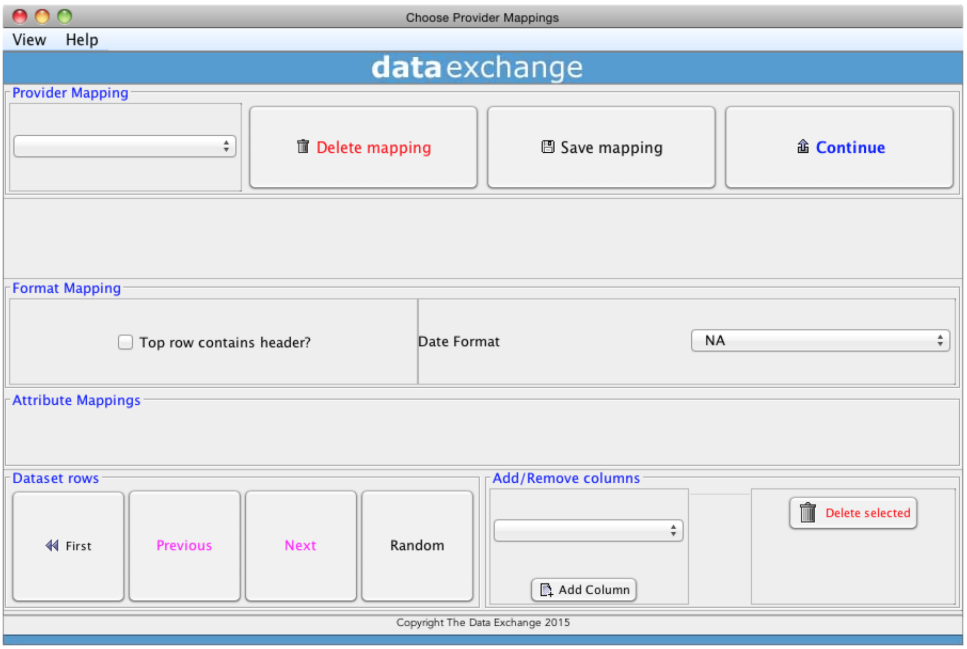

Figure 4.10: The Mappings screen as it appears in the WindowBuilder designer window is shown here. The *Attribute Mappings* panel is empty as it will be programmatically loaded based on the columns given in the current Provider Mapping.

The screen in Figure 4.11 is very important as this is where the user can select what important attributes of a blockType are present in the raw data. The attributes for the chosen blockType are added as Drop down boxes (referred to as a ComboBox in Java) to the Attribute Mappings panel. The raw data string is parsed into substrings which are then loaded into an array. This array is added as a ComboBoxModel to each of the drop down boxes. The client needs to select which substring from the raw data matches the attribute names of the blockType.

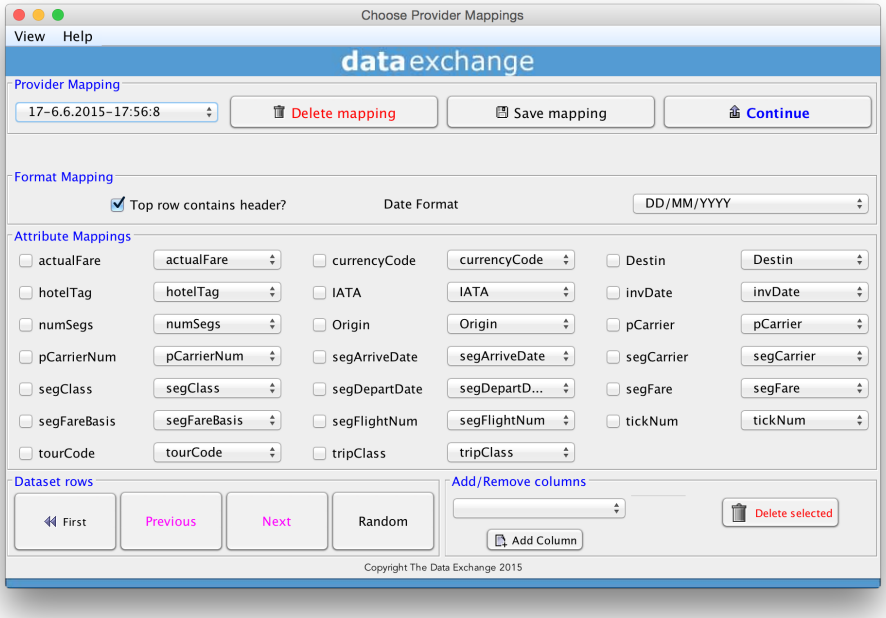

Figure 4.11: The Mappings screen after the AIRTRIP blockType has been selected in the previous screen.

Once the client selects the appropriate substring for each attribute satisfactorily, they can proceed. If a particular attribute has no appropriate entry in the raw data, then they can set the drop down list to “NA” indicating that this attribute is not to be found in this dataset. Another option that is provided is the ability to delete inappropriate attributes, for instance, if a particular agency’s datasets never contain a particular attribute.

If a blockType contains 20 attributes, then each of the attributes are rendered in the panel as drop down list. There are checkboxes next to the attributes which can be checked, and the “Delete Selected” button can be pressed in the

bottom-right corner of the screen, which will delete the attributes from the panel. Suppose in the future this particular agency starts sending datasets with this deleted attribute. In the Add/Remove columns panel, there is a drop down list with deleted attributes for the current agency that can be added again (Figure 4.12).

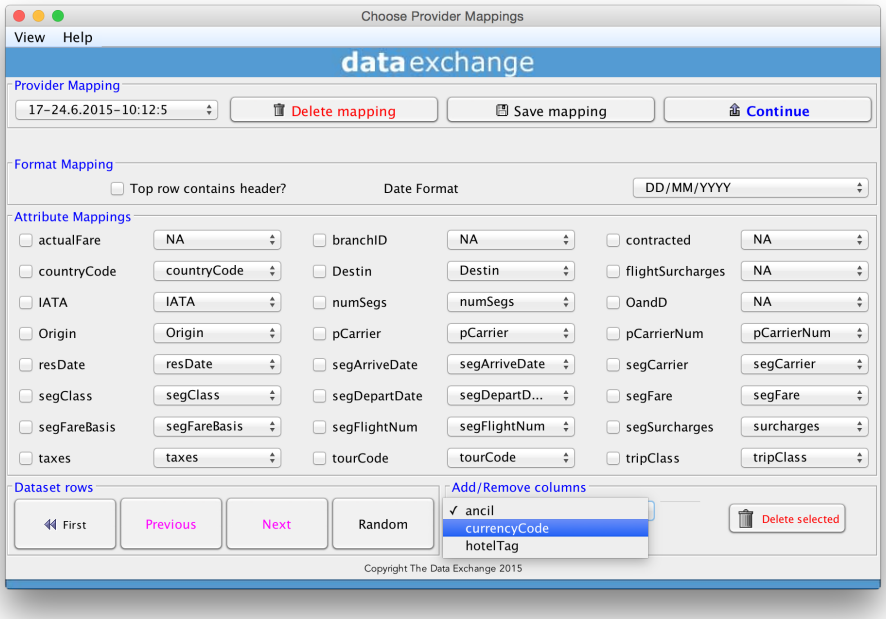

Figure 4.12: Adding deleted attributes back to the *Attribute Mappings* panel is possible through the controls in the *Add \Remove columns* panel. The user needs to select which deleted attribute to restore and confirm the selection.

The buttons in the Dataset rows panel control which row in the raw data is the source for the substrings present inside the attributes' drop down lists. Some datasets provide a header row which is often the first row of the dataset. The header row is very helpful as it contains the attribute name in the correct index of the string, so all the client has to do is select the attribute name that matches the correct drop down list. When the top row contains the header, a checkbox in the Format Mapping panel needs to be checked to ensure that the program doesn't parse the header row as part of the dataset, as the header row is metadata.

Using the Dataset rows panel, the client can browse through each row in the raw data individually to ensure that the correct substring index is selected for each attribute name's mapping. The "First" button takes the user to the first row in the raw data. The "Previous" and "Next" buttons help iterate through

the raw data. “Random” as the name implies takes the user to a random row in the raw data.

A key feature of this program is the ability to save mappings. If the user was able to select the correct mapping for each column and was not able to save this information, there would be many redundant steps each time the client wanted to use the application. Therefore, functionality has been provided to save the mappings to a database and retrieve them when the client encounters data from the same agency in the future. Figure 4.13 shows an example of the processes being described here with the CAR blockType.

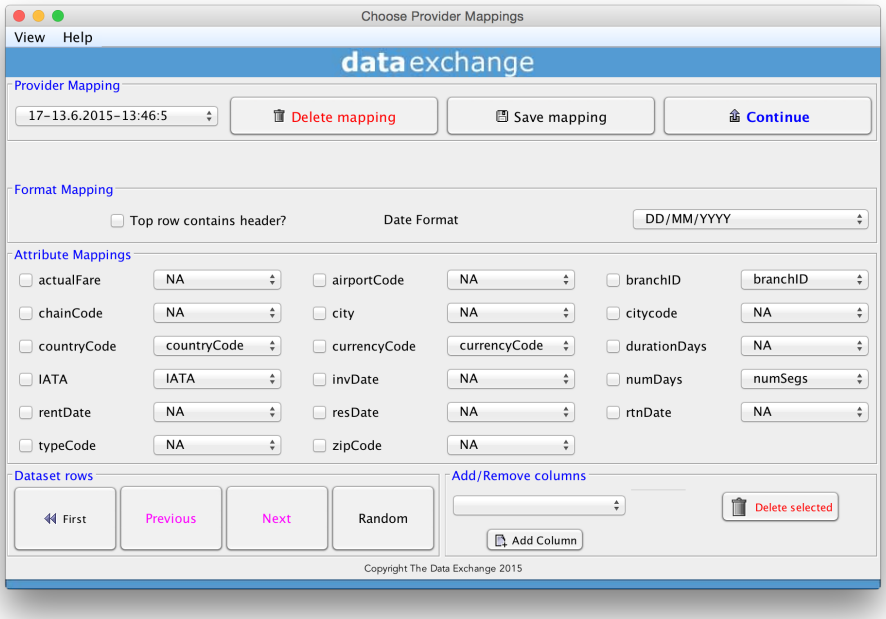

Figure 4.13: When a different blockType is loaded in the previous screen, different attributes and provider mappings are loaded from the database

The top panel contains the components used for this purpose. In the “Provider Mapping” drop-down list, all entries in the mappings database table for the currently selected agency and blockType combination are loaded. New mappings can be saved, and are done by adding a time-stamp to the mappings selection so the user can differentiate between different mappings. If the user does not require a particular set of mappings any further, then the mappings set can be deleted by using the “Delete mapping” button.

Another interesting feature that is provided in this screen is the ability to cater for different date formats. Using the “Date format” drop-down list, the user selects the international date format being used in the particular dataset, and this is used for error checking later on in the program. Since each agency is consistent in the dataset that it publishes, once a particular setting is saved for each agency, they rarely need to be changed for another dataset from the same agency for the same blockType.

4.3.6 Error metrics generation

The 10 error metrics and statistics being used as a baseline measurement for the health of the dataset have been described previously. This section aims to deal with how they were programmatically addressed to generate the appropriate heuristic.

Once the key attributes present in the dataset are identified, the attribute characteristics for these are loaded from the *attributeDescriptor* table. Then, the program compares these characteristics with each row and column of the raw data to see if these requirements match.

A table was created in the MySQL database to store the error metrics, for ease of future retrieval. The table had the following SQL creation syntax:

```
CREATE TABLE ‘errorLogs‘ (
  ‘rowNum‘ int(10) DEFAULT ’0’,
  ‘colNum‘ int(2) DEFAULT ’0’,
  ‘type‘ varchar(15) DEFAULT NULL,
  ‘fullError‘ text,
  ‘ID‘ int(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (‘ID‘),
  UNIQUE KEY ‘ID_UNIQUE‘ (‘ID‘)
) ENGINE=InnoDB AUTO_INCREMENT=283885 DEFAULT CHARSET=utf8;
```

rowNum and **colNum** store the location of the error in the raw data. **Type** indicates the type of error e.g. Missing value or format check failed etc. **fullError** provides a brief description of what triggered the error for non-simple error metrics where there could be a number of causes. This field is also used when specific information regarding an error is required, for instance, to log the number of unique rows when the Unique Ticket numbers check is carried out.

Syntactic errors are carried out on a cell by cell basis. As the program iterates through the parsed data, each cell is inspected to see if:

- it contains a value (Missing entry check)
- the entry's format matches the required format of the attribute (Format)
- length of the entry is above or beyond a certain range Length
- entry is between a certain acceptable domain (Integrity)

Semantic errors are carried out on each row to see if the row contains any violations. This is done by checking:

- if attribute of row is selected as a primaryKey then no duplicate(s) of that attribute value exist (Duplicate check)
- if attribute of row has another attribute to check against to see if there are contradictions, then this check is made and reported (contradiction check)
- if more than half the attributes in a particular row are empty, then an error is flagged (Semi-empty row)

Coverage errors are unique to certain datasets and blockTypes, and as such were only included as part of a health heuristic because of its usefulness to the client. The checks for such errors are done if:

- an attribute's calculateSum column in attributeDescriptor is selected, then all values of that attribute are tallied to generate a sum (Calculate Sum)

- a blockType has a unique identifier for each type of entry it accepts. At present, this is only applicable to the AIRTRIP blockType and its ticket number attribute (Unique ticket number)
- a blockType exceeds the number of allocated segments. This is applicable to AIRTRIP and AIRREFUND blockTypes only (Segment Excess)

4.3.7 Health Score

The premise of the project was to generate a definitive heuristic based on which the client could decide whether the dataset that was checked was healthy or not. The heuristic needed to be accurate, and not merely qualitative. This is because if improved datasets are sent subsequently, the client needs to be able to compare the progress made. A qualitative scale would tell the client whether any improvements were made, but would provide insufficient information as to how much improvements were made.

For this purpose, a weighted average of the error metrics was calculated and used as the final health score. The weighting was assigned based on the relative importance assigned to the error by the client. Despite the subjective nature of this method, it was chosen because the data consumer knew their requirements best and would be the primary user of the tool in the future.

The errors were ordered linearly from most important metric to least important metric. Next, the error metric outputs were normalised between 1 and 100. This was done by converting them to percentages so as to denote what percentage of data checked contained these errors. The score denotes health, thus the error percentages were subtracted from 100 to give a healthiness score rather than an error score.

The client rated the errors from most important to least important in this manner: Semi-empty, Missing, Contradictions, Format, Integrity, Length, Duplicates, Segment Excess.

The calculations made on this basis are shown below:

```
Syntactic score = 100 - (0.35*Missing_errors + 0.13*
   Length_errors + 0.22*Integrity_errors + 0.30*
   Format_errors)

Semantic score = 100 - (0.40*Contradiction_errors +
   0.15*Duplicates_errors+ 0.45*Semi_empty_rows)

Coverage score = 100 - Segment_Excess_errors.

finalScore = 0.5 * semantic score + 0.4 * syntactic
   score + 0.1 * coverage score.
```

4.4 Analysis

The design of the data parsing and interpretation section was done in three stages as identified in the Design section. However, it is to be noted that this was not a fully linear process. During successive prototypes, various error metrics that were initially not considered were added to the implementation, triggering a rethinking in the design.

This occasional intervention wasn't unexpected either, as it is an active part of developing a system using the agile methodology.

The characteristics of the data that are expected to be input to the program by the user were analysed. It was based on this empirical hands-on approach that the final program was designed, keeping the client's requirements in the forefront of the design process.

Next, the categories of errors to be highlighted and a preliminary list of the errors within each category which were to be flagged were also noted. Once

the architecture and underlying database mechanisms etc. were decided, the implementation was begun by building a client-friendly GUI.

It was necessary to ensure that the program produced the right kind of error metrics. The principle of Garbage In Garbage Out applies here. If the right numbers are not generated, the visualization will be entirely without merit as it will be visualizing faulty data. Hence, the error metrics were carefully designed and tested to ensure accurate numbers were being reported.

As part of the agile development methodology that has been pursued in this project, testing of the program needed to be continuous and integrated into the project workflow. Therefore unit testing and behaviour testing was done on an ongoing basis. This was made all the more essential because of the regular updates made to the project where new behaviours were added to the application; behaviours that needed to be shown to the client on a regular basis. Before the demonstration, it was essential to test for bugs and to do state checks: ensure that the results were as expected.

