March 2016

Angelo Gordon: Loan Data Governance

Alexander K. Shoop
Worcester Polytechnic Institute

Benjamin John Sharron
Worcester Polytechnic Institute

Khazhismel Kumykov
Worcester Polytechnic Institute

Follow this and additional works at: https://digitalcommons.wpi.edu/mqp-all

Angelo Gordon: Loan Data Governance

Richard O'Brien
Ben Sharron
Alex Shoop
Khazhy Kumykov

Worcester Polytechnic Institute

Loan Data Governance

A Major Qualifying Project

Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfilment of the requirements for the

degree of Bachelor of Science.

Date: December 18th, 2015

Project ID: KMS-AG11.

Report Submitted to:
Kevin Sweeney, Jon Abraham, Michael Ciaraldi, Arthur Gerstenfeld, Huang Xinming

ABSTRACT

The purpose of this project was to build a quick, intuitive, customizable reconciliation tool for Angelo, Gordon & Co. The current tool was strong in functionality but weak in user interface. New reconciliations took weeks of setup and generated hard-to-read reports. Further, every report cost money. We worked closely with the sponsor to develop an extremely intuitive and user-friendly tool that is free to use, and only take minutes to create new, large reconciliations.

ACKNOWLEDGEMENTS

EXECUTIVE SUMMARY

During the course of business, Angelo, Gordon & Co. exchanges and creates large amounts of data regarding loan pools that they buy, sell, and transform. This means that data comes from third party vendors, who are buying and selling, and data is created internally due to transformations and financial maneuvers performed on these pools. As well, since these pools are so large, the transactions can take a lengthy amount time to complete. This means that values of the exchange can change between the time when the exchange happened and when the loans are actually received. With all these different sources of data, and no standardized means of entering the data across the board, as well as the time delay of actually acquiring product, mismatches in data of different types, whether it be name value or otherwise, are bound to occur.

To become more transparent, Angelo, Gordon & Co. has outsourced its official accounting to a third party firm. The company, though, still maintains its own accounting records, meaning that there is yet another outside source of data that the company must keep their books in sync with.

The current tool used to reconcile data is good for very detailed reconciliations, for example if the firm wanted to reconcile everything down to the penny. However, every report generated costs money because they require custom reports, the user interface is not very intuitive, and generating any new reports takes an abundance of time (on the order of weeks).

Our project was to create a simple, quick reconciliation tool that could be used for reconciling loan taxonomies. This tool could be run from the terminal using one command, and would check loan taxonomies between databases, then email out an error report to users notifying them of mismatches on multiple levels of data. The tool could also have a translation table input into it, so that it would not output already known and accepted differences among databases as well as to provide a different key mapping between sources. The input and output format was chosen to be Excel files, so that reports could be opened and manipulated easily by any member of the company. As well, the entire user interface was designed to be extremely intuitive, requiring virtually no learning curve, compared to the current confusing tool.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1   CHAPTER 1: INTRODUCTION

1.1   Current Process

Before understanding the process that Angelo, Gordon & Co. currently uses, it is important to note that the firm has outsourced its official accounting to a third party for the sake of transparency with clients. Though this is true, they still maintain internal accounting and this actually adds another outside source of data that the company reconciles with.

The current reconciliation tool the firm uses is called IVP Recon. It is a very detailed reconciliation tool useful for very accurate checks (for example reconciling loan values down to the penny) and has an abundance of functionality. It is a very useful tool if the firm wants to set up something like a cash reconciliation and then run the same reconciliation over and over for a long time. The image below shows only some of its functionality and gives an idea of how detailed this software can be.
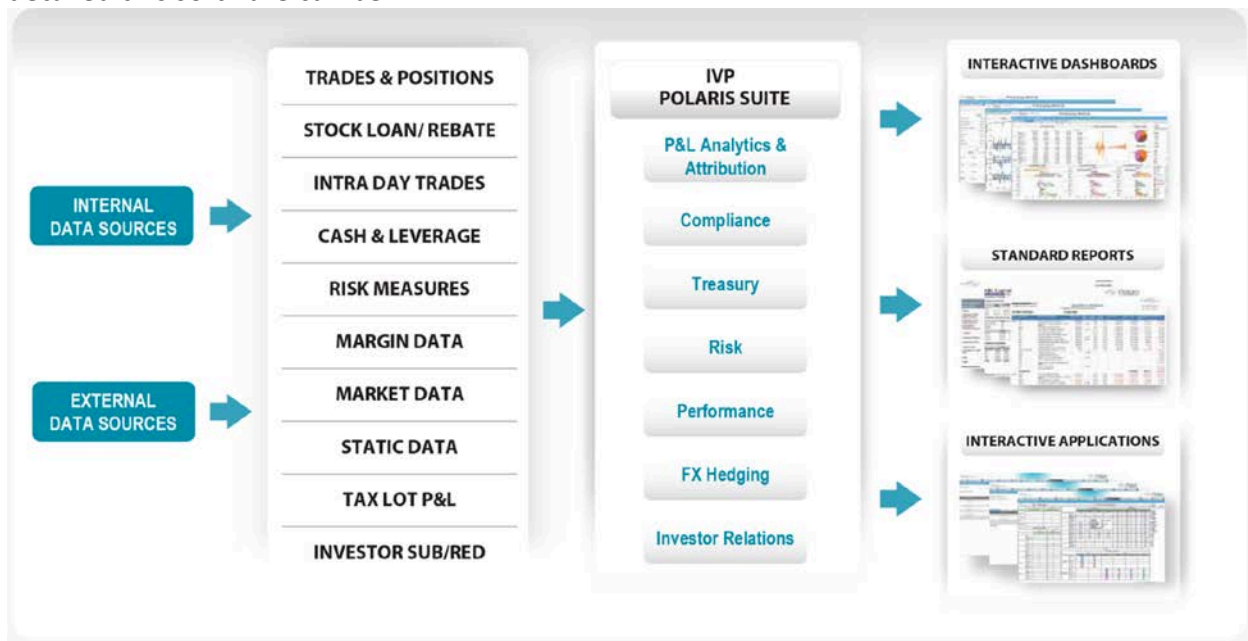
Figure 1: IVP Functionality

The tool has a few major issues: It can be very time consuming to set up new reconciliations, the user interface is not very intuitive and every new reconciliation costs a fee for custom reports.

The software takes time on the order of days simply to set up a new reconciliation. One of the main reasons for this is that IVP Recon does not allow for reconciliations on only select columns of database tables and on top of this, in one of the first steps, the program makes the user input all column pairs that are being reconciled. This single step alone takes days to complete and is generally outsourced to India. Due to the time difference, this means that this step in the

process alone won't be completed for roughly two days. After this lengthy step, there are many more steps that help define the functionality the user wants to be displayed in the report they receive, but all could be completed in a matter of hours. Overall though, by the time the new reconciliation is set up days have passed, potentially causing it to already be obsolete.

The user interface of the software also takes a long time to learn due to its complex nature. Due to the depth of the functionality of this tool, many problems arise with the user interface such as there being many sub-menus within sub-menus for each step. Since the software is also so precise as well as having a wide range of functionality, the user must enter specific sets of data in very specific places in order to accomplish the desired functions. For example if the user wanted to have "functionality 1" be in the output, there is a specific area perhaps within a specific menu of a specific step in the set-up process that pertains to "functionality 1". If the user doesn't know where this is, prepare for a lot of clicking and searching. Unfortunately the immense functionality of the software also leads to immense difficulty for the user entering data.

The final key issue that the firm has with this software is that every new reconciliation they generate costs a fee, because they have to have a custom report generated for them. The generic output file that the IVP Recon produces is not geared towards intuitive usability. The generic report outputs massive amounts of data regarding mismatches into an immense table with no obvious organization, no filterability, and a lack of most other details that would allow a user to easily understand what is going on. Thus the firm has a custom formatted report generated with every new reconciliation. This not only costs money, every single time a new reconciliation is needed, but also takes time because the software company needs to build Angelo, Gordon & Co. this new report. This may take time on the order of weeks, which further adds to the time issue.

Overall the current process works very well for very accurate reconciliations that they want to run repetitively for a long time. It is not suitable for smaller and frequently updated reconciliations that don't need pinpoint precision.

1.2   Problem Statement

Angelo, Gordon and Co. often wants to do quick checks of loan taxonomies between databases or Excel files to ensure that no information is being lost, entered incorrectly, or corrupted in some other way. Since there are both internal and external sources of data being input, these frequent comfort checks are very important. However if the firm wants a simple taxonomy check or aggregation check between loan pools on separate databases or Excel files, the current software is cumbersome taking a large amount of time to set up, having a complicated user interface, not allowing for reconciling of only certain columns of tables and costing money with each new report. The company needs a tool for quick, customizable, and easy reconciliations, which any user could perform. The tool also needs to be built so it is easily extended for other uses.

Simple examples of a taxonomy or an aggregation the tool may be used for can be seen below.
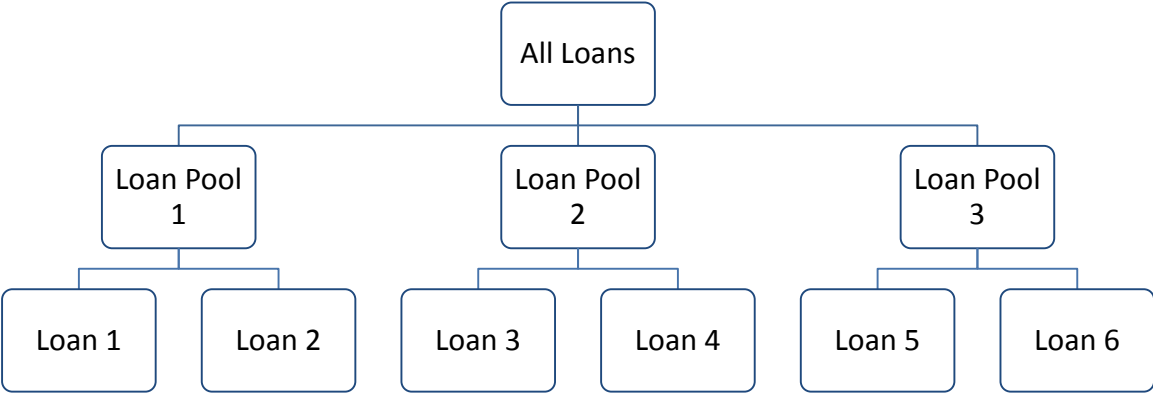
Figure 2: Simple Loan Pool Taxonomy
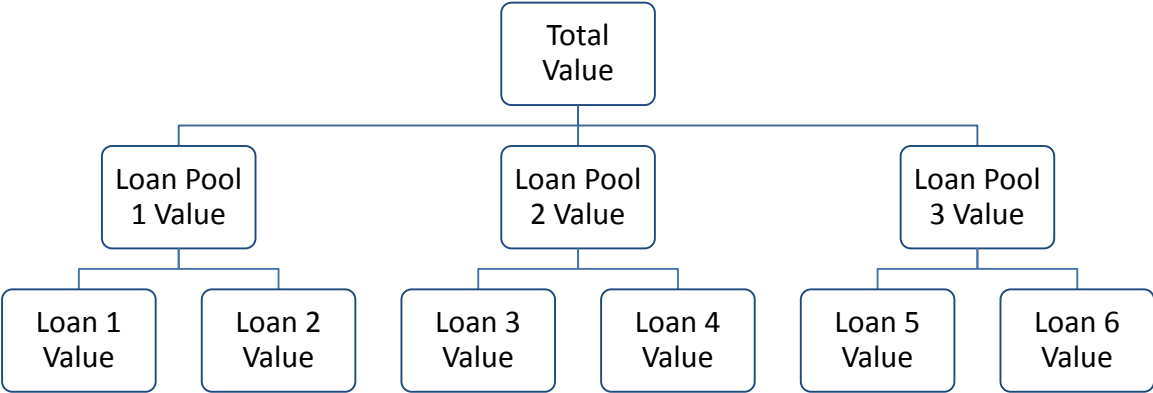
Figure 3: Simple Loan Aggregation

1.3   Quick Rec Tool Overview

The program our group built will be referred to as the Quick Rec Tool throughout this report. The Quick Rec Tool was built using Python along with downloaded packages such as openpyxl and smtplib. It also takes SQL statements as input to communicate with databases. The Quick Rec Tool can be run entirely from the command line. The run time is very short. In order to run a reconciliaton on two tables, each containing roughly 10,000 rows, the program only needs about five minutes. To avoid a user having to learn a complicated front end and navigation through menus, all the input and output files are Excel files. The Excel file is an ideal input because of its versatility, natural structure, and widespread use throughout Angelo, Gordon & Co.

1.4   Purpose

The project had a dual mandate, to deliver the functionality of a quick, customizable and simple reconciliation tool, and to deliver the tool such that it can be altered and built upon for further use.

- First and foremost, the tool's purpose was to be customizable. The reconciliations that were going to be run using the Quick Rec Tool were often not going to be on entire tables but only certain columns. There had to be an easy way for the user to enter this customization.

- The tool had to be intuitive, and all functionality and data had to be able to be altered simply, and by a non-programmer. Someone had to be able to sit down and easily figure the tool out.

- The Quick Rec Tool had to be powerful. Users had to be able to reconcile names or numbers as well as perform aggregations on columns, input translations tables, and more.

- The tool had to be quick. Generating reports needed to be done in seconds or minutes as opposed to days or weeks.

- The tool was built so it could be easily altered and expanded upon. This doesn't simply mean building flexible, and well organized code. This also meant thorough documentation of the program and input files in order to ensure that future users and builders know exactly what they need. We developed both video and written documentation.

2   CHAPTER 2: BACKGROUND INFORMATION

2.1   Python

"Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with duck typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance" (Long, 2014). Python's syntax is more similar to common English than some other popular languages, and its format forces appropriate indentation for coding, using this to erase the need for ending statements at the end of for loops, if statements, and logical structures of that nature.

"Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed" (Long, 2014). The project conducted by this team used numerous packages to perform functions specifically designed for interactions with other software such as Excel and email.

Openpyxl was an open source Python package designed to allow for easy interaction with Excel documents. The package allows opening, reading, writing, editing and more.

Smtplib is another open source Python package that allows for easy interaction with the SMTP email protocol from a Python program. This package, as well as MIME, was used to send emails to end-users and attach output reports to those emails.

There are also many modules within Python such as re, which can aid in performing smaller functions. For example the module re allowed the program to parse input using regular expressions.

2.2   SQL

SQL stands for Structured Query Language and it is a computer language used to communicate with databases (sqlcourse.com, 2015). "SQL is a common language used for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems (DBMS) that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc." (sqlcourse.com, 2015).

A relational database system contains objects called tables. The data or information for the database is stored in these tables. "Tables are uniquely identified by their names and are comprised of columns and rows. Columns contain the column name, data type, and any other attributes for the column. Rows contain the records or data for the columns" (sqlcourse.com, 2015).

2.3   Hedge Funds

"Hedge funds are alternative investments using pooled funds that may use a number of different strategies in order to earn active return, or alpha, for their investors" (Hedge Funds Investopedia, 2015). They are typically far smaller than banks and other financial services companies, with a large fund having employees numbering in the hundreds as opposed to the thousands. Hedge funds cater to high net worth clients, whether that be individual investors or corporate. In order to invest in a hedge fund as an individual one must have greater than $1 million in net worth, and have greater than $200,000 annual income.

"Hedge funds may be aggressively managed or make use of derivatives and leverage in both domestic and international markets with the goal of generating high returns. Because hedge funds may have low correlations with a traditional portfolio of stocks and bonds, allocating an exposure to hedge funds can be a good diversifier" (Hedge Funds Investopedia, 2015). Hedge funds are also largely unregulated, though they can be considered large players in some markets.

The term hedge fund is a very broad and flexible term that covers a wide range of investment management styles. The basic idea of the whole industry though is that they invest other peoples' money using strategies designed to create an absolute return regardless of the direction of the market.

2.4   Angelo Gordon

Angelo, Gordon & Co. is a privately held registered investment advisor and hedge fund headquartered in New York City, NY with tens of billions in assets under management. Though headquartered in New York, Angelo, Gordon & Co. has locations all over the globe, in Los Angeles, Hong Kong, London and more. (Angelo Gordon, 2014) The firm may appear small compared to a large bank, yet with over 360 employees they are large with respect to most hedge funds.

A hedge fund is essentially a company that invests on behalf of high net-worth individuals and other high net worth clients. They are largely unregulated due to the fact that they cater to sophisticated investors.

The company is dedicated to alternative investing, meaning that they seek to generate an absolute return regardless of the direction of the market by exploiting inefficiencies in selected

markets and capitalizing on situations that are not in the mainstream of investment opportunities. The markets they focus on are the Credit, Real Estate, and Private Equity.

A major part of the business model is buying large pools of loans, the size of which could never be purchased by any individual investor (think hundreds of millions of dollars), and then splitting, packaging, or performing some other function to them that allows the firm to sell these loans for a maximized value and a profit. The following figure shows the process of securitization of a loan. Hedge funds like Angelo, Gordon & Co. buy in the capital markets.
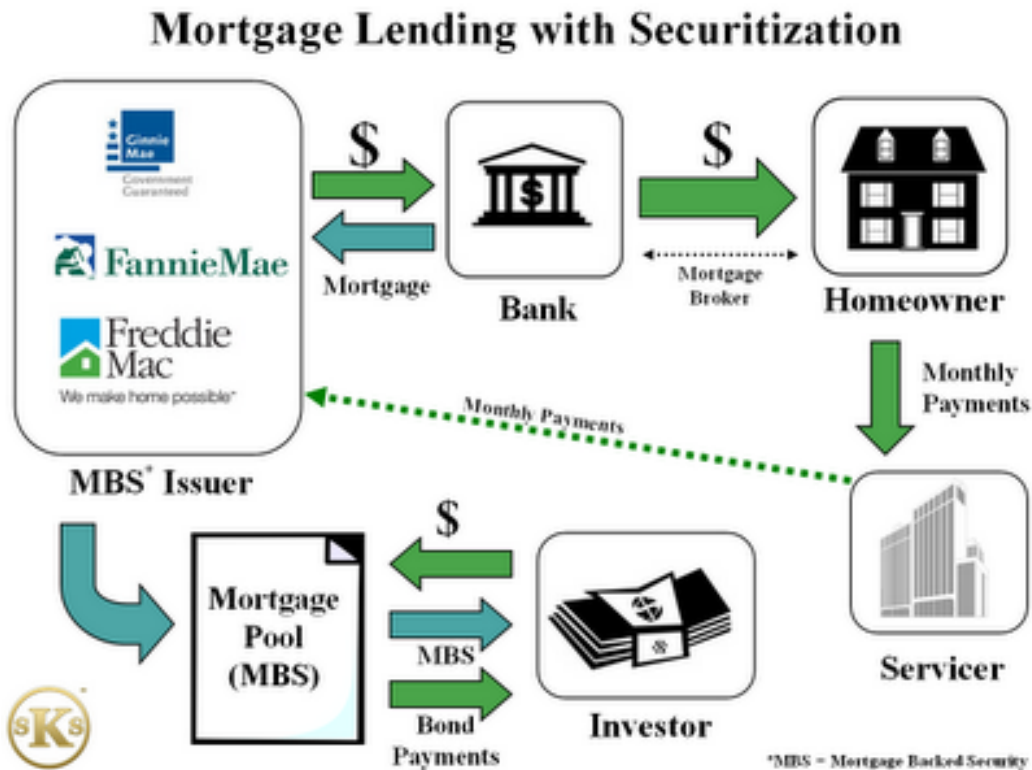
Figure 4: Securitization of Loans

Below is another illustration of where hedge funds are involved in this specific market. This diagram specifically also references what happens when the system breaks down as it did during 2008. Notice where it says "Hedge funds stop buying."

Crunch time

During the recent turbulence, the markets broke down in several areas that were stops in the complicated journey a home mortgage can make from original issuer to ultimate buyer.

Issuers stop buying

ABCP Issuers

Buyers: Institutional Investors, other

Buyers: Hedge funds, other aggressive investors

Institutional Investors stop buying

CDO

Senior

Mezzanine

Equity

MBS Issuers
Fannie and Freddie, and private label

CDO Issuers stop buying

Hedge funds stop buying

Secondary market

Relief
Fannie and Fredddie increase their buying of conforming mortgages

Originator
underwrite and hold

Originators cease issuing new subprime and other nonconforming mortgages

Borrower-Homeowner

Note: ABCP = asset-backed commercial paper; CDO = collateralized debt obligation; MBS = mortgage-backed securities.

Figure 5: Another Look at Loan Securitization

In order facilitate the buying and selling of these massive pools of loans, the fund has databases that store data regarding the pools they have bought and sold. For example these databases store a pool name, the value of the pool and other data of that nature. These databases need to be accessed by multiple different departments within the firm in order for the company to understand its current risk, obligations, equity, and more. Problems can arise with consistency of data between databases since data is originating from multiple sources with no standardized method of labeling between the sources.

2.5   Loan Trading

Loan trading differs based on who the loan is being issued to, an individual or a company. Since mortgages, loans issued to individuals, are what is most pertinent to Angelo, Gordon & Co. the

focus of this section will be on the primary mortgage markets and it will ignore syndicated loans (loans offered by a group of lenders generally to a large company, government, etc.). A simplified illustration of the primary and secondary Mortgage market can be seen below.
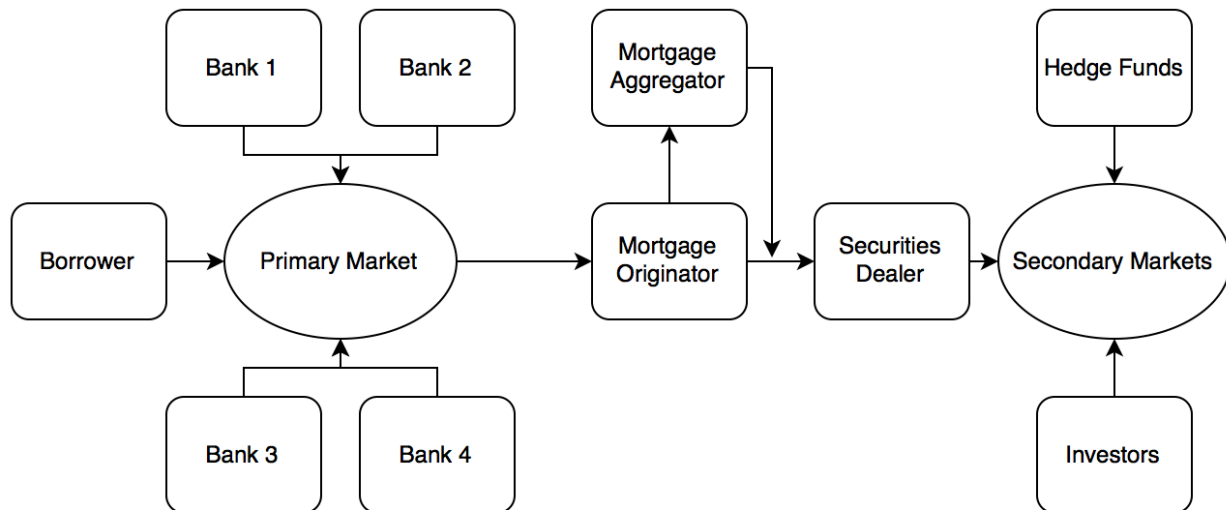
Figure 6: Mortgage Market Simplified

The first step in mortgage market is that the borrower enters the primary market. Banks, mortgage banks and mortgage brokers make up this primary market, though there is little relevant difference between the three, so we will simply consider them all banks. The banks compete over the business of the borrower and eventually the borrower selects a bank to take out a mortgage with. Once the borrower picks a company, then that bank becomes the mortgage originator. The originator generally makes money from fees charged to originate the mortgage. From here the process moves to the aggregator. (Your Mortgage Investopedia, 2015)

The originator generally sells the new mortgage to the aggregator very quickly since they used their own funds to issue it. Aggregators have close ties to Wall Street and to government entities like Fannie Mae and Freddie Mac and primarily securitize the loans into Mortgage Backed Securities (MBS). They are large financial institutions that collect loans and aggregate them into pools (securities) to be sold again. They can also split larger pools into smaller ones called tranching. If the originator didn't immediately sell the new mortgage on market, it is generally because they are acting as the aggregator. (Your Mortgage Investopedia, 2015)

The third step is a securities dealer. Most Wall Street brokerage firms have an MBS trading desk. The goal of these institutions is to sell the mortgages to investors by using a multitude of strategies. Major investors here are hedge funds like Angelo, Gordon & Co. There are a lot of

creative strategies that are used here, some that were responsible for the near financial collapse in 2008.

The last step is the investors buying in the secondary market. Investors aren't just hedge funds though; they are banks, pension funds, insurance companies and more. After this step a loan can follow many different paths until it's completely paid off (Secondary Markets Investopedia, 2015).

2.6 Data management

2.6.1 Quantity of Data

When buying a stock or a bond, individuals and companies are not physically handed a piece of paper signifying their ownership of that one individual asset; instead it is all done virtually. However, the amount of assets any individual owns does not necessarily require any data management on their part. When firms buy pool of loans worth hundreds of millions of dollars, the amount of data they now have to manage becomes massive due to the sheer number of individual loans involved in the transaction. The quantity and granularity of this data requires firms who are involved in this process to incorporate strict and careful data management. This means having multiple databases where the data is stored, as well as having backups, security, and careful data management standards.

2.6.2 Implementing Standards

Since the amount of data that is being managed is not only very detailed, going as far down as describing each individual loan, but also is very important, for losing some data means losing a loan, standards become a key requirement to be implemented in any firm. By standards, we mean naming and storage standards that allow universal understanding of which data is where within the firm. Though this may seem obvious and sound easy, there are many obstacles that get in the way of this.

The first is that often the data comes into the firm in the form of massive tables from third party vendors. This means that the data coming in may be organized and labeled in many different ways depending on whom it is received from. The question then becomes how to, or if it is even worth it to, convert so much data to a standard the firm has established, as opposed to just leaving it and dealing with the difficulties that arise from different labeling conventions within their system down the road.

The second issue that often arises is that many different organizations within a firm may be utilizing and organizing the data differently. The traders may like the data organized one way for their ease of use, but accounting may want the data organized another way for their calculations. This issue is a lot easier for financial firms to control though. By implementing a cross-company policy, though some divisions may be less happy than others, the data will remain consistent.