However, due to the nature of the technologies used in this project, it was difficult to set up a testing suite to automate the testing process. Nonetheless, tests were conducted on a frequent basis to circumvent this difficulty. In the Results and Evaluation chapter, an extensive critique is provided to analyse the design and implementation undertaken in this chapter.

4.5 Summary

The requirements for the design were finalized with the client and then an iterative design process began. The existing computing setup of the client was taken into consideration when designing a solution. This made it easier for the iterative testing process, as the client could seamlessly check the efficacy of the solutions being developed. The error metrics and a final health score were also generated, by ordering the errors in the order of importance, which meant that the client could get an accurate picture of the data quality

instantly. Additional functionality like iterating through rows from raw data within the mappings screen were not initially considered; they were added as they were deemed an important feature from client interactions. After the raw data is parsed, and errors have been generated, the visual depiction of the same can take place. This is described in the next chapter.

Chapter 5

Visual Depiction

5.1 Overview

In this chapter, the design and implementation of the visualization component is described. At this stage, only a tabular representation of the data statistics and error metrics exist. A suitable way to present the data to the client must be designed and implemented. This was an ongoing iterative process, as shown in the sections below.

5.2 Design

5.2.1 Rationale

Generating the error metrics and related statistics is not enough. One must present it to the user, so the trends and information that is the focus of the visualization is shown appropriately. In this section, the aim is to design a visualization that captures the essence of the data quality. The nature of errors in the data is highlighted and is to be used by the user to make future alterations to incoming data.

The design process needs to cater for an agile development workflow, as the designs will go through multiple iterations, as important data error

dimensions are uncovered and as client interaction provides novel insight into the importance to be placed on individual error metrics.

5.2.2 Design process

Since the final visualization output is meant to be non-simple, a number of different aspects need to be presented within a single depiction. For instance, three distinct categories of error metrics need to be shown in the visualization, along with a summary of the incoming dataset. A final requirement is to relate the data quality heuristic - the data health, to the user. To help facilitate this complex design, it was decided to use the Five Design Sheet [CITE] methodology to design an information visualization tool. It is documented to work well within an Agile workflow, thus fitting the design needs of the project adeptly.

As the name implies, this methodology creates five design sheets, each of which involves varying levels of client-designer interaction. The initial sheet is called the “brainstorming sheet” where the data is considered and possible solutions are sketched. After this, the next three sheets are “design sheets” where three possibilities from the brainstorming sheet are sketched as they would appear on the screen, and the visualization’s operations and pros/cons are discussed. The final sheet is the “realization sheet” and is arrived at after the 3 design sheets are discussed with the client and appropriate changes are made to the visualization concept. The requirements and estimated time that would be needed for completion of this final design is also discussed on this sheet.

The first sheet was used for brainstorming initial ideas. Each of the error metric categories like Syntactic errors, Semantic errors and Coverage errors were visualised in different manners to see different implementations of presenting the data. Another consideration that evolved during this stage was the need to present the Data Overview (summary of the data and the

errors) alongside the error metrics, so that a summary of the summaries was presented.

Continuing this thought, a simple visualization also needed to be shown to present a quick snapshot of the health of the dataset, without requiring the user to interpret all the information being shown in this complex visualization. Thus it was decided to show a Final Score section too, a section where a quick glance will give the user a conclusive answer about how healthy the dataset is. The Final Score attribute needs to be normalised between a certain range, so that the result is intuitively understood. A percentage value, normalised between 0 and 100 would be ideal.

Design Sheet 2

After the brainstorming session, the first design was put forward. This is shown in Figure 5.1. This sheet shows the design of a tabbed interface that was put forward after some initial discussions with the client. As discussed above, the amount of information to be conveyed to the user could be overwhelming so a tabbed interface was suggested as a way to overcome that. However from further discussions with visualization experts and the client, it was realised that a tabbed interface is not an effective way to present summaries. A summary visualization needs to show the entire summary, whereas a tabbed interface by its nature works best with information that needs to be hidden from view for particular views. Final Score was presented as a quick metric in the bottom right corner, and as a metric that required minimal mental processing it was felt to be a good choice.

Design Sheet 3

The concept of a summary view was further developed in the design of Sheet 3. An attractive visualization interface was developed that included all the aspects that needed to be shown in the final visualization with the added

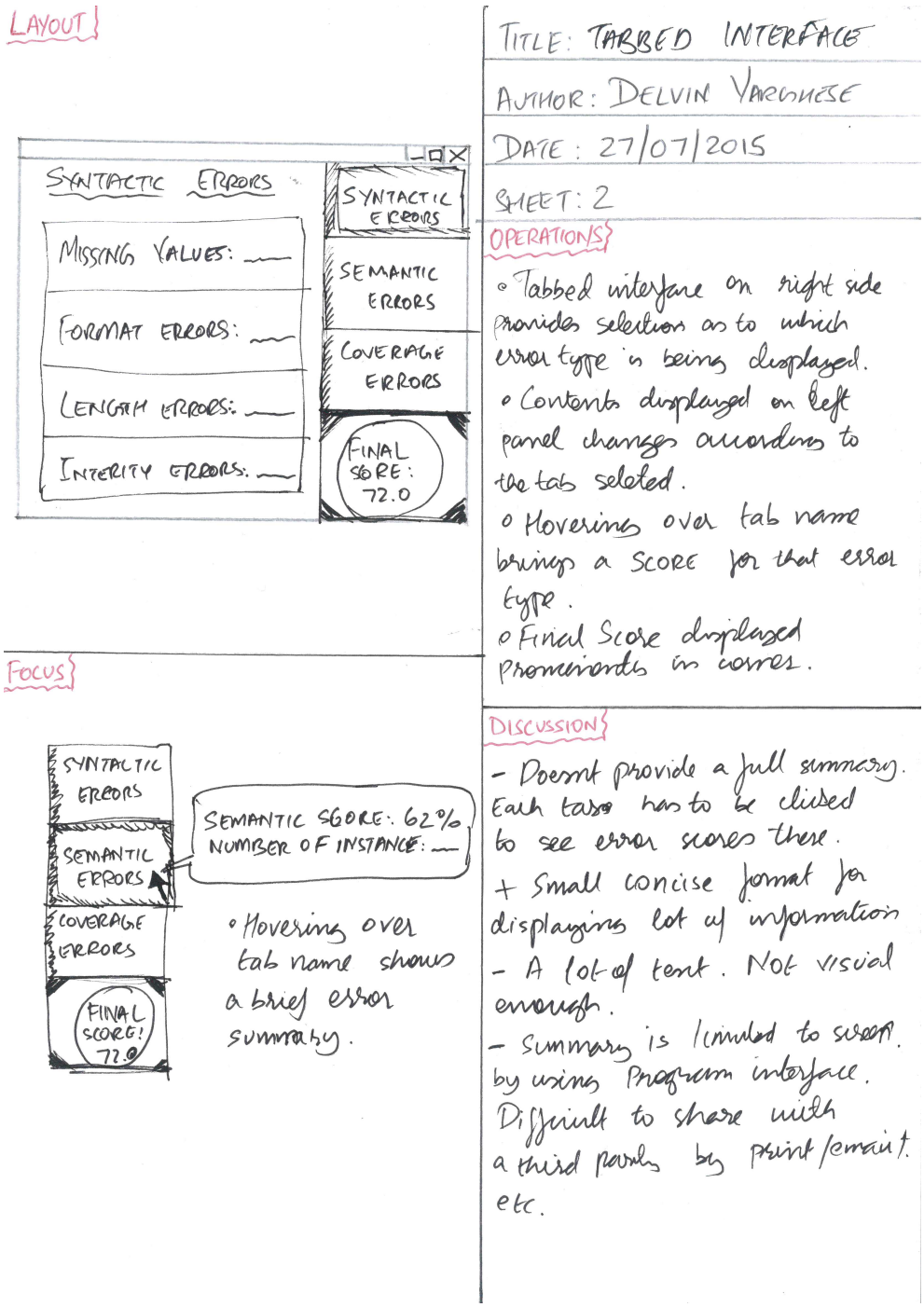

Figure 5.1: FDS Design sheet 2

benefit that of an aesthetically pleasing circular design (Figure 5.2). A table grid was incorporated into the Syntactic errors section to show the different types of errors (missing values, format errors, length errors and integrity errors) in a single view. This was especially required as the design shows that using a circular design meant sacrificing space which would be otherwise useful with a rectangular design.

Similarly in the Semantic errors section, since rows are represented (as opposed to errors at a cell level in Syntactic errors section), 3 single column grids were used to represent each of the semantic errors (contradiction, duplication and semi-empty row errors) and the spatial location of these errors in the dataset.

A discussion with the client revealed that showing all four errors in a single grid for the syntactic errors section but obscure certain errors or if the colours used to visualise them are not chosen with care, data might be incorrectly interpreted by the human visual system.

Design Sheet 4

An alternative design that was proposed is shown in Figure 5.3. This time the syntactic section was visualized using four distinct grids to make error interpretation less confusing. Aesthetic design principles were scraped for this design, with a minimalist approach being used.

Design Sheet 5

In the fifth and final sheet, designated as the realization sheet, the clients feedback from the previous three sheets is used to develop a design that incorporates many of the changes put forward to existing designs and developing a design that looks like the ideal output at this stage of the process. Aesthetics from Design Sheet 3 were incorporated with the structure from Design Sheet 4 to produce this final sheet (Figure 5.4).

The Final Score section is centred in the visualization to immediately grab the attention of the viewer as the most important metric. The syntactic errors are visualized individually, so the user can compare the prevalence of different errors and compare their spread relative to the position of the offending cell in the dataset.

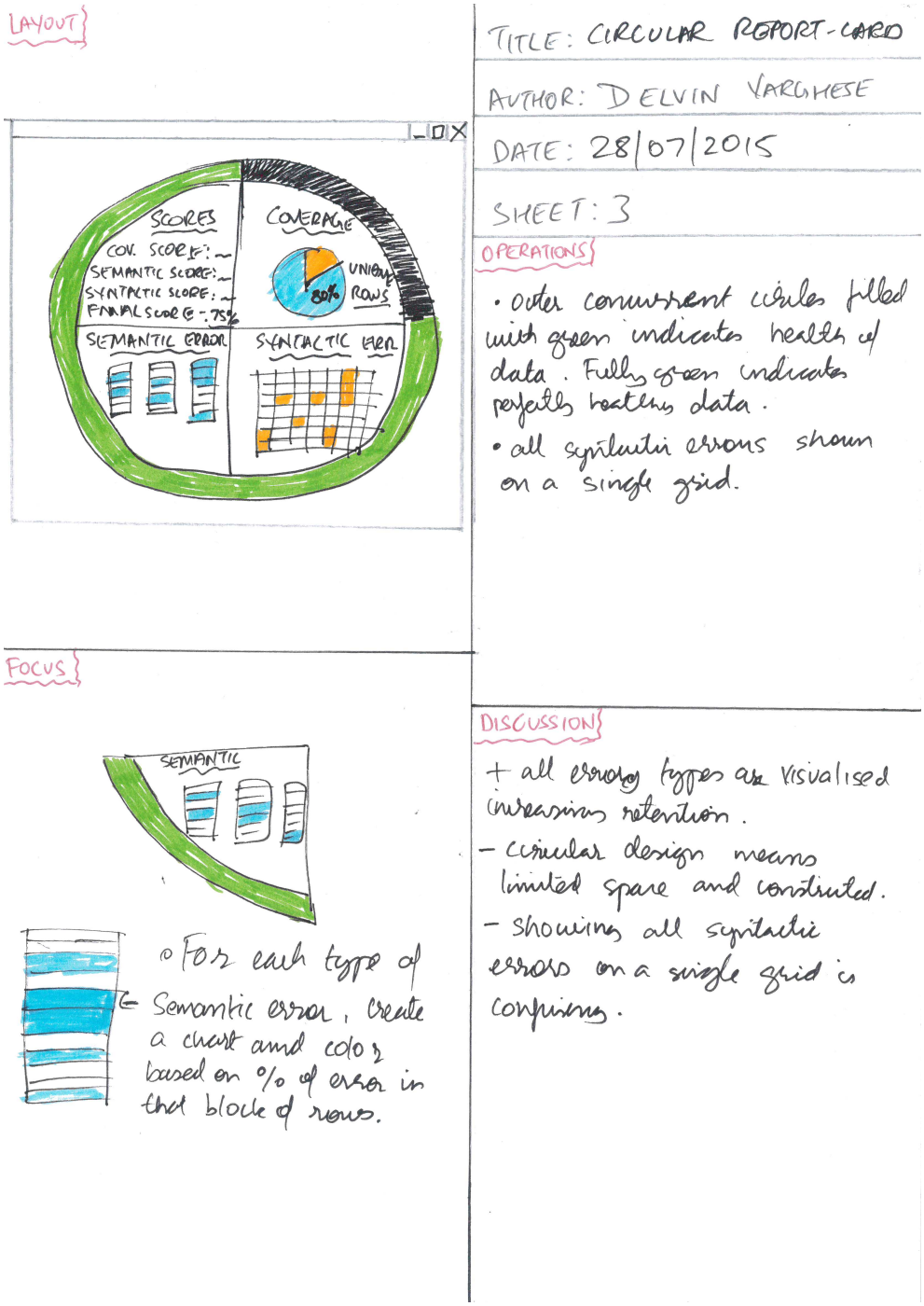

Figure 5.2: FDS Design sheet 3

In the Semantic errors section, alongside a vertical representation of the offending rows, a bar chart is shown to compare the frequency of error occurrence. This will further help analysis as the Semantic errors function on a level higher than the errors caught in the syntactic checks.

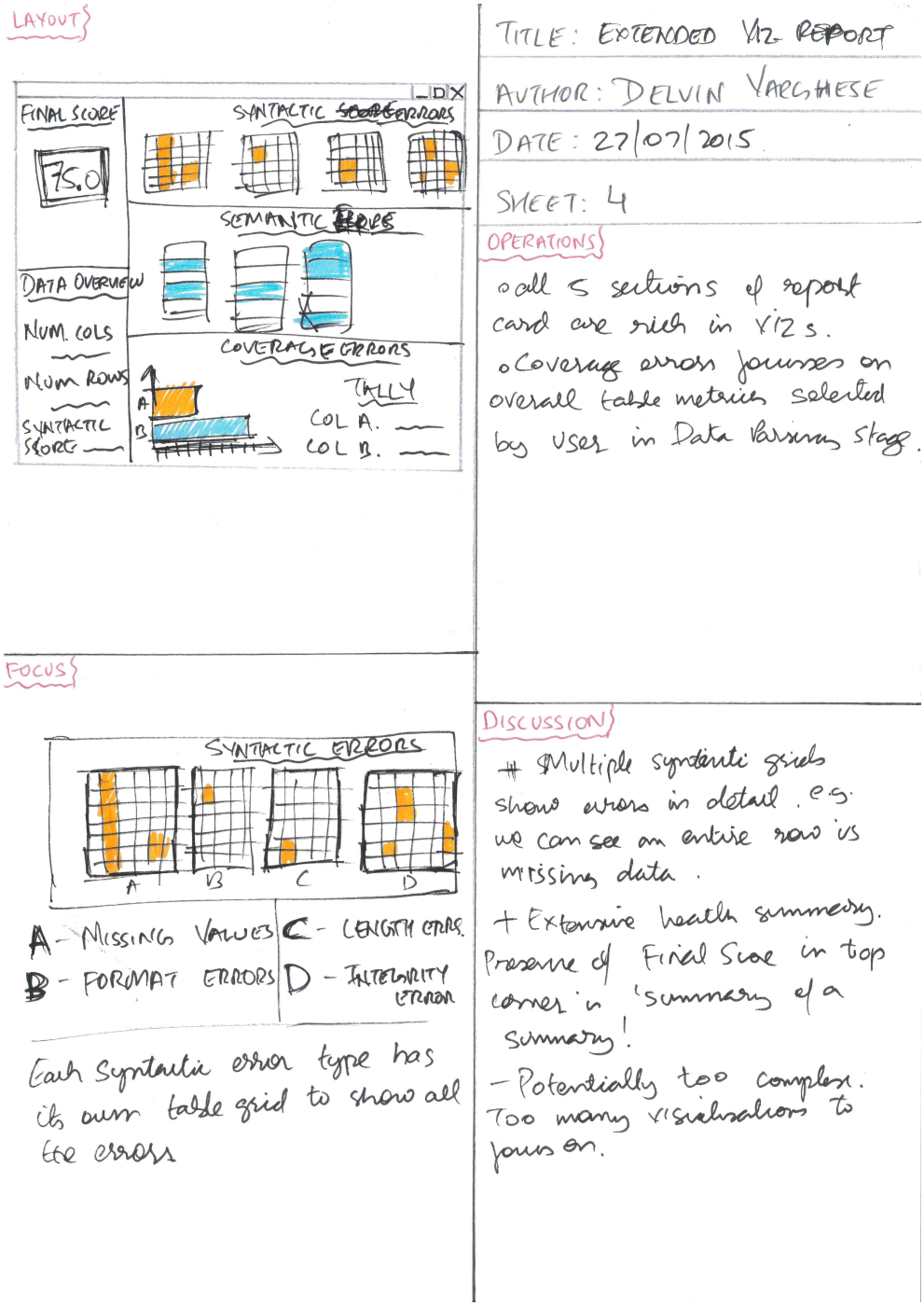

Figure 5.3: FDS Design sheet 4

5.2.3 Summary

This section dealt with the design process undertaken to decide the final output of the visualization tool. The FDS process was incredibly effective in putting forward multiple ideas and facilitating discussion on them to arrive at the best possible design, incorporating the best elements from multiple design sheets. Another advantage is the relative ease with which a client can

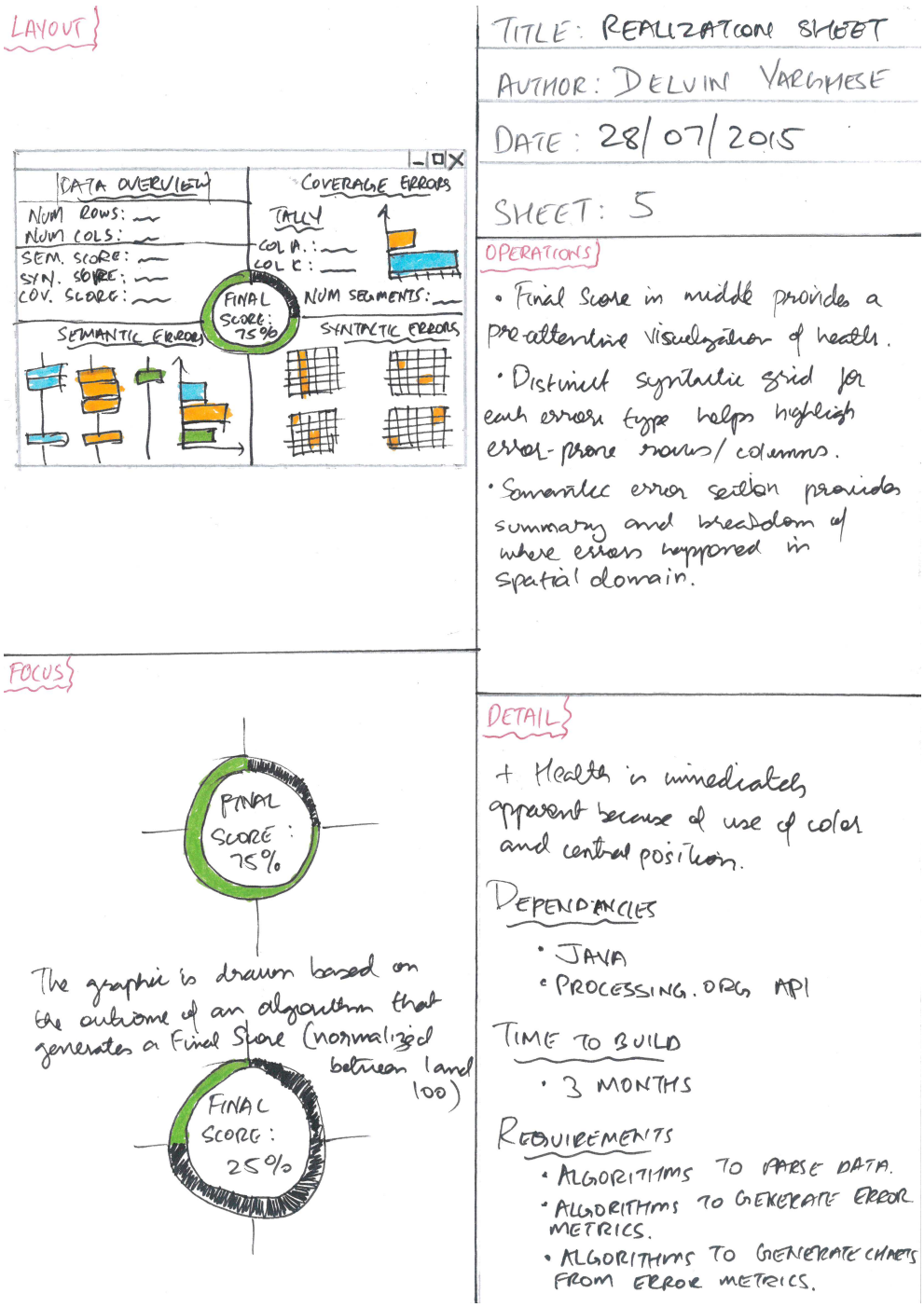

Figure 5.4: FDS Design sheet 5 which was the realization sheet. The final program design was based on this sheet.

interact with potential designs and suggest changes, a process that would be harder with sketching or design done in software.

5.3 Implementation

5.3.1 Rationale

The realization sheet from the Design stage is an ideal starting point for the implementation of the visualization tool. This is the template that will be used as the reference for designing the prototypes. As part of the agile development workflow that has been embraced in this project, regular prototypes are made which are then discussed with the client for feedback. This feedback becomes a strong basis for the changes and improvements introduced in the next prototype.

Several prototypes were produced and discussed with the client using this methodology over the course of project development. For the sake of brevity, only a select few prototypes are discussed here. This will enable discussion of new features and improvements introduced over the course of several prototypes and help track the achievements of the overall product objectives.

5.3.2 Prototype A

Once the template for the visualization tool was finalized, as discussed in the previous chapter, priority was given to development of error metrics and statistics for the data parsing and interpretation aspects of the program. Once a sufficient amount of work was done there, attention was turned to setting the bare-bones structure of the visualization output.

Using Eclipse IDE as the programming editor of choice, and after setting up Processing.org for use within Eclipse (as a Java library), work began. After the first series of prototypes, the prototype shown in Figure 5.5 was arrived at. Similar to the template, 5 distinct sections are drawn in the visualization: Data Overview in the top-left corner, Coverage Errors in the top-right corner,

Semantic Errors in the bottom-left corner, Syntactic Errors in the bottom-right corner, and Final Score in the middle.

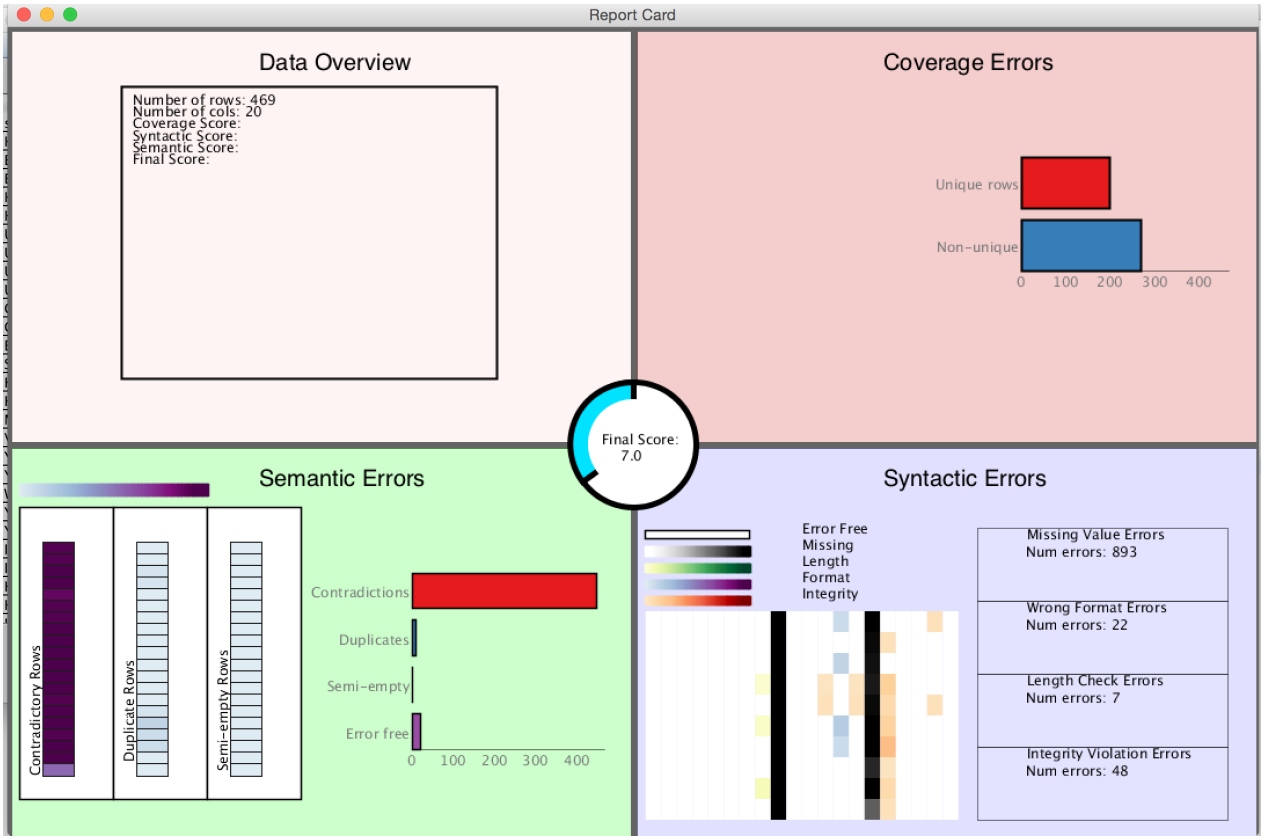

Figure 5.5: Iteration 1 showing initial implementation efforts towards the FDS realization sheet from the previous section.

The Data Overview section contains a block of text that aims to provide an overview of the data. Currently, the number of rows from that dataset that are parsed and the number of columns in the dataset are displayed in a simple manner.

The Coverage Errors section shows a single bar chart visualization that shows a comparison of the number of rows with unique ticket numbers against rows that have duplicate ticket numbers.

The Semantic Errors section contains two visualizations showing different dimensions of the same error statistics. On the left side, three columns are each each depicting the spread of errors (contradiction, duplication and semi-empty respectively) in the rows of the dataset. Only two rows are shown, but each row is an aggregate of a number of rows. For instance for a 1000 row dataset, each of the 20 rows will represent an average error of $1000 \div 20 \equiv 50$

rows in the order that it is present. A colour table is shown as a legend above it, to show what the colours in the rows signify.

The bar-chart on the right side of the Semantic errors section shows totals for each of the error categories. These error totals are shown alongside the total for the rows with no errors, as a comparison measure.

In the Syntactic Errors section, a large grid of cells is shown which is coloured with values from five different colour tables. Similar to the Semantic errors visualization, rows are aggregated such that only 20 rows are shown in the table, however, these are representative of the entire selection of rows in the dataset. Due to the limited number of columns, all of them are shown. Each cell possesses an error value for its contents' missing values, length errors, format errors and integrity errors. It was realised during the creation of this grid that priority will have to be given to certain error measures over others. For instance, missing values are more important to visualize than Length values. Similarly, the other measures were ordered and the table was rendered, showing all 4 types of errors (or a blank white cell where all 4 types of errors didn't occur). A textual representation of the number of errors for each category was presented on the right of the grid visualization.

Finally, a circular section was added in the middle to depict the Final Score quickly to the user. Based on the final score value between 1 and 100, the circle is "filled" with the cyan concentric circle.

Despite the significant progress made in the last few prototypes, the culmination of which was discussed here, a number of flaws in the approach were made apparent. These flaws were taken into account in subsequent iterations. Of particular note, is the decision to draw the syntactic errors as a single grid, despite discussion in the design stage that proposed multiple grids as the more salient choice. This was undertaken in the prototype to test that particular opinion. It was worthwhile to see if a single grid could have concisely presented the information, which would otherwise be redundant if shown using multiple grids.

5.3.3 Prototype B

After some time of iterative development of Prototype A, the prototype being discussed here (Figure 5.6) was arrived at. Improvements were made mainly in the Data Overview and Syntactic errors section.

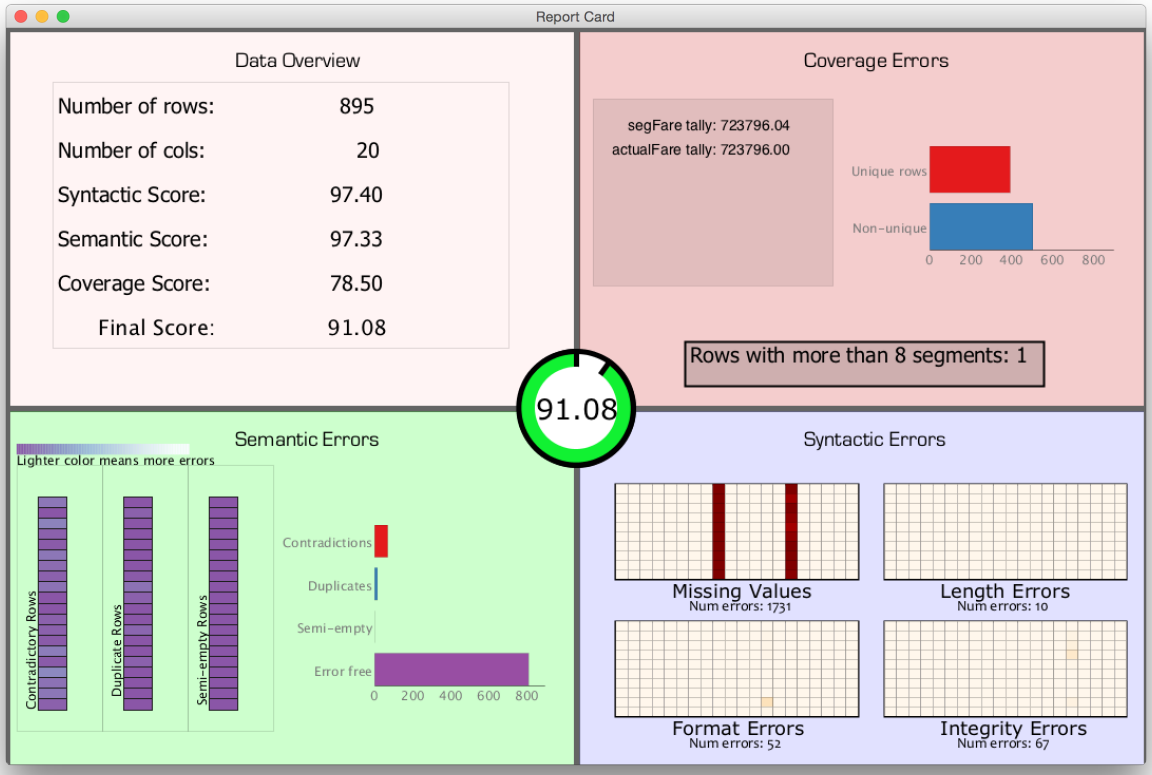

Figure 5.6: Iteration 2, showing the improvements made to the previous design. Data Overview section has been redesigned to be more aesthetically pleasing. The syntactic errors section has been changed by introducing individual grids for each error type.

In the Data overview section, with the help of additional metrics that were generated in the error parsing stage, estimates of the error scores were published alongside the row and column count information. The final score that present here is also visualized in the Final Score section in the middle (which has been improved from previous iterations).

In the Syntactic errors section, the single grid view was scrapped in favour of multiple grids: 4 grids representing each of the syntactic error measures. The decision was immediately justified by the clarity of the information that could

be gleaned from the visualization. The statistics that were initially shown as a separate entity in the previous prototype were integrated next to the grids, for a more unified and coherent understanding of the errors.

However as seen in the screenshot, cells with very few errors were not easily picked up. This is particularly the case in the Format Errors part of the screenshot, where the penultimate row has an error being shown in one of the middle columns but it is not very apparent. In future iterations, an alternate set of colours needs to be used to represent the range of errors better visually.