3 CHAPTER 3: METHODOLOGY

3.1 Planning and Analysis

3.1.1 Version management

Before even beginning to build the program our group knew that there would be a large amount of functionality in this tool, which invariably meant many different iterations of certain aspects as well as many bugs. Version control was going to be an essential part of the project, and so while building the group set up a Git repository using Dropbox to store the files. This way we could always revert back to a previous working version, were we to destroy something while changing an aspect of the program. Both are free software easily located and installed on the Internet.

3.1.2 Feasibility

Initial feasibility analysis determined that the project along with all accompanying aspects of the project, such as documentation, would be feasible to complete in the term. The very first day, the group sat down with the sponsor to define the exact expectations and deliverables of the project. As the project progressed, brief daily meetings were had with the sponsor to ensure the projects goals remained reasonable. Additionally, weekly conference calls with project advisors helped ensure the project stayed on course and expectations were being managed appropriately.

Though the project was not budgeted with extra time, due to the pace at which functionalities of the Quick Rec Tool were completed, multiple times entire parts of the project were completely overhauled to account for a new format of input for a user, or a simpler and more logical method of running the program or outputting results due to new information coming to light.

It was immediately recognized that the project would not expand into standardizing the data entry methods into the databases, and simply remain building the reconciliation tool. Standardizing data entry was still a work in progress for the firm. It is also a logical area of expansion for the project given more time.

3.1.3 Discussions With Potential Users

During the planning stage and throughout the project, the group had frequent discussions with potential users in order to ensure that the product was being built to their liking. Meetings with users were held weekly at a minimum and meetings with the project sponsor were held roughly daily. Our group determined that weekly meetings with potential users was the best way to mediate the conflict of interests between the sponsor, who cared more about functionality of the program, and the end users, who cared more about intuitive interaction with the program. Though the meetings were generally focused on making features more user-friendly and often

resulted in changes to the output file or the input file, they were also responsible for some of the key functionality that was built into the tool such as the translation table functionality.

A common theme throughout these meetings was focusing on creating quick and efficient recs. The idea of quick recs was that users didn't want to have to wait days of even hours for reconciliations. The two primary uses of this system according to potential end users was to be able to start many recs at the end of a day, let them work for a while, and then have all of the outputted reports emailed to employees before the next morning and on demand comfort checks of data between databases. This meant set-up couldn't be a hassle or take an inordinate amount of time. The idea of efficient recs was that the tool would be able to create reconciliations focused only around certain elements of the table that the user deemed important. Essentially the users didn't want to sift through excess data when doing the recs, and this would also help them be quicker.

The discussions with the users throughout the project put a strong emphasis on balancing usability with functionality, as well as having the recs be customizable.

3.2   Design

3.2.1   Balancing Usability With Functionality.

Usability was a top priority throughout the entire process of designing and building features of the application. The user interface was one of the weaknesses of the current reconciliation tool mostly caused by its extensive functionality. The Quick Rec Tool was designed to be simpler and extremely user friendly, yet still get the job done. Think of the current tool as driving a stick shift, whereas the Quick Rec Tool was designed to be driving an automatic. Since the group also knew that the end user of the program hadn't been determined, the Quick Rec Tool had to be designed so that someone of any background could easily learn, run, and use it.

The Quick Rec Tool had a fairly large amount of functionality and flexibility. The core functionality of the programs was to run reconciliations on multiple columns in either databases or Excel files. However, the Quick Rec Tool could also run aggregations over multiple columns and output exceptions within a specified tolerance for each attribute, as well as have translations input by the user so it would ignore specified mismatches in data. The application then would output all exceptions to an Excel file in a predetermined format and then email this file to employees who would go fix the mismatches. Choosing the functionality in any reconciliation is completely up to the user and what they input. Aggregations, translations, and emails was completely optional.

Having all this functionality and flexibility for the user raised two main concerns with usability: having an intuitive input and having a readable output.

Having an intuitive configuration file meant making the file a format that was familiar to all users, had obvious structure, and could be handled easily by the program. After testing multiple file types, the group decided that the optimal input format for both the general configuration and translation table was an Excel file. An Excel file provides natural structure due to its grid format, was familiar to everyone in the firm, and there already existed Python packages specially designed to handle Excel input. Excel files also allowed all of the configuration and translation input to be in one workbook, simply on different sheets.

The top concern regarding the output was readability. The file format was defined at the start of the project to be a CSV file so it could be opened in Excel, eventually simply being a straight Excel (xlsx) file. After consulting with employees in the firm, and using the custom outputs from the current reconciliation tool as a template, we crafted the exception output to be broken into two sheets. The first sheet in the file shows mismatches between input in the keys, and the second sheet shows mismatches in the attributes being checked on. Below is an example of exceptions in the output file.

	A	B	C
1	Source Key: name; owner (pools.xlsx)	Target Key: pool_name; pool_owner (loans.xlsx)	Error Types
2	MERCURY; Bob	mercury; bob	Case Mismatch; Case Mismatch
3	MERCURY; Mary	mercury; Mary	Case Mismatch; Match
4	PREAKNESS; John	Preakness; John	Case Mismatch; Match
5	ABBEY ROAD; Bob	ABBEY ROAD; bob	Match; Case Mismatch
6	PREAKNESS; Dave	preakness; Dave C.	Case Mismatch; Partial Match
7	BELMONT; Jim		In Source Not Target
8		BEMLONT; Jim	In Target Not Source
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			
26			
27			
28			
29			

Key Exceptions | Exceptions | Source | Target | +

Figure 7: Example Output File

The naming of the mismatches was also decided upon by surveying potential end users. The name of the mismatch was meant to provide enough clarity such that the reader could easily see the error, but also is broad enough to describe a full bin of problems. The types of mismatches can be seen in the table below.

Table 1: Mismatch Types

Types Of Mismatch	Meaning
Case Mismatch	There is a mismatch in the case somewhere between the two names. (Ex: bob → Bob)
Style Mismatch	There is a difference in either non-alphanumeric formatting or there is the word "Project" before one name and not the other. (Ex: Project bob → bob
Partial Match	After a style mismatch has been checked, if part of either name is within the other then it is a partial mismatch. (Ex: bob → bobby)
Numeric Mismatch	There is a difference outside the specified tolerance between two numbers, regardless of whether aggregation occurs or not. (4.2 → 4 with tolerance of .1)
In Source Not Target (No Match)	Applies only to keys: The value is in the source table but not in the target table. (Ex: Bob → ___)
In Target Not Source (No Match)	Applies only to keys: The value is in the target table but not in the source table. (Ex ___ → Bob)

Finally, to ensure usability of the finished program, all portions that have interactions with the user (the input file consisting of both the configuration and translation tables as well as the output) have detailed written and video documentation. The written documentation comes in the form of Word file guides as well as Excel tutorial files. This means that even if the user doesn't find the interface intuitive, there are many resources a user could go to in order to understand the software.

3.2.2   Setting Direction

Though the size of the team had been split in half, the original project description had been thrown out, and the new one involved building an entire Python program from scratch, the group had little issues setting direction.

The first step was a meeting with the sponsor to clarify the vision of the project. This happened on the very first day within hours of arriving in the office. During this meeting, we discussed the immediate and long-term goals of the project, as well as resources our group would need, and

other project essential matters. After leaving this meeting the group felt we understood the macro concepts of the project and the immediate path forwards well enough to draw out a workflow diagram for the application as well as identify the major components that would be included. The workflow diagram of the program is reproduced below.
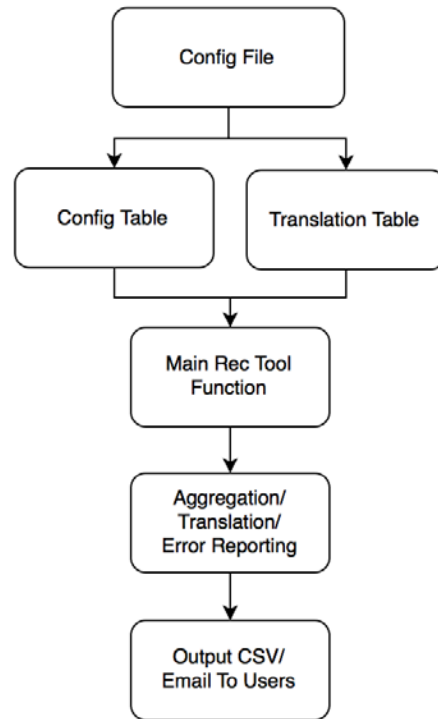
Figure 8: Finalized Workflow Diagram

Once we had this diagram in place, the immediate priority was to get core functionality of the main rec tool portion completed. This meant completing the configuration handler, the database handler, and the output methods.

As the project progressed, the group set direction by maintaining a to-do list, and updating it after each meeting with the sponsor or other employees. These meetings with the sponsor ranged from approximately three times a week during the early build, to approximately daily during the later portion of the build. Meetings with other employees were held approximately once or twice a week. Frequent meetings and a to do list ensured that the project always headed in the direction the sponsor and employees wished, and that if any issues arose that may change that direction, the sponsor knew of them immediately so the project wouldn't flounder for days on end.

The group also maintained direction by scheduling weekly conference calls with its group of advisors. These meetings helped address issues the team wasn't yet aware of, and suggest solutions to problems the team may have been having at the time. The status reports from each meeting can be seen in Appendix B.

3.2.3   Front End

The Quick Rec Tool's front end was designed to be extremely user friendly. The entire Quick Rec Tool is run from the command line, which means that the user interface is only the terminal and the input file.

To simplify the front end even more, the user doesn't even have to open the terminal and type anything in. There are batch files that will run the program by simply double clicking the file.

The other part of the front end is the input file, an Excel file that was designed to be as user friendly as possible. The group went through many iterations and consultations with users to arrive at the optimal format. The input file is further discussed in chapter four.

3.2.4   Human or Automated Mismatch Correction

Early during the build of the Quick Rec Tool there was uncertainty as to whether the program was supposed go into the source of the data, the database or Excel file, and actually change the table entries to correct mismatches or output a list of exceptions with error type for employees to manually change.

The program was initially designed and built with automated error correction as an optional functionality, under the assumption that if it weren't wanted it would be easier to remove than to build later if it were to become wanted. After multiple meetings with sponsors and potential users to further define the Quick Rec Tool's purpose, it was decided that automated error correction would not be a feature of the tool for three key reasons: the security of having a human change data entries was desired, often human correction was going to be required anyways due to illogical data in the sources, and it required a lot of extra permissions.

The idea of the program going into an input data source and changing entries has a lot of potential repercussions that neither side, the project team nor the company, thought were worth the potential time saved by implementing an auto-fix feature. The first repercussion was that if there was ever an error with how the program read a mismatch, or how the program was run that affected what it thought was "correct" and the Quick Rec Tool was allowed to go into source data and change values, it could potentially destroy a vast amount of correct data. If the tool even had the potential to destroy correct data, the risk versus reward of using it often for quick checks didn't make sense. Not to mention, that if the end user wanted to see what data was changed, the end user error reporting functionality had to be in the program regardless. Having no automated fix allowed for the security of an employee seeing the error report and then interpreting changes that needed to be made, as opposed to seeing changes

that were made and having to fix bad corrections if they existed. The latter destroys trust in the program.

The second major reason the automated error reporting was unanimously decided against was due to the fact that there were values in the tables that didn't match up logically, and so a computer wouldn't be able to correct them. A mock example below shows data similar to what existed.

Table 2: Example of Illogical Data Mismatches

Source Data	Target Data
Example 1	Project Example 1231231453432-asbd
Example 2	Project Example 1233453242643-ajfnfj
Example 3	Project Example 1244536450393-ajnfb
None	Project Example 1245432221256-ajneh

As you can see above, there is clearly some sort of relationship between the values in the two databases, though there are three values in the source with Example in their name and four in the target. Yet, there is no clear mapping that would logically make sense, such as Example 1 mapping to Example A or something along those lines. Thus a computer wouldn't be able to correct these mismatches or even pair them up without some sort of extra information. Data mismatches like this appeared in many tests the group ran, so having a human go in and input the corrections would often be unavoidable anyways. This also influenced simply making all the corrections done by human interaction with the data source.

Finally in order to have the Quick Rec Tool go into a data source and edit, for databases the users would have to have permissions to write data entries into many databases. This simply wasn't a realistic expectation for the user to have that many permissions unless the person running the program was in a position like a database administrator.

In the end, it simply made more sense to have the Quick Rec Tool do the reporting of data errors and then have a human go in and fix them.

3.3   Development

3.3.1   Creating The Loan Pool Name Reconciliation Tool

The first step of the development process was creating the loan pool name reconciliation tool. This was the highest-level functionality of the tool; essentially the tool needed to be able check one name against another and determine if they matched. If they didn't match, the Quick Rec Tool also needed to provide some sort of helpful information to the user as to why they weren't matching, like a mismatch in letter case for example.

Since we had no database access, this core functionality was developed using Excel workbooks as input. The program was being built to reference cells, then check the values between the cells. This very first step eventually became influential in determining the input types as well.

In order to make the program work with whatever input we needed, and to better organize the code, a decision was made early on to modularize the Quick Rec Tool. From the instant of first building, it was determined that the Quick Rec Tool would have a main function responsible for passing data from handler to handler, but the calculations and inputting and outputting would all be done in different handlers. Notice the modular nature in the original design of the program in the figure below.
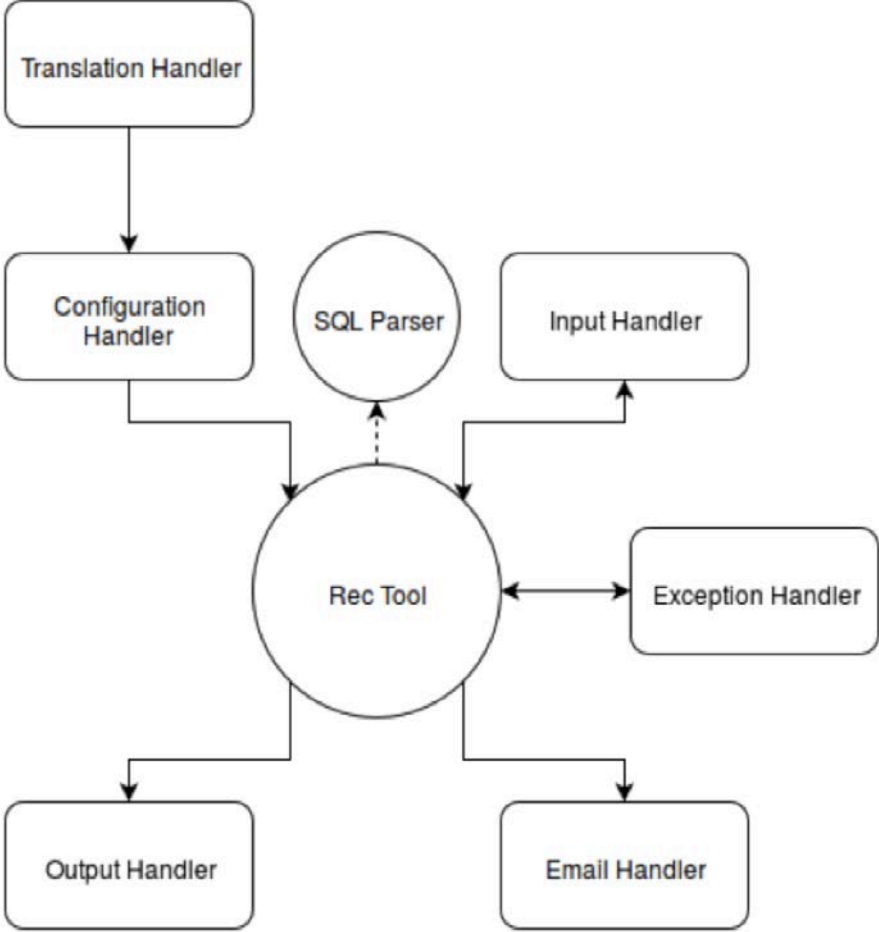

Figure 9: Modular Nature of the Quick Rec Tool

The creation of the loan pool name reconciliation functionality was actually the inception of the Input Handler, Configuration Handler, Exception Handler and Output Handler.

The Input Handler was built to read an Excel file and import data from Excel files then pass that data along. It was mostly completed during this stage, since it is designed to read any generic Excel file based on the specifications of the configuration file.

The Configuration Handler was written during the original build, though was by no means complete. The very first configuration type was a simple text file, so the Configuration Handler was built to be able to import and pass along data from a plain text file. Since there wasn't much functionality, the Configuration Handler didn't have to take in much data, and so it was very simplistic as well at this stage.

The Exception Handler was designed to parse through all the data and then store all the exceptions in a list of some form, which was then to be passed to the Output Handler. During this original build it was decided that the exception handler would store its data in a list of objects holding dictionaries with the exception data.

The Output Handler was originally designed to simply be passed data and then output the data to the Excel file. Though we believed the major functionality of the Output Handler was mostly done during the pool name reconciliation build as well, it turned out later that this was not the case because the layout of the output file changed and the file type of the output changed from CSV to Excel.

Overall, the original build of the highest-level core functionality took roughly a week. After which, the Quick Rec Tool was tested against mock data, and then against data pulled from databases and transposed into Excel files. It required several small bug fixes, though no major overhauls in logic of any of the methods or the main function immediately after.

3.3.2   Integrating a Configuration File

Along with integrating a configuration file, the program required building a Configuration Handler. This was the portion of the Quick Rec Tool that was responsible for reading through the configuration file and passing the configuration data to the rest of the program so that the Quick Rec Tool could pull data and find exceptions. As the configuration file format changed though, so did the Configuration Handler.

Originally the configuration file format was a plain text document. The justification for this was that every user has familiarity with a text file. The Configuration Handler was originally created to be able to parse data from a plain text file, by recognizing certain strings in the configuration text file. The data from the Configuration Handler would then be passed to the other handlers so that they could do their specific functions. The nature of the Configuration Handler in the way that the program needed to be taken in by the program forced the text file to have a format. However the file lacked any evident structure, which resulted in the necessity of memorizing the format adding an additional burden on the user. As well, since a lot of the data the sponsor wanted to do reconciliations on was already in Excel files, putting information in the proper format into a text file for longer reconciliations would be very laborious.

The next iteration of the file was an .ini file. The Configuration Handler changed to support this, but since an .ini file is very similar to a text file except that it has a defined structure, this didn't involve many major changes to the Configuration Handler. Switching to an .ini file format lost

some familiarity with the user, but solved the issue of the configuration file having no evident structure, since this file format comes with headings and then defines property names and property values.

```
[SectionName]
PropertyName1=PropertyValue1
PropertyName2=PropertyValue2
```

Figure 10: .ini File Format

This however did not solve the issue that reconciliations of multiple columns were difficult to input due to the fact that they needed to be put in pairwise ordered comma delineated lists. Around this time, the translation table was also being implemented into the program and an additional issue arose because the configuration file and translation table were in two different files. For convenience and intuitive user interface it was essential that everything be in one place.

The final iteration for the configuration file format was a change to an Excel file format. This required a major overhaul of the entire Configuration Handler. The Configuration Handler changed from recognizing strings in lines of text documents, to recognizing values in specific cells. It required the use of a whole new module, openpyxl, so that it would even be able to read the Excel files. The Configuration Handler now pulled values from cells based on keys in the first column, among other things. The input process became more consistent with this change.

As well, the Excel format had the same familiarity as a generic text file with the users. It also had a built in format with its grids. This format made the most sense as an input file since the output went to an Excel file as well. Additionally, the firm has many existing spreadsheets it wanted to run reconciliations on and copying cell values between Excel files is easier. We were also able to add comments that would ensure the user understood what to insert into the fields, further instilling intuitive use into the program. The translation table was also migrated to other tabs within the configuration file, since Excel has the functionality to have multiple worksheets within the same workbook.

	A	B	C
1	**Field Name (DO NOT EDIT)**	**Field Value**	**Comments**
2	input_type	excel	excel or database
3	source_server		Server to access source database in
4	source_file	pools.xlsx	File for source table
5	source_table	pools	Name of source table
6	target_server		Server to access target database in
7	target_file	pools2.xlsx	File for target table
8	target_table	pools2	Name of target table
9	source_key_columns	name, owner	Names of key columns for source table
10	target_key_columns	pool_name, pool_owner	Names of key columns for target table
11	source_columns	seller, num_loans, total_loan_value	Names of attribute columns for source table
12	target_columns	pool_seller, num_loans, total_loan_value	Names of attribute columns for target table
13	tolerances	0, 1	Tolerances for numeric attributes
14	source_aggregation	off	Aggregation functionality for source table
15	target_aggregation	off	Aggregation functionality for target table
16	source_aggregation_types		Aggregation types for source table (count, sum, avg, min, max, or none)
17	target_aggregation_types		Aggregation types for target table (count, sum, avg, min, max, or none)
18	email	off	Email functionality
19	subject_line	Rec Tool	Subject line of email
20	to_addresses		To addresses for email
21	from_addresses	rectool@example.com	From address for email
22	body_file	email_body.txt	Text file holding the contents of the email's body

Config | Translation | +

Normal View | Ready | Sum = 0

Figure 11: 1ˢᵗ Excel File Iteration

On the second iteration of the Excel file the group took care of the issue of comma delineated lists causing the user to have to put a lot of labor into running reconciliations on multiple columns in a table. Moving the comments to the left, and allowing each table column name to be entered in a separate cell solved the problem. This also meant that users could just cut and paste selections of headers into the designated column name cells if they wanted to, speeding up the process of entering more detailed reconciliations significantly.

The Configuration Handler did not need any significant changes for the remainder of the iterations since all that was changed was where the file looked for data within an Excel file, not the file format it had to deal with.

Color-coding and helpful additional text was also instilled in this second iteration of the file. As well, the one large table was separated into two smaller ones at the sponsor's request. The purple table below deals with source info, such as whether it is a database or Excel or what the table name is. The yellow table deals with individual features of the table such as what the key column names are. The additional text helps guide the user where to enter data by appearing or disappearing next to the purple table.

21

	A	B	C	D	E	F	G	H	I	J
1	Field Name (DO NOT EDIT)	Comments				Field Values				
2			Input Type		Server		File		Table	
3	source	Values for source input	excel				Enter sou	pools.xlsx	Enter tab	pools
4	target	Values for target input	excel				Enter tar	loans.xlsx	Enter tab	loans
5										
6	source_key_columns	Names of key columns for source table	name		owner					
7	target_key_columns	Names of key columns for source table	pool_name		pool_owner					
8										
9	source_columns	Names of attribute columns for source table	seller		num_loans	total_loan_value				
10	source_aggregation_types	Aggregation types for source table								
11	target_columns	Names of attribute columns for target table	pool_seller		loan_name	loan_value				
12	target_aggregation_types	Aggregation types for target table	none		count	sum				
13	tolerances	Tolerances for numeric attributes (none for non-numeric attributes)	none		0	1%				
14										
15	verbose	Output additional attribute information for keys that don't have a match	on							
16	email	If set to "on" will email a link to the exceptions file generated. "off" otherwise	on							
17	subject_line	Subject line of email	Rec Tool							
18	to_addresses	To: addresses for email	bs.treofspades@gmail.com							
19	from_address	From: address for email	rectool@example.com							
20										
21										
22										
23										
24										
25										
26										
27										
28										
29										
30										
31										
32										
33										
34										
35										

Config / Translation / +

Figure 12: 2nd Excel File Iteration

The third and final iteration of the Excel configuration file incorporated data validation, and compatibility with the preprocessor. The preprocessor was a functionality added later in the project which basically took database input and converted it to an Excel file to be read by the Quick Rec Tool. It is described in detail later in the report. The configuration file had some important fields changed in order to be compatible with this. The third iteration of the Excel file also added data validation drop down lists to display the user their options in entering certain fields like "Server" or "on" and "off" switches as well as splitting the translation table up into two separate tabs.

Field Name (DO NOT EDIT)	Comments	Input Type		Input File		Server		Database
source	Values for source input	excel		Enter source excel file:	pools.xlsx			
target	Values for target input	excel		Enter target excel file:	loans.xlsx			
source_key_columns	Names of key columns for source table	name		owner				
target_key_columns	Names of key columns for target table	pool_name		pool_owner				
source_columns	Names of attribute columns for source table	seller		num_loans				
source_aggregation_types	Aggregation types for source table							
target_columns	Names of attribute columns for target table	pool_seller		loan_name		loan_value		
target_aggregation_types	Aggregation types for target table	none		count		sum		
tolerances	Tolerances for numeric attributes (none for non-numeric attributes)	none		0				
verbose	Output additional attribute information for keys that don't have a match	on						
email	If set to "on" will email a link to the exceptions file generated. "off" otherwise	off						
subject_line	Subject line of email	Rec Tool						
to_addresses	To: addresses for email	email@example.com						

Dropdown options (next to "sum"): count, ✓ sum, avg, min, max, none

Sheet tabs: Config | Source Translation | Target Translation | Server Data | +

Figure 13: 3ʳᵈ Excel File Iteration

3.3.3 Choosing Input Types

At the beginning of the project, while the group and the sponsor were still trying to feel out the purpose of the Quick Rec Tool, the input type was solely Excel files. The immediate and obvious choice for how to use the program was that the firm had many spreadsheets just hanging around and they wanted to know which were up to date, and what was mismatching between them. For the original build of the main functionality, the program was only compatible with Excel files.

The group created the Input Handler module in order to encapsulate reading Excel files. The Input Handler's functionality is so basic that it virtually hasn't changed since its inception.

However, then the project switched to more of a user focus. The end user was unknown, and the amount of knowledge and their potential uses were also unknown. This gave the Quick Rec Tool a necessity for broad functionality. The program was changed to add database input types. In order to query a database, the user wouldn't have a file they input like the Excel input type, but instead would fill in specific fields, then the tool would build a SQL query out of those fields.

After this was completed, the sponsor took time to refine the focus of the tool and the vision of whom the end user would be. The end user was likely going to be an employee who knew how to construct SQL statements, needed to run rather complex reconciliations and the purpose of the tool still needed to remain as broad as possible. Therefore the input type was readjusted a final time. The Quick Rec Tool could still handle both Excel and database inputs however for

database inputs, the user now only had to input a SQL file which had the query they wanted inside of it. This allowed more complex recs and simplified the user interface greatly.

3.3.4 Creating The Output File

The output file needed to be in a format that could be easily handled and manipulated by an end user. The original output file format was designed to be a CSV so that it could be opened in Excel, a program we were sure all users were familiar with.

Like many other features, the output file took multiple iterations. Originally, an Output Handler was created with the job of taking the exceptions and outputting them into the CSV file. The two test tables we used can be seen below. The first figure is the very first output table ever created. It had non-descriptive headers, and was very simplistic.