Despite the ability to cater for four blockTypes in the program, one is primarily more relevant for the client: AIRTRIP. The client requested the ability to show a number of coverage errors that were specific to AIRTRIP blockType but not for the others. Thus abilities to view the tally of key columns as well as a count of the ticket numbers that have more than 8 segments were added to the Coverage Errors section. This is not applicable to the other blockTypes, so when another blockType is selected, these two statistics are not present in the final output.

5.3.4 Prototype C

The prototype that is showcased here (Figure 5.7) shows a radical overhaul of the look and feel of the report card. A uniform set of colours was used consistently across the sections of the report card. The previous designs used colours that did not signify anything, being there for decorative purposes. This was felt to be chart-junk and discarded in favour of the current design. The result is a more aesthetically pleasing design.

Study was done into the best practices for using colour in designs, and the ColorBrewer principles[17] were used. This helped the design by suggesting appropriate colour schemes to use. Using the *gicentre* Processing library available in Java, ColorBrewer principles are made available in the program. Previously a sequential colour table was used in the grids to show different

levels of errors. But due to limited perception, a divergent colour table was applied, and the differentiation in errors improved.

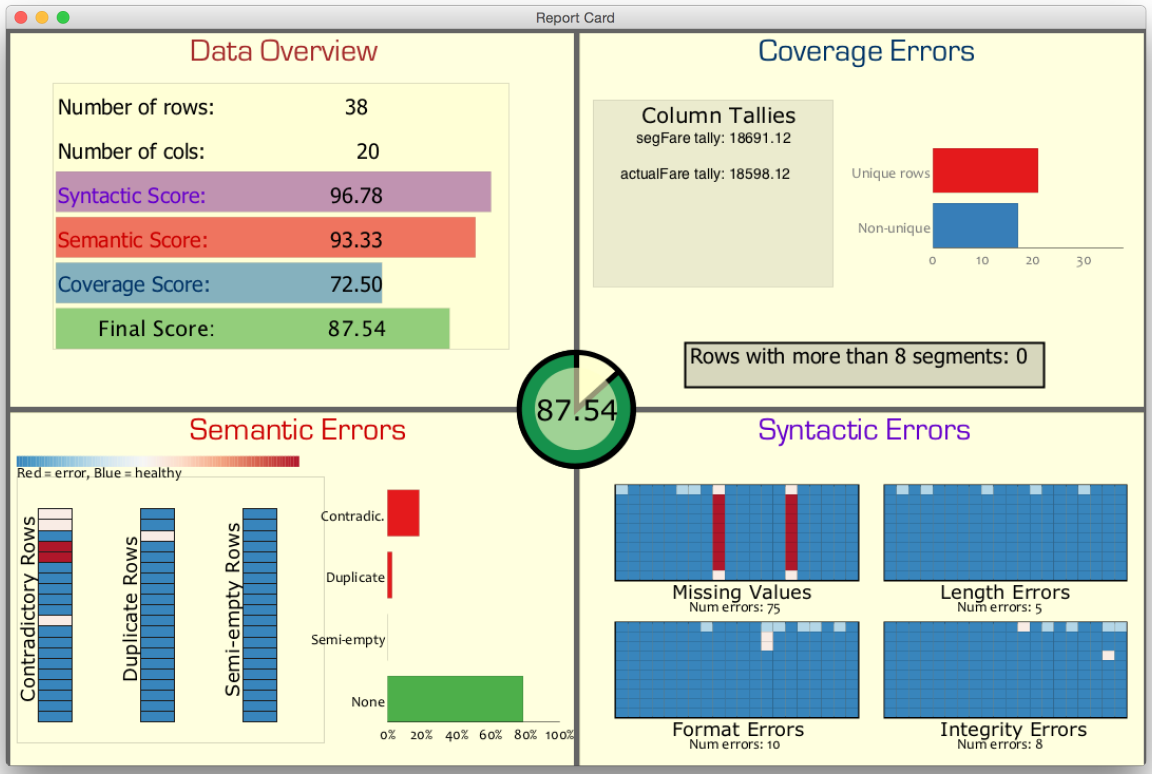

Figure 5.7: Iteration 3 is shown here. The colour scheme has been changed to introduce a look and feel which is more consistent.

The use of the blue-white-red colour table as the basis for visualizing errors in the Semantic and Syntactic errors section has dramatically increased the perception and the spatial spread of errors in the dataset. For instance, in the Syntactic Errors section it is obvious from the Missing Values grid where significant chunks of errors are occurring. However, for the Length Errors where the number of errors are few and far between, the light blue indicates this fact effectively. This is also the case with the Semantic Errors section, where not only the presence of errors but the density of errors comes across very effectively from the colours that are used to visualize the errors.

Apart from the redesign in the look and feel of the visualization, Data Overview also shows the presence of a new visualization. Each of the scores are not only represented textually but also as a horizontal bar that shows reinforces this information. The width of the bar is proportional to the particular error

score it visualizes. The colours that are used to represent the bars are used to colour the title of the associated section i.e. Syntactic Score is coloured with the same colours as the title of Syntactic Errors section. This makes the visual link between the two more obvious and helps the user to navigate the program.

The use of fonts is similarly tailored to highlight what a particular string of text represents. For instance, the titles of the different sections use the same font to show that they summarise what the section stands for.

5.3.5 Prototype D

This prototype that is presented here(Figure 5.8) shows a significant development in the Semantic row visualization. So far the philosophy undergirding the design process has been creating an output that the client can share with agencies that they interact with. It is the agencies after all, that send the datasets to the client. But interacting with the data and trying to decipher trends in the data has shown the gap in understanding because of a lack of context for the errors shown. For instance, the aggregate rows are coloured according to their contradiction, duplicate and semi-empty status error levels. But missing from this visualization is the understanding of how error prone the particular segment of rows are. Another question, that was felt to be inadequately addressed by a simple gradient colouring of errors was a visual cue as to where the highest percentage of errors was.

It was this thought process that led to the adding of interaction and creation of additional error metrics in the data interpretation process. Yellow bounding boxes are added to the top 20 percentage of errors. For instance, in a 20 row visualization with 5 rows that generate an error measure, the single highest error-filled row (20 percent of 5) is fitted with the bounding box.

When the user hovers their mouse over a row with a bounding box, the colour of the bounding box turns to red, and an error description pops up alongside it,

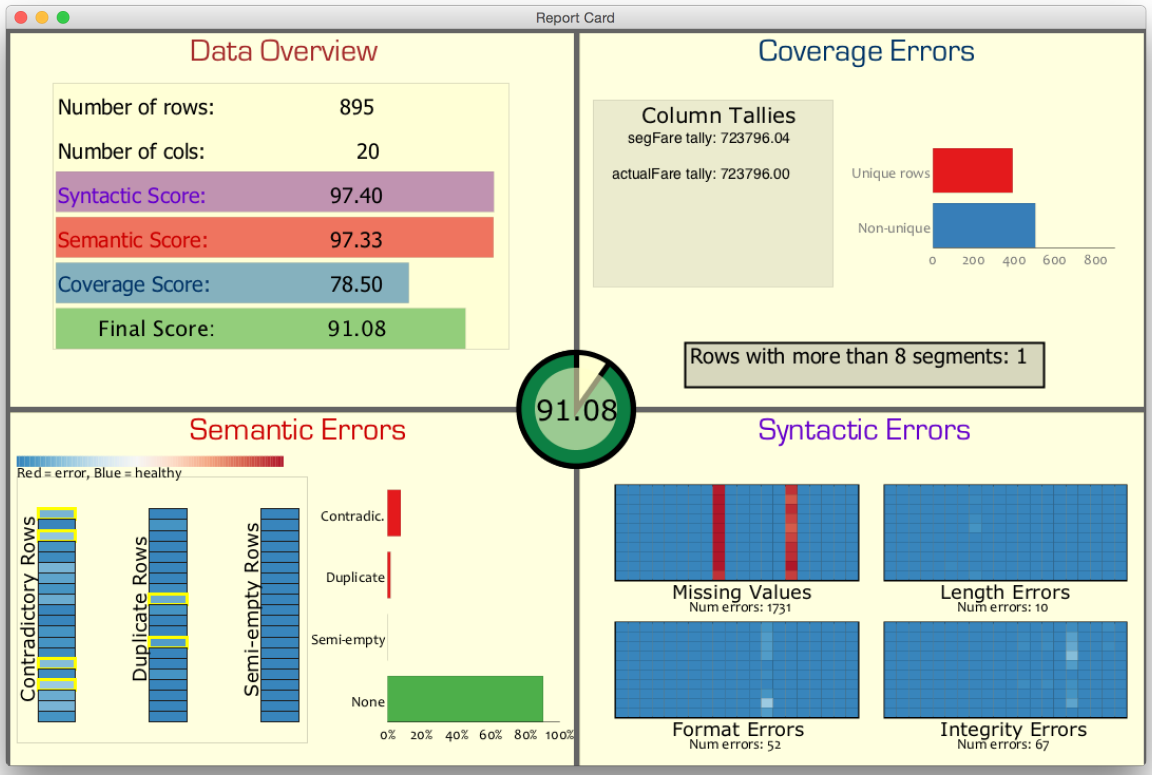

Figure 5.8: Iteration 4 is shown here. Interaction is added in the Semantic Errors section to provide additional information about the location of errors. Yellow bounding boxes are present on blocks of rows that contain the top 20 % of errors.

giving more detail (Figure 5.9). The detail includes the row numbers that the aggregate row is referencing, the index of the aggregate row and the number of error filled rows. This interaction has been added to all three semantic error visualizations.

It is not enough to generate visualizations. Visualizations need to be shared by the client with providers to begin the discussion about data health. Thus the functionality to save an image to disk was added. When the user presses 's' key on the keyboard, a copy of the current visualization is saved to the *img* directory of the program, and a reference to the file alongside meta-information about the report card are saved to a table in MySQL. This information will be used to view and retrieve generated visualizations at a later stage, using the report card viewer.

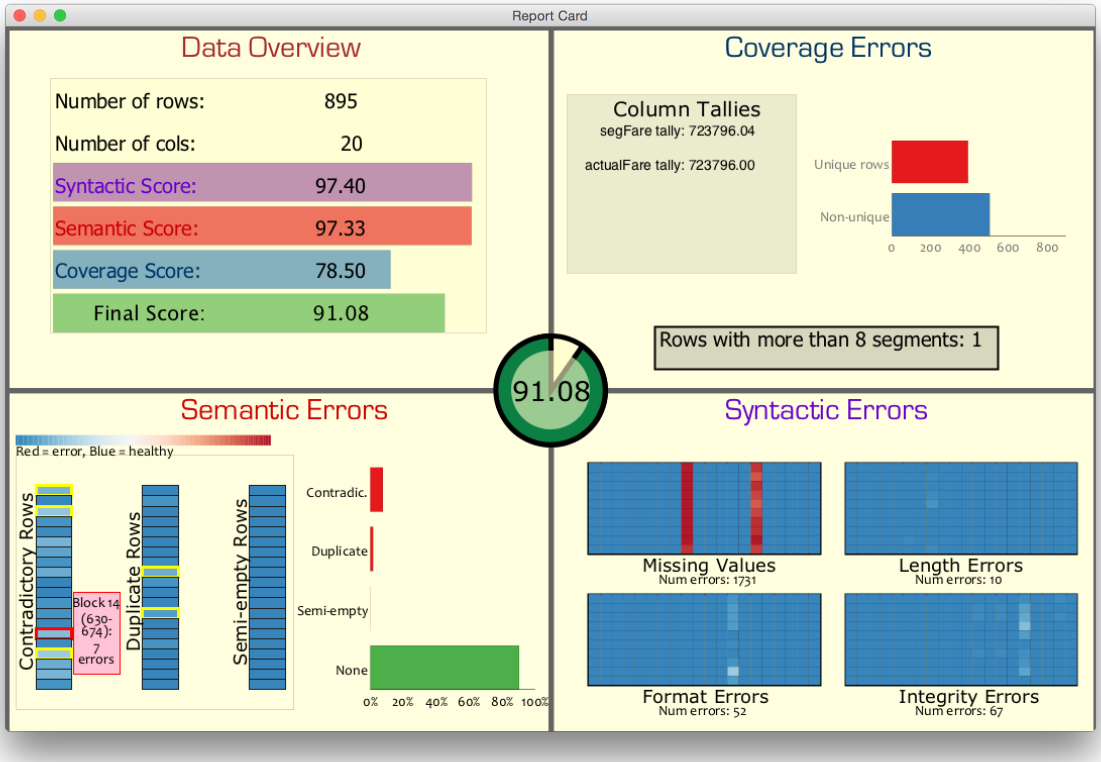

Figure 5.9: Hovering over the segments with yellow bounding boxes reveals the rows in the dataset that are represented by this segment and the percentage of rows therein that contain Semantic errors.

5.3.6 Report Card Viewer

Initially it was envisioned that comparisons could be made between reports generated for one dataset and another. This was important when improvements are made to a dataset and another report card is required, the program could generate a new visualization to show how the health score has improved.

A report card viewing screen was created (Figure 5.10) to retrieve a list of previously generated report cards for the current provider. The interface enables the user to view how well previous datasets did on the final score measure. Unfortunately, due to time restrictions, a full side by side comparative analysis of report cards was not implemented.

Available report cards for the current provider were placed in a scrollable list (implemented using a custom JList wrapped in a JScrollPane). When a report

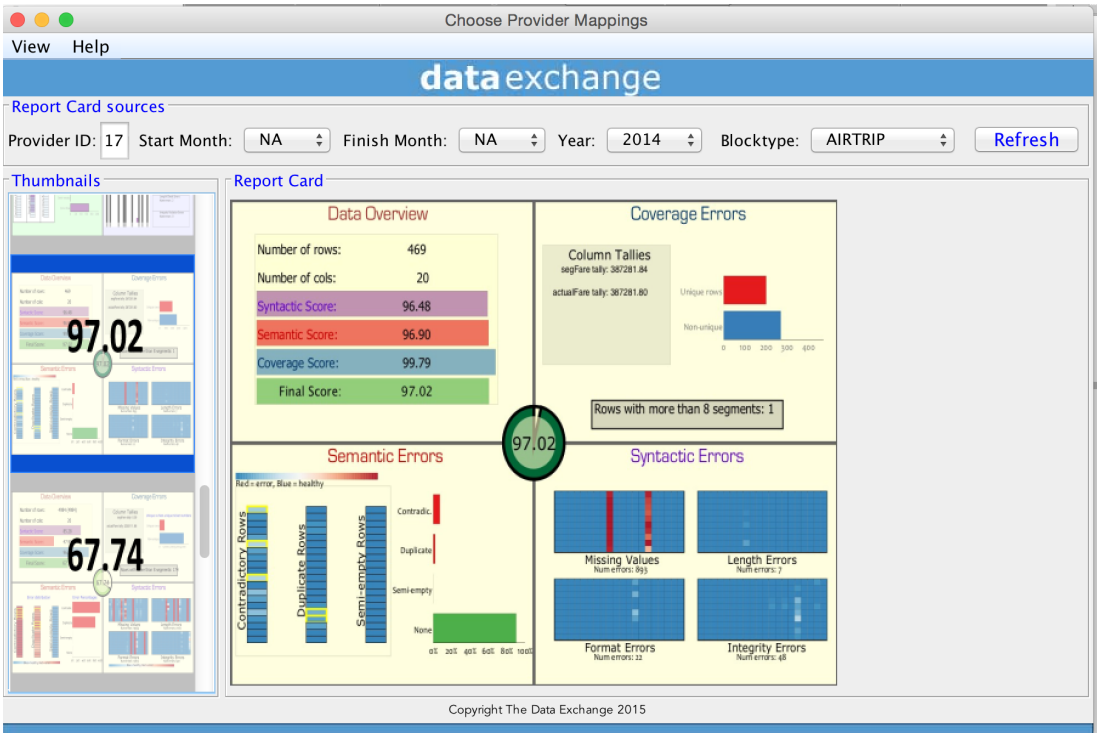

Figure 5.10: Report card viewing screen. The range of report cards available to view can be filtered using the controls at the top including provider ID for which to view report cards and the time-frame to which the particular report cards belong. Once this is entered, a list of thumbnails on the left side is shown in a Scroll pane. Clicking on the items in the list shows the report cards in an enlarged view in the main panel.

card is selected in the list, it is shown in an expanded view on the panel on the right. By double clicking on the image, or by pressing the button next to the image, the user can open the report card image in the default system viewer for the user's operating system.

A filtering mechanism is also provided at the top, so that the user can select another provider, for whom report cards need to be seen. If the report cards to be viewed are to be within a certain time-frame, this can be set at the top too. Clicking Refresh reloads the JList with report cards that match the criteria set here.

A logical suggestion was making available another JList container and image viewer besides the current one, to make side by side comparison possible. However, this was felt to be a cheap hack, and with the implementation of functionality to open in the system viewer, it was reasoned that side by

side comparison of images is much easier in the operating system;s image viewers.

5.3.7 Summary

The key stages in the iterative development of the visualization were shown here. Starting from the realization sheet in the Design chapter as the template, progress was made in creating each section. Interaction although not stressed as part of the philosophy of the design due to the outcome of the program to create a shareable visualisation was nonetheless added. This was added to facilitate greater understanding of the errors in the Semantic errors section, and to get more clarity on clear error trends that would otherwise be missing from the visualization. The visualization that is generated can be saved and viewed later through a report card viewer interface.

5.4 Analysis

Traditionally, the design process is based on client interaction but there is rarely a framework for facilitating this. Communication is haphazard and unstructured, making it hard to keep track of design process and change in desired outcomes or requirements.

Using the Five Design Methodology, the agile development workflow was reconciled with the design process. Iterative designs were produced, where the client actively contributed. Whenever an agreement was reached on any aspect of the program, implementation was promptly undertaken on that aspect, and brought forward to the client for critique and approval before any iterative improvements were made.

The use of this methodology produced a smooth gradual development in the conception of the final look and design of the Report Card.

The Report Card was designed to be shared between the client and the agency whose data was checked and visualized by the report card visualization. Therefore it was deemed necessary at the start to make sure the visualization contained a summary view of all that information that needed to be presented, and use of interaction techniques were limited. However, at a later stage it was seen as the logical next step to add small interactive processes to help understand error trends in greater detail. This would have been missed otherwise, from a static visualization.

The output is visually appealing and it uses colour and size as two key retinal variables in getting the message across about the health of the source dataset. In the next chapter, the output will be critiqued in more detail to see if project aims and objectives have been achieved.

5.5 Summary

The designing of the report card was done with a well tested existing methodology called the Five Design Sheet methodology. This introduced structure and flow to an otherwise haphazard creative process of visual interface design. After an amicable process of collaborative sketching and requirements analysis with the client, the visualization tool was implemented in Java and Processing.org. A viewer interface was also added to give the user ability to view previously generated report cards.

Chapter 6

Results and Evaluation

6.1 Overview

The methodology espoused for this project stipulated that design, implementation and testing can go hand in hand (unlike a traditional waterfall model of design followed by implementation followed by testing). In the previous couple chapters, the design and implementation has been discussed in detail. Testing has been deliberately collated in a specific chapter for both parts of the program to enable an easier comparison between the aims and objectives originally set out for this project and the final outcomes.

6.2 User Scenarios

At the outset of the project, two specific use cases for the program were mentioned:

1. **Report Card generation**: generating a report card for a dataset that the client receives from an agency. The application should be able to parse an incoming dataset and find out the errors present therein and categorise those errors, presenting them visually in a report card to the client.
2. **Report Card comparison**: making health comparisons between datasets. There should be functionality to compare report cards. This is important when a report card prompts an agency to resend the data

after making corrections. The updated dataset's health status should be compared against the original dataset's to measure the improvement.

6.2.1 Scenario 1

Scenario 1 was implemented successfully. Functionality was provided in the software to cater for both data stored on disk (CSV and Excel files) and data read from a MySQL database. The client's current architecture was based around the MySQL RDBMS so the program was built around that. These are the steps the user has to take to complete this scenario:

1. Click on Import Data on the start screen
2. On the next screen, user is asked to choose the blockType from the incoming data that they wish to parse and the specific table in the MySQL database that holds the raw data using two drop-down boxes (Figure 6.1).

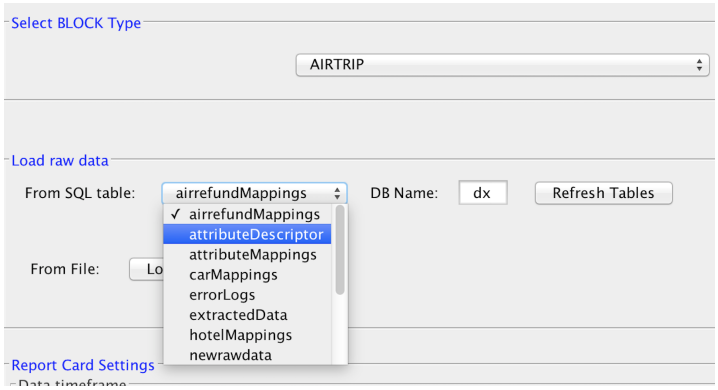

Figure 6.1: Selecting the dataset source from a table list dynamically populated from the user-selected MySQL database.

3. Additional options can be configured like the option to choose which Coverage Errors are checked for and the time-frame that is covered by the data. (Figure 6.2). Once these are chosen, the user presses Continue.

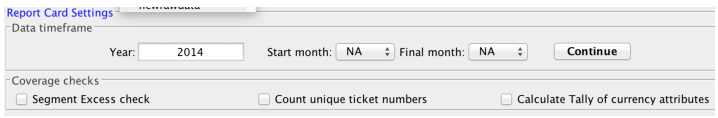

Figure 6.2: Selecting additional error-checking related options. These options are used in setting the time-frame of the generated report cards.

4. The user is presented with the Mappings screen. Here the user can select a pre-existing mapping from the wanted attributes to columns

present in the raw data or create a new set of mappings which can be saved for retrieval next time dataset from the same provider are parsed.

5. There are a lot of configurations available on this screen, with the ability to tweak mappings, delete undesirable attributes (which may not be relevant to datasets from particular agencies). Once these are set, the user presses Continue.
6. The user is presented with a report card showing the error status and health of the dataset. Some interaction is available on the Semantic errors section to see where the highest concentration of errors are.
7. The generated report card can be saved to disk as an image by pressing Cmd + S on a Mac or Ctrl + S on Windows/Linux. The generated file can be viewed either through the application's Report Card viewer interface or through the operating system's file browser. It is in the *img* subdirectory of the directory where the application resides.

6.2.2 Scenario 2

The second Scenario was not fully implemented according to the initial designs due to time constraints and programming complexity issues. The originally intended plan of having direct report card comparisons was scrapped and a Report Card viewer interface was implemented in its stead to overcome this shortcoming. The steps to navigate this interface are given below:

1. Click on Report Cards on the start screen.
2. The next screen shows all available report cards for the currently chosen Provider ID and blockType combination as thumbnails in the left side. Clicking on the thumbnails loads the report card in a larger panel on the right (Fig 6.3).

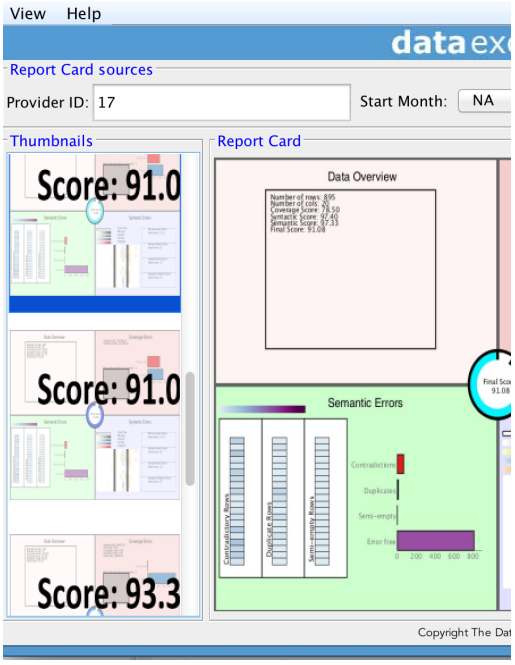

Figure 6.3: Selecting a report card thumbnail to enlarge the view

3. The settings panel at the top of the screen can be used to narrow the range of report cards that are shown. Report Cards can be filtered by the time-frame which needs to be checked by the user as well as the ProviderID to select a different provider. (Future implementations may add additional filters and sorting mechanisms to sort the report cards by their final score etc.)

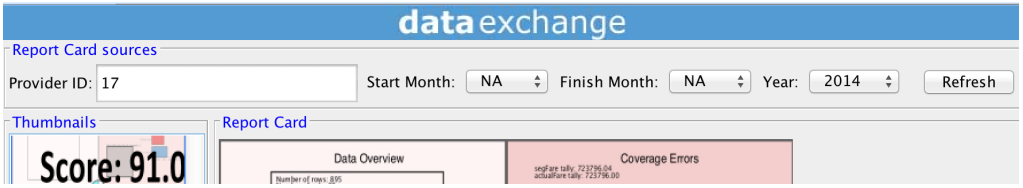

Figure 6.4: Filtering the list of report cards shown

4. Some comparisons can be made between multiple report cards by scrolling through the list of thumbnails once the filtering has been done using the controls at the top. The Final Score is overlaid on the thumbnails to make quick comparisons. Using the arrow keys or a mouse, the user can switch between different report cards.

6.3 Analysing Data Parsing interface

Unit and integrative testing has been implemented throughout the development stage to ensure code integrity and methods successfully

achieved states that they were programmed to achieve. In this section, a system test will be conducted to check for bugs in the chain of processes that culminate in the report card production.

The client provided numerous datasets to check the system against, and these were representative of the datasets the program will be used to check in the future. Where different datasets will be handled, the extensibility of the program, a key feature developed in discussions with the client will ensure those can be adequately catered for by further developments in the program.

The timeliness of the data parsing process will be undertaken, followed by accuracy and usefulness of the generated report card. Feedback has been obtained from the client using a System Usability Study which will also be beneficial for a complete analysis of the progress of the project.

6.3.1 Timeliness for error checking

Timeliness is one of the most important attributes of data parsing and visualization [4], [16]. The process of visualizing data quality errors should not be intractable i.e. the time taken for the generation of the output should be reasonable. Thus timeliness measures as the time between when data is expected and when it is readily available for use.

It was mentioned previously that the client had stated a desired time of less than 2 seconds per 1000 rows of parsing and error checking. In the initial prototypes this was not achieved (initially it took as long as 10 seconds for 1000 rows), partly as a result of following the agile methodology. Quick “dirty” methods to add functionality, at the expense of bad coding practices. In subsequent iterations, improvements were made to the methods and code quality was improved. This significantly improved the time taken for data parsing and error checking to an average of 1.5 seconds per 1000 rows. Table 6.1 shows the timeliness factor for datasets of different sizes. The *newrawdata* dataset is shown at the bottom of the table, which contains over

Dataset Source	Rows parsed (total rows)	Total Parsing Time (ms)	Parsing rate (ms)	Error Analysis Time (ms)	Error Analysis rate (ms)
testrawdata	38 (44)	43.49	1.14	196.70	5.18
testrawdata1	92 (100)	67.29	0.73	380.95	4.14
testrawdata3	469 (500)	204.82	0.44	876.69	1.87
testrawdata4	895 (1000)	311.84	0.35	1,296.33	1.45
ztest2	4984 (4984)	955.88	0.19	20,468.73	4.11
ztest3	16170 (16170)	1,770.81	0.11	27,101.53	1.68
ztest4	1862 (1862)	597.50	0.32	6,382.28	3.43
ztest5	457 (457)	155.02	0.34	2,018.70	4.42
ztest6	1862 (1968)	508.84	0.27	9,258.44	4.97
ztest7	4215 (4215)	708.01	0.17	9,315.76	2.21
ztest8	2352 (2352)	467.84	0.20	7,841.02	3.33
newrawdata	*69349 (76079)*	*6,757.53*	*0.10*	*55,124.35*	*0.79*

Table 6.1: The time taken for each dataset to be parsed and error-checked completely is shown here.

70,000 rows. The size is many times higher than the maximum size expected to be used with this tool by the client, but is shown to stress test the system.

Some explanatory notes on the table are given below:

- The dataset source names have been assigned new names to protect the confidentiality of the source data's company of origin.
- **Rows parsed (total rows)** column shows the number of rows that were parsed and error checked by the program. These are the rows with the chosen blockType as the row blockType. The number is in brackets is the total number of rows in the dataset counting all the different blockTypes present.
- **Total Parsing Time (ms)** shows the amount of time, in milliseconds, it took for the program to read the data from the source dataset into memory in the form of a 2D String array.
- **Parsing rate (ms)** shows the amount of time it took on average to parse one row of the raw data.
- **Error Analysis Time (ms)** is the total amount of time it took to check the errors of all the rows in the raw data.
- **Error Analysis rate (ms)** shows the average amount of time it took to check the errors in one row of the dataset. All time calculations are shown in milliseconds.

Dataset Source	Semantic Errors	Syntactic Errors	Coverage Errors	Total Errors	Errors per row
testrawdata	8	98	0	106	2.79
testrawdata1	8	186	0	194	2.11
testrawdata3	44	970	1	1,015	2.16
testrawdata4	84	1,860	1	1,945	2.17
ztest2	9,130	58,020	174	67,324	13.51
ztest3	22	86,852	1	86,875	5.37
ztest4	1,071	17,987	1	19,059	10.24
ztest5	456	4,372	0	4,828	10.56
ztest6	1,863	24,206	0	26,069	14.00
ztest7	1,374	23,411	4	24,789	5.88
ztest8	2,399	16,629	1	19,029	8.09
newrawdata	*14,025*	*137,994*	*102*	*152,121*	*2.19*

Table 6.2: Analysing error check output. The number of errors generated for each category is also shown. The *Errors per row* column is a good guide of how many error prone the dataset is.
A visual summary of this can be found in Figure 6.5.

testrawdata, *testrawdata1, testrawdata3* and *testrawdata4* contain 44, 100, 500, and 1000 rows respectively from the *newrawdata* dataset. These were extracted into different tables due to the sheer size of the *newrawdata* dataset and the need to test on smaller tables. As can be seen from the table, the parsing rate is inversely proportional to the number of rows that are parsed. The program positively favours larger datasets. The error analysis rate on the other hand is dependant on the number of errors that are generated in course of checking each row, so it is a variable rate from one dataset to another. However, for the above mentioned samples (of *testrawdata*) which contain varying quantities of data from the same dataset source, it can be observed that the error analysis rate is again inversely proportional to the number of rows being checked for errors.

However, the time taken for the application to parse data and check errors is not only dependant on the number of rows to check but also on how error-filled the rows are. Table 6.2 shows the number of errors generated in each dataset.

Figure 6.6 shows a chart that plots the relationship between the number of rows in a dataset and the time it takes to check the errors and output statistics

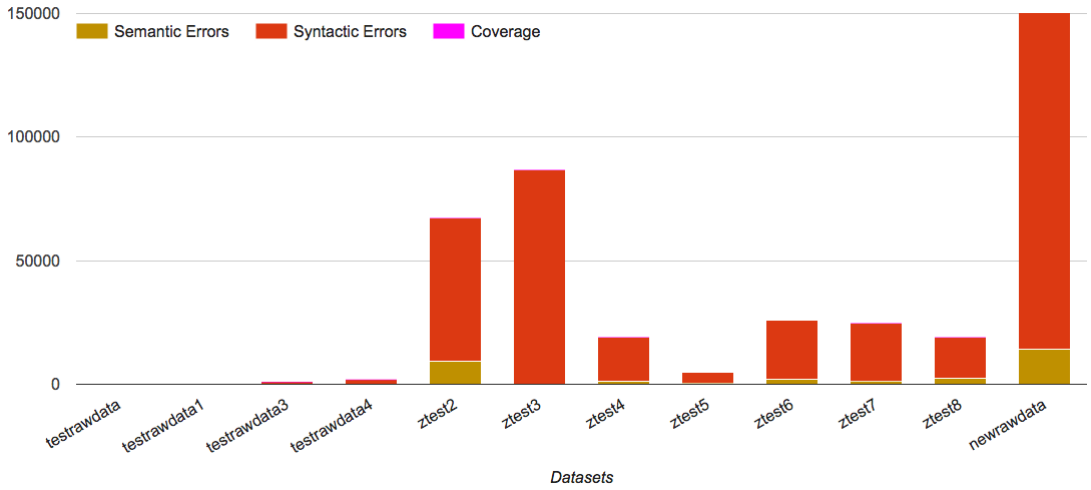

Figure 6.5: Total errors present in each dataset (graphical representation of Table 6.2).

for that dataset. Figure 6.7 on the other hand, plots the total number of errors in a dataset against the time it takes to check the errors for that dataset.