	A	B	C	D
1	name1	name2	err_type	
2	POOL1	Pool1	Case Mismatch	
3	POOL2	Pool 2	Style Mismatch	
4	POOL4	Project pool-4	Other Mismatch	
5				
6				
7				
8				

Figure 14: 1st Test Output File

	A	B	C	D	E	F	G	H
1	Source (pools1: name; owner)	Target (pools2: pool_name; pool_owner)	Error Types					
2	MERCURY; Bob	mercury; bob	Case Mismatch; Case Mismatch					
3	MERCURY; Mary	Mercury; Mary	Case Mismatch; Match					
4	PREAKNESS; John	Project Preakness; john	Other Mismatch; Case Mismatch					
5	PREAKNESS; Dave	preakness; Dave C.	Case Mismatch; Other Mismatch					
6	ABBEY ROAD; Bob	Abbey-Road; bob	Style Mismatch; Case Mismatch					
7	PREAKNESS; Ryan		Missing Name					
8		kentucky; Mary	Extra Name					
9								
10	Source (pools1: num_loans)	Target (pools2: num_loans)						
11	40	38						
12	failing	source Key	target Key	Source Value	Target value	Type of break		
13	num loans	MERCURY; Bob	mercury; bob	38	40	numeric		
14								
15	Source Key	Target Key	Source Attribute	Target Attribute	Source Value	Target Value	diff	Match,Case Mismathc, Numeric
16	Mer,Bob	mer,bob	Num Loans	Number Loans	39	45	6	
17								
18								
19								
20								
21								
22								
23								
24								
25								
26								
27								
28								
29								
30								
31								
32								
33								
34								
35								
36								
37								
38								
39								

exceptions_20151030-102556.csv +

Figure 15: Possible Attribute Output Formats

Figure 13 shows the group's attempts to determine the best format for displaying attribute exceptions, taking creative input from the sponsor and tweaking it based on the functionality that was already in place.

The original, truly functioning, output file was named by the date and time it was created, to distinguish it from others the user may make. A screen shot of the original output file and format can be seen in the figure below. It contained data separated into two tables, one for key exceptions and the other for attribute exceptions. It also had descriptive headers.

	A	B	C	D	E	F	G
1	Source Key (pools1.sqlite: pools: name; owner)	Target Key (pools2.sqlite: pools: pool_name; pool_owner)	Error Types				
2	MERCURY; Bob	mercury; bob	Case Mismatch; Case Mismatch				
3	MERCURY; Mary	Mercury; Mary	Case Mismatch; Match				
4	PREAKNESS; John	Project Preakness; john	Other Mismatch; Case Mismatch				
5	PREAKNESS; Dave	preakness; Dave C.	Case Mismatch; Other Mismatch				
6	ABBEY ROAD; Bob	beatles; bob	Translation Table Mismatch; Case Mismatch				
7	KENTUCKY; Ryan		Missing Name				
8		kentucky; Mary	Extra Name				
9							
10	Source Key (pools: name; owner)	Target Key (pools: pool_name; pool_owner)	Source Attribute	Target Attribute	Source Value	Target Value	Error Type
11	MERCURY; Bob	mercury; bob	seller	pool_seller	Richard	Dick	Translation Table Mismatch
12	MERCURY; Mary	Mercury; Mary	seller	pool_seller	Glenn	glenn	Case Mismatch
13	PREAKNESS; John	Project Preakness; john	seller	pool_seller	Jacob	jake	Translation Table Mismatch
14	PREAKNESS; Dave	preakness; Dave C.	seller	pool_seller	Mark	Marcus	No Match
15	ABBEY ROAD; Bob	beatles; bob	seller	pool_seller	Richard	Rich	Other Mismatch
16							
17							

exceptions_20151104-091920.csv

Figure 16: 1ˢᵗ Iteration of the Full Output

The next iteration of the output file required an overhaul of the original Output Handler. The sponsor wanted two additional tabs to be added to the output, one containing a table of all the values associated with keys that had matches found in the source, and the other for all values associated with keys that had matches in the target. This meant rewriting the Output Handler because a CSV can't have tabs added to it, although it is easily opened in Excel. The output file type would now have to be an Excel file. The Output Handler was changed to handle outputting to Excel and formatting with three tabs, the first with exceptions and the other two listing data from the source and target input streams. To help organize the output, the tabs were named Exceptions, Source, and Target. By sponsor request, the group changed the output file to be named for the configuration file used as opposed to the date it was created. The name was now always EXCEPTIONS_*configfilenamehere*. The second iteration of the output file can be seen in the figures below, highlighting the Exceptions tab and the Source tab.

Source Key: name; owner (pools.xlsx)	Target Key: pool_name; pool_owner (loans.xlsx)	Error Types
MERCURY; Bob	mercury; bob	Case Mismatch; Case Mismatch
MERCURY; Mary	mercury; Mary	Case Mismatch; Match
PREAKNESS; John	Preakness; John	Case Mismatch; Match
ABBEY ROAD; Bob	ABBEY ROAD; bob	Match; Case Mismatch
PREAKNESS; Dave	preakness; Dave C.	Case Mismatch; Partial Match
BELMONT; Jim		In Source Not Target
	BEMLONT; Jim	In Target Not Source

Source Key: name; owner (pools.xlsx)	Target Key: pool_name; pool_owner (loans.xlsx)	Source Attribute	Target Attribute	Source Value	Target Value	Error Type
MERCURY; Mary	mercury; Mary	seller	pool_seller	Glenn	glenn	Case Mismatch
ABBEY ROAD; Bob	beatles; bob	seller	pool_seller	Richard	richard	Case Mismatch
PREAKNESS; Dave	preakness; Dave C.	seller	pool_seller	Mark	Marcus	No Match
BELMONT; Jim		seller		Larry		In Source Not Target
	BEMLONT; Jim		pool_seller		Larry	In Target Not Source
MERCURY; Bob	mercury; bob	num_loans	count(loan_name)	4.00	3.00	Numeric Mismatch (tolerance: 0.0)
BELMONT; Jim		num_loans		1.00		In Source Not Target
	BEMLONT; Jim		count(loan_name)		1.00	In Target Not Source
MERCURY; Bob	mercury; bob	total_loan_value	sum(loan_value)	100.00	95.00	Numeric Mismatch (tolerance: 1.0%)
PREAKNESS; John	Preakness; John	total_loan_value	sum(loan_value)	80.00	78.90	Numeric Mismatch (tolerance: 1.0%)
PREAKNESS; John	Preakness; John	total_loan_value	sum(loan_value)	60.00	59.00	Numeric Mismatch (tolerance: 1.0%)
BELMONT; Jim		total_loan_value		30.00		In Source Not Target
	BEMLONT; Jim		sum(loan_value)		30.00	In Target Not Source

Exceptions / Source / Target / +

Figure 17: 2nd Iteration Output File

name; owner (pools.xlsx)	seller	num_loans	total_loan_value
MERCURY; Bob	Richard	3.00	100.00
MERCURY; Mary	Glenn	2.00	500.00
PREAKNESS; John	Jacob	4.00	300.00
KENTUCKY; Ryan	Carol	2.00	25,000.00
ABBEY ROAD; Bob	Richard	3.00	450.00
BELMONT; Frank	Lenny	1.00	80.00
PREAKNESS; Dave	Mark	2.00	60.00

Exceptions / **Source** / Target / +

Figure 18: 2nd Output File Iteration Source Tab

The final iteration of the output file didn't involve any major changes to the Output Handler, but did reorganize the file. Instead of having three tabs where the data was displayed and two tables on the first tab, the two tables were separated and one was put on a new tab. The new iteration had the first tab dedicated to key exceptions, the second tab dedicated to attribute exceptions, the third to source matches and the fourth to target matches. Additionally, a percent difference ("diff") was added to the attributes table so the user could see by what percentage numeric mismatches were incorrect. An example showing the Key Exceptions and Attribute Exceptions tabs can be seen in the figures below.

	A	B	C
1	Source Key: name; owner (pools.xlsx)	Target Key: pool_name; pool_owner (loans.xlsx)	Error Types
2	MERCURY; Bob	mercury; bob	Case Mismatch; Case Mismatch
3	MERCURY; Mary	mercury; Mary	Case Mismatch; Match
4	PREAKNESS; John	Preakness; John	Case Mismatch; Match
5	ABBEY ROAD; Bob	ABBEY ROAD; bob	Match; Case Mismatch
6	PREAKNESS; Dave	preakness; Dave C.	Case Mismatch; Partial Match
7	BELMONT; Jim		In Source Not Target
8		BEMLONT; Jim	In Target Not Source
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Key Exceptions / Exceptions / Source / Target / +

Figure 19: 3ʳᵈ Output File Iteration Key Exceptions

Source Key: name; owner (pools.xlsx)	Target Key: pool_name; pool_owner (loans.xlsx)	Source Attribute	Target Attribute	Source Value	Target Value	Error Type	Diff
MERCURY; Mary	mercury; Mary	seller	pool_seller	Glenn	glenn	Case Mismatch	
ABBEY ROAD; Bob	beatles; bob	seller	pool_seller	Richard	richard	Case Mismatch	
PREAKNESS; Dave	preakness; Dave C.	seller	pool_seller	Mark	Marcus	No Match	
BELMONT; Jim		seller		Larry		In Source Not Target	
	BEMLONT; Jim		pool_seller		Larry	In Target Not Source	
MERCURY; Bob	mercury; bob	num_loans	count(loan_name)	4.00	3.00	Numeric Mismatch (tolerance: 0.0)	-25.00%
BELMONT; Jim		num_loans		1.00		In Source Not Target	
	BEMLONT; Jim		count(loan_name)		1.00	In Target Not Source	
MERCURY; Bob	mercury; bob	total_loan_value	sum(loan_value)	100.00	95.00	Numeric Mismatch (tolerance: 1.0%)	-5.00%
PREAKNESS; John	Preakness; John	total_loan_value	sum(loan_value)	80.00	78.90	Numeric Mismatch (tolerance: 1.0%)	-1.37%
PREAKNESS; John	Preakness; John	total_loan_value	sum(loan_value)	60.00	59.00	Numeric Mismatch (tolerance: 1.0%)	-1.67%
BELMONT; Jim		total_loan_value		30.00		In Source Not Target	
	BEMLONT; Jim		sum(loan_value)		30.00	In Target Not Source	

Key Exceptions | **Exceptions** | Source | Target | +

Figure 20: 3rd Iteration Output File Attribute Exceptions

3.3.5 Email Reporting

The Email Handler gives the Quick Rec Tool the functionality to automatically generate an email, attach the exceptions file, and send it to end users. This handler uses a few different Python packages such as smtplib and MIME to generate the email and attach the exception file. To do this the Email Handler gets passed information from the Configuration Handler about the email body and addresses. Once the output file has been created, it is attached in the email, and the email information has been received from the configuration file, the Email Handler sends out the finished exception report to the user.

The default subject line that is given to any email is, "Rec Tool: Found _____ Exceptions!" The blank is the number of exceptions found. The default body statement is, "This is an automated message from the rec tool. Do not respond." The body text can be changed by editing the text file named *email_body.txt* found in the same directory as the Quick Rec Tool.

3.3.6 Integrating a Translation Table

Figure 21: Translating Values

The Translation Handler was part of the original core functionality of the Quick Rec Tool. In the group's first meeting with a potential user a few days into the build, the user's first suggestion was to include a translation table to associate keys that didn't match and avoid outputting known breaks that weren't considered incorrect.

The Translation Handler functionality took a few iterations to complete. The first iteration was building the original core functionality, where the handler would take the values the program received from the inputs, convert any translations, and then pass them to the rest of the program for exception processing. This was designed to work with an .ini file on the first iteration.

The second iteration of the Translation Handler was when the translation table format changed to an Excel file. The logic of the code remained the same, however similar to the Configuration Handler changes when the configuration file switched to Excel, the Translation Handler had to be changed to be able to receive data in a new format. Further changes also had to be made because the translation table was then moved into the same file as the configuration table.

The last major iteration of adding a translation table emerged during original testing of the feature. It was after it was moved into the correct location, where it would remain, albeit split into two tabs in the configuration file (one for source, one for target). The Translation Handler had a major bug when checking attributes. The handler would change any key names that were being translated, but then when it went to go look up attributes for the translated values, they didn't exist in the original file they had come from, because they had been translated. This meant that the program couldn't find any attributes for the translated value, and thus couldn't properly reconcile them. In order to fix this, the group created an "untranslate" feature which was held within the Exception Handler, that would switch translated values back specifically for the purpose of looking up their attributes. This sounds extremely inefficient, but we needed to translate the key values back before retrieving and checking the attributes, because otherwise the program wouldn't be able to create the key pairing to know which attributes to check against each other.

After this the Translation Handler had some small bug fixes, but most of the work was done to decrease the run time of translating all the value. The run time was decreased significantly towards the end of the project, so that translating many (think thousands) of values didn't noticeably change the overall run time.

3.3.7   Reconciling On Multiple Attributes

After the original build of the core functionality, the obvious next step was to begin reconciling attributes. This meant adding to the Exception Handler, which already had the functionality to check for matches in strings from its original build purpose of checking keys. Notice an example table below that has the key highlighted in yellow and the attributes highlighted in orange. This may be something like what users wanted to check.

Table 3: Example Source Attribute Table

Source					
loan_id	loan_name	pool_name	pool_owner	pool_seller	loan_value
1	Mercury 1	mercury	bob	Richard	20
2	Mercury 2	mercury	bob	Richard	30
3	Mercury 3	mercury	bob	Dick	45
4	Mercury 4	mercury	Mary	glenn	250
5	Mercury 5	mercury	Mary	Glenn	250
6	Preakness 1	Preakness	John	Jake	100
7	Preakness 2	Preakness	John	Jacob	100
8	Preakness 3	Preakness	John	Jacob	100
9	preakness 4	preakness	Dave C.	Marcus	40
10	preakness 5	preakness	Dave C.	Mark	19
11	kentucky 1	KENTUCKY	Ryan	Carol	125
12	kentucky 2	KENTUCKY	Ryan	Carol	126
13	beatles 1	beatles	bob	Richard	150
14	beatles 2	beatles	bob	Dick	150
15	beatles 3	beatles	bob	richard	151.1
16	Belmont 1	BELMONT	Frank	Lenny	78.9
17	Belmont 2	BEMLONT	Jim	Larry	30

Table 4: Example Target Attribute Table

Target					
loan_id	name	pool	owner	seller	value
1	Mercury 1	mercury	bob	Richard	20
2	Mercury-2	Mercury	bob	Richard	25
3	mercury 3	mercury	bob	Dick	45
4	Mercury 4	mercury	Mary	glenn	249
5	Mercury 5	mercury	Mary	Glenn	250
6	Preakness1	Preakness	John	Jake	100
7	Preakness 2	Preakness	John	Jacob	99.9
8	PREAKNESS 3	PREAKNESS	John	Jacob	100
9	preakness 4	preakness	Dave C.	Marcus	40
10	preakness 5	preakness	Dave C.	Mark	20
11	kentucky	KENTUCKY	Ryan	Carol	125
12	kentucky 2	KENTUCKY	Ryan	Carol	126
13	beatles 1	ABBEY ROAD	bob	Richard	150
14	beatles 2	ABBEY ROAD	bob	Dick	150
15	beatles 3	ABBEY ROAD	bob	richard	151
16	Belmont 1	BELMONT	Frank	Lenny	80
17	Bemlont 2	BEMLONT	Jim	Larry	30

Checking multiple attributes meant that the Exception Handler was given the functionality to be able to take key pairings it created from the original key check, and then use those pairings to check each pair of attributes that were under the keys.

As well, the Configuration Handler and configuration file both needed to be changed to bring the attributes' data into the program. The meant adding the necessary fields to the configuration file and adding the functionality to read and recognize the fields as attributes into the Configuration Handler.

3.3.8   Reconciling Numeric Values

After adding the functionality to reconcile multiple attributes, the next step was adding the ability to reconcile numeric values. Before this feature was added the Quick Rec Tool could only reconcile strings.

This step involved adding additional functionality to the Exception Handler. The handler was given the ability to check for exceptions between both string and numeric values. For the Exception Handler to check numeric values, it was roughly the same process as matching strings; it simply looked for the exact value in the other data source.

However, the group recognized that having to have an absolutely perfect match between numeric values isn't always realistic with rounding, and larger numbers. Thus, the group added tolerance functionality for numeric attributes.

Upon original development, the Exception Handler was programmed to be able to have numeric inputs in the configuration file that would correspond to an absolute tolerance of by how much two numbers could vary from each other and still be considered a match. Notice the arrow pointing to the tolerance in the figure of the configuration file below.

Figure 22: Absolute Tolerance Example

An absolute tolerance, though, can be equally limiting if the user is dealing with a wide range of values. Say for example the user had a column of loan pool values and one pool had a value of a million dollars while the other had a value of a hundred dollars. An absolute tolerance wouldn't do nearly as much for the million-dollar pool as the hundred-dollar pool. Thus an optional percentage tolerance was added. The percentage tolerance takes a percent of the source value, sets it as the tolerance, and then checks to see if the target value is within that tolerance. Notice the arrow pointing to the percentage tolerance below.

Figure 23: Percent Tolerance Example

A bug the team encountered upon testing the numeric attribute reconciliation functionality after creating it was that sometimes when the tolerance was set to 0, two numbers that appeared exact matches would be output as mismatches. The reason was that in rounding the numbers would be off by a minuscule decimal such as .0000001, and so technically their difference was not zero. To avoid this we instituted that a tolerance of 0 is actually a tolerance of .0001 and rounded off long decimals, this way errors that are too small to take any effect like minute rounding errors wouldn't be output.

3.3.9 Aggregation

Once the functionality to reconcile numeric attributes was added, the next step was adding aggregation functionality to the Quick Rec Tool. The new capability was programmed into the Input Handler since it was a process of manipulating the input as opposed to being somewhere like the Exception Handler, since aggregation alone had nothing to do with exceptions. There were many instances where Angelo, Gordon & Co. would need to aggregate numeric values, such as any instance when they needed to aggregate loan values up to a pool value that many of the loans belonged to.

There would also be instances where the firm would want to aggregate strings. The only aggregation that can be performed on strings is counting. The tables below show an example where the strings of the loan names in the table Target: Individual Loans need to be counted with respect to what pool they are in and then reconciled against the Number of Loans in the table Source: Pools.

Table 5: Aggregation Example 1

Source: Pools		
Pool Name	Number of Loans	Pool Location
Pool 1	2	On Shore
Pool 2	1	Off Shore

Table 6: Aggregation Example 2

Target: Individual Loans

Loan Name	Pool Name	Loan Location
Loan 1	Pool 1	On Shore
Loan 2	Pool 1	Off Shore
Loan 3	Pool 2	Off Shore
Loan 4	Pool 2	Off Shore

In the source table above Pool 1 correctly has two loans listed under Number of Loans, however Pool 2 has only one listed, which is incorrect. The program knows which loans belong to which pool by using a distinguishing key, such as Pool Name in the target table in this example.

Aggregation can also be combined with simple string checking functionality. The columns Pool Location and Loan Location in the source and target table respectively illustrate this purpose. Though the function counts the number of loans and reconciles that attribute using aggregation, in the same reconciliation, the Quick Rec Tool could also check if the loan locations were properly matching the pool locations. The program would notify the user running a reconciliation of the tables above that Loan 2 in the target table has an incorrect loan location.
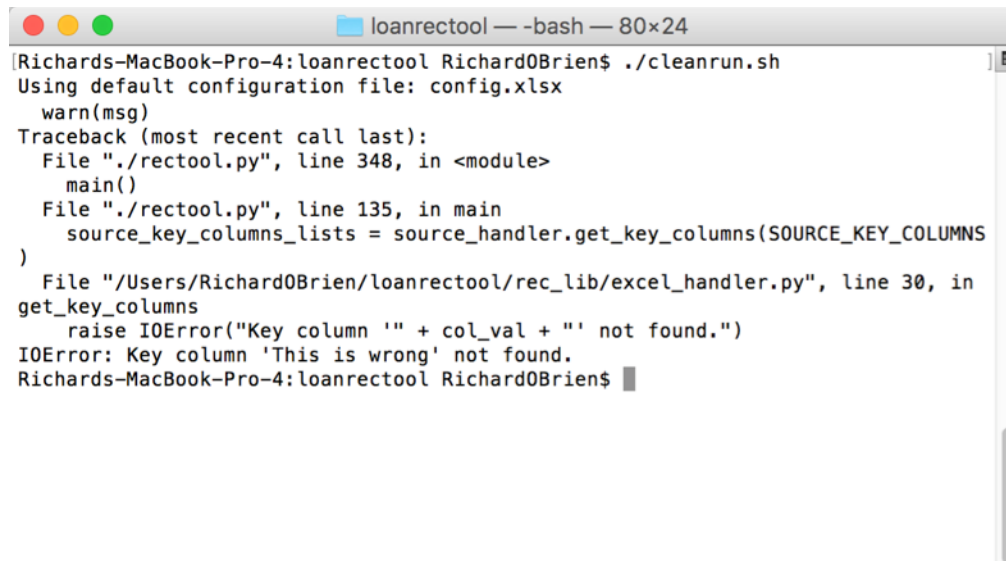
3.3.10 User Reporting

An ongoing process throughout development was ensuring the Quick Rec Tool had clear and concise error reporting. This meant both that the program would give the user an appropriate amount of updates regarding the internal workings of the program and that it would fail gracefully when it encountered input errors.

The user needs to be aware of what the program is doing in order to judge how long the program will run and to be sure it is actually running. Just hitting run and watching a blank screen would leave a lot of uncertainty as to what is going on behind the scenes and make the Quick Rec Tool seem like a black box. As well, if something is input incorrectly, there is no functionality we could code in to try to guess what the user would mean, like auto-correct, so instead our group decided to provide custom alerts that give the user useful information as to what they did wrong.

The user reporting is housed mostly in the main function of the Quick Rec Tool, though appears in other places like the Input Handler, where potentially erroneous information would cause breaks that the main function is unaware of.

The first iteration of user reporting was only console error reporting. The program used the built in reporting that Python automatically comes with. The terminal displayed a generic error report when something went wrong automatically, and pointed to specific locations in the code where the error occurred. An example can be seen in the figure below.

```
[Richards-MacBook-Pro-4:loanrectool RichardOBrien$ ./cleanrun.sh
Using default configuration file: config.xlsx
  warn(msg)
Traceback (most recent call last):
  File "./rectool.py", line 348, in <module>
    main()
  File "./rectool.py", line 135, in main
    source_key_columns_lists = source_handler.get_key_columns(SOURCE_KEY_COLUMNS
)
  File "/Users/RichardOBrien/loanrectool/rec_lib/excel_handler.py", line 30, in
get_key_columns
    raise IOError("Key column '" + col_val + "' not found.")
IOError: Key column 'This is wrong' not found.
Richards-MacBook-Pro-4:loanrectool RichardOBrien$ ▊
```

Figure 24: 1ˢᵗ Iteration of Error Reporting

However, our group realized that the person running this tool will not understand these cryptic error messages, and leaving this sort of dirty dump is bad practice. It also breaks the fourth wall of our program because it forces the user to go into code. Thus the first iteration did not truly accomplish its goal of making the program more transparent to users and didn't tell the user anything about the internal process as it happened.

The second iteration added print statements and a log to the user reporting. The print statements printed out a report of the major processes in real time to the terminal, such as when it was translating values. The log file was of the file type .log and would have a description of the error that had occurred. This way the user knew what was happening as the program ran and, instead of navigating a code dump to find an error, the person was told in plain English what had gone wrong. Another addition to the program was that the user now needed to press enter in order to exit the program. This way the print statements wouldn't just disappear the second the program stopped due to an error. An example of the second iteration of user reporting can be seen in the two figures below.