A comparison of these two charts is significant since it reveals the relationship the number of errors and the number of rows has with the time it takes to check a dataset for errors. The number of errors has a direct relationship, resulting in the straight line that can be seen when visually connecting the points in Figure 6.7. Figure 6.6 on the other hand, shows the points forming a curve with a positive slope. To verify this, a test was conducted with datasets of sizes varying from 20,000 rows to 70,000 rows to ensure the missing range was accurately represented when curve fitting.

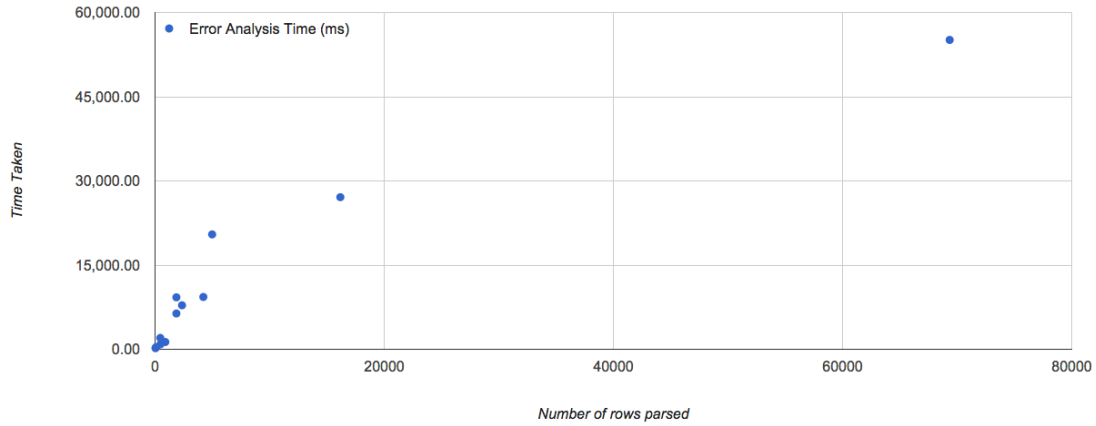

Figure 6.6: Number of *rows* in dataset against Time taken to do the error-checking process

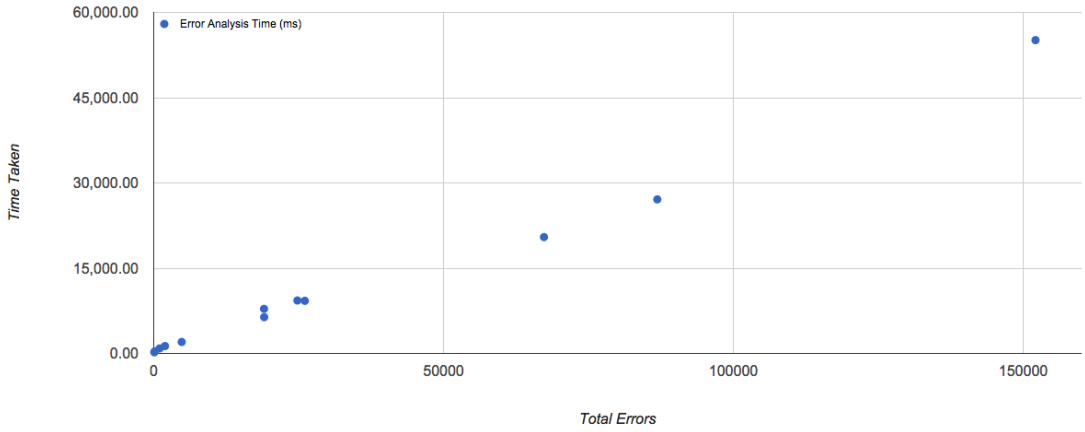

Figure 6.7: Number of *errors* in dataset against Time taken to do the error-checking process. Cf. Figure 6.6

Dataset Source	Rows parsed (or n)	Error Analysis Time (ms)
testrawdata1	92	380.95
testrawdata3	469	876.69
testrawdata4	895	1,296.33
newrawdata (limited to 2000 rows)	2000	2686.47
newrawdata (limited to 5000 rows)	5000	5604.19
newrawdata (limited to 10000 rows)	10000	9899.31
newrawdata (limited to 20000 rows)	20000	15707.83
newrawdata (limited to 30000 rows)	30000	20423.39
newrawdata (limited to 40000 rows)	40000	25231.00
newrawdata (limited to 45000 rows)	45000	38553.63
newrawdata (limited to 55000 rows)	55000	48826.41
newrawdata (limited to 60000 rows)	60000	52610.71
newrawdata	69349	55,124.35

Table 6.3: Comparative analysis of number of rows and time taken for error analysis.

The test was conducted by extracting the first n rows from the large *newrawdata* dataset. *n* was set to predetermined numbers to accurately map varying dataset sizes and the time it would take to check them for errors. Adding the LIMIT clause in the MySQL data retrieval command, the data was checked for errors and the results of the time taken are listed in Table 6.3. When plotted in a scatter plot, some deductions can be made regarding the relationship between the number of rows and error checking time.

This scatter plot is shown in Figure 6.8. It was initially thought that the higher values would be brought down by the application process i.e. the higher the number of rows being processed, the amount of time spent per row would

decrease (this was deduced from visual analysis of another chart shown in Figure 6.9 where the error parsing rate is clearly declining with size). However this was not the case, and the relationship between the two axis is more nuanced. A sixth degree polynomial curve is the best fitting curve for the data points. The “lumpy” nature of the curve could not be explained, and this needs further analysis as to what the precise cause of this is.

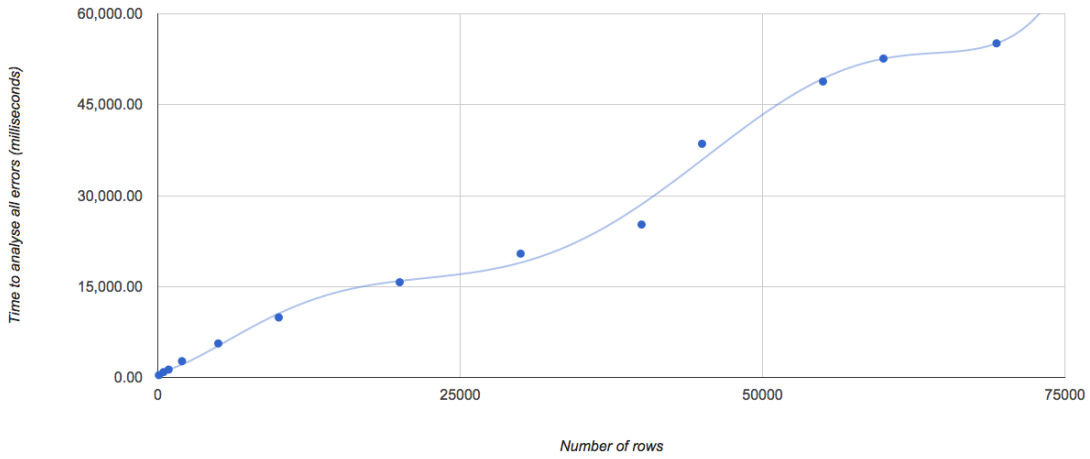

Figure 6.8: Scatter plot analysing time taken for error analysis for varying sizes of the *newrawdata* dataset

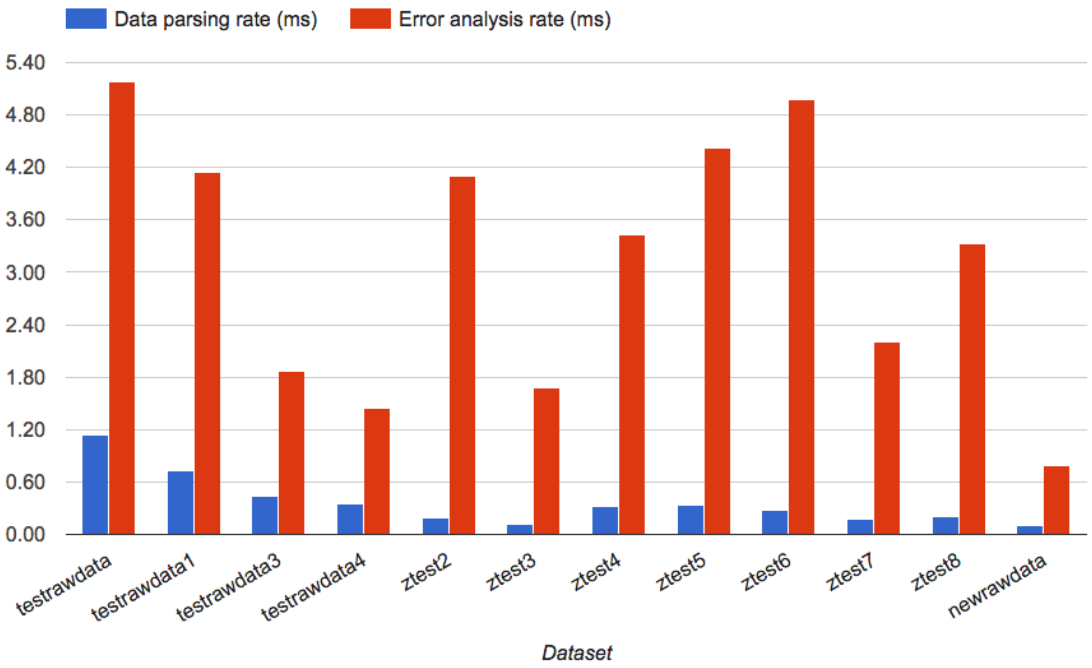

Figure 6.9: Graph comparing the data parsing rate and error analysis rate for each dataset. *ztest6* and *testrawdata* are comparable in the time it takes to check the errors in the rows.

6.3.2 Evaluation of GUI

The GUI has been discussed and numerous screen-shots have been provided in previous chapters, repetition of the same won't be necessary here. However, to evaluate the tool, a behaviour test of how it processes incoming data might be beneficial. For this scenario, if the *newrawdata* dataset is selected as the data source in the GUI, the stages the data goes through from parsing to generation of error metrics can be tracked to see if the behaviour is as desired. Figure 6.10 shows the first three lines of this dataset, visualized with font styles and colours for emphasis.

```
215741  17  data/Admin/Archive/17/ATC_airfareBookings_20140101.csv  AIRTRIP
actualFare,branchID,conjTickNum,countryCode,currencyCode,BusLei,eTkt,FlightNo,hotelTag,IATA,i
nvDate,LowestLogicalFare,netInd,numSegs,OandD,Origin,Destin,pCarrier,pCarrierNum,resDate,se
gArriveDate,segCarrier,segClass,segDepartDate,segFare,segFareBasis,segFlightNum,surcharges,s
urcharges,taxes,tickNum,tourCode,tripClass,contracted,ancil,Exchange P/R,Exchange
A/C,AirTrip/Refund,clientIdentifier
--
215742  17  data/Admin/Archive/17/ATC_airfareBookings_20140101.csv  AIRTRIP
30,6,,GB,GBP,not set,Y,1442,645488,91278040,01/01/2014,0,not set,1,not
set,LHR,EDI,BA,125,01/01/2014,02/01/2014,BA,K,02/01/2014,30,,1442,,,0,4631677295,not
set,Y,not set,not set,not set,not set,AirTrip,10001092
--
215743  17  data/Admin/Archive/17/ATC_airfareBookings_20140101.csv  AIRTRIP
698.2,6,,GB,GBP,not set,Y,5109,645488,91278040,01/01/2014,733.11,not set,1,not
set,DXB,LHR,EK,176,01/01/2014,02/01/2014,EK,E,02/01/2014,698.2,,5109,,,13.2,4631677296,not
set,Y,not set,not set,not set,not set,AirTrip,10001092
--
```

Figure 6.10: An extract of the first three lines from the *newrawdata* dataset.

The text in red is the ProviderID. The text in purple colour is the blockType. The text in blue is the actual raw data. If the user has selected the AIRTRIP blockType as the data type to be parsed in the GUI, these rows of the data will be processed, otherwise they will be ignored. For this particular dataset, the first row of the raw data contains the attribute headers. The position of the attribute headers within the string indicates the type of attribute based on the index of sub-strings in subsequent rows of data.

The program will use the first raw to help the user map columns according to their attribute characteristics. For instance, row 2 can be deciphered using

row 1 as a guide, and likewise with row 3 and so on. Table 6.4 shows this breakdown effectively.

Of the 39 attributes used by this dataset, only 19 are important attributes (as identified by the client a priori). These 19 attributes are selected in the dataset by storing the index where it is located. This set of attributes (called a mapping) is saved in another table, able to be retrieved next time another dataset from the same agency is loaded. The retrieval and setting of saved mappings is done in Java through HashMaps, which are an efficient data structure to hold key-value pairs.

Each key attribute has some characteristics that need to be checked against the characteristics of the data in the raw data. For instance, the key attribute **invDate** has the following characteristics:

- **maximum length of field**: 12
- **format**: Date
- **minimum size**: 6
- **allowable range**: 01/01/2014 - current date
- **Check for duplicates?** False
- **Check for contradiction?** None
- **Calculate tally for column?** False

The data in the row is stripped and parsed into a two dimensional String array. The program then iterates through the array and checks each value against the characteristics described above.

The Syntactic errors like Missing Values and Format errors etc. are done on a cell by cell basis as the program iterates through the 2D array. In the data extract table, for instance, row attributes which contain empty columns (“ ”) or “not set” will be treated as a Missing value, and syntactic errors will be flagged for all such cases. Semantic errors like contradictions and duplicates are done on a row by row basis. Before the array is iterated through, the program stores the list of columns names which need checking for contradictions. During the iteration loop, the program checks the current

index	row 1 data	row 2 data	row 3 data
1	actualFare	30	698.2
2	branchID	6	6
3	conjTickNum		
4	countryCode	GB	GB
5	currencyCode	GBP	GBP
6	BusLei	not set	not set
7	eTkt	Y	Y
8	FlightNo	1442	5109
9	hotelTag	645488	645488
10	IATA	91278040	91278040
11	invDate	01/01/2014	01/01/2014
12	LowestLogicalFare	0	733.11
13	netInd	not set	not set
14	numSegs	1	1
15	OandD	not set	not set
16	Origin	LHR	DXB
17	Destin	EDI	LHR
18	pCarrier	BA	EK
19	pCarrierNum	125	176
20	resDate	01/01/2014	01/01/2014
21	segArriveDate	02/01/2014	02/01/2014
22	segCarrier	BA	EK
23	segClass	K	E
24	segDepartDate	02/01/2014	02/01/2014
25	segFare	30	698.2
26	segFareBasis		
27	segFlightNum	1442	5109
28	surcharges		
29	surcharges		
30	taxes	0	13.2
31	tickNum	4631677295	4631677296
32	tourCode	not set	not set
33	tripClass	Y	Y
34	contracted	not set	not set
35	ancil	not set	not set
36	Exchange P/R	not set	not set
37	Exchange A/C	not set	not set
38	AirTrip/Refund	AirTrip	AirTrip
39	clientIdentifier	10001092	10001092

Table 6.4: Initial stage of data parsing is shown here. The dataset sample from figure 6.10 is split as shown in this table, with row 1 being used as header information.

column name against the stored list of columns (stored in a hashmap). If there is a match, the entry in the current cell is stored. After the row is iterated through, a comparison is made between the contradiction check columns, and if there is a contradiction, then an error is generated.

For the duplicate checks, a HashSet is used because of its feature of holding unique values. If multiple columns have been indicated to be unique in the *attributeDescriptor* table for the current dataset's blockType, then those column values are concatenated in the same order and then added to the HashSet. This is repeated on each row. If a particular row's concatenated value cannot be added to a HashSet, it is because it is a duplicate value being rejected by the Set data structure and then an duplication error is generated.

These characteristics which are checked against (duplicate values, contradiction check values etc.) are retrieved from the *attributeDescriptor* table. If there is a mismatch between expectation and actual value, then an entry is added to error logs (saved in MySQL). These error logs are then checked by the report card generator to create the summary visualization.

The GUI has been tested extensively against provided datasets and works to specification. All the requirements laid out have been met successfully. However, for an application dealing with custom datasets, there are always improvements that can be made. The author recognises the ever changing nature of data, and the need for a data scientist to constantly interact with the data sources and tweak the application code accordingly.