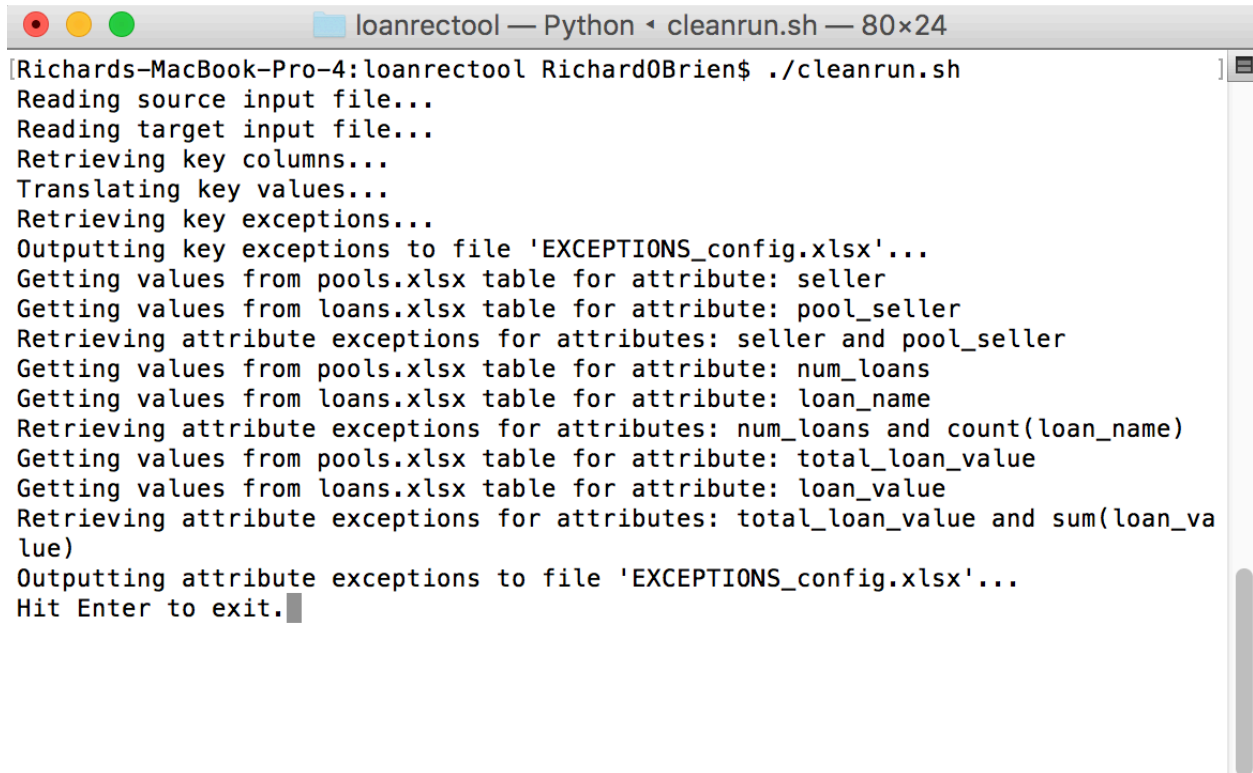**Figure 25: 2nd Iteration of Error Reporting**
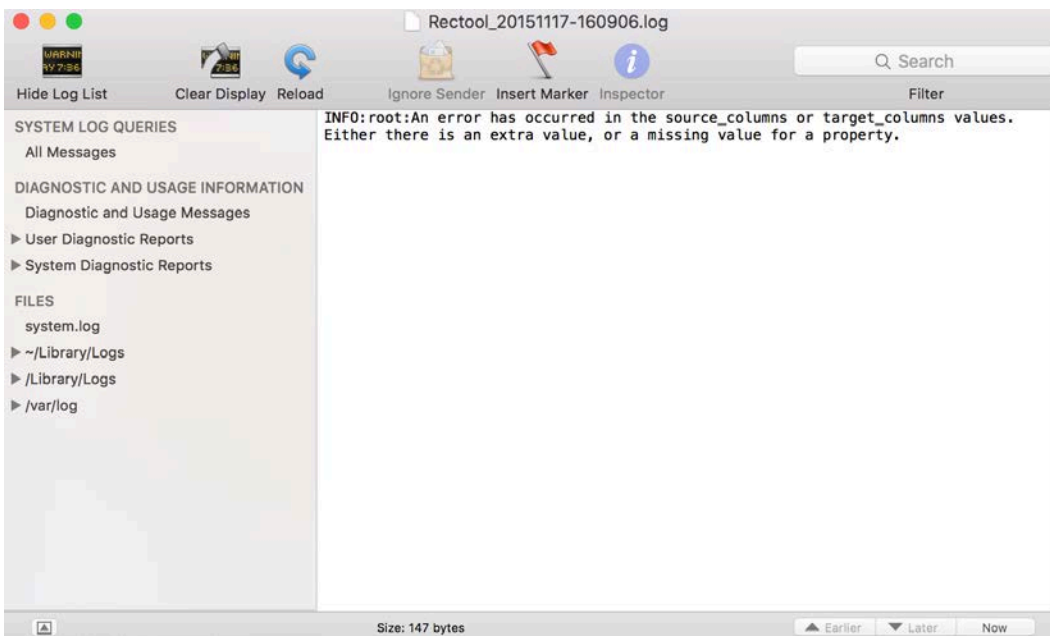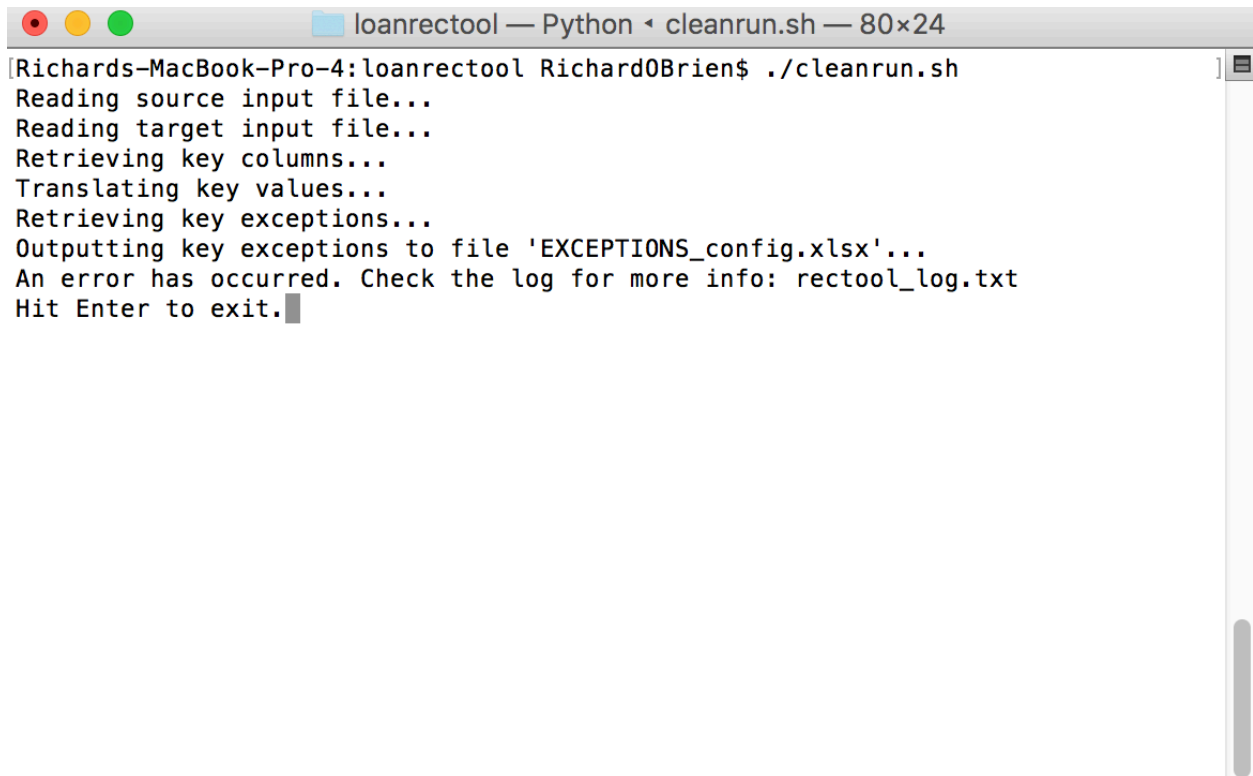

Figure 26: 2nd Iteration Error Log

The last iteration of user reporting involved refining the existing structure that was in place. The log was moved to a .txt file since it was simply a more familiar file type. The print statements were also copied into the log, so the user could see exactly where the program stopped when looking at the error. Finally, the print statements were made more precise. Instead of being only over major functionality like aggregation or translation, the print statements were made to

be more granular, such as notifying the user on every hundred pairs of matched keys. Below is an example of the terminal when the program executes without a problem, the terminal when it encounters an error, and the log once an error is encountered for the last iteration of user reporting.

```
loanrectool — Python ‹ cleanrun.sh — 80×24
[Richards-MacBook-Pro-4:loanrectool RichardOBrien$ ./cleanrun.sh
Reading source input file...
Reading target input file...
Retrieving key columns...
Translating key values...
Retrieving key exceptions...
Outputting key exceptions to file 'EXCEPTIONS_config.xlsx'...
Getting values from pools.xlsx table for attribute: seller
Getting values from loans.xlsx table for attribute: pool_seller
Retrieving attribute exceptions for attributes: seller and pool_seller
Getting values from pools.xlsx table for attribute: num_loans
Getting values from loans.xlsx table for attribute: loan_name
Retrieving attribute exceptions for attributes: num_loans and count(loan_name)
Getting values from pools.xlsx table for attribute: total_loan_value
Getting values from loans.xlsx table for attribute: loan_value
Retrieving attribute exceptions for attributes: total_loan_value and sum(loan_va
lue)
Outputting attribute exceptions to file 'EXCEPTIONS_config.xlsx'...
Hit Enter to exit.
```
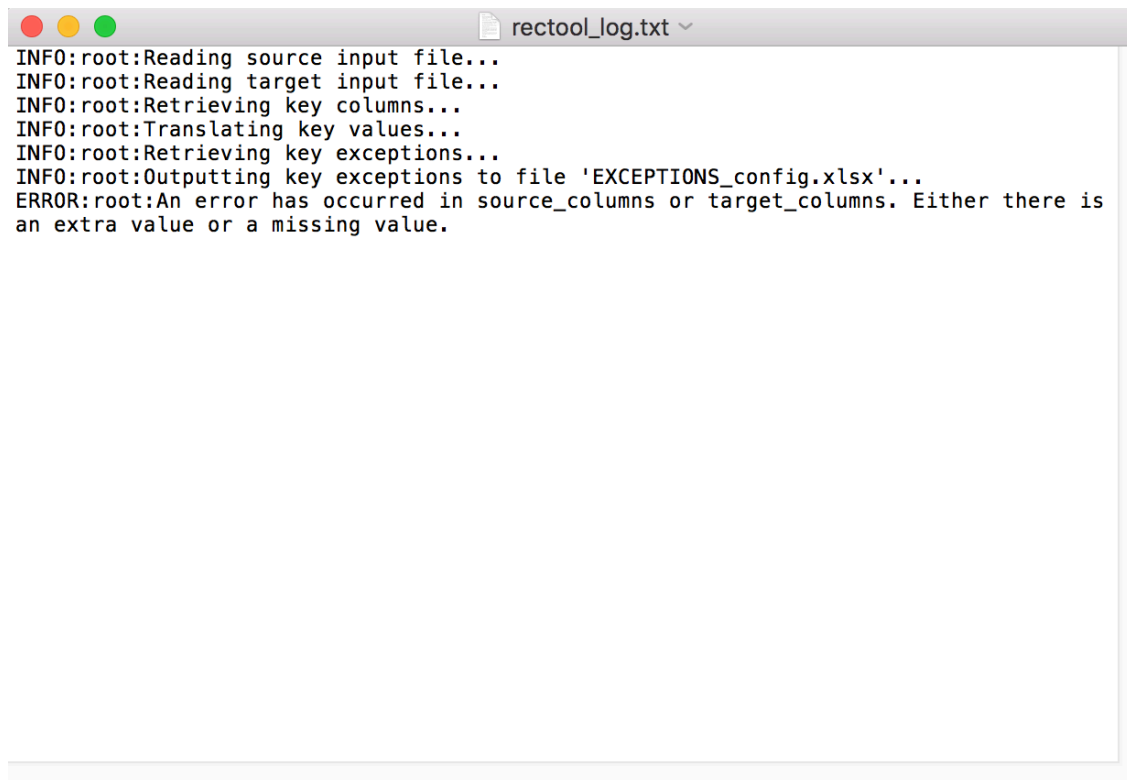
Figure 27: User Reporting Last Iteration Complete Run

Figure 28: User Reporting Last Iteration Error

Figure 29: User Reporting Last Iteration Log

3.3.11 Adding SQL Preprocessor

The SQL preprocessor was added after numerous previous changes to how databases should be handled by the Quick Rec Tool. Originally, the prevailing idea regarding the program's handling of databases, was that the tool would be designed so the user would never have to input SQL in order to access a database table. This meant that the configuration file essentially made the user build fake SQL statement by filling in many different cells. There were a few different iterations on how this was done, creating different layouts of the configuration file with different boxes to be filled in by the user. Some of them were rather confusing. One notable iteration was when there was a cell to be filled in called, "Table," but if someone were using an Excel file as input there would be no table name associated with the file. Another notable example was that we wanted more functionality added to the potential database queries, so we added a cell where the user could optionally type in a SQL "where" clause, and then the program would use that in its SQL query. In this case the user needed to know SQL anyways, so it defeated the purpose of having the user dodge the SQL statement with all the other fields being filled in.

As the project progressed this method of having the user dodge SQL began to not only seem clunky, but also ineffective at accomplishing the queries the sponsor wanted. Since a lot of the data comes to the firm from third parties, in order to query a specific set of data a user may want, it often results in many join statements and a where clause. Using the fill-in-the-blank style we had couldn't accomplish such complex queries. As well, the sponsor had the ability to save SQL queries in SQL files and so once a query was typed out correctly, as long as the user had some way of knowing which file was which, they would never need to type out that SQL query again. Additionally, it became clear that the target user base would likely know SQL.

As a result, the SQL Preprocessor was created. This was designed to be an entirely separate function that could act on its own, but could also be utilized by the Quick Rec Tool. The Preprocessor takes a SQL file, essentially just a file with a SQL statement in it, as input and then queries the necessary server and database, parses through the table, then outputs the data to an Excel file in a way that can be easily taken in by the Quick Rec Tool. Though the Preprocessor can be run on its own, and used for separate actions other than working with the Quick Rec Tool, its primary purpose is to act as the intermediary between the Quick Rec Tool and Databases. Since the Preprocessor was added, the Quick Rec Tool only takes Excel input into the program itself, because any database input is preprocessed automatically.

3.3.12 Reducing Runtime

Towards the end of the project, runtime became a major concern. All the major functionality had been essentially completed, but it didn't matter how perfectly the program caught exceptions or how easy it was to use if the Quick Rec Tool didn't run in a reasonable amount of time.

Upon initial completion of most of the functionality, running a larger reconciliation, on two Excel files with somewhere around 10,000 rows each, would take upwards of forty-five minutes to an hour. If translations were added in then the program would take even longer. This wasn't a very feasible time for the purpose of the program, to deliver a quick customizable comfort check

Among other minor changes, some data structures were altered and many of the calculations were front-loaded allowing for a large reduction in time spent checking for exceptions and doing translations. After all of these optimizations, the Quick Rec Tool could run a large reconciliation on two Excel files with roughly 10,000 rows a piece in around five minutes, a reasonable amount of time for comfort checks.

3.3.13 Refactoring and Commenting the Code

A concerted effort was put into making the code understandable. The code was refactored after almost every major completion in functionality. As well, once all the functionality was complete, the group took some time to focus on making the code readable and logically ordered as well as sorted into the different modules of the overall program for ease of access to any future developers.

Comments can be found regularly throughout all aspects of the Quick Rec Tool as well. They mention the major functionality of every method as well as describe important parts of the code that may not be completely obvious to the reader.

4   CHAPTER 4: IMPLEMENTATION

4.1   Testing

4.1.1   The Testing Process

Testing the software was not a final step, saved for once the project was believed to be complete, but instead was done every time a new core functionality was completed. For example, the Quick Rec Tool was thoroughly tested when it originally was only able to check pool names, then again when the translation table was implemented and every time that functionality was changed, then when aggregation was added, and so on. This process of constant testing may have been time-intensive, but ensured we were building the program upon a solid foundation, as opposed to one riddled with bugs that would cause trouble later.

The testing process typically followed a three-step method. The first step was that our project team would test the functionality in question against mock data that would simulate a scenario requiring the new functionality. Once all of the bugs were fixed from that, we would move on to step two and test the functionality on Excel files populated with data that we were given by our sponsor or databases we had read only access too. Finally, once the functionality checked out, we would notify our sponsor and he would do some testing of his own, and then give us feedback on his findings. Even when he found no additional errors, it also gave him a chance to give immediate feedback on the usability of the program with the new functionality, and offer suggestions to make the added complexity of the new feature more user-friendly. This process probably saved time in the end, because bug fixes were generally small as opposed to major overhauls of the program.

4.1.2   Mock Data Testing

The first step in testing every major change to the Quick Rec tool was to test the program against Excel spreadsheets and local, custom databases holding mock data. The mock data was easily customized to test whichever feature we wanted, and generally held less data. This step can be likened to having the program take a standardized test, because it would give us general knowledge of the big issues, but not knowledge of most of the small abstract ones or any corner cases that were unforeseen when designing the test case. To test, we would deliberately break parts of the data in the populated spreadsheets or tables, and make sure that the program caught all instances of the break. If it didn't, then we knew something was wrong with the function and it had to be debugged.

A frequent bug we found while using this methodology of testing was that the function we had added would only catch the first instance of the exceptions occurring. Had we not known exactly what we broke, then we may have been fooled because we see the function correctly output an exception, however, it would only output the first instance, which wasn't helpful.

That would create a false sense of trust in the user and more importantly, leave incorrect data behind.

This form of testing was also nice because the group had finished the first major step in functionality, having the program be able to check loan pool names against each other, before we had access to databases. Having this step in the beginning allowed us to completely nullify what would have been a delay.

Testing with mock data was very effective in correcting larger logical errors in the program, since we were specifically able to design and run data through the program that would test for them. To test for all possible anomalies we moved to real data testing in phase two.

4.1.3 Real Data Testing

The second step in the testing process was using real data as input sources to see how the Quick Rec Tool handled large amounts of actual data. Testing using some actual use cases ensured that the program would be exposed to all possible errors that could arise, even the ones we couldn't think of. This step in testing can be likened to holding a microscope to the program, or putting it through final exams week. After this, we would know all of the little abstract anomalies in data that revealed bugs.

This testing often made the team add more case-by-case input checking. An example of a functionality that was further refined using this method was the programs ability to differentiate a number and a string. For example, when the key value is a number (such as a numeric ID) the program must convert it to a string in order to properly compare it. Even though humans are able to ignore this fact when comparing the values, at the first iteration something like this confused the program because it was expecting a string value for the keys and it found a number. However, after noticing errors like this with the program during testing we refined the code to make type-checking more robust. For example, the program now assumes that keys will always be treated as strings and convert appropriately. This way it wouldn't become confused over something like this and mistake it for a numeric value.

The real data testing instilled confidence in the team that any form of data would not trip up that the program's new functionality. The next step was exposing the new part of the Quick Rec Tool to the sponsor.

4.1.4 Sponsor Testing

During earlier phases of the build, sponsor testing was often done along with the group, and was brief. Later in the build it became a more distinct and formal stage of testing.

For the first few completed parts of the core functionality, the group would coordinate a quick meeting with the sponsor where he would guide us through different additional data sets he

wanted to see the Quick Rec Tool run on, and simultaneously the meeting would also function as a status update meeting.

As the program further neared completion, this stage in testing became crucial because the sponsor would test the program on many additional data sets, specifically data sets that he intended to reconcile using the finished product at a later date. Most of the errors found here were minor and required small tweaks or bug fixes, though as previously mentioned, the test run often resulted in suggestions about user interaction of the new feature or about macro changes to the program.

Once this phase of testing was completed the functionality was considered complete as well, and the group would move on to the next major step in the build.

4.2   Documentation

4.2.1   The Purpose and Process of Documentation

The purpose of producing documentation for the Quick Rec Tool was to ensure usability and total user comprehension of the program once the project team had left and was no longer around to answer questions. Since the group was unaware who would be using the program, or trying to build further on top of the current functionality, the team added both internal and external sources of documentation to make the program as transparent as possible.

The internal documentation consists of comments, data validation, and other helpful text inserted into the input files. The comments section is a series of statements that explains what is supposed to go into each individual field. The data validation of the server, database, and other cells helps ensure the reader understands the valid choices of inputs. This data validation creates dynamic lists so that the user can choose a value to input into the cell as opposed to typing one in. The file also includes other helpful text which helps notify the user which cells should be filled in or not. Notice the comments section in the second column, the data validation on the server cell, and the helpful text next to the input file in the figure below.

Field Name (DO NOT EDIT)	Comments	Input Type		Input File	Server	Database
source	Values for source input	excel	Enter source excel file:	pools.xlsx		
target	Values for target input	excel	Enter target excel file:	loans.xlsx		
source_key_columns	Names of key columns for source table	name	owner			
target_key_columns	Names of key columns for target table	pool_name	pool_owner			
source_columns	Names of attribute columns for source table	seller	num_loans			
source_aggregation_types	Aggregation types for source table					
target_columns	Names of attribute columns for target table	pool_seller	loan_name	loan_value		
target_aggregation_types	Aggregation types for target table	none	count	sum		
tolerances	Tolerances for numeric attributes (none for non-numeric attributes)	none	0			
verbose	Output additional attribute information for keys that don't have a match	on				
email	If set to "on" will email a link to the exceptions file generated. "off" otherwise	off				
subject_line	Subject line of email	Rec Tool				
to_addresses	To: addresses for email	email@example.com				

Dropdown: count, sum ✓, avg, min, max, none

Tabs: Config / Source Translation / Target Translation / Server Data / +

Figure 30: Documentation

For external documentation, the group created both a guide to running the program as well as a guide to the software and packages used to create the program. Documentation was also produced in the form of use cases since the project group recognized that some ways of conveying the information might be more effective than others depending on who was the end-user.

The process of documentation was a continuous one. After completion or changes to any major functionality in the program, written documentation was reproduced for two reasons: to ensure nothing would be forgotten, and to ensure the sponsor would be able to test the new functionality appropriately.

All of the documentation can be seen in Appendix A.

4.2.2   Internal Documentation

The first documentation a user would likely encounter is the internal documentation. It is essentially the first line of defense against a user not understanding how to operate the Quick Rec Tool. The internal documentation consists of in-file comments, data validation, and other helpful text that will appear depending on which cells are filled in.

The internal documentation is only designed to be quick reminders to the user. If users need a little nudge in the right direction, internal documentation will likely be what they refer too and will likely help them. It is mainly there for convenience as opposed to deep understanding.

Multiple different forms of comments and helpful hints add some depth to the internal documentation. If the comments do fail, there is the added security that the comments combined with the data validation or additional helpful text may succeed.

4.2.3 Written Documentation

The second line of defense against user confusion is the external documentation. This documentation was designed so that a user could quickly search for the answer to a question if the internal documentation wasn't enough to solve their issue. The written documentation contains detailed descriptions, notes, examples, and images of the different parts of the application that the user would interact with. The written documentation is included in Appendix A.

The written documentation is separated into two different files, a program document and a software document. The program document gives detailed explanations of the input file, output file, logs, and launching the program, so if the user wants a detailed explanation on any specific aspect of the program, they should reference the written documentation. The figure below illustrates how the document combines many different methods such as screen shots, bullet points and more to explaining different aspects of the program.

Formatting The Input File.

Description: The input file format is that of an Excel file. The default configuration file is named config.xlsx. If the user wishes to make their own configuration file, they are free to do so as long as they follow the same structure as the default. The format must remain the same because values are pulled and identified by their position in the spreadsheet. If the user chooses to use their own configuration file, they simply input that files name at the appropriate position in the command line argument to launch the file.

Important Notes:
- Both the configuration table and the translation table are in the same file.
- A user should change neither configuration, nor the translation tables' formats.
- The configuration file consists of a configuration table and translation tables. The first sheet in the file is the configuration table; the second and third sheets are the translation table for source and target.
- The tab "Server Data" is simply data used to create data validation on the configuration table. There is never any input on this tab.

Screen Shots:

Figure 1: Configuration File

————————Page Break————————

Figure 31: Written Documentation Example

4.2.4 Use Cases

The goal of the use cases, which were specifically requested by the sponsor, is that they are designed to act like training wheels for the user. There are six use case documents, which all thoroughly show the process of performing a single function. They also build on top of each other since the more complex processes naturally build on top of the simpler ones. The first two use case is a simple key comparison, one for Excel sources, the other for database, the second is an attribute comparison (which requires a key comparison to be done in order to do so), the third employs a translation, the fourth is an aggregation (which requires attribute comparisons), and the fifth is a complex reconciliation with aggregation as well as roll ups. The figures below are screenshots of the six files.

Figure 32: Use Case 1 Excel Key Pair Reconciliation Description Tab

Figure 33: Use Case 1 Excel Key Pair Reconciliation

Figure 34: Use Case 1 Database Key Pair Reconciliation

Figure 35: Use Case 2 Attribute Reconciliation

Figure 36: Use Case 3 Translation

Figure 37: Use Case 4 Aggregation

Figure 38: Use Case 4 Advanced Aggregation & Rolling up

The use cases each have a description tab, which runs the reader through the reconciliation that will be displayed in the configuration tab. This first tab also often displays both tables that will be reconciled next to the text box holding the description. The configuration tab has red numbers inserted into the files. These numbers correspond to a flow chart immediately below the configuration table. This process as well as the order of the numbers indicates the order to fill out the cells of the spreadsheet in. The process describes exactly what has been filled out in the example, and what should be filled out or left blank in order to accomplish the goal. The files also have additional in-cell comments added to them to clarify what to enter into each cell. In addition to a walkthrough, these files could also be used as a shortcut if the user knows what functionality they want, but doesn't want to fill out a configuration sheet from scratch. The use cases are organized in a subdirectory within the same directory that the program is stored. Each use case has another sub directory of its own with an example source and target file that go along with the use case configuration file.

Inside the first use case subdirectory, the use cases describes how to do a simple key reconciliation on one key pair with two Excel files. It is the most basic reconciliation that can possibly be done. There is a use case on performing a key reconciliation between two Excel files and between one Excel file and one SQL file. This way the user has a tutorial on filling out the sheet with database inputs.

The second use case subdirectory holds a file that describes how to reconcile attributes. The reconciliation builds off the first, creating continuity between this file and the double Excel source in use case one.

50

The third use case subdirectory holds a file that adds in translations. It also builds off the first two files and gives clear directions on how to add translations.

The fourth and fifth use case subdirectories hold the most complex reconciliations. The fourth use case discusses a basic aggregation from a table containing loans and their values, to a table containing pools with their respective values. The fifth use case explains a complex reconciliation with three keys and five attributes where some attributes are simply rolled up and may look as though they need aggregation but don't while other attributes do in fact require aggregation. It tries to clarify the difference between aggregations and simply performing one-to-many or many-to-many attribute comparisons.

5 CHAPTER 5: RESULTS

5.1 Finished Product

5.1.1 Logic Overview

The logical overview of the finished Quick Rec Tool can be seen in the figure below.
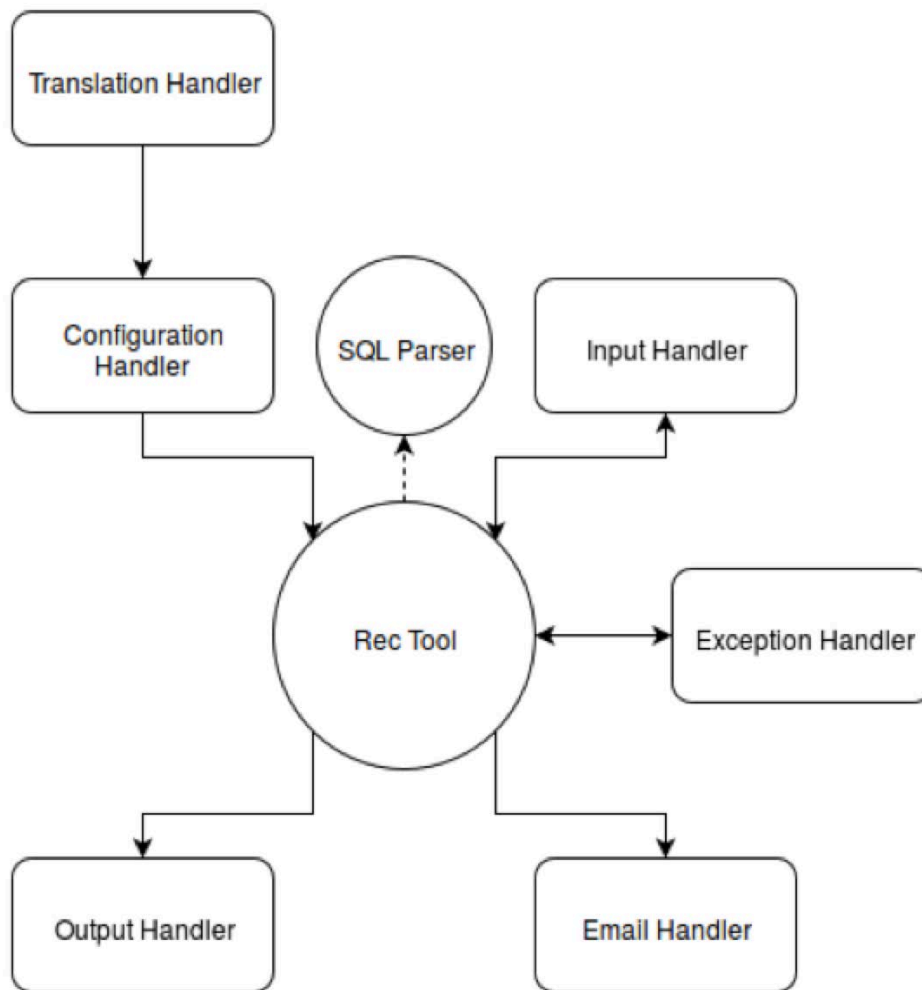
Figure 39: Quick Rec Tool Logical Overview

The main function in the program acts as the hub. Different functionalities are stored in separate modules. For example the function of generating exceptions falls to the Exception Handler.

On the top level of the logic resides the Translation Handler. This module does not directly feed into the main, but instead feeds into the Configuration Handler since the translation table resides in the configuration file. The Translation Handler works with the Configuration Handler

to create a translation table that is then passed to the main function and used to translate the data that is retrieved from the Input Handler.

The SQL Parser takes SQL files as input and converts them to Excel files that are then passed to the Input Handler along with any other Excel inputs. The SQL Parser also has the capability to be run independently to simply perform the conversion of SQL output to an Excel format.

The Input Handler and the Configuration Handler both feed directly into the main function of the Quick Rec tool. The Input Handler takes in data from the main function that was gathered by the Configuration Handler. The Input Handler performs the function of extracting data from the input data sources as well as performing aggregation. This data is then passed back into the main function.

Once source data has entered the main function, the Exception Handler begins working. The Exception Handler takes in source data and generates all exceptions. The matches and exceptions are then returned to the main function.

The Output Handler and the Email Handler are the two potential end points. The Output Handler performs the function of outputting all the data to the output Excel file and formatting it appropriately. If email functionality is off then the program ends here.

If the email functionality is turned on, the Email Handler formats an email, and attaches the output file to it. The email is then sent to all specified email addresses. The program then ends here.

5.1.2   Final Functionality

The finished Quick Rec Tool can find exceptions between numeric types within a tolerance and string types in data from either Excel worksheets or databases. The program can reconcile two tables each containing multiple key and attribute columns. It can avoid outputting mismatches a user already knows exists or create non-standard key pairings using a translation table, and can aggregate over source attributes, target attributes, or both. All exceptions generated are output to an easy to understand Excel file along with data regarding the target and source matches. The program also has a verbose functionality that allows the user to decide whether to output attribute information for unmatched keys. The Quick rec Tool can lastly email the output file to specified recipients for easy dissemination of information.

5.1.3   Final User Interface

The final user interface is merely two Excel files and a batch file. The configuration file and the output file are the two Excel files that the user has to interact with. The configuration file is designed to be user friendly, having data validation, comments, and copious documentation. The output is formatted, specifically to the sponsor's liking, ensuring it is easy to consume and data is presented in a logical and readable way. The batch file provides the user with a more

hands-on approach to initiate the program than typing in the command manually. The user then has to hit enter to exit the program. The final user interface is simple and intuitive.

6 CHAPTER 6: CONCLUSION

6.1 Limitations

6.1.1 Two Table Reconciliations

The current limitation to the amount of tables a user can reconcile is two. If someone wants to reconcile more tables, that person must use the other reconciliation tool IVP Recon. In order to add a feature for reconciling more than two tables, it would involve extreme complications in the user interface. Though the user can only reconcile two tables, within each table there is no limitation on how many keys or attributes that the user can do reconciliations on.

6.1.2 SQL Statements

Another limitation of the tool is that in order to do reconciliation on a database, the user has to input a SQL file. This means that the tool is inaccessible to someone who wants to reconcile a table in a database, but isn't capable of creating a SQL query for that database.

Originally, a user could simply input the server, the database, the table, and the columns that he or she wished to reconcile into the configuration file, and then the Quick Rec Tool would go retrieve the data. No knowledge of SQL was necessary. However, after testing this feature, and further exploring the potential uses of the program, the sponsor determined that this wouldn't work for many of the SQL searches that the company needed to perform. Since the company often receives their data and tables from third party vendors, this means that the queries for specific data they want to reconcile can be massive SQL statements with many joins and specific conditions. An example of this is performing a reconciliation only wanting data as of a certain date. These complex searches couldn't be done easily with the relatively simple user interface. In order to add the capability of doing such SQL queries, and not have the user just input SQL would have made the configuration file a complete mess. Thus it was decided that if the user wanted to query data from a database, they would simply use a file containing the full SQL statement and run it through the preprocessor. This turns it into an Excel file which would then be automatically run through the rec tool.

6.1.3 Recognizing Logical Groups with No Logical Pair

If the Quick Rec Tool finds a group of data as is displayed in the table below, there isn't very much it can do with it. There appears to be no logical pairing between any of these values.

Table 7: Logical Group Without A Logical Pair

Source	Target

Hud 1	Hud-128gcsjq827ey9ei
Hud 2	Hud-128uhckhbcwfbwe
Hud 3	Hud-12832icndsicn2ocj
	Hud-128wejncqbciekek

However, this is a limitation of the program. Instead of outputting the group to the user and mentioning that they have some sort of clear relationship, though the specifics of this relationship can't be determined, instead it simply outputs all the values in the source as missing in the target and all the values in the target as missing in the source. It is difficult to match these types of groups in such a vague way without unintentionally causing more precise pairs to match other similar values as a group, so it was avoided.

6.2   Extension

6.2.1   Ability to Reconcile More Than Two Tables

The most feasible extension would be adding in the functionality to reconcile more than two tables. The logic and complexity of eliminating this limitation would be considerable, though can be done. Changes would end up happening to almost all of the different handlers within the Quick Rec Tool, which assume a source-target relationship. An additional difficulty would be formatting the new configuration file in the most intuitive possible way.

6.2.2   Recognizing Logical Groups

Another extension for the program would be an ability to recognize logical groups that don't have a logical pair. This functionality is difficult to program and it could potentially add a lot of run time to the program with all of the additional checks. The Quick Rec Tool would have to comb through all names that don't have a match, and look for very high-level matches, then output these matches to the user in a logical sense. Therefore these high level matches would probably have to be output into a third table in the output file, which could add unnecessary complication to the output of the tool.

6.2.3   GUI For Configuration and Execution

If all the functionality that Angelo, Gordon & Co. could ever want was complete, then a graphical user interface could be developed. The danger in this is that the GUI is what causes the main issues in the current reconciliation tool that the company uses, IVP Recon. The GUI would have to be designed in a minimalistic way, with the least possible navigating necessary

by the user. We would want to avoid the mistakes of having menus within menus within menus, and forcing the completion of certain steps before the ability to fill out others, which cause issues currently.

6.3   Recommendations

The primary recommendation the group has after completion of this project is to continue implementing standardization of labeling values in databases and Excel files. The group recommends this be done both internally, and to possibly convert data upon reception from external sources.

The Quick Rec Tool could have dramatically increased functionality if this were the case. There is even a small possibility that the tool would become obsolete if a standard was implemented if there weren't ever any breaks in the data, but it is unlikely. It is more likely that the Quick Rec Tool would be able to be extended to automatically make corrections to mismatches since it knew exactly what to change values from and to. This would eliminate all the man-hours that still exist in the process. Humans have to take the error report the Quick Rec Tool generates and fix the breaks manually since they aren't sure what naming conventions the tool will find in the database, and don't necessarily know to change breaks in the data without getting into each individual data set and seeing how it works.

Implementing a clearly defined standard of labeling values, and determining which values are correct would make it possible to transform the Quick Rec Tool from a quick, customizable reconciliation tool, to a fully automated reconciliation and break fixing tool.

7   INDEPENDENT STUDIES FOR THE PROJECT

7.1   Richard O'Brien

7.1.1   Independent Computer Science Studies

In preparation for this project and over the course of it, I was largely working outside my comfort zone. Though I was at home in the world of finance and mathematics, the project more implicitly called upon the fields and directly called for a large amount of computer science. Having only ever taken one computer science course in my life I was playing catch up early and often. Having a group of two ensured I was going to be intimately involved in every part of the project.

Fortunately my one computer science course had been in Python, which happened to be the language we used to build the program. The complexity of the programming involved in creating the Quick Rec Tools functionality required me to do more than simply brush up on my Python, but instead forced me to expand my knowledge in the subject. I used the book <u>Python Programming For the Absolute Beginner</u> as well as online resources like CodeAcademy.com to expand my knowledge.

Probably the most crucial independent learning I did pertained to version management and using the command line. I had never done projects using version management before, so the tools and concept were entirely new to me. Version management is essentially that you save different copies of the project in different places, so if you mess anything up while you are building a part of the program, you always have a safe and working copy to revert back to. This was done using software called Git, and all the commands to control Git are done through the command line. I learned about Git and version management at CodeAcademy.com as well.

The previous extent of my knowledge using the command line was typing in one command to run a game I used to play in high school. Thus not only did I have to learn all about version management but also how to use the command line to run the version management as well as navigate my computer and install Python packages needed to run the Quick Rec Tool. I learned about the command line at CodeAcademy.com, though Stackoverflow.com also played a key role.

Finally the program had to connect to Microsoft SQL servers, so I had to learn how to do SQL queries. CodeAcademy.com again came to my rescue, teaching me the basics of SQL.

8 WORKS CITED

ANGELO, GORDON. (2014). Retrieved December 9, 2015, from
https://www.angelogordon.com/ NJ: Pearson/Prentice Hall.

Behind The Scenes Of Your Mortgage. (2015, April 18). Retrieved December 9, 2015, from
http://www.investopedia.com/articles/pf/07/secondary_mortgage.asp

Finance and Development. (2015). Retrieved December 9, 2015, from
http://www.imf.org/external/pubs/ft/fandd/2007/12/dodd.htm

Hedge Fund Definition | Investopedia. (2015, November 20). Retrieved December 9, 2015, from
http://www.investopedia.com/terms/h/hedgefund.asp

IVP Polaris. (n.d.). Retrieved December 9, 2015, from https://www.ivp.in/portfolio/ivp-polaris/

Long, J. (2014). *Python Programming For Beginners: Quick And Easy Guide For Python Programmers*. CreateSpace Independent Publishing Platform.

Secondary Mortgage Market Definition | Investopedia. (2015, April 17). Retrieved December 9, 2015, from http://www.investopedia.com/terms/s/secondary_mortgage_market.asp

Securitization Audit. (n.d.). Retrieved December 9, 2015, from
http://thepatriotswar.com/index.php/securitization-audit/

Welcome to SQLCourse.com! (2015). Retrieved December 9, 2015, from
http://www.sqlcourse.com/

9 APPENDIX A: DOCUMENTATION

9.1 Program Document

Formatting The Input File.

Description: The input file format is that of an Excel file. The default configuration file is named config.xlsx. If the user wishes to make their own configuration file, they are free to do so as long as they follow the same structure as the default. The format must remain the same because values are pulled and identified by their position in the spreadsheet. If the user chooses to use their own configuration file, they simply input that files name at the appropriate position in the command line argument to launch the file.

Important Notes:

- Both the configuration table and the translation table are in the same file.
- A user should change neither configuration, nor the translation tables' formats.
- The configuration file consists of a configuration table and translation tables. The first sheet in the file is the configuration table; the second and third sheets are the translation table for source and target.
- The tab "Server Data" is simply data used to create data validation on the configuration table. There is never any input on this tab.

Screen Shots:

Figure 40: Configuration File

Figure 41: Source Translation Table

Figure 42: Target Translation Table

Formatting The Configuration Table

Description: The "Config" sheet is separated into two tables. The purple table specifies input type, and macro information about the sources of data. The yellow table specifies the detailed information about the sources of data.

Important Notes:
- The Source is the set of data that is considered "correct." The Target is the set of data being reconciled against the source. If neither set of data is considered "correct" these terms are arbitrary.
- If the cell is highlighted in red, do not change it.

Format: Below is the Format of the input file.

Top Table

Field Name (DO NOT EDIT)	Field Value
Source	\<Enter "Excel" or "database" Here\> • *If "Excel" is entered, will take Excel files as inputs* • *If database is entered, will take tables from databases as inputs*
Target	\<Enter Server Name Here\> • *Server to access source database in.* • *May be the same as the target_server.*
Input Type	\<Name of the Source Table Here\> • *Names must be entered exactly (case sensitive)* • *For Excel files this name is arbitrary.* *Don't leave blank, used in translations and aggregation.*
Input File	\<Enter Name of File for Excel/Database\> • *Names must be entered exactly (case sensitive)* • *If database, this will be a SQL file.*
Server	\<Enter Server Name Here\> • *Names must be entered exactly (case sensitive)* • *Server to access database in.* • *If name doesn't appear on drop down list, add it to the table on "Server Data" tab.*
Database	\<Enter Server Name Here\> • *Names must be entered exactly (case sensitive)* • *Server to access database in.* • *If name doesn't appear on drop down list, add it to the table on "Server Data" tab.*

Bottom Table

source_key_columns	<Key Column Names From Source File/Database Here> • *Each element in key column should be unique* • *One value per cell. Unlimited number of cells.* • *Names must be entered exactly (case sensitive)* • *Order should reflect the order of target_key_columns.* • *Ex. 1st entry in target_key_columns reconciled to first entry in source_key_columns.*
target_key_columns	<Key Column Names From Target File/Database Here> • *Each element in key column should be unique* • *One value per cell. Unlimited number of cells.* • *Names must be entered exactly (case sensitive)* • *Order should reflect the source_key_columns order.* • *Ex. 1st entry in source_key_columns reconciled to first entry in target_key_columns.*
source_columns	<Attribute Column Names From Source File/Database Here> • *One value per cell. Unlimited number of cells.* • *Names must be entered exactly (case sensitive)* • *Order should reflect the order of target_columns.* • *Ex. 1st entry in target_columns reconciled to first entry in source_columns.*
source_aggregation_types	<Aggregation Types For Source Table Here> • *Aggregations performed on source columns.* • *One value per cell. Unlimited number of cells.* • *Choose none for a column not aggregated.* • *Order of types matches order of columns.* *Types: count, sum, min, max, avg, none.*
target_columns	<Attribute Column Names From Target File/Database Here> • *One value per cell. Unlimited number of cells.* • *Names must be entered exactly (case sensitive)* • *Order should reflect the order of source_columns.* • *Ex. 1st entry in source_columns reconciled to first entry in target_columns.*
target_aggregation_types	<Aggregation Types For Target Table Here> • *Aggregations performed on target columns.* • *One value per cell. Unlimited number of cells.* • *Choose none for a column not aggregated.* • *Order of types matches order of columns.* *Types: count, sum, min, max, avg, none.*
tolerances	<List of Tolerances for Numerical Value Here> • *Only apply to numerical values in source/target columns.* • *Order of tolerances reflects order of numerical values.* • *Do not need to write none for non-numeric values*
verbose	<"on" Or "off" Here> • *Lists all the key mismatches again with each individual attribute in the attributes mismatches table on the output file.*
email	<"on" Or "off" Here>

	• *Toggle email output feature.*
subject_line	<Email Subject Line Here> • *Ex: Rec Tool Output.*
to_addresses	<To Addresses For Email Here> • *One value per cell. Unlimited number of cells.* • *Ex: email1 @site.com, email2 @example.com…*
from_addresses	<*Rec Tool Email Address Here*> • *Ex: rectool@example.com*

Full Example: Below is an example of the file with all fields filled in.

	A	B	C	D	E	F	G	H	I
1	Field Name (DO NOT EDIT)	Comments			Field Values				
2			Input Type			Input File		Server	Database
3	source	Values for source input	excel	Enter source excel file	pools.xlsx				
4	target	Values for target input	excel	Enter target excel file	loans.xlsx				
5									
6	source_key_columns	Names of key columns for source table	name	owner					
7	target_key_columns	Names of key columns for source table	pool_name	pool_owner					
8									
9	source_columns	Names of attribute columns for source table	seller	num_loans	total_loan_value				
10	source_aggregation_types	Aggregation types for source table							
11	target_columns	Names of attribute columns for target table	pool_seller	loan_name	loan_value				
12	target_aggregation_types	Aggregation types for target table	none	count	sum				
13	tolerances	Tolerances for numeric attributes (none for non-numeric attributes)	none	0	1%				
14									
15	verbose	Output additional attribute information for keys that don't have a match	on						
16	email	If set to "on" will email a link to the exceptions file generated. "off" otherwise	off						
17	subject_line	Subject line of email	Rec Tool						
18	to_addresses	To: addresses for email	email@example.com						
19	from_address	From: address for email	rectool@example.com						
20									
21									
22									
23									
24									
25									
26									
27									
28									
29									
30									
31									
32									
33									
34									
35									

Config / Source Translation / Target Translation / Server Data / +

Translation Table

Description: The translation table provides an opportunity for known differences between databases to avoid being output as mismatches. By entering the corresponding data in the translation_table.xlsx file, the user can have the program ignore known mismatches between the databases. It is located on the "Translation" sheet of the input file.

Important Notes:
1. Case and non-alphanumeric values matter. This is to avoid incorrect translations.
2. The translation table only handles one to one translations. In the full example below, "beatles" should only translate to "ABBEY ROAD", not to an additional value as well, such as "ABBEY ROAD" and "Richard".
3. The tab "Source" is for translating values in the source to values in the target data set
4. The tab "Target" is for translating values in the target to values in the source data set.
5. Both "Source" and "Target" tabs have the same format.

Example:

TAB NAME: Data set where the value is located

Column	Source Value	Target Value
Name of column where value is located	Value you want to translate	Value you want to translate to

Full Example: Below is an example of the file with all fields filled in.

Column	Source Value	Target Value
pool_name	beatles	ABBEY ROAD
pool_seller	Dick	Richard
pool_seller	Jake	Jacob

Running the Program

Description: The whole program can be run from the terminal or by double clicking the .bat file named rectool.bat

Important Notes:
- If you encounter any errors, there are two logs that will give a detailed message as to why the error was encountered
    - Rec Tool Log: rectool_log.txt
    - SQL Parser Log: sqlparser_log.txt

Description: The output file consists of four tabs: Key Exceptions, Exceptions, Target, and Source. The exceptions tab displays any mismatches in data found between the two data sets that were input. The key exceptions are listing in a table on the first tab, and attribute exceptions are listed in a table on the second tab. The Source tab has all the values in the source data set that had matches found in the target data set. Similarly the Target tab has all the values in the target data set that had matches in the source.

Important Notes:
- The verbose option in the configuration file changes how much data is displayed in the output. If verbose is on, then any key that has no match, or no close match (Style, Case) all its attributes are listed in the attribute mismatch table. If verbose is off, the attributes are not listed in the table below, and the complete mismatch is simply noted in the top table containing key mismatches.

Examples:

Consider these example source and target sets of data as well as their output file.

	A	B	C	D	E	F
1	pool_id	name	owner	seller	num_loans	total_loan_value
2	1	MERCURY	Bob	Richard	3	100
3	2	MERCURY	Mary	Glenn	2	500
4	3	PREAKNESS	John	Jacob	4	300
5	4	PREAKNESS	Dave	Mark	2	60
6	5	KENTUCKY	Ryan	Carol	2	250
7	6	ABBEY ROAD	Bob	Richard	3	450
8	7	BELMONT	Frank	Lenny	1	80
9	8	BELMONT	Jim	Larry	1	30

Figure 43: Example Source

	A	B	C	D	E	F
1	loan_id	loan_name	pool_name	pool_owner	pool_seller	loan_value
2	1	Mercury 1	mercury	bob	Richard	20
3	2	Mercury 2	mercury	bob	Richard	30
4	3	Mercury 3	mercury	bob	Dick	45
5	4	Mercury 4	mercury	Mary	glenn	250
6	5	Mercury 5	mercury	Mary	Glenn	250
7	6	Preakness 1	Preakness	John	jake	100
8	7	Preakness 2	Preakness	John	Jacob	100
9	8	Preakness 3	Preakness	John	Jacob	100
10	9	preakness 4	preakness	Dave C.	Marcus	40
11	10	preakness 5	preakness	Dave C.	Mark	19
12	11	kentucky 1	kentucky	Ryan	Carol	125
13	12	kentucky 2	kentucky	Ryan	Carol	126
14	13	beatles 1	beatles	bob	Richard	150
15	14	beatles 2	beatles	bob	Dick	150
16	15	beatles 3	beatles	bob	richard	151.1
17	16	Belmont 1	BELMONT	Frank	Lenny	78.9
18	17	Belmont 2	BEMLONT	Jim	Larry	30
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						
29						
30						
31						
32						

Ex Target +

Figure 44: Example Target

	A	B	C
1	Source Key: name; owner (pools.xlsx)	Target Key: pool_name; pool_owner (loans.xlsx)	Error Types
2	MERCURY; Bob	mercury; bob	Case Mismatch; Case Mismatch
3	MERCURY; Mary	mercury; Mary	Case Mismatch; Match
4	PREAKNESS; John	Preakness; John	Case Mismatch; Match
5	ABBEY ROAD; Bob	ABBEY ROAD; bob	Match; Case Mismatch
6	PREAKNESS; Dave	preakness; Dave C.	Case Mismatch; Partial Match
7	BELMONT; Jim		In Source Not Target
8		BEMLONT; Jim	In Target Not Source

Figure 45: Key Output From Example Data

	A	B	C	D	E	F	G	H
1	Source Key: name; owner (pools.xlsx)	Target Key: pool_name; pool_owner (loans.xlsx)	Source Attribute	Target Attribute	Source Value	Target Value	Error Type	Diff
2	MERCURY; Mary	mercury; Mary	seller	pool_seller	Glenn	glenn	Case Mismatch	
3	ABBEY ROAD; Bob	beatles; bob	seller	pool_seller	Richard	richard	Case Mismatch	
4	PREAKNESS; Dave	preakness; Dave C.	seller	pool_seller	Mark	Marcus	No Match	
5	BELMONT; Jim		seller		Larry		In Source Not Target	
6		BEMLONT; Jim		pool_seller		Larry	In Target Not Source	
7	MERCURY; Bob	mercury; bob	num_loans	count(loan_name)	4.00	3.00	Numeric Mismatch (tolerance: 0.0)	-25.00%
8	BELMONT; Jim		num_loans		1.00		In Source Not Target	
9		BEMLONT; Jim		count(loan_name)		1.00	In Target Not Source	
10	MERCURY; Bob	mercury; bob	total_loan_value	sum(loan_value)	100.00	95.00	Numeric Mismatch (tolerance: 1.0%)	-5.00%
11	PREAKNESS; John	Preakness; John	total_loan_value	sum(loan_value)	80.00	78.90	Numeric Mismatch (tolerance: 1.0%)	-1.37%
12	PREAKNESS; John	Preakness; John	total_loan_value	sum(loan_value)	60.00	59.00	Numeric Mismatch (tolerance: 1.0%)	-1.67%
13	BELMONT; Jim		total_loan_value		30.00		In Source Not Target	
14		BEMLONT; Jim		sum(loan_value)		30.00	In Target Not Source	

Figure 46: Attribute Exceptions Output From Example

	A	B	C	D
1	name; owner (pools.xlsx)	seller	num_loans	total_loan_value
2	MERCURY; Bob	Richard	3	100
3	MERCURY; Mary	Glenn	2	500
4	PREAKNESS; John	Jacob	4	300
5	KENTUCKY; Ryan	Carol	2	250
6	ABBEY ROAD; Bob	Richard	3	450
7	BELMONT; Frank	Lenny	1	80
8	PREAKNESS; Dave	Mark	2	60
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				

Exceptions | **Source** | Target | +

Figure 47: Source Tab

9.2   Software Document

Python Version 2.7.10
Description: This is the version of Python used.
Location: https://www.Python.org/downloads/
Important Notes:
- ConfigParser in 2.7.10 → configparser in 3.5.0

Openpyxl
Description: This package is necessary for the manipulation of Excel files.
Download Instructions: Can be downloaded through pip.

```
Don't have Pip
    Install Pip
    Go to Terminal
    Type "sudo easy_install pip"
    Follow "Have Pip" instructions

Have Pip
    Go to Terminal
    type "pip install openpyxl"
```
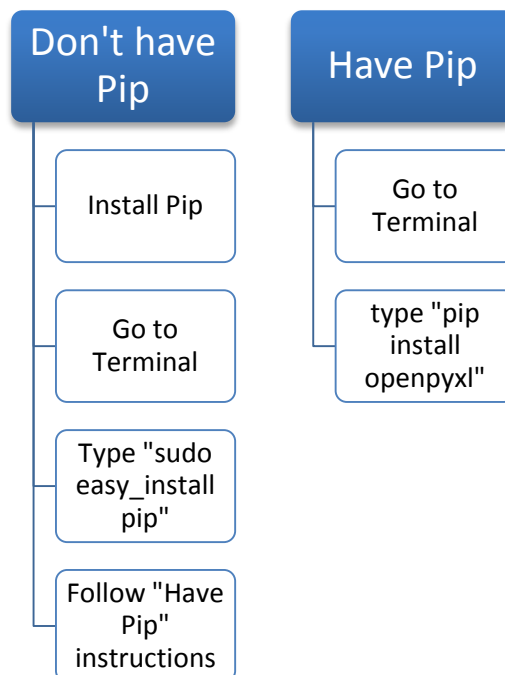
Input File Format
Description: The import file is an Excel file with the configuration info on the first sheet and translation table on the second. For example, the document should be saved as mysample.xlsx
More info: See the program setup guide.

Mail Server
Description: In order to send the e-mail with the attached CSV, a mail server must be set up on the computer. If there is none, SendMail was used while building the program.
Location: https://www.sendmail.com/sm/open_source/
Important Notes:
- This link is to the open source version.

- If you encounter problems and an email server is installed, ensure you have permissions to access it.

PYODBC: Server Package

Description: pyodbc makes it easy to connect to databases using ODBC. It implements the DB API 2.0 specification, but adds even more convenience. The library is designed to be stand-alone. It does not have other dependencies and only uses Python's built-in data types.