One of the suggested areas of improvement is the automatic reading of header information from the first row of the data. This would be a particular time saver, as it would save the client time spent clicking through dozens of drop-down boxes each time a new agency sends a dataset. If the program were able to search for certain keywords and map them as a suggestion (which could be tentative, needing ratification from the user to be confirmed), this would make the process of data parsing more efficient.

6.4 Report Card

In the previous section, a sample dataset (*newrawdata*) was loaded into the program, and its operations on the data were discussed. The report card that is generated for this dataset is shown in Figure 6.11. Five distinct regions can be seen in the program, each of which describes a different dimension of the data health measure.

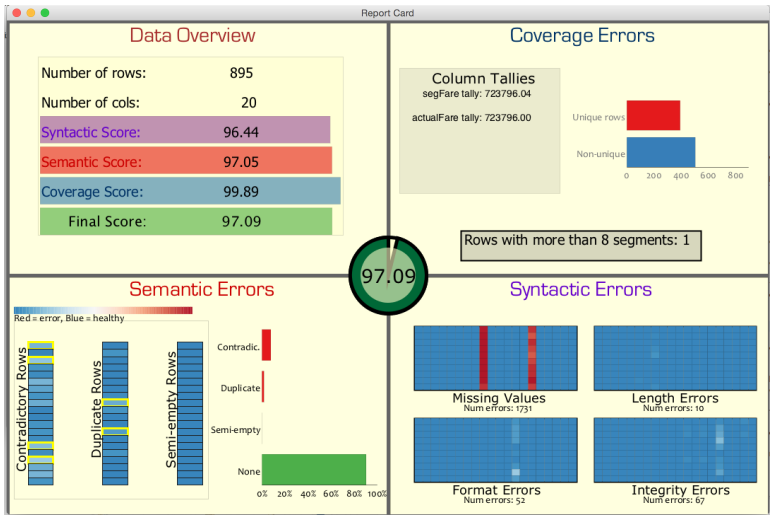

Figure 6.11: Report card of *newrawdata* dataset, generated after the data parsing and error checking process.

A key objective of the project was to show visually to the user how error prone the data was. This is done by locating the errors spatially and by showing the quantity of the error-filled cells compared to the quantity of the error-free cells. The visualization achieves this very well in its rendering of grids to show spatial information, and using aggregate statistics to show quantities.

The change in direction from a simple static report card to a report card with interactive abilities was justified to give more detail where it was lacking. Using aggregates in the Semantic Error grid visualization failed to convey effectively where the most errors were. The presence and relative quantity of errors were apparent. However, regarding the location of the rows with the most errors, which was to be the top priority if the dataset were to be corrected, there was little help from the visualization.

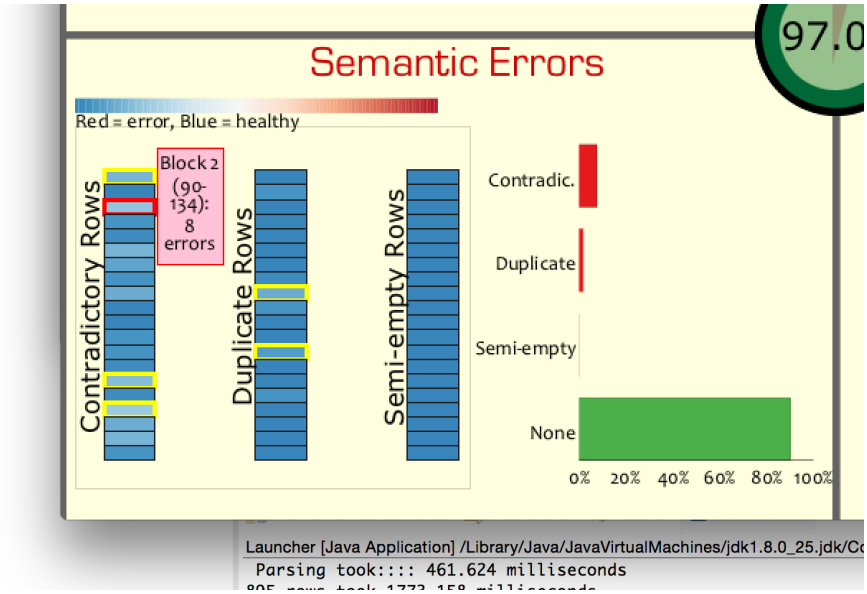

Figure 6.12: Zoomed in view of the interactive abilities in the Semantic Errors section. Hovering over the segments with yellow bounding boxes shows an additional box that contains information about the rows that the segment applies to, and the total number of errors contained within.

The addition of bounding yellow boxes (Figure 6.12) to show the presence of the top 20% of errors, was crucial in gleaning a key trend from the data. The user can gain additional information about the quantity of errors in the worst error-filled areas by hovering over the yellow boxed rows.

A drawback of this approach was that this additional information, shown only through interaction with the tool, is not available for viewing to the originator of the dataset. When the client sends the report card, digitally or by print, this interactive component will not be present. The provider will see the bounding boxes, showing the location of the most error-prone rows, but additional information regarding the density of errors will not be as apparent.

For datasets with an uneven spread of errors, i.e. blocks of contradictory rows followed by blocks of error-free rows, this will not be a problem. The user can glean error information accurately from the error colour-gradient. However, for datasets that contain errors spread evenly across the spectrum of rows, the lack of this additional information restricts the reading of errors.

On the other hand, it could be argues that for datasets with an even spread of errors, particular areas do not need to be highlighted, since the problem is not restricted to certain areas, but throughout the data.

6.4.1 Effectiveness of Presentation

Comparing against a dataset with a huge quantity of errors (*ztest3*, shown in Figure 6.13), an analysis can be made of the report card's effectiveness. The quantity of errors present in this dataset can be seen from Table 6.2. Despite the presence of 86,875 errors in the dataset, the report card shows a rating of 97.02 for this dataset. This might seem an inaccurate representation, but closer analysis reveals a methodical basis. There are only 22 semantic errors in the 16170 rows analysed for three different types of errors (contradiction, semi-empty and duplicate checks). This reveals an error rate of 0.0045%. For the syntactic errors section, there are 86,852 errors in total. However this is after four syntactic checks are done on each cell resulting in $16170 \times 20 \times 4$ (1,293,600). This results in an error rate of 6.71 %.

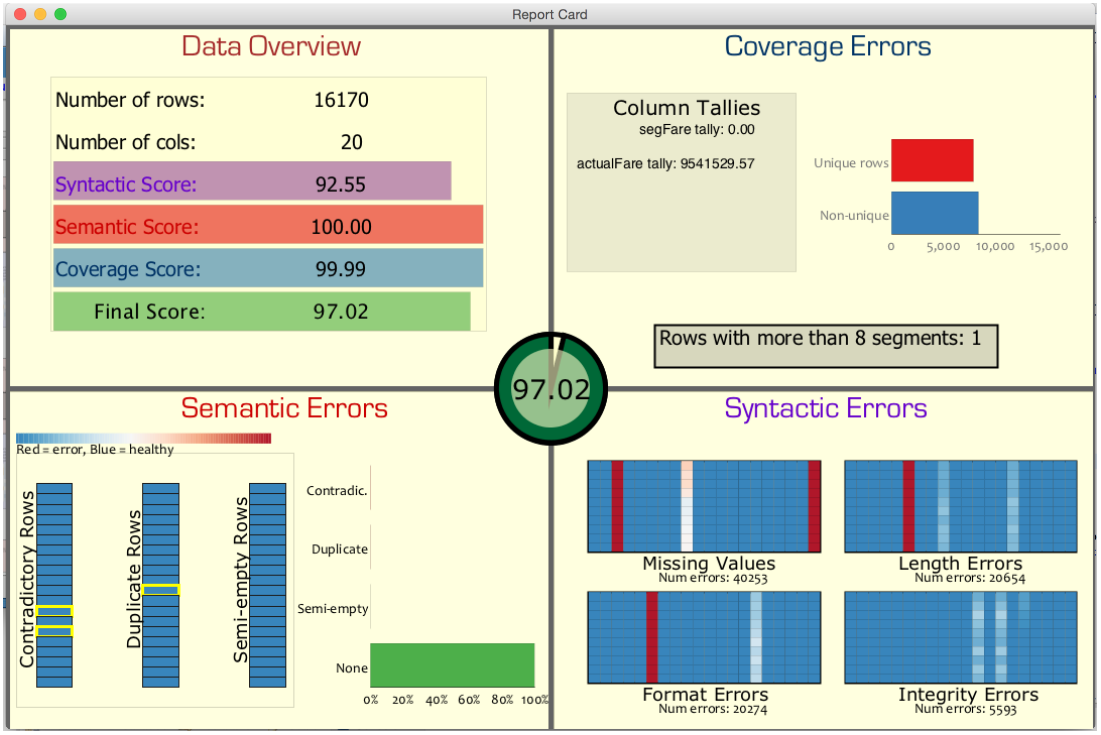

Figure 6.13: Report Card for *ztest3* dataset]. Despite the presence of 86,875 errors in the dataset, the report card shows a rating of 97.02 for this dataset. This might seem an inaccurate representation, but closer analysis reveals a methodical basis.

Despite the difference in the quantity of errors present in *testrawdata4* dataset and *ztest3*, the similarity in the error rate and the derivation of the scores shows the errors in context, grounded in the size of the dataset and the amount of non-error values.

The visualization accurately depicts the error status summary of the entire dataset without being overwhelmed by the quantity of errors thanks to the aggregate measures used. Over-plotting does not occur regardless of the size of the dataset. The summary of the number of errors below the syntactic grids, provides context by highlighting the quantity too. Thus the visualization is very effective in achieving its objective.

A drawback of the tool is apparent in the categorization of coverage errors alongside Semantic and Syntactic errors. The statistics generated in the Coverage Errors section are not universally applicable to all incoming data that the client parses, but only for AIRTRIP blockType data. This means that for non-AIRTRIP blockTypes, some of the statistics will not be displayed, leaving the section with a high amount of blank space and looking very bare. The very specific and niche nature of the errors generated here, for instance, the number of non-unique ticket numbers, differentiates it significantly from the errors in the Semantic and Syntactic error section.

An area of improvement for future iterations of the project could be to provide an alternate categorization method that effectively differentiates between specific statistics (Coverage Errors and Data Overview) and data error metrics (Semantic Errors and Syntactic Errors).

6.5 Client feedback

An effective appraisal of the system would not be complete without feedback from the client regarding their thoughts on the outcome of the implementation. This is especially the case in this project since the visualization tool has been built around the needs and requirements of the client, keeping in mind their

critique and thought processes at each stage of the iterative development process. The client was actively involved in the design stages as well, in part due to the open nature of FDS based designs.

6.5.1 System Usability

A Usability study was conducted to collect the client's feedback on the system. Usability, being a subjective concept [5], is still a key component in assessing the program's *appropriateness to a purpose*. The System Usability Scale (SUS) is a "quick and dirty" method of assessing usability.

The SUS is a easy to use survey that outputs a single score on a scale that is easily understood. It has been well researched over the 19 years since its inception [3]; research that can be used to analyse the score from a SUS undertaking and puts it in context of how the score reflects on the usability of the system.

The original questions from the SUS instrument were modified slightly by Bangor et al. These modifications were made to clarify terms and word usage that was potentially affecting the scores. However for the usage for this thesis, the original questions were used verbatim because the justification for the changes by Bangor et al. were not applicable to the current system's usability checking. The questions in the survey and the scoring are given in Table 6.5, with responses from the client recorded there as well.

In according with Brooke's [5] stipulations for using the SUS, it was used after the client had an opportunity to use the system being evaluated, but before any discussion had taken place. The client was asked to record their immediate response to each question, rather than deliberating over the question. All questions on the scale are mandatory; if the client feels they cannot answer a question, they should mark the centre point for that question.

1	I think that I would like to use this system frequently				X		
		1	2	3	4	5	
2	I found the system unnecessarily complex			X			
		1	2	3	4	5	
3	I thought the system was easy to use.				X		
		1	2	3	4	5	
4	I think that I would need the support of a technical person to be able to use this system			X			
		1	2	3	4	5	
5	I found the various functions in this system were well integrated				X		
		1	2	3	4	5	
6	I thought there was too much inconsistency in this system		X				
		1	2	3	4	5	
7	I would imagine that most people would learn to use this system very quickly.		X				
		1	2	3	4	5	
8	I found the system very cumbersome to use				X		
		1	2	3	4	5	
9	I felt very confident using the system		X				
		1	2	3	4	5	
10	I needed to learn a lot of things before I could get going with this system		X				
		1	2	3	4	5	

Table 6.5: Original SUS questions
Original SUS questions, used verbatim. The client's answers are given on the right. Key: *1 = Strongly Disagree, 5 = Strongly Agree*

In order to remove any bias in the testing process, the SUS was conducted with minimal commentary from the author. The client was informed that they were required to use all aspects of the program, trying out the program on multiple datasets. They were instructed to complete the short survey as soon as they had finished testing the program. No other communication took place. As soon as the client started using the program to test it, the author left the immediate vicinity to ensure no pressure was felt by our presence; communication was ceased and the client was left in silence to complete the task at hand.

The marking of the SUS were done as instructed in the original paper:

> To calculate the SUS score, first sum the score contributions from each item. Each item's score contribution will range from 0 to 4. For items 1,3,5,7,and 9 the score contribution is the scale position minus 1. For items 2,4,6,8 and 10, the contribution is 5 minus the scale position. Multiply the sum of the scores by 2.5 to obtain the overall value of SU.

Based on the client's answers, a SUS score of **55** was achieved. This is considered a below average score. It was decided based on this result, to

increase the usability of the program. A number of changes were introduced in the program including minimising use of pop-ups and number of options on the Dataset loading page to help the user navigate in a clear logical manner etc.

There were other inconsistencies in the initial implementation that were addressed, to increase its usability. For instance, the program relied on mouse clicks and buttons throughout its various screens to check user choice. However, this approach was not present in the initial report card viewer interface, where the user was required to double click on the currently selected report card to view it in an external viewer.

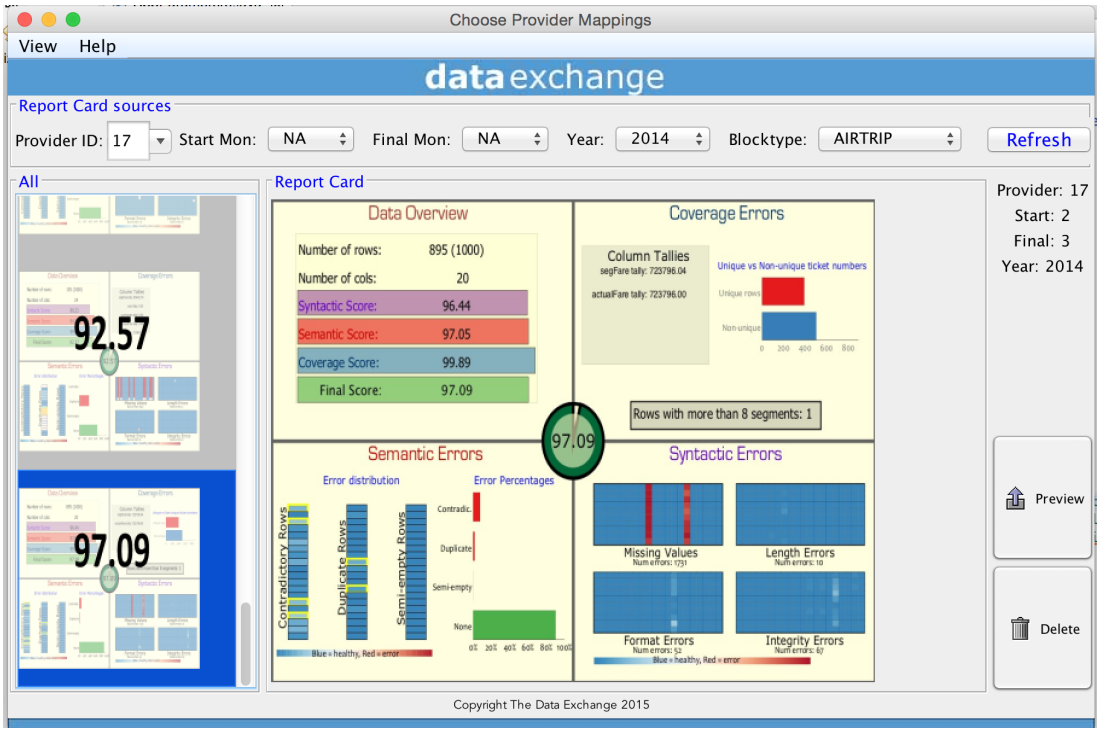

Figure 6.14: The updated report card viewer interface, showing the addition of the right panel. The panel contains report card metadata, and manipulation controls, introduced for usability reasons.