More Info: https://github.com/mkleehammer/pyodbc

Download Instructions:

```
Don't have Pip
    Install Pip
    Go to Terminal
    Type "sudo easy_install pip"
    Follow "Have Pip" instructions

Have Pip
    Go to Terminal
    type "pip install pyodbc"
```
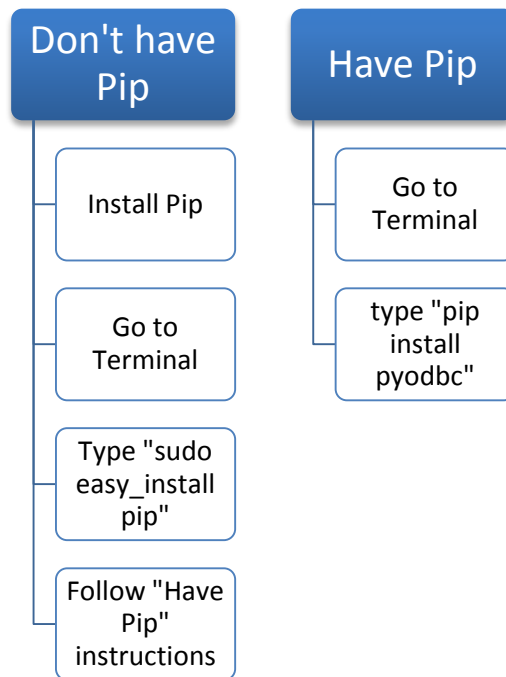
9.3   Use Case Document

The use case documentation is uploaded in separate documents along with the report.

10 APPENDIX B: WEEKLY STATUS REPORTS

Angelo Gordon Team 1
Week 1 Status Report

Prepared by	Richard O'Brien, Ben Sharron
Date	10/31/15
Reporting Period	10/25/2015 – 10/30/2015

Top Concerns and Outstanding Items

- Scheduling a meeting with Scott when he returns to work on Monday after having been out for 4 days the past week.
- Ensuring that we are combining intuitive usability for the end users with the functionality and vision Scott has for the project.
- Keeping the code flexible for potential changes in the future.
- Getting access to the databases (Need Scott).

Key Accomplishments and Decisions

- Met with Scott and defined the path for the project for the week ahead as well as the clarified the vision of the project long term.
  - The project will be building a quick and simple reconciliation tool for loan data taxonomies between databases. The first step is creating a rec tool for the pool names between databases, then drilling down on the checks within the database. This program would be run regularly and emails out a report of problems for people them to go fix.
- Met with Andrew (in Accounting) to get feed back on how the users want the outputted file to look.
- Started creating the program running off local servers hosted on our laptops. Program currently…
  - Checks loan pool names, identifies, and labels the mismatches in the field.
  - Can rec multiple fields at once. Ex pool names and owner of pool below that. Checks, identifies, and labels mismatches on both levels.
  - Can rec numerical values and match them to those in another database within a tolerance.
  - Outputs mismatches to a CSV file in the format specified by asking users.
  - Emails the CSV file to users.
  - Code is remaining flexible and program is run from terminal.
- On the Saturday before work, visited Time Square, resisted impulse buying in the Toys R Us store, the Disney store and the M&M's store.

Open or Deferred Tasks

- Change the database handler to be work with their specific SQL server. Currently using SQLite and running of a locally hosted server on our laptops.
- Get access to data bases to see how the data is structured.

Planned Activities

- Get Access to databases
- Meet with Scott and get his input of current state of the program

Angelo Gordon Team 1
Week 2 Status Report

Members	Richard O'Brien, Ben Sharron
Date	11/04/15
Reporting Period	11/01/2015 – 11/04/2015

Top Concerns and Outstanding Items

- Ensuring that we are keeping input files intuitive and usable.
- Keeping the code flexible for potential changes in the future.
- Getting access to the databases

Key Accomplishments and Decisions

- Had a meeting with Scott when he returned on Monday from being sick.
  - Updated him with current progress.
  - Got his opinion on how to format input documents.
  - Discussed path-moving forwards. Next step is testing with real data.
- Changed the Quick Rec Tool to be able to function with Excel files as input to it as well as connect to databases.
- Added aggregation rec functionality. (Untested)
  - Can now reconcile between tables where one table has aggregations of another.
  - Ex. One table has loan values; the other has total value of the pool. Loan value → total value
- Got excel files with taxonomies of actual loan data in the system.
  - Ran a reconciliation on real data for first time.
    - Worked as planned on a high level.
    - Found some new errors.
- Added error reporting to users.
  - If something is input incorrectly in one of the input files, users are told what fields are incorrect.
- Began documentation process.
  - Created a document listing what software packages are needed to run the Quick Rec Tool.
    - Python 2.7.10, permissions to mail server, etc…
  - Created documentation with detailed instructions on setting up input files.
  - Acquired software for instructional video documentation (Camtasia).
- Setup outline of report, and began intro/background and other sections.

Open or Deferred Tasks

- Change the database handler to be work with their specific SQL server. Currently using SQLite and running of a locally hosted server on our laptops.
- Get access read only access to databases.
- Install Camtasia on windows machine.

Planned Activities

- Get Access to databases.
- Fix newly found errors.
  - Fixing error where names with #'s behind them in a target table only have the first instance matching to all instances in the source.
    - Ex. Mercury, Mercury, Mercury in source and Mercury 1, Mercury 2, Mercury 3 in the target.
    - Mercury matches to Mercury 1
    - Mercury Matches to Mercury 1
    - Mercury Matches to Mercury 1
    - Mercury 2 and Mercury 3 considered "Missing Names."
- Begin video documentation.
- Test aggregation functionality.
- Update Documentation to reflect addition of aggregation functionality.

Angelo Gordon Team 1
Week 3 Status Update

Members	Richard O'Brien, Ben Sharron
Date	11/11/15
Reporting Period	11/05/2015 – 11/11/2015

Top Concerns and Outstanding Items

- Ensuring that we are keeping input files intuitive and usable.
- Keeping the code flexible for potential changes in the future.
- The format of input and outputs (excel seems to be preferred (all in one workbook or 2 different), outputs to csv, outputs in new tab of workbook)

Key Accomplishments and Decisions

- Checked with Scott about thanksgiving.
    - Wednesday half day, Friday off.
- Had regular meetings with Scott as we completed keystone parts of the application.
    - Updated him with current progress.
    - Got his opinion on path forwards.
    - Next step is changing config file to also be an excel file.]
    - No need for currency translations.
- Met with Andrew
    - Got opinions on usability, and how intuitive inputs were.
    - Suggested changing translation handler to excel.
- Overhauled translation table handler.
    - Changed the input format from excel to .ini.
    - Better for ease of use, although had to completely change the handler and edit the main function.
    - Changed translation table to be case sensitive so as to avoid incorrect translations.
- Refactored main function code to increase readability and avoid repetitiveness.
- Updated error reporting to users and commenting in code.
- Continue documentation process.
    - Learn Camtasia.
    - Update paper documents (config and translation docs).
- Continued writing report.

Open or Deferred Tasks

- Change the database handler to be work with their specific SQL server. Currently using SQLite and running of a locally hosted server on our laptops.

Planned Activities

- Continue testing and debugging on actual data.
    - Continue expanding functionality based on findings
- Begin video documentation.
- Update Documentation to reflect changes in translation handler
- Test with databases
- Potentially change config file to an Excel file as well
    - Put both config and translation table in same file for convenience.

Angelo Gordon Team 1
Week 4 Status Update

Members	Richard O'Brien, Ben Sharron
Date	11/18/15
Reporting Period	11/12/2015 – 11/18/2015

Top Concerns and Outstanding Items

- Finalizing the configuration table input
- Literature review necessary for report? (Notice the current report on our basecamp)
- Keeping the code flexible.
- The format of input and outputs (excel seems to be preferred (all in one workbook or 2 different), outputs to csv, outputs in new tab of workbook)

Key Accomplishments and Decisions

- Changed config file from an .ini to an Excel file input.
  - Changed code base to allow for the new file format to be read and dealt with appropriately.
- Combined translation and config files
  - Used to be in two separate workbooks
  - Now in same workbook on separate sheets.
- Added user feedback print statements
  - If the program is running for a long time, you know what it is doing. See Page 2 for Example.
- Converted error reporting to a log.
  - See images on page 2.
- Added user input to exit the program. "Press Enter." Also see page 2.
- Finalized layout of translation table.
- Adjusted layout of configuration table to sponsors recommendations.
  - Changed layout of the cells to be more intuitive
  - Added data validation to reduce user error.
- Added percentage functionality for reconciling with a tolerance. Ex: is it within 10% of source value.
- Added a where clause to the configuration table. Ex: daily rec can be done "as of yesterday" every day
- Continue documentation process.
  - Update paper documents (config and translation docs).
- Had regular meetings with Scott as we completed keystone parts of the application.
  - Updated him with current progress.
  - Got his opinion on path forwards.
- Met with Andrew for user interface/ Other suggestions
- Continued writing report.

Open or Deferred Tasks

- Changing the output file from a CSV to an Excel so that we can add a data dump the sponsor wants.
  - Returns mismatches on one sheet, source data on another, target on another

Planned Activities

- Continue testing and debugging on actual data.
  - Continue expanding functionality based on findings/fixing bugs.
- Finalize the configuration input
  - Finalize documentation once this is done.
- Continue meeting with sponsor for additional recommendations.

Angelo Gordon Team 1
Weeks 5&6 Status Update

Members	Richard O'Brien, Ben Sharron
Date	12/3/15
Reporting Period	11/19/2015 – 12/3/2015

Top Concerns and Outstanding Items

- Literature review necessary for report?
- Reducing run time of the program.
- Finalizing input so can preform video documentation.

Key Accomplishments and Decisions

- Added Preprocessor
    - Have many long SQL queries with a lot of joins and all that jazz.
    - Input is an SQL query, output is an excel file formatted to be an input to Quick Rec Tool.
- Added Capability for email to tell how many exceptions are in the file
    - Tells user in subject line
- Finalized layout of translation table.
- Adjusted layout of configuration table to sponsors recommendations.
    - Moved things around slightly
    - Made all database files (SQL) all automatically run through the preprocessor and become excel.
- Continue documentation process.
    - Created use case documentation. **See use case file in base camp.**
    - Updated paper documents (config and translation docs).
- Had regular meetings with Scott as we completed keystone parts of the application.
    - Updated him with current progress.
    - Got his opinion on path forwards.
- Changed output file to having two tabs for attribute and key mismatches.
- Drastically reduced the run time by changing logic in how key pairs are checked within code.
- Fixed bugs with translations as well as email functionality.
- Completed most of the final presentation
- Continued writing report.

Open or Deferred Tasks

- Reducing run time.
- Finalizing configuration file.

Planned Activities

- Continue testing and debugging on actual data.
    - Continue expanding functionality based on findings/fixing bugs.
- Complete video documentation

Angelo Gordon Team 1
Week 7 Status Update

Members	Richard O'Brien, Ben Sharron
Date	12/9/15
Reporting Period	12/3/2015 – 12/9/2015

Top Concerns and Outstanding Items
- Getting the report edited.
- Preparing for final presentation.
- Deciding whether or not to do video documentation.

Key Accomplishments and Decisions
- Finalized layout of Configuration file.
- Continue documentation process.
  - Finished use case documentation.
  - Updated paper documents.
- Had regular meetings with Scott as we tested the program.
  - Performed bug fixes.
- Reduced the run time
- Completed most of the final presentation.
  - Went through it with professors Ciaraldi and Sweeney.
  - Presentation has been shared with all advisors.
- Continued writing report.
  - Internally edited.
  - Sent to Advisors for their comments.
  - Finished all major sections.

Open or Deferred Tasks
- Preparing for final presentation.
- Finalizing report.

Planned Activities
- Continue testing and debugging on actual data.
- Complete video documentation.
- Preparing for final presentation.
- Finalizing report.

11 APPENDIX C: CODE FOR QUICK REC TOOL

Rectool (Main Function)

```python
#!/usr/local/bin/Python
"""
Main method and rec checks.

Angelo Gordon
Ben Sharron and Richard O'Brien
Rec Tool
"""

import re
import sys
import os
import logging
import getpass
from sqlparser import SqlParser
from rec_lib.config_handler import ConfigHandler
from rec_lib.input_handler import InputHandler
from rec_lib.exception_handler import ExceptionHandler
from rec_lib.output_handler import OutputHandler
from rec_lib.email_handler import EmailHandler


def _recursive_string_check(element):
    """Check for two levels of strings."""
    if isinstance(element, str) or isinstance(element, unicode):
        return True
    elif isinstance(element, list):
        return _recursive_string_check(element[0])
    else:
        return False


def _strip_aggr(element):
    """Strip aggr(col) to col."""
    pattern = re.compile(r'(.*\(|\))')
    return pattern.sub('', element)


def _to_boolean(element):
    """Convert config string to boolean, raising an error on bad input."""
    true_vals = ('on', 'true', 'yes', 1)
    false_vals = ('off', 'false', 'no', 0)

    if element in true_vals:
        return True
```

```python
        elif element in false_vals:
            return False
        else:
            raise IOError('%s must be one of on, off, yes, no, '
                    'true, false, 0 or 1.', element)


def _parse_tol(tol):
    """Turn inputted tolerance value into a dictionary object."""
    if tol[-1] == '%':
        return {'type': 'percent', 'value': float(tol[0:-1])}
    elif tol.lower() != 'none':
        return {'type': 'number', 'value': float(tol)}
    else:
        return None


def _strip_filename(filename):
    """Strip extension off inputted filename."""
    pattern = re.compile(r'(.*[\\/])|(\..*)')
    return pattern.sub('', filename)


def _join_columns(col_list, sep):
    """Convert a list of list of strings to a joined list of strings."""
    return [sep.join(column) for column in col_list]


def _get_source_input_file(config):
    """Helper function to determine input for source."""
    source_input_type = config.get('source_input_type')
    source_file = config.get('source_file')

    # Perform database-style check if source input type is database
    if source_input_type == 'database':
        print 'Connecting to Source server...'
        logging.info('Connecting to Source server...')
        source_server = config.get('source_server')
        source_database = config.get('source_database')

        # Parse SQL file and retrieve Excel file
        source_input_file = SqlParser(
            source_server, source_database, source_file, None).execute()
    # Perform Excel-style check if source input type is Excel
    elif source_input_type == 'Excel':
        source_input_file = source_file
    else:
        raise IOError('Source input type is neither Excel nor database: %s',
                source_input_type)

    return source_input_file
```

```python
def _get_target_input_file(config):
    """Helper function to determine input for target."""
    target_input_type = config.get('target_input_type')
    target_file = config.get('target_file')

    # Perform database-style check if target input type is database
    if target_input_type == 'database':
        print 'Connecting to Target server...'
        logging.info('Connecting to Target server...')
        target_server = config.get('target_server')
        target_database = config.get('target_database')

        # Parse SQL file and retrieve Excel file
        target_input_file = SqlParser(
            target_server, target_database, target_file, None).execute()
    # Perform Excel-style check if target input type is Excel
    elif target_input_type == 'Excel':
        target_input_file = target_file
    else:
        raise IOError('Target input type is neither Excel nor database: %s',
                target_input_type)

    return target_input_file


def main():
    """Main method where program execution begins."""
    # Set up logger
    open('rectool_log.txt', 'w').close()
    logging.basicConfig(filename='rectool_log.txt', filemode='w',
                level=logging.INFO)

    # Parse config filename from command line argument
    if len(sys.argv) == 2:
        config_file = sys.argv[1]
    else:
        raise IOError('Expected 1 command line argument, received %s: %s',
                str(len(sys.argv) - 1), str(sys.argv[1:]))

    # Set up config handler
    config = ConfigHandler(config_file)

    source_key_columns = config.get_list('source_key_columns')
    target_key_columns = config.get_list('target_key_columns')
    source_columns = config.get_list('source_columns')
    target_columns = config.get_list('target_columns')
    tolerances = config.get_list('tolerances')

    email = _to_boolean(config.get('email'))
    verbose = _to_boolean(config.get('verbose'))

    print 'Reading source input file...'
    logging.info('Reading source input file...')
```

```
source_input_file = _get_source_input_file(config)
source_handler = InputHandler(source_input_file)
source_key_columns_str = \
    '; '.join(source_key_columns) + ' ({})'.format(source_input_file)

print 'Reading target input file...'
logging.info('Reading target input file...')
target_input_file = _get_target_input_file(config)
target_handler = InputHandler(target_input_file)
target_key_columns_str = \
    '; '.join(target_key_columns) + ' ({})'.format(target_input_file)

output_file = 'EXCEPTIONS_{}.xlsx'.format(_strip_filename(config_file))
output_handler = OutputHandler(output_file)

source_aggregation_types = config.get_list('source_aggregation_types')
target_aggregation_types = config.get_list('target_aggregation_types')

source_aggr_on = len(source_aggregation_types) > 0
target_aggr_on = len(target_aggregation_types) > 0

# Generate headers
key_header_list = ['Source Key: {}'.format(source_key_columns_str),
          'Target Key: {}'.format(target_key_columns_str),
          'Error Types']

attr_header_list = ['Source Key: {}'.format(source_key_columns_str),
          'Target Key: {}'.format(target_key_columns_str),
          'Source Attribute',
          'Target Attribute',
          'Source Value',
          'Target Value',
          'Error Type',
          'Diff']

print 'Retrieving key columns...'
logging.info('Retrieving key columns...')
source_key_columns_lists = \
    source_handler.get_key_columns(source_key_columns)
target_key_columns_lists = \
    target_handler.get_key_columns(target_key_columns)

# Translation handler
trans_handler = config.get_translation_handler()

print 'Translating key values...'
logging.info('Translating key values...')
trans_source_key_columns_lists = trans_handler.translate_source_multiple(
    source_key_columns, source_key_columns_lists)
trans_target_key_columns_lists = trans_handler.translate_target_multiple(
    target_key_columns, target_key_columns_lists)

# Set up key name exception handler, retrieve exceptions, and output them
```

```python
print 'Retrieving key exceptions...'
logging.info('Retrieving key exceptions...')
exc_handler = ExceptionHandler()
exc_handler.retrieve_key_exceptions(trans_source_key_columns_lists,
                    trans_target_key_columns_lists)

print "Outputting key exceptions to file '{}'...".format(output_file)
logging.info("Outputting key exceptions to file '%s'...", output_file)
key_exc_list = exc_handler.get_key_exc_list()

if len(key_exc_list) > 0:
    output_handler.output_key_header(key_header_list)
    output_handler.output_key_exceptions(key_exc_list)

# Un-translate key pairing into original values
exc_handler.untranslate(source_key_columns_lists,
            target_key_columns_lists,
            trans_source_key_columns_lists,
            trans_target_key_columns_lists)

# Extract individual primary key lists
(paired_source_key_list, paired_target_key_list) = (
    exc_handler.split_key_pairing())

# Output core data
output_handler.output_source_column(
    source_key_columns_str, _join_columns(paired_source_key_list, '; '))
output_handler.output_target_column(
    target_key_columns_str, _join_columns(paired_target_key_list, '; '))

# Check if column_name and target_name lists are same length
if len(source_columns) != len(target_columns):
    raise IOError(
        'An error has occurred in source_columns or target_columns. '
        'Either there is an extra value or a missing value.')

# Iterate through attributes
for source_col_name, target_col_name in zip(
        source_columns, target_columns):
    # Get relevant source columns (aggregating if necessary)
    print 'Getting values from {} table for attribute: {}'.format(
        source_input_file, source_col_name)
    logging.info('Getting values from %s table for attribute: %s',
            source_input_file, source_col_name)

    if source_aggr_on:
        try:
            src_aggr_type = source_aggregation_types.pop(0)
        except IndexError:
            print 'WARNING: Missing aggregation type for attribute {} ' \
                'for the source table. Assuming none.'.format(
                    source_col_name)
            logging.warning(
```

```python
            'WARNING: Missing aggregation type for attribute %s '
            'for the source table. Assuming none.', source_col_name)
        src_aggr_type = 'none'

    if source_aggr_on and src_aggr_type != 'none':
        source_attr_columns_list = (
            source_handler.get_column_with_aggregation(
                paired_source_key_list, source_col_name, src_aggr_type))

        source_col_name = '{}({})'.format(src_aggr_type, source_col_name)
    else:
        source_attr_columns_list = source_handler.get_column(
            paired_source_key_list, source_col_name)

    # Get relevant target columns (aggregating if necessary)
    print 'Getting values from {} table for attribute: {}'.format(
        target_input_file, target_col_name)
    logging.info('Getting values from %s table for attribute: %s',
            target_input_file, target_col_name)

    if target_aggr_on:
        try:
            tgt_aggr_type = target_aggregation_types.pop(0)
        except IndexError:
            print 'WARNING: Missing aggregation type for attribute {} ' \
                'for the target table. Assuming none.'.format(
                    target_col_name)
            logging.warning(
                'WARNING: Missing aggregation type for attribute %s '
                'for the target table. Assuming none.', target_col_name)
            tgt_aggr_type = 'none'

    if target_aggr_on and tgt_aggr_type != 'none':
        target_attr_columns_list = (
            target_handler.get_column_with_aggregation(
                paired_target_key_list, target_col_name, tgt_aggr_type))

        target_col_name = '{}({})'.format(tgt_aggr_type, target_col_name)
    else:
        target_attr_columns_list = target_handler.get_column(
            paired_target_key_list, target_col_name)

    # Check if attributes are strings
    try:
        source_is_str = _recursive_string_check(source_attr_columns_list)
    except:
        raise IOError('Error accessing attribute %s in source table.',
                source_col_name)

    try:
        target_is_str = _recursive_string_check(target_attr_columns_list)
    except:
        raise IOError('Error accessing attribute %s in target table.',
```

```python
        target_col_name)

# Get tolerance
try:
    tol = _parse_tol(tolerances.pop(0))
except (TypeError, IndexError):
    tol = None

# If the attributes are strings, translate and retrieve exceptions
if source_is_str and target_is_str:
    if tol is not None:
        print 'WARNING: tolerance {} given for non-numeric ' \
            'attributes: {} and {}'.format(
                str(tol['value']), source_col_name, target_col_name)
        logging.warning(
            'WARNING: tolerance %s given for non-numeric attributes: '
            '%s and %s', str(tol['value']),
            source_col_name, target_col_name)
        tol = None

    # Translate attribute exceptions
    if source_aggr_on:
        for index, attr_list in enumerate(source_attr_columns_list):
            source_attr_columns_list[index] = (
                trans_handler.translate_source(source_col_name,
                            attr_list))
    else:
        source_attr_columns_list = trans_handler.translate_source(
            source_col_name, source_attr_columns_list)

    if target_aggr_on:
        for index, attr_list in enumerate(target_attr_columns_list):
            target_attr_columns_list[index] = (
                trans_handler.translate_target(target_col_name,
                            attr_list))
    else:
        target_attr_columns_list = trans_handler.translate_target(
            target_col_name, target_attr_columns_list)
# Attribute types don't match
elif source_is_str or target_is_str:
    raise IOError('Corresponding attributes {} and {} are not '
            'of the same type.'.format(
                source_col_name, target_col_name))
# Check tolerance for None when comparing numbers
else:
    if tol is None:
        print 'WARNING: No tolerance given for numeric attributes: '
        '{} and {}. Using default of 0.'.format(
            source_col_name, target_col_name)
        logging.warning(
            'WARNING: No tolerance given for numeric attributes: '
            '%s and %s. Using default of 0.',
            source_col_name, target_col_name)
```

```
        tol = {'type': 'number', 'value': 0}

print 'Retrieving attribute exceptions for attributes: ' \
    '{} and {}'.format(source_col_name, target_col_name)
logging.info(
    'Retrieving attribute exceptions for attributes: %s and %s',
    source_col_name, target_col_name)

# Retrieve attribute exceptions
exc_handler.retrieve_exceptions(source_col_name,
                    target_col_name,
                    source_attr_columns_list,
                    target_attr_columns_list,
                    tol)

# Output core data
if source_is_str and source_aggr_on:
    output_handler.output_source_column(
        source_col_name, _join_columns(source_attr_columns_list, '; '))
else:
    output_handler.output_source_column(
        source_col_name, source_attr_columns_list)

if target_is_str and target_aggr_on:
    output_handler.output_target_column(
        target_col_name, _join_columns(target_attr_columns_list, '; '))
else:
    output_handler.output_target_column(
        target_col_name, target_attr_columns_list)

# If verbose mode, add additional info for missing or extra names
if verbose:
    # Strip output fluff off to use source handler again
    source_col_name = _strip_aggr(source_col_name)

    # Get missing name list for this attribute
    missing_names = exc_handler.get_missing_names()

    if len(missing_names) > 0:
        # Check for aggregation
        if source_aggr_on and src_aggr_type != 'none':
            missing_name_vals = \
                source_handler.get_column_with_aggregation(
                    missing_names,
                    source_col_name,
                    src_aggr_type)

            source_col_name = '{}({})'.format(
                src_aggr_type, source_col_name)
        else:
            missing_name_vals = source_handler.get_column(
                missing_names, source_col_name)
```

```python
        # Attribute is a string
        if _recursive_string_check(missing_name_vals):
            # Translate attributes
            if source_aggr_on and src_aggr_type != 'none':
                for index, attr_list in enumerate(missing_name_vals):
                    missing_name_vals[index] = \
                        trans_handler.translate_source(
                            source_col_name, attr_list)
            else:
                missing_name_vals = trans_handler.translate_source(
                    source_col_name, missing_name_vals)

        # Retrieve exceptions for missing names
        exc_handler.retrieve_missing_name_exc(source_col_name,
                             missing_name_vals)

# Strip output fluff off to use target handler again
target_col_name = _strip_aggr(target_col_name)

# Get extra name list for this attribute
extra_names = exc_handler.get_extra_names()

if len(extra_names) > 0:
    # Check for aggregation
    if target_aggr_on and tgt_aggr_type != 'none':
        extra_name_vals = \
            target_handler.get_column_with_aggregation(
                extra_names,
                target_col_name,
                tgt_aggr_type)

        target_col_name = '{}({})'.format(
            tgt_aggr_type, target_col_name)
    else:
        extra_name_vals = target_handler.get_column(
            extra_names, target_col_name)

    # Attribute is a string
    if _recursive_string_check(extra_name_vals):
        # Translate attributes
        if target_aggr_on and tgt_aggr_type != 'none':
            for index, attr_list in enumerate(extra_name_vals):
                extra_name_vals[index] = \
                    trans_handler.translate_target(
                        target_col_name, attr_list)
        else:
            extra_name_vals = trans_handler.translate_target(
                target_col_name, extra_name_vals)

    # Retrieve exceptions for extra names
    exc_handler.retrieve_extra_name_exc(target_col_name,
                         extra_name_vals)
```

```python
    # Output exceptions for attributes
    if len(source_columns) > 0:
        print "Outputting attribute exceptions to file '{}'...".format(
            output_file)
        logging.info("Outputting Attribute Exceptions to file '%s'...",
                output_file)
        attr_exc_list = exc_handler.get_attr_exc_list()

        if len(attr_exc_list) > 0:
            output_handler.output_attr_header(attr_header_list)
            output_handler.output_attr_exceptions(attr_exc_list)

    # Get number of exceptions found
    num_exceptions = exc_handler.get_num_exceptions()

    # Set up email handler and send email if EMAIL is turned on
    if email:
        print 'Configuring and sending email...'
        logging.info('Configuring and sending email...')
        subject_line = config.get('subject_line')
        to_addresses = config.get_list('to_addresses')
        from_address = '{}@angelogordon.com'.format(getpass.getuser())
        body_filename = 'email_body.txt'

        script_dir = os.path.dirname(__file__)
        body_file = os.path.join(script_dir, body_filename)

        mail_server = 'mail.angelogordon.com'

        if num_exceptions > 0:
            subject_line += ': Found {} exceptions!'.format(
                str(num_exceptions))
        else:
            subject_line += ': No exceptions found.'

        email = EmailHandler(mail_server)

        email.add_file(output_file)
        email.send(from_address, to_addresses, subject_line, body_file)

if __name__ == '__main__':
    # pylint: disable=broad-except
    try:
        main()
    except Exception:
        ERROR = sys.exc_info()[1]
        logging.error(ERROR)
        print 'An error has occurred. ' \
            'Check the log for more info: rectool_log.txt'
    finally:
        raw_input('Hit Enter to exit.')
```

SQL Parser

```python
#!/usr/local/bin/Python
"""
Main method and SQL parsing.

Angelo Gordon
Ben Sharron and Richard O'Brien
SQL Parser
"""

import re
import sys
import logging
from sql_lib.sql_handler import SQLHandler
from sql_lib.output_handler import OutputHandler


def _strip_filename(filename):
    """Strip extension of inputted filename."""
    pattern = re.compile(r'(.*[\\\/])|(\..*)')
    return pattern.sub('', filename)


class SqlParser(object):
    """Parse a SQL file and converts the result to an Excel output."""

    def __init__(self, server_name, database_name, sql_file, output_file):
        """Initialize SqlParser object and set up log file."""
        open('sqlparser_log.txt', 'w').close()
        logging.basicConfig(
            filename='sqlparser_log.txt', filemode='a', level=logging.INFO)

        self.server_name = server_name
        self.database_name = database_name
        self.sql_file = sql_file

        if output_file is None:
            self.output_file = '{}.xlsx'.format(_strip_filename(sql_file))
            print 'Using default output file: {}'.format(self.output_file)
            logging.info('Using default output file: %s', self.output_file)
        else:
            self.output_file = output_file

    def execute(self):
        """Execute SQL code and output as Excel file."""
        # Set up input and retrieve dictionary of results
        print 'Querying database...'
        logging.info('Querying database...')
        in_handler = SQLHandler(self.server_name, self.database_name)
        try:
            query_dict = in_handler.execute(self.sql_file)
```

```python
        except:
            error = sys.exc_info()[1]
            raise IOError(
                'Could not execute SQL statement in {} on server {}: '
                '{}'.format(self.sql_file, self.server_name, error))

        # Set up output handler
        print 'Outputting result...'
        logging.info('Outputting result...')
        out_handler = OutputHandler(self.output_file)

        for col_name, col_vals in query_dict.iteritems():
            print 'Outputting {}'.format(col_name)
            logging.info('Outputting %s', col_name)
            try:
                out_handler.output_column(col_name, col_vals)
            except:
                raise IOError(
                    'Could not output results to {}'.format(self.output_file))

        print 'Saving result in {}'.format(self.output_file)
        logging.info('Saving result in %s', self.output_file)
        try:
            out_handler.save()
        except:
            raise IOError(
                'Could not save results in {}'.format(self.output_file))

        return self.output_file

if __name__ == '__main__':
    # pylint: disable=broad-except
    try:
        if len(sys.argv) == 4:
            SERVER_NAME = sys.argv[1]
            DATABASE_NAME = sys.argv[2]
            SQL_FILE = sys.argv[3]
            OUTPUT_FILE = None
        elif len(sys.argv) == 5:
            SERVER_NAME = sys.argv[1]
            DATABASE_NAME = sys.argv[2]
            SQL_FILE = sys.argv[3]
            OUTPUT_FILE = sys.argv[4]
        else:
            raise IOError('Expected 3 or 4 command line arguments, '
                    'received {}: {}'.format(
                        str(len(sys.argv) - 1), str(sys.argv[1:])))

        SQL_FILE_NAME = _strip_filename(SQL_FILE)

        SqlParser(SERVER_NAME, DATABASE_NAME, SQL_FILE, OUTPUT_FILE).execute()
    except Exception:
        ERROR = sys.exc_info()[1]
```

```
    logging.error(ERROR)
    print 'An error has occurred.' \
        'Check the log for more info: sqlparser_log.txt'
finally:
    raw_input('Hit Enter to exit.')
```

SQL Parser SQL Handler

```python
#!/usr/local/bin/Python
"""
Performs statement-driven database retrieval given an SQL file as input.

Angelo Gordon
Ben Sharron and Richard O'Brien
SQL Handler module
"""

import pyodbc


class SQLHandler(object):
    """Handler for accessing database with a statement and return as a dict."""

    def __init__(self, server, database):
        """Initialize DB_Handler object."""
        self.server = server
        self.database = database

    def execute(self, sql_filename):
        """Read SQL file as input and execute SQL on saved server."""
        with open(sql_filename, 'r') as sql_file:
            query = sql_file.read()

        # Throttles non-results
        query = "SET NOCOUNT ON;\n" + query

        if self.database and len(self.database):
            conn = pyodbc.connect(
                'DRIVER={{SQL Server}};SERVER={};DATABASE={}'
                ';TrustedConnection=yes'.format(self.server, self.database))
        else:
            conn = pyodbc.connect(
                'DRIVER={{SQL Server}};SERVER={};TrustedConnection=yes'.format(
                    self.server))
        cursor = conn.cursor()

        result = {}

        for row in cursor.execute(query).fetchall():
            cols = [t[0] for t in cursor.description]

            for col, value in zip(cols, row):
                if col not in result:
                    result[col] = []
                result[col].append(value)
        conn.close()
        return result
```

SQL Parser Output Handler

```python
#!/usr/local/bin/Python
"""
Handles outputting values to an Excel file.

Angelo Gordon
Ben Sharron and Richard O'Brien
Output Handler module
"""

from openpyxl import Workbook


class OutputHandler(object):
    """Handler for outputting inputted columns to given Excel file."""

    def __init__(self, filename):
        """Initialize output state and workbook."""
        self.filename = filename
        self.out_col = 1
        self.workbook = Workbook()
        self.sheet = self.workbook.active
        self.sheet.title = 'SQL Output'

    def output_column(self, col_name, col_list):
        """Output a column of data into the given sheet."""
        row_num = 1

        self.sheet.cell(row=row_num, column=self.out_col).value = col_name

        for col_val in col_list:
            row_num += 1
            self.sheet.cell(row=row_num, column=self.out_col).value = col_val

        self.out_col += 1

    def save(self):
        """Save workbook."""
        self.workbook.save(self.filename)
```

Rec Tool Abstract Exceptions

```python
#!/usr/local/bin/Python
"""

Provides abstract functionality for exceptions.

Angelo Gordon
Ben Sharron and Richard O'Brien
Abstract Exception module
"""


class AbsException(object):
    """Generic abstract class to factor out exception methods."""

    _KEY_LIST = []

    def __init__(self, source_val, target_val, err_type):
        """Initialize Abstract Exception object."""
        self.exc = {
            'source_val': source_val,
            'target_val': target_val,
            'err_type': err_type
        }

    def get_exc_strings(self):
        """Return exception dictionary with list values converted to strings."""
        str_exc = {}

        for (key, value) in self.exc.iteritems():
            if isinstance(value, list):
                value = '; '.join(value)

            str_exc[key] = value

        return str_exc

    def add_vals(self, exc_dict):
        """Add new values to the exception object."""
        for (key, value) in exc_dict.iteritems():
            self.exc[key] = value

    @classmethod
    def get_keys(cls):
        """Return constant list of keys for exception object."""
        return cls._KEY_LIST
```

Rec Tool Attribute Exceptions

```python
#!/usr/local/bin/Python
"""
Object for exceptions found betweens attribute values.

Angelo Gordon
Ben Sharron and Richard O'Brien
Attribute Exception class
"""

from rec_lib.abs_exceptions import AbsException


class AttrException(AbsException):
    """An Exception found in an attribute."""

    _KEY_LIST = ['source_key', 'target_key', 'source_attr', 'target_attr',
            'source_val', 'target_val', 'err_type', 'diff']

    def __init__(self, source_val, target_val, err_type):
        """Initialize exception object."""
        super(AttrException, self).__init__(source_val, target_val, err_type)

    def add_values(
            self, source_key, target_key, source_attr, target_attr, diff):
        """Add name values to exception object."""
        super(AttrException, self).add_vals({
            'source_key': source_key,
            'target_key': target_key,
            'source_attr': source_attr,
            'target_attr': target_attr,
            'diff': diff
        })
```

Rec Tool Configuration Handler

```python
#!/usr/local/bin/Python
"""
Handles the configuration file parsing.

Angelo Gordon
Ben Sharron and Richard O'Brien
Config Handler module
"""

import openpyxl
import warnings
from rec_lib.translation_handler import TranslationTableHandler


class ConfigHandler(object):
    """Handler for configuring the program attributes."""

    def _get_single_value(self, row, col):
        """Get value from known location."""
        val = self.sheet.cell(row=row, column=col).value

        if val is None:
            return ['']
        else:
            return [val]

    def _initialize_attributes(self):
        """Initialize attributes dicitonary and translation handler."""
        self.attributes = {}

        warnings.filterwarnings("ignore")
        self.workbook = openpyxl.load_workbook(self.filename)
        self.sheet = self.workbook.get_sheet_by_name('Config')
        self.source_trans_sheet = self.workbook.get_sheet_by_name(
            'Source Translation')
        self.target_trans_sheet = self.workbook.get_sheet_by_name(
            'Target Translation')
        self.trans_handler = TranslationTableHandler(
            self.source_trans_sheet, self.target_trans_sheet)

        for row_num in range(2, self.sheet.max_row + 1):
            # Only get up to the first space for key
            try:
                row_key = self.sheet.cell(
                    row=row_num, column=1).value.split(' ')[0]
            except (IndexError, AttributeError):
                continue

            if row_key == 'source':
                self.attributes['source_input_type'] = (
```

```python
                self._get_single_value(row_num, 3))
            self.attributes['source_file'] = (
                self._get_single_value(row_num, 5))
            self.attributes['source_server'] = (
                self._get_single_value(row_num, 7))
            self.attributes['source_database'] = (
                self._get_single_value(row_num, 9))
            continue

        if row_key == 'target':
            self.attributes['target_input_type'] = (
                self._get_single_value(row_num, 3))
            self.attributes['target_file'] = (
                self._get_single_value(row_num, 5))
            self.attributes['target_server'] = (
                self._get_single_value(row_num, 7))
            self.attributes['target_database'] = (
                self._get_single_value(row_num, 9))
            continue

        # Get values from next non-empty cells
        row_vals = []

        column = 3
        row_val = self.sheet.cell(row=row_num, column=column).value
        while not row_val is None:
            if isinstance(row_val, str) or isinstance(row_val, unicode):
                row_val = row_val.strip()
            else:
                row_val = str(row_val)

            row_vals.append(row_val)
            column += 1
            row_val = self.sheet.cell(row=row_num, column=column).value

        self.attributes[row_key] = row_vals

def __init__(self, filename):
    """Initialize config handler."""
    self.filename = filename
    self._initialize_attributes()

def get(self, attribute):
    """Retrieve an attribute's singular value from the dictionary."""
    try:
        return self.attributes[attribute][0]
    except:
        raise KeyError('Key in the config file '
                'is incorrect/missing: {}'.format(attribute))

def get_num(self, attribute):
    """Retrieve an attribute's singular value as an integer."""
    try:
```

```python
            return int(self.attributes[attribute][0])
        except:
            raise KeyError('Key in the config file  '
                    'is incorrect/missing: {}'.format(attribute))

    def get_list(self, attribute):
        """Retrieve an attribute's values as a list."""
        try:
            return self.attributes[attribute]
        except:
            raise KeyError('Key in the config file '
                    'is incorrect/missing: {}'.format(attribute))

    def get_translation_handler(self):
        """Retrieve and return the translation handler."""
        return self.trans_handler
```

Rec tool Email Handler

```python
#!/usr/local/bin/Python
"""
Creates and sends email with attachment.

Angelo Gordon
Ben Sharron and Richard O'Brien
Email functionality
"""

from os.path import basename
import smtplib
from email.mime.text import MIMEText
from email.mime.application import MIMEApplication
from email.mime.multipart import MIMEMultipart


class EmailHandler(object):
    """Handle email functionality and sending emails."""

    def __init__(self, server):
        """Initialize email message."""
        self.server = server
        self.filename_list = []

    def add_file(self, filename):
        """Add file to the email."""
        self.filename_list.append(filename)

    def send(self, from_address, to_addresses, subject, body_file):
        """Send email."""
        msg = MIMEMultipart()

        msg['Subject'] = subject
        msg['From'] = from_address
        msg['To'] = ', '.join(to_addresses)

        try:
            with open(body_file, 'rb') as body:
                msg.attach(MIMEText(body.read()))
        except IOError:
            print "Body file not found, leaving body blank."
        except:
            raise IOError("Error opening body file: %s" % body_file)

        # Iterate over each file and attach it to the email
        for filename in self.filename_list:
            with open(filename, 'rb') as attached_file:
                attachment = MIMEApplication(
                    attached_file.read(),
                    Content_Disposition='attachment; filename=%s' %
```

```
        basename(filename),
        Name=basename(filename))
    msg.attach(attachment)

server = smtplib.SMTP(self.server)
server.sendmail(from_address, to_addresses, msg.as_string())
```

Rec Tool Exception Handler

```python
#!/usr/local/bin/Python
"""
Uses matching rules to validate input and create exception lists.

Angelo Gordon
Ben Sharron and Richard O'Brien
Generating and Storing Exceptions
"""

import re
import copy
from rec_lib.key_exceptions import KeyException
from rec_lib.attr_exceptions import AttrException

_ALPHA = .0001


def _check_names(source_names, target_names):
    """Check two key name lists for equivalency."""
    match_types = []

    for source_name, target_name in zip(source_names, target_names):
        if source_name == target_name:
            match_types.append('Match')
            continue

        stripped_source_name = source_name.upper()
        stripped_target_name = target_name.upper()

        if stripped_source_name == stripped_target_name:
            match_types.append('Case Mismatch')
            continue

        stripped_source_name = re.sub(r'\W+', '', stripped_source_name)
        stripped_target_name = re.sub(r'\W+', '', stripped_target_name)

        if stripped_source_name[0:7] == 'PROJECT':
            stripped_source_name = stripped_source_name[7:]
        if stripped_target_name[0:7] == 'PROJECT':
            stripped_target_name = stripped_target_name[7:]

        if stripped_source_name == stripped_target_name:
            match_types.append('Style Mismatch')
            continue

        if stripped_source_name in stripped_target_name or \
           stripped_target_name in stripped_source_name:
            match_types.append('Partial Match')
            continue
```

```python
                match_types.append('No Match')

        return match_types


    def _check_numbers(source_numbers, target_numbers, tolerance):
        """Check two key name lists for equivalency."""
        match_types = []

        for source_number, target_number in zip(source_numbers, target_numbers):
            if tolerance['type'] == 'number':
                if round(abs(source_number - target_number), 6) <= \
                    tolerance['value'] + _ALPHA:
                    match_types.append('Match')
                else:
                    match_types.append('Numeric Mismatch (tolerance: {})'.format(
                        str(tolerance['value'])))
            elif tolerance['type'] == 'percent':
                if round(abs(source_number - target_number), 6) <= \
                    ((tolerance['value'] + _ALPHA) / 100) * source_number:
                    match_types.append('Match')
                else:
                    match_types.append('Numeric Mismatch (tolerance: {}%)'.format(
                        str(tolerance['value'])))

        return match_types


    def _flatten_lists(key_pairing, sources_list, targets_list):
        """Flatten out imperfect list pairs."""
        if isinstance(sources_list[0], list):
            new_key_pairing = []
            new_sources_list = []
            new_targets_list = []

            for (key_set, src_list, tgt) in \
                    zip(key_pairing, sources_list, targets_list):
                for elt in src_list:
                    new_key_pairing.append(key_set)
                    new_sources_list.append(elt)
                    new_targets_list.append(tgt)

            return _flatten_lists(
                new_key_pairing, new_sources_list, new_targets_list)
        elif isinstance(targets_list[0], list):
            new_key_pairing = []
            new_sources_list = []
            new_targets_list = []

            for (key_set, src, tgt_list) in \
                    zip(key_pairing, sources_list, targets_list):
                for elt in tgt_list:
                    new_key_pairing.append(key_set)
```

```python
        new_sources_list.append(src)
        new_targets_list.append(elt)

    return _flatten_lists(
        new_key_pairing, new_sources_list, new_targets_list)
else:
    return (key_pairing, sources_list, targets_list)


def _get_diff(source_val, target_val):
    """Retrieve percentage difference between values."""
    if isinstance(source_val, str) or isinstance(source_val, unicode):
        return ''
    if isinstance(target_val, str) or isinstance(target_val, unicode):
        return ''

    if source_val == 0:
        return "100%"

    return '{}%'.format(
        str(round(100 * (target_val - source_val) / source_val, 2)))


class ExceptionHandler(object):
    """Handler for exceptions found during execution."""

    def __init__(self):
        """Initialize list of exceptions and headers for outputting."""
        self.key_pairing = []
        self.key_exc_list = []
        self.attr_exc_list = []
        self.missing_names = []
        self.extra_names = []

    def _add_key_exception(self, exc):
        """Add key exception to list."""
        self.key_exc_list.append(exc)

    def _add_attr_exception(self, exc):
        """Add attribute exception to list."""
        self.attr_exc_list.append(exc)

    def get_num_exceptions(self):
        """Get number of exceptions."""
        return len(self.key_exc_list) + len(self.attr_exc_list)

    def retrieve_key_exceptions(self, sources_list, targets_list):
        """Generate key name exceptions for two lists and save key pairing."""
        sources_list_copy = copy.deepcopy(sources_list)
        targets_list_copy = copy.deepcopy(targets_list)

        found_source_keys = {
            tuple(source_entry): False for source_entry in sources_list_copy[:]
```

```python
    }
    found_target_keys = {
        tuple(target_entry): False for target_entry in targets_list_copy[:]
    }

    # Prioritize matches that aren't partial
    for source_entry in sources_list_copy[:]:
        if found_source_keys[tuple(source_entry)]:
            continue

        for target_entry in targets_list_copy[:]:
            if found_target_keys[tuple(target_entry)]:
                continue

            check_result = _check_names(source_entry, target_entry)

            if 'No Match' not in check_result and \
                'Partial Match' not in check_result:
                if check_result != ['Match']*len(check_result):
                    exc = KeyException(
                        source_entry, target_entry, check_result)
                    self._add_key_exception(exc)

                self.key_pairing.append([source_entry, target_entry])
                found_source_keys[tuple(source_entry)] = True
                found_target_keys[tuple(target_entry)] = True

                # Output every hundreth pair
                if len(self.key_pairing) % 100 == 0:
                    print 'Found key pair {} of up to {}: ' \
                        '{} and {}'.format(
                            str(len(self.key_pairing)),
                            str(len(sources_list_copy)),
                            source_entry, target_entry)

                break

    for source_entry in sources_list_copy[:]:
        if found_source_keys[tuple(source_entry)]:
            continue
        for target_entry in targets_list_copy[:]:
            if found_target_keys[tuple(target_entry)]:
                continue

            check_result = _check_names(source_entry, target_entry)

            if 'No Match' not in check_result:
                exc = KeyException(
                    source_entry, target_entry, check_result)
                self._add_key_exception(exc)

                self.key_pairing.append([source_entry, target_entry])
                found_source_keys[tuple(source_entry)] = True
```

```python
                found_target_keys[tuple(target_entry)] = True

                # Output every hundreth pair
                if len(self.key_pairing) % 100 == 0:
                    print 'Found key pair {}: {} and {}'.format(
                        str(len(self.key_pairing)),
                        source_entry, target_entry)

                break

        for source_entry in sources_list_copy[:]:
            if not found_source_keys[tuple(source_entry)]:
                found_source_keys[tuple(source_entry)] = True
                exc = KeyException(source_entry, '', ['In Source Not Target'])
                self._add_key_exception(exc)
                self.missing_names.append(source_entry)

        for target_entry in targets_list_copy[:]:
            if not found_target_keys[tuple(target_entry)]:
                found_target_keys[tuple(target_entry)] = True
                exc = KeyException('', target_entry, ['In Target Not Source'])
                self._add_key_exception(exc)
                self.extra_names.append(target_entry)

    def untranslate(
            self, orig_source, orig_target, trans_source, trans_target):
        """Revert values of translated key pairing and missing/extra names."""
        # Return early if there was no translation
        if orig_source == trans_source and orig_target == trans_target:
            return

        key_pairing_copy = copy.deepcopy(self.key_pairing)
        missing_names_copy = copy.deepcopy(self.missing_names)
        extra_names_copy = copy.deepcopy(self.extra_names)

        for index, key_pair in enumerate(key_pairing_copy[:]):
            source_vals = key_pair[0]
            target_vals = key_pair[1]

            if source_vals not in orig_source:
                orig_val = orig_source[trans_source.index(source_vals)][:]

                key_pairing_copy[index][0] = orig_val

            if target_vals not in orig_target:
                orig_val = orig_target[trans_target.index(target_vals)][:]

                key_pairing_copy[index][1] = orig_val

        for (index, name) in enumerate(missing_names_copy[:]):
            if name not in orig_source:
                orig_val = orig_source[trans_source.index(name)][:]
```

```python
            missing_names_copy[index] = orig_val

        for (index, name) in enumerate(extra_names_copy[:]):
            if name not in orig_target:
                orig_val = orig_target[trans_target.index(name)][:]

                extra_names_copy[index] = orig_val

        self.key_pairing = key_pairing_copy
        self.missing_names = missing_names_copy
        self.extra_names = extra_names_copy

    def retrieve_exceptions(self, source_attr, target_attr,
                    sources_list, targets_list, tolerance):
        """Generate exceptions for two lists given a key pairing."""
        (key_pairing, sources_list, targets_list) = _flatten_lists(
            self.key_pairing, sources_list, targets_list)

        if tolerance is None:
            check_results = _check_names(sources_list, targets_list)
        else:
            check_results = _check_numbers(
                sources_list, targets_list, tolerance)

        for (key_pair, source_val, target_val, result) in zip(
                key_pairing, sources_list, targets_list, check_results):
            if result != 'Match':
                exc = AttrException(source_val, target_val, result)
                exc.add_values(key_pair[0], key_pair[1],
                        source_attr, target_attr,
                        _get_diff(source_val, target_val))

                self._add_attr_exception(exc)

    def retrieve_missing_name_exc(self, source_attr, sources_list):
        """Generate missing name exceptions for a sources list."""
        for (key_name, source_val) in zip(self.missing_names, sources_list):
            exc = AttrException(source_val, '', 'In Source Not Target')
            exc.add_values(key_name, '', source_attr, '', '')
            self._add_attr_exception(exc)

    def retrieve_extra_name_exc(self, target_attr, targets_list):
        """Generate extra name exceptions for a targets list."""
        for (key_name, target_val) in zip(self.extra_names, targets_list):
            exc = AttrException('', target_val, 'In Target Not Source')
            exc.add_values('', key_name, '', target_attr, '')
            self._add_attr_exception(exc)

    def get_missing_names(self):
        """Return list of source key names that don't have a match."""
        return self.missing_names

    def get_extra_names(self):
```

```python
        """Return list of target key names that don't have a match."""
        return self.extra_names

    def get_key_exc_list(self):
        """Return the list of key exceptions."""
        return self.key_exc_list

    def get_attr_exc_list(self):
        """Return the list of attribute exceptions."""
        return self.attr_exc_list

    def split_key_pairing(self):
        """Return a split up version of the key pairing."""
        return ([key[0] for key in self.key_pairing],
                [key[1] for key in self.key_pairing])
```

Rec Tool Key Exceptions

```python
#!/usr/local/bin/Python
"""
Object for exceptions found betweens key name values.

Angelo Gordon
Ben Sharron and Richard O'Brien
Key_Exception class
"""

from rec_lib.abs_exceptions import AbsException


class KeyException(AbsException):
    """An Exception found in the primary key."""

    _KEY_LIST = ['source_val', 'target_val', 'err_type']

    def __init__(self, source_val, target_val, err_type):
        """Initialize exception object."""
        super(KeyException, self).__init__(source_val, target_val, err_type)
```