This was considered a key usability culprit, and buttons were implemented on this interface to address this issue. Similarly, when generating a report card, in the initial implementations the user was required to press a key 'S' to save a report card. On later reflection and in light of the usability score findings, it was changed to allow automatic saving of report cards and remove this cumbersome additional keyboard shortcut. A delete button was added to the viewer interface to allow the user to remove unwanted report cards

1	I think that I would like to use this system frequently				X		
		1	2	3	4	5	
2	I found the system unnecessarily complex		X				
		1	2	3	4	5	
3	I thought the system was easy to use.				X		
		1	2	3	4	5	
4	I think that I would need the support of a technical person to be able to use this system			X			
		1	2	3	4	5	
5	I found the various functions in this system were well integrated				X		
		1	2	3	4	5	
6	I thought there was too much inconsistency in this system		X				
		1	2	3	4	5	
7	I would imagine that most people would learn to use this system very quickly.			X			
		1	2	3	4	5	
8	I found the system very cumbersome to use		X				
		1	2	3	4	5	
9	I felt very confident using the system				X		
		1	2	3	4	5	
10	I needed to learn a lot of things before I could get going with this system	X					
		1	2	3	4	5	

Table 6.6: Results from second SUS
Results from the second Usability study. Key: *1 = Strongly Disagree, 5 = Strongly Agree*

(Figure 6.14). Finally, a tab order was implemented to enable the user to cycle through the key buttons and components on the screen using the Tab key for additional navigational help.

A second SUS was done with the client after these changes were made. The results are published in Table 6.6. The final SUS score was **72.5**. This is considered a very good increase in usability. A score of more than 70 is considered a satisfactorily usable program. Based on feedback received from open-ended questions asked to the client, they noted finding the interface improvements made the program more intuitive: confirming the SUS score empirically.

6.6 Summary

ISO 9241-11[18] is an important standard that deals with usability of a system. It defines usability as "*the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use*".

Usabilitynet.org provides some helpful questions that help address each aspect of this definition:

> Effectiveness - can users complete tasks, achieve goals with the product, i.e. do what they want to do?
> Efficiency - how much effort do users require to do this? (Often measured in time)
> Satisfaction – what do users think about the products ease of use?

By carrying out a SUS, this project has provided demonstrable satisfaction from the client. The efficiency was studied in depth, with time-taken by the system a key consideration right from the start of the project. This chapter has also demonstrated the effectiveness of the program in addressing the aims and objectives set at the start of the project. One of the objectives, of making direct comparisons between report cards was not achieved due to complexity of the task. But aside from this task, the other requirements have been met to a sufficient degree.

Chapter 7

Conclusions

7.1 Achievements

A reflection on the aims and objectives set as part of the research plan, and the outcomes achieved reveal a satisfactory result. The project can be said to have been successful and achieved a significant portion of the desired goals.

The raw data was sufficiently analysed in light of the paradigms highlighted in existing literature for querying data quality issues. An empirical approach was used to ensure that the client's requirements were adequately represented in the health score generation. Numerous tests on various datasets with different levels of errors show that the program was successful in highlighting the errors that the client wanted to be highlighted.

Several visualization techniques were used to show the summary report card, in particular the use of aggregate measures to ensure over-plotting was not a problem when plotting large datasets. Regardless of the size of raw data, the report card would remain aesthetic and readable to the human viewer. The use of agile methodology enabled the design, implementation and testing process to be carried our seamlessly and iteratively, making changes in each stage where and when necessary. Using the FDS design methodology, a satisfactory design output was generated in collaboration with the program's chief consumer, the client.

Timeliness was an objective that was a key priority during the implementation stages. Initial prototypes, achieved the desired results, but were expensive in terms of time taken. This was addressed to ensure that the product was usable in a business environment, where time is of the essence. Efficient ways to make database calls and reducing number of database operations was key in achieving this optimization.

The report card generation required several parameters that varied with each dataset. This meant that client input had to be taken at each dataset import iteration using user input (either via keyboard or mouse). Using a GUI designed in Java, various Swing components like ComboBoxes and buttons were used to render a usable user-intuitive interface. The usability of the whole system in general, and this interface in particular was tested using an SUS study and found to be highly satisfactory. Verbal feedback confirmed this.

The initial objective of the ability to work with and differentiate between multiple datasets from multiple providers, with varying blockTypes was also achieved. Using the paradigm of “provider mappings”, each provider’s providerID was used in combination with the blockType as a unique identifier in each dataset’s list of columns. The raw data was parsed and stored in the appropriate attributes, using the index allocation of the header string provided by the user in the GUI interface.

A viewer interface was also implemented to enable the user to view previously generated report cards. The interface allowed the user to search report cards by providerID, blockType, timeframe of data represented etc. Some comparison between report cards was also provided by the use of Final Score information overlaid on top of the report card thumbnails, viewed in a scrolling list.

This project is an excellent demonstration of the agile methodology in practice. It’s benefits were fully utilised, and possibly crucial to the outcome that was achieved. A different methodology would have resulted in a inferior final

product because the client's input wouldn't be sufficiently utilised. The reader is referred to the *Introduction* chapter and see the objectives envisioned for the project to assess how well the project hypothesis has been met. Objectives 1 and 2 were successfully implemented, as shown in Chapter 4. Objectives 3 and 4 were demonstrated successfully in Chapter 5.

7.2 Limitations & further work

The program's objective of providing the ability to make comparison between data health report cards was not fully implemented. The presence of a report card viewer interface, provided some means of indirect comparison but a side-by-side comparison is not present. This is something that needs to be introduced as a priority in future iterations.

Initial implementations provided a way for the user to load CSV and Excel files as an alternative data source and to check for errors in them. However, this was discarded in later implementations due to the need to use a MySQL database for dataset configurations and mappings settings. Discussions with the client showed that the file loading feature was not a necessity due to their sole reliance on MySQL as the data storage architecture. In future versions, file loading can be reintroduced if client requirements expand.

For debugging, a JTable was added to the program GUI. This was for the author's benefit, to check if the data that was loaded and parsed from the MySQL raw data source was satisfactorily parsed into their respective attributes as per the mappings set by the user. However, in future versions, this JTable interface could be shown in the program GUI for the user to check if the correct dataset has been loaded, and to see the parsed contents of the data themselves. Due to concerns of complexity (a concern that more features being added without addressing a direct need), this was left out of the final product; but can be easily introduced with a few additional changes.

This would require care so as not to affect usability, and additional SUS studies may be required to ensure this.

To get the most out of the program, the user needs to have visual access to the MySQL tables separately, either through the command line interface or through a GUI (the author used a program called *SequelPro*). Future iterations of the program should look at introducing more independence, so the user can view MySQL table information from within the program. Currently, only the list of tables and individual lines of the parsed raw data are visible, but with the introduction of the Table viewer interface and a panel for MySQL interaction, functionality will be greatly increased.

Appendix A

References

[1] M. Ankerst, ‘Visual Data Mining with Pixel-oriented Visualization Techniques’, *Proceedings of the ACM SIGKDD Workshop on Visual Data Mining*, 2001 (p. 23).

[2] D. P. Ballou and H. L. Pazer, ‘Modeling data and process quality in multi-input, multi-output information systems’, *Management science*, vol. 31, no. 2, pp. 150–162, 1985 (p. 18).

[3] A. Bangor, P. T. Kortum and J. T. Miller, ‘An Empirical Evaluation of the System Usability Scale’, *International Journal of Human-Computer Interaction*, vol. 24, no. 6, pp. 574–594, 2008, ISSN: 1044-7318. DOI: 10.1080/10447310802205776 (p. 99).

[4] C. Batini, C. Cappiello, C. Francalanci and A. Maurino, ‘Methodologies for data quality assessment and improvement’, *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–52, 2009, ISSN: 03600300. DOI: 10.1145/1541880.1541883 (p. 85).

[5] J. Brooke, ‘Sus-a quick and dirty usability scale’, *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996 (p. 99).

[6] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, T. Vo and H. T. Silva, ‘VisTrails : Visualization meets Data Management’, *2006 ACM SIGMOD iInternational Conference on Management of Data*, pp. 745–747, 2006, ISSN: 07308078. DOI: 10.1145/1142473.1142574 (p. 24).

[7] I. Chengalur-Smith, D. Ballou and H. Pazer, ‘The Impact of Data Quality Information on Decision Making’, *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 6, pp. 853–864, 1999, ISSN: 10414347. DOI: 10.1109/69.824597 (p. 19).

[8] Q. Cui, M. O. Ward, E. a. Rundensteiner and J. Yang, 'Measuring data abstraction quality in multiresolution visualizations', *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 709–716, 2006, ISSN: 10772626. DOI: 10.1109/TVCG.2006.161 (p. 23).

[9] T. Dasu, G. T. Vesonder and J. R. Wright, 'Data quality through knowledge engineering', in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2003, pp. 705–710 (p. 17).

[10] T. Davenport and D. Patil, *Data scientist: the sexiest job of the 21st century*, https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century/, [Online; accessed 20-July-2015], 2012 (p. 1).

[11] C. Eaton, C. Plaisant and T. Drizd, 'The Challenge of Missing and Uncertain Data (Poster)', pp. 2–3, 1991 (p. 23).

[12] J.-D. Fekete and C. Plaisant, 'Interactive information visualization of a million items', in *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, IEEE, 2002, pp. 117–124 (p. 23).

[13] H. Galhardas, D. Florescu, D. Shasha and E. Simon, 'Ajax: an extensible data cleaning tool', in *ACM Sigmod Record*, ACM, vol. 29, 2000, p. 590 (p. 23).

[14] H. Galhardas, D. Florescu, D. Shasha, E. Simon and C. Saita, 'Declarative data cleaning: language, model, and algorithms', 2001 (p. 19).

[15] H. Griethe and H. Schumann, 'The Visualization of Uncertain Data : Methods and Problems', *SimVis*, no. July, pp. 143–156, 2006 (p. 23).

[16] J. Harris, *Timeliness is the most important data quality dimension*, http://www.ocdqblog.com/home/timeliness-is-the-most-important-data-quality-dimension.html, [Online; accessed 21-August-2015], 2014 (p. 85).

[17] M. Harrower and C. A. Brewer, 'Colorbrewer. org: an online tool for selecting colour schemes for maps', *The Cartographic Journal*, vol. 40, no. 1, pp. 27–37, 2003 (p. 73).

[18] ISO, 'Iso 9241-11:1998 ergonomic requirements for office work with visual display terminals (vdt)s - part 11 guidance on usability',

International Organization for Standardization, Geneva, Switzerland, ISO, 1998 (p. 102).

[19] S. Kandel, J. Heer, C. Plaisant, J. Kennedy, F. V. Ham, N. H. Riche, C. Weaver, B. Lee, D. Brodbeck and P. Buono, 'Research directions in data wrangling: Visualizations and transformations for usable and credible data.', 2011, ISSN: 1473-8716. DOI: 10.1177/1473871611415994 (pp. 17, 23).

[20] S. Kandel, A. Paepcke, J. Hellerstein and J. Heer, 'Wrangler: Interactive Visual Specification of Data Transformation Scripts', *Human factors in computing systems. ACM*, pp. 3363–3372, 2011, ISSN: 1450302289. DOI: 10.1145/1978942.1979444 (p. 23).

[21] D. A. Keim, 'Pixel-oriented visualization techniques for exploring very large databases', *Journal of Computational and Graphical Statistics*, vol. 5, no. 1, pp. 58–77, 1996, ISSN: 10618600. DOI: 10.2307/1390753 (p. 23).

[22] D. A. Keim, H. Kriegel and M. Ankerst, 'Recursive pattern: a technique for visualizing very large amounts of data', *Proceedings Visualization '95*, pp. 279–286, 1995, ISSN: 1070-2385. DOI: 10.1109/VISUAL.1995.485140 (p. 23).

[23] W. Kim, B. J. Choi, E. K. Hong, S. K. Kim and D. Lee, 'A Taxonomy of Dirty Data', *Data Mining and Knowledge Discovery*, vol. 7, no. 1, pp. 81–99, 2003, ISSN: 13845810. DOI: 10.1023/A:1021564703268 (p. 22).

[24] J. D. Mackinlay, P. Hanrahan and C. Stolte, 'Show Me Automatic Presentation for Visual Analysis Background • Polaris : commercialized by Tableau Software', *Analysis*, 2007 (p. 24).

[25] S. Madnick, 'Improving Data Quality Through Effective Use of Data Semantics', 2003 (p. 22).

[26] H. Müller and J.-C. Freytag, 'Problems, Methods, and Challenges in Comprehensive Data Cleansing', *Challenges*, no. HUB-IB-164, pp. 1–23, 2003 (p. 22).

[27] P. Oliveira, F. Henriques and P. Henriques, 'A formal definition of data quality problems', *2005 International Conference on Information Quality*, pp. 1–14, 2005 (p. 22).

[28] K. Orr, ‘Data quality and systems theory’, *Communications of the ACM*, vol. 41, no. 2, pp. 66–71, 1998 (p. 17).

[29] L. L. Pipino, Y. W. Lee and R. Y. Wang, ‘Data quality assessment’, *Communications of the ACM*, vol. 45, no. 4, p. 211, 2002, ISSN: 00010782. DOI: 10.1145/505248.506010 (p. 21).

[30] E. Rahm, ‘Data Cleaning : Problems and Current Approaches’, *Informatica*, pp. 1–11, 2000 (p. 20).

[31] V. Raman and J. M. Hellerstein, ‘Potter’s Wheel: An Interactive Data Cleaning System’, *Data Base*, vol. 01, pp. 381–390, 2001, ISSN: 10477349. DOI: 10.1.1.39.2790 (p. 23).

[32] T. C. Redman, ‘The impact of poor data quality on the typical enterprise’, *Communications of the ACM*, vol. 41, no. 2, pp. 79–82, 1998 (p. 18).

[33] K.-U. Sattler and E. Schallehn, ‘A data preparation framework based on a multidatabase language’, in *Database Engineering and Applications, 2001 International Symposium on.*, IEEE, 2001, pp. 219–228 (p. 17).

[34] B. Shneiderman, ‘Extreme visualization: squeezing a billion records into a million pixels’, *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 3–12, 2008, ISSN: 07308078. DOI: 10.1145/1376616.1376618 (p. 23).

[35] J. Shore *et al.*, *The art of agile development*. " O’Reilly Media, Inc.", 2007 (p. 5).

[36] S. Skeels Lee, ‘Revealing uncertainty for information visualization’, *Information Visualization*, 2010 (p. 23).

[37] G. K. Tayi and D. P. Ballou, ‘Examining data quality’, *Communications of the ACM*, vol. 41, no. 2, pp. 54–57, 1998, ISSN: 00010782. DOI: 10.1145/269012.269021 (pp. 18, 19).

[38] G. Tejay, G. Dhillon and A. G. Chin, ‘Data quality dimentions for information systems security: A theoretical exposition (Invited paper)’, *Security Management, Integrity, and Internal Control in Information Systems*, no. 1995, pp. 21–39, 2006, ISSN: 15715736 (p. 22).

[39] A. Treisman, ‘Preattentive processing in vision’, *Computer vision, graphics, and image processing*, vol. 31, no. 2, pp. 156–177, 1985 (p. 7).

[40] R. W. Wang and D. M. Strong, 'Beyond Accuracy: What Data Quality Means to Data Consumers.', *Journal of Management Information Systems*, vol. 12, no. 4, p. 5, 1996, ISSN: 07421222. DOI: 10.2307/40398176 (p. 19).

[41] R. Y. Wang, 'A product perspective on total data quality management', *Communications of the ACM*, vol. 41, no. 2, pp. 58–65, 1998 (p. 18).

[42] Z. Wen and M. X. Zhou, 'Evaluating the use of data transformation for information visualization', *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1309–1316, 2008, ISSN: 10772626. DOI: 10.1109/TVCG.2008.129 (p. 18).