Rec Tool Input Handler

```python
#!/usr/local/bin/Python
"""
Excel workbook retrieval and storage.

Angelo Gordon
Ben Sharron and Richard O'Brien
Input Handler module
"""

import openpyxl
import warnings


def _aggregate(input_list, aggr_type):
    """Helper function to perform aggregation."""
    if aggr_type == 'count':
        return len(input_list)
    elif aggr_type == 'max':
        return max(input_list)
    elif aggr_type == 'min':
        return min(input_list)
    elif aggr_type == 'sum':
        total = 0.0

        for num in input_list:
            total += float(num)

        return total
    elif aggr_type == 'avg':
        total = 0.0

        for num in input_list:
            total += float(num)

        return total / len(input_list)
    else:
        raise IOError("Unknown aggregation type '{}'".format(aggr_type))


def _clean_list(input_list):
    """Clean up list if the inner lists all have one element each."""
    new_list = []

    for sublist in input_list:
        if len(sublist) > 1:
            return input_list
        else:
            new_list.append(sublist[0])

    return new_list
```

```python
class InputHandler(object):
    """Accesses values from an Excel file and parses and stores them."""

    def __init__(self, filename):
        """Initialize workbook, as well as the separate sheets and state."""
        self.filename = filename
        self.table_key_nums = []
        self.key_rows_hash = {}
        warnings.filterwarnings("ignore")
        self.workbook = openpyxl.load_workbook(filename)
        self.sheet_names = self.workbook.get_sheet_names()
        self.sheet = self.workbook.get_sheet_by_name(self.sheet_names[0])

    def get_key_columns(self, col_names):
        """Return unique key column values and save key columns."""
        result = []

        for col_val in col_names:
            for col_num in range(1, self.sheet.max_column + 1):
                if self.sheet.cell(row=1, column=col_num).value == col_val:
                    self.table_key_nums.append(col_num)
                    break
            else:
                raise IOError("Key column '{}' not found in file: {}.".format(
                    col_val, self.filename))

        for row_num in range(2, self.sheet.max_row + 1):
            row_vals = []

            for col_num in self.table_key_nums:
                val = str(self.sheet.cell(row=row_num, column=col_num).value)
                if val:
                    if isinstance(val, unicode):
                        val = val.encode('ascii', 'ignore')
                    row_vals.append(val)

            if len(row_vals) > 0:
                if not tuple(row_vals) in self.key_rows_hash:
                    result.append(row_vals)
                    self.key_rows_hash[tuple(row_vals)] = [row_num]
                else:
                    self.key_rows_hash[tuple(row_vals)].append(row_num)

        return result

    def get_column(self, key_list, col_name):
        """Return row values for column in source table as list."""
        result = []

        for col_num in range(1, self.sheet.max_column + 1):
            if self.sheet.cell(row=1, column=col_num).value == col_name:
```

```python
                col_number = col_num
                break
        else:
            raise IOError("Column '{}' not found in file: {}.".format(
                col_name, self.filename))

        for key_set in key_list:
            entry_set = []

            # If key is not in file
            if not tuple(key_set) in self.key_rows_hash:
                return []

            for row_num in self.key_rows_hash[tuple(key_set)]:
                val = self.sheet.cell(row=row_num, column=col_number).value

                if val:
                    if isinstance(val, unicode):
                        val = val.encode('ascii', 'ignore')
                    entry_set.append(val)
                else:
                    entry_set.append('')

            result.append(entry_set)

        return _clean_list(result)

    def get_column_with_aggregation(self, key_list, col_name, aggr_type):
        """Return given column with aggregation from given table as list."""
        result = []

        for col_num in range(1, self.sheet.max_column + 1):
            if self.sheet.cell(row=1, column=col_num).value == col_name:
                col_number = col_num
                break
        else:
            raise IOError("Column '{}' not found in file: {}.".format(
                col_name, self.filename))

        for key_set in key_list:
            entry_set = []

            # If key is not in file
            if not tuple(key_set) in self.key_rows_hash:
                return []

            for row_num in self.key_rows_hash[tuple(key_set)]:
                val = self.sheet.cell(row=row_num, column=col_number).value

                if val:
                    entry_set.append(val)

            result.append(_aggregate(entry_set, aggr_type))
```

```
    return result
```

Rec Tool Output Handler

```python
#!/usr/local/bin/Python
"""

Handles the outputting of found exceptions to an Excel file.

Angelo Gordon
Ben Sharron and Richard O'Brien
Output Handler
"""


from openpyxl.workbook import Workbook
from openpyxl.styles import numbers
from rec_lib.key_exceptions import KeyException
from rec_lib.attr_exceptions import AttrException


def _is_num(val):
    """Return if the value is a number."""
    return isinstance(val, int) or isinstance(
        val, float) or isinstance(val, long)


def _output_column(sheet, col_num, col_name, col_list):
    """Helper function that outputs a column of data into the given sheet."""
    row_num = 1

    sheet.cell(row=row_num, column=col_num).value = col_name

    for col_val in col_list:
        row_num += 1

        sheet.cell(row=row_num, column=col_num).value = col_val

        if _is_num(col_val):
            sheet.cell(
                row=row_num, column=col_num).number_format = (
                    numbers.FORMAT_NUMBER_COMMA_SEPARATED1)


def _output_header(header_list, sheet):
    """Output header list."""
    col_num = 1

    for header_val in header_list:
        sheet.cell(row=1, column=col_num).value = header_val
        col_num += 1


class OutputHandler(object):
    """Handler for outputting exceptions and data found during execution."""
```

```python
def __init__(self, filename):
    """Initialize output state and workbook."""
    self.filename = filename

    self.source_col = 1
    self.target_col = 1

    self.workbook = Workbook()
    self.sheets = {
        'Key Exceptions': self.workbook.active,
        'Attribute Exceptions': self.workbook.create_sheet(),
        'Source': self.workbook.create_sheet(),
        'Target': self.workbook.create_sheet()
    }

    self.sheets['Key Exceptions'].title = 'Key Exceptions'
    self.sheets['Attribute Exceptions'].title = 'Exceptions'
    self.sheets['Source'].title = 'Source'
    self.sheets['Target'].title = 'Target'

def output_key_header(self, header_list):
    """Output header list for keys."""
    _output_header(header_list, self.sheets['Key Exceptions'])

    self.workbook.save(self.filename)

def output_attr_header(self, header_list):
    """Output header list for attributes."""
    _output_header(header_list, self.sheets['Attribute Exceptions'])

    self.workbook.save(self.filename)

def output_key_exceptions(self, key_exc_list):
    """Output key exception list into exceptions sheet."""
    output_row = 2

    key_list = KeyException.get_keys()

    for exc in key_exc_list:
        col_num = 1

        for key in key_list:
            self.sheets['Key Exceptions'].cell(
                row=output_row, column=col_num).value = (
                    exc.get_exc_strings()[key])
            col_num += 1

        output_row += 1

    output_row += 1

    self.workbook.save(self.filename)
```

```python
def output_attr_exceptions(self, attr_exc_list):
    """Output attribute exception list into exceptions sheet."""
    output_row = 2

    key_list = AttrException.get_keys()

    for exc in attr_exc_list:
        col_num = 1

        for key in key_list:
            val = exc.get_exc_strings()[key]

            if _is_num(val):
                self.sheets['Attribute Exceptions'].cell(
                    row=output_row, column=col_num).value = float(val)
                self.sheets['Attribute Exceptions'].cell(
                    row=output_row, column=col_num).number_format = (
                        numbers.FORMAT_NUMBER_COMMA_SEPARATED1)
            elif isinstance(val, str) and len(val) > 0 and val[-1] == '%':
                self.sheets['Attribute Exceptions'].cell(
                    row=output_row, column=col_num).value = (
                        float(val[:-1]) / 100)
                self.sheets['Attribute Exceptions'].cell(
                    row=output_row, column=col_num).number_format = (
                        numbers.FORMAT_PERCENTAGE_00)
            else:
                self.sheets['Attribute Exceptions'].cell(
                    row=output_row, column=col_num).value = val

            col_num += 1

        output_row += 1

    self.workbook.save(self.filename)

def output_source_column(self, col_name, col_list):
    """Output given list under given header into source sheet."""
    _output_column(self.sheets['Source'], self.source_col,
            col_name, col_list)
    self.workbook.save(self.filename)
    self.source_col += 1

def output_target_column(self, col_name, col_list):
    """Output given list under given header into target sheet."""
    _output_column(self.sheets['Target'], self.target_col,
            col_name, col_list)
    self.workbook.save(self.filename)
    self.target_col += 1
```

Rec Tool Translation Handler

```python
#!/usr/local/bin/Python
"""
Module for handling translation for source and target input.

Angelo Gordon
Ben Sharron and Richard O'Brien
Translation Table Handler module
"""

import copy


def _translate(sheet, mapping, column, source_values):
    """Translate list for given sheet and column."""
    source_values_copy = copy.deepcopy(source_values)

    if sheet.max_row == 1 or column not in mapping:
        return source_values_copy

    # Check for translations
    for index, given_name in enumerate(source_values[:]):
        if given_name in mapping[column]:
            source_values_copy[index] = mapping[column][given_name]

    return source_values_copy


def _translate_multiple(sheet, mapping, columns, source_values_list):
    """Translate list of lists for given sheet and columns."""
    source_values_list_copy = copy.deepcopy(source_values_list)

    if sheet.max_row == 1:
        return source_values_list_copy

    # Check for translations
    for source_index, source_values in enumerate(
            source_values_list_copy[:]):
        for index, (col_name, col_value) in enumerate(
                zip(columns[:], source_values[:])):
            if col_name in mapping and col_value in mapping[col_name]:
                source_values_list_copy[source_index][index] = (
                    mapping[col_name][col_value])

    return source_values_list_copy


def _get_mapping(sheet):
    """Get mappings of column names to sheet rows for both sheets."""
    mapping = {}

    # Iterate through all source file rows
    for row_num in range(2, sheet.max_row + 1):
        row_column = str(sheet.cell(row=row_num, column=1).value)
```

```python
        row_source_value = str(sheet.cell(row=row_num, column=2).value)
        row_target_value = str(sheet.cell(row=row_num, column=3).value)

        if isinstance(row_column, unicode):
            row_column = row_column.encode('ascii', 'ignore')

        if isinstance(row_source_value, unicode):
            row_source_value = row_source_value.encode('ascii', 'ignore')

        if isinstance(row_target_value, unicode):
            row_target_value = row_target_value.encode('ascii', 'ignore')

        if row_column not in mapping:
            mapping[row_column] = {row_source_value: row_target_value}
        else:
            mapping[row_column][row_source_value] = row_target_value

    return mapping


class TranslationTableHandler(object):
    """Handler for translation table instance."""

    def __init__(self, source_trans_sheet, target_trans_sheet):
        """Initialize workbook, as well as the separate sheets."""
        self.source_sheet = source_trans_sheet
        self.target_sheet = target_trans_sheet

        self.source_mapping = _get_mapping(self.source_sheet)
        self.target_mapping = _get_mapping(self.target_sheet)

    def translate_source(self, column, source_values):
        """Translate column in source sheet."""
        return _translate(
            self.source_sheet, self.source_mapping, column, source_values)

    def translate_target(self, column, source_values):
        """Translate column in target sheet."""
        return _translate(
            self.target_sheet, self.target_mapping, column, source_values)

    def translate_source_multiple(self, columns, source_values):
        """Translate multiple columns in source sheet."""
        return _translate_multiple(
            self.source_sheet, self.source_mapping, columns, source_values)

    def translate_target_multiple(self, columns, source_values):
        """Translate multiple columns in target sheet."""
        return _translate_multiple(
            self.target_sheet, self.target_mapping, columns, source_values)
```

APPENDIX D: EXTENSION OF HIGH SPEED EQUITIES TRADING RESEARCH STUDY

1. Introduction

In January 2015, Hans R. Stoll published a study in the *Asia-Pacific Journal of Financial Studies* examining High Speed Equities Trading, A.K.A. High Frequency Trading (HFT), and its effect on the general market during the years 1993 to 2012 (Stoll, 2015). This appendix seeks to examine Stoll's study and attempt to replicate and extend a number of his findings using publicly and freely available data. In particular we aim to expand his observations through the most recent few years to determine if the conclusions that he had drawn remain relevant and accurate in the context of this new information. While Stoll used data from both the New York Stock Exchange (NYSE) and Nasdaq/OMX to support his claims of the effect that HFT has on the market, our team focuses only on collecting and analyzing data from the NYSE which would both limit the scope of our research and allow more focus on extending the existing conclusions to the current day. Our conclusions are drawn from this research and seek to bolster the conclusions Stoll formed as well as introduce our own analysis of the data to determine the implications of HFT on the greater market.

2. Background

With the rise of computing technology, and the introduction of this technology into the financial sector, a family of algorithmic trading strategies known broadly as High-Frequency Trading (HFT) emerged. This set of strategies takes advantage of electronic trading tools that have replaced human brokers, as well as low-latency technology and high-speed connections to perform trades precisely and efficiently. These tools allow traders to implement new techniques such as algorithmically trading short-term positions quickly and in high volumes to make a profit on the minute margins of change in equity prices. They also, however, allow techniques such as "quote stuffing" which attempts to deny information to other traders by overwhelming the trade queue with quotes that are then rescinded at the last minute. These strategies, and other derived strategies, however, have been criticized as contributing factors to so-called "flash crashes" that have cost investors trillions of dollars.

One of the more well known flash crashes was the crash of May 6, 2010. In less than an hour, securities such as the Dow Jones, Nasdaq Composite, and S&P 500 all collapsed and rebounded very quickly; the Dow Jones in particular dropped more than 600 points before rebounding. Nobody expected or could predict the 2010 flash crash. One of the most discussed potential reasons for the flash crash is HFT, since one can execute a technique in HFT where the computer program would set up a large trade to be placed, but then cancel it at the very last second. This technique known as quote stuffing, as previously defined, allows traders to artificially manipulate the market. It can also disrupt the market and cause extreme anomalies as seen in the 2010 flash crash.

3. Data

To investigate the effect of HFT on the market, and to replicate as well as extend Stoll's research paper, our team analyzed data from the New York Stock Exchange (NYSE), both for the full exchange and individual equities. Due to time constraints as well as the resources available to us, we chose to only extend the study's analysis for the NYSE rather than also including analysis on the Nasdaq as seen in Stoll's paper. In order to reproduce the results from the study, the team examined 300 total NYSE stocks with data available from 1993 until present; we grouped 100 stocks from each of three market capitalization groups: large, mid, and small cap. Market capitalization is calculated by multiplying a company's shares outstanding by the current market price of one share (Market Cap, 2016). This data was pulled and analyzed from Yahoo Finance using python scripts to access their API. We then used the available data for these stocks to calculate several relevant measures to determine the potential impact of HFT on the NYSE during this time. The result of the team's investigation follows.

4. Trade size and trade frequency

The first measurement the team attempted to replicate from the study was the change in average trade size of transactions on the NYSE during the period in question. Since the requisite data to calculate this measure for individual stocks was not freely available, the team compromised to compare Stoll's study's findings to our examination of trade size changes across the entire exchange of the NYSE. We used the data available on the NYSE's website (NYSE, 2016) to find the daily values for the amount of available shares in the exchange and the amount of trades on the given day. We had to concatenate the 1993-2000 data with the recent 2000-present data in order to consider all values. The average trade size on a given day was calculated by taking of the total amount of shares available and dividing by the amount of trades done. The average trade size in shares is plotted for comparison and analysis in Figure 1.
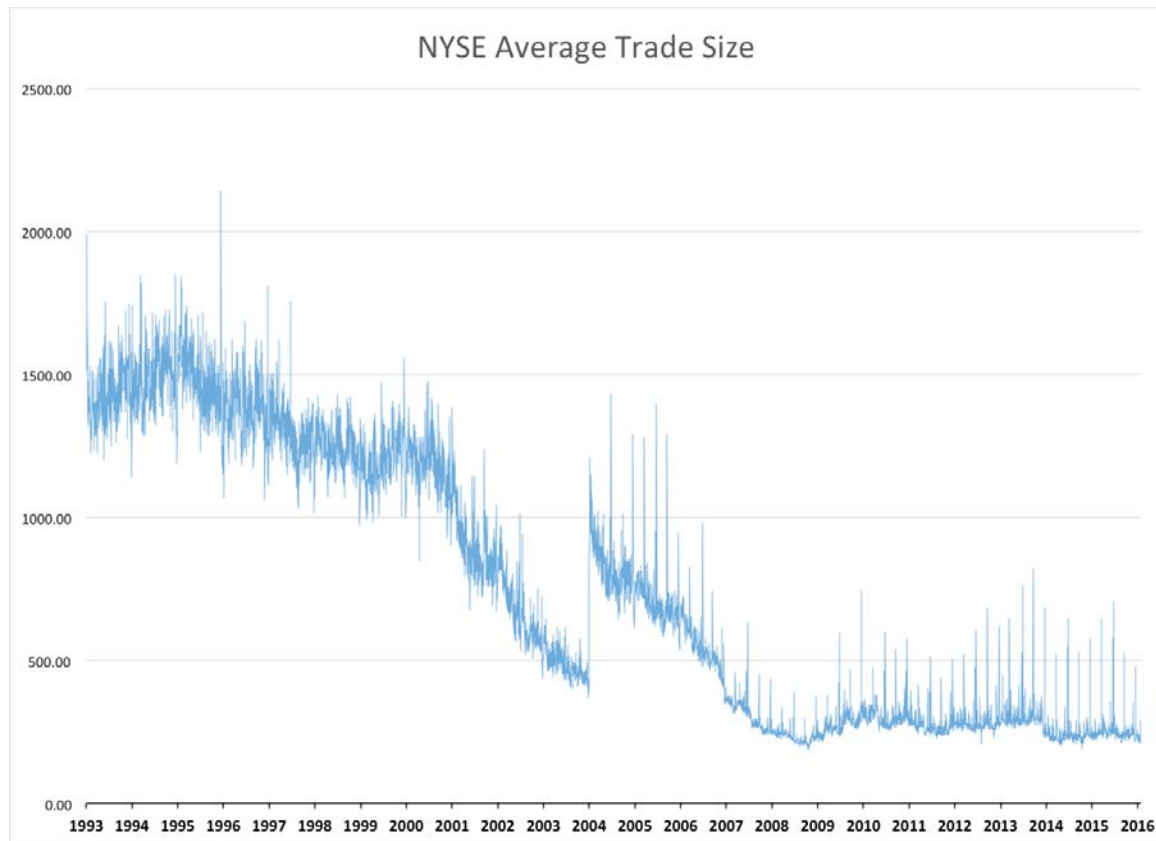
NYSE Average Trade Size

Figure 1: Average trade size across the NYSE for years 1993-2015

It's worth noting the spike anomaly in our graph at around the beginning of 2004. This is due to an inconsistency in the data that is provided by the NYSE through the Consolidated Tape Association (CTA) which provides "supplemental" trade data for trades prior to January 2004, but not for trade data after this date. This inconsistency manifests itself as a sharp jump in the calculated Average Trade Size at that time due to a decrease in the reported number of trades performed.

When looking at the overall trend of the NYSE's average trade size across time, Stoll's research paper found a steady decrease in recent years in average trade size. This correlates with an increase in the use of algorithms to perform trades quickly as well as a decline in the auxiliary costs in performing trades that would allow for traders to perform a higher number of

smaller volume trades. Fortunately, ignoring the spike in the beginning of 2004, our data fits

this trend as well, demonstrating that the effect is consistent even across the exchange as a

whole and in the recent years of 2012-2015.

5. Effect of high frequency trading on market quality

5.1 Bid-Ask spread

One of the suggested effects of the introduction of high speed trading was a decrease in the bid-ask spread on average. The Stoll paper suggests a trend downwards with a sharp decline in 2001 that they attribute to a switch from fractional to decimal stocks, and a general downward trend thereafter.

In order to calculate bid-ask spread we used an estimator as described in Corwin and Shultz's research paper using daily high and low prices (Corwin et al., 2012). As seen in Figure 2, we do not see a similar dramatic drop in 2001 that was attributed to a change from fractional to decimal pricing, which may be a result of the estimator assuming that bid-ask spread is continuous rather than discrete. We do however see, from 2003 to 2007 bid-ask spread is mostly constant, and after 2010 it is as well, which suggests that even with the introduction and adoption of high speed trading, we do not see a significant drop in the average bid-ask spread in the market. This matches Stoll's observation that "there is no indication that spreads increased after 2005 as HF trading became more common," (Stoll, 2015, p.777). We do see a slight downward trend for bid-ask spread for the high-cap group continuing after 2013, which may suggest an effect, however note that the decrease is seen after average trade size has stabilized.
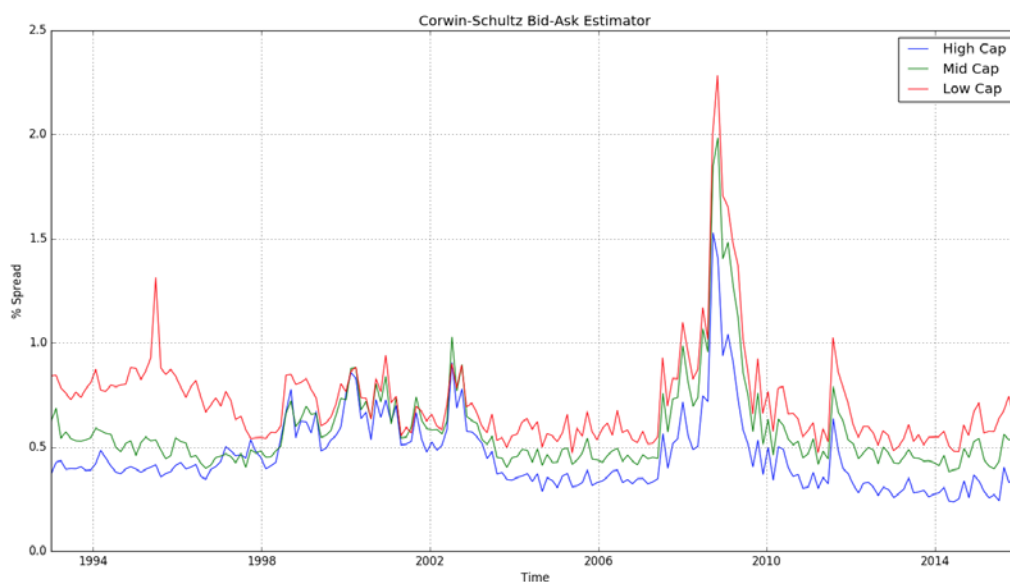
Figure 2: Percentage spread of Corwin-Schultz bid-ask estimator.

5.2 Volatility

In a similar vein to Stoll's paper, we wanted to see if the effect of HFT has any effect on volatility in the market, especially when looking at recent years. Generally speaking, one would guess an increase in volatility because of the large and rapid amount of trades occurring during HFT. One of the methods of calculating volatility that Stoll used was to compute the intra-day variance of trade-to-trade prices and then average them for the designated month. Unfortunately our team did not have access to intra-day data for our chosen stocks. We decided to use the second method from Stoll's paper which involves calculating variance of daily stock returns.

For each of the market cap stock groups, we retrieved the daily close prices and calculated the average of all the stocks for a designated day. We then calculated the 365-day variance of our average daily close prices and plotted our results across time as seen in Figure 3.
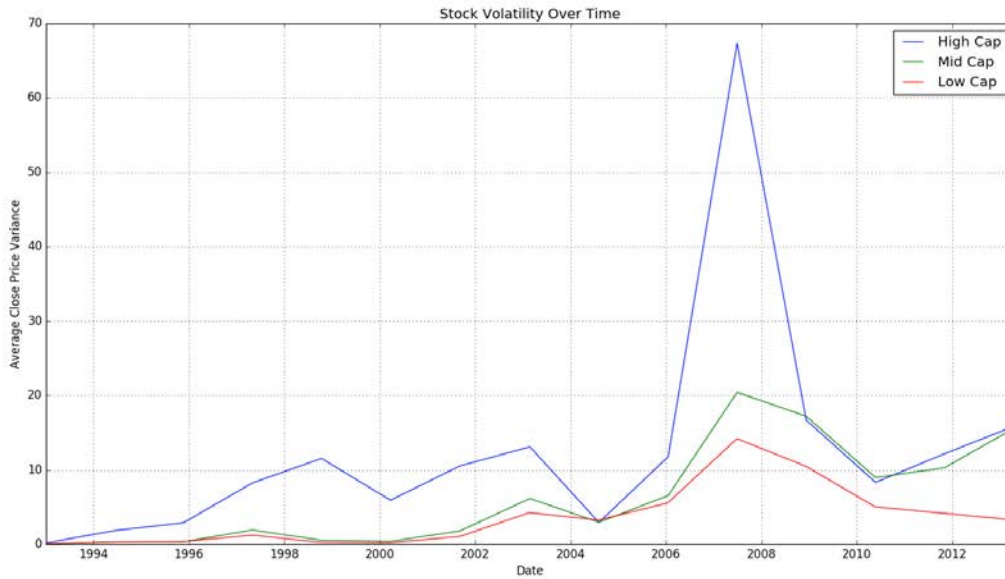
Figure 3: Variance of daily NYSE stock returns.

When comparing our figure to Stoll's daily variance volatility graph, our figure follows

the same general trend. From just looking at the high cap stocks, the dot-com bubble and the

2007-2008 financial crisis are very pronounced. The stock volatility generally hovers around the

average close price variance of 10, with the exception of anomalies. Our volatility analysis

suggests that HFT can be correlated with the gradual increase in volatility over the recent years.

6. Conclusion and comments

Although the introduction of electronic trading began around 1999 after electronic exchanges were authorized by the Securities and Exchanges Commission in 1998 (SEC), it is important to remember that the introduction and usage of HFT started becoming more prevalent around 2005. Taking this into consideration and when looking at the average trade size trend, the average trade size has decreased as expected. This downward trend coincides with the introduction and continued adoption of HFT. We also can observe that the average trade size reaches a level of saturation after the 2007-2008 financial crisis. One might infer that for the average trade size to reach this steady level, the proportion of traders who use HFT has plateaued, leading to a stability in this measure.

When looking at our results for bid-ask spread, although our analysis used a bid-ask estimation formula, it is clear that bid-ask spread has stayed steady since the introduction of HFT around 2005 - ignoring the 2007-2008 crisis anomaly. Even in the years of increased HFT adoption in the later years, bid-ask spread does not seem to correspond with the adoption of HFT.

In regards to HFT's effect on volatility on the NYSE, it can be inferred from our volatility analysis that HFT could be a factor in the gradual increase in volatility on the overall market. After 2005, even when ignoring the spike of the financial crisis, volatility has steadily increased across the recent years. This can be correlated with the increase in trade activity thanks to the rapid nature of HFT.

We can therefore conclude that there is no evidence that the introduction of HFT has a meaningful, negative impact on the market, at least in the case of the NYSE and based on the

measurements we have examined. Our result is consistent with the result of Stoll's paper

where he also concludes that HFT failed to have a negative effect on the market.

References:

Corwin, Shane A., and Paul Schultz, 2012, "A simple way to estimate bid-ask spreads from daily high and low prices", *Journal of Finance,* 67, pp. 719–759.

Market Capitalization Definition. Website. Retrieved from:
http://www.investopedia.com/terms/m/marketcapitalization.asp

Quote Stuffing Definition. Website. Retrieved from:
http://www.investopedia.com/terms/q/quote-stuffing.asp

NYSE Data Library. Website. Retrieved from: https://www.nyse.com/data/transactions-statistics-data-library

(SEC) SECURITIES AND EXCHANGE COMMISSION, "Regulation of Exchanges and Alternative Trading Systems", 17 CFR Parts 202, 240, 242 and 249, Release No. 34-40760; File No. S7-12-98; RIN 3235-AH41. Retrieved from: https://www.sec.gov/rules/final/34-40760.txt

Stoll, H., 2015, "High Speed Equities Trading: 1993–2012", *Asia-Pacific Journal of Financial Studies*, 43, pp. 767-797.