Data Analytic Tools for Inconsistency Detection in Large Data Sets

Project Plan

Team 27
Client - Kingland
Advisers - Cai Ying
Team Members/Roles -
Logan Heitz (Team Lead), Camden Voigt (Technical Lead),
CJ Konopka (Communication Lead), TJ Rogers (QA Lead)
Team Email - sdmay18-27@iastate.edu
Team Website - http://sdmay18-27.sd.ece.iastate.edu/
Revised: December 1, 2017 Version 3

Table of Contents

1 Introduction

1.1 Acknowledgement

This project would not be possible without the assistance of the faculty advisor Dr. Cai. Working with Dr. Cai on this project are two graduate students Guolei Yang and Zehua Li who have provided invaluable assistance in design and implementation of this project. Finally, this project relies on the support of Kingland Systems for testing data and any other needed materials for implementation of the project.

1.2 Problem and Project Statement

Kingland processes a large amount of data that it receives from its clients everyday. This data can be relating to customers, companies, or agreements between entities. This data is compared to a central inconsistency database in order to detect inconsistencies and then added to the database. An example of an inconsistency would be two customer records containing the same social security number, but different names. This is an issue, since a social security number should be unique. The database contains over 100 million records, and around 10% of these records are updated or inserted daily. Due to its size, this comparison takes several hours to run every day. This time stems from the fact the entire database cannot be loaded into main memory at one time and the use of SQL inner join statements to check for inconsistencies, which is inefficient. Kingland would like to process 100 million records for inconsistencies in an hour or less. Additionally this detection must begin with the latest version of the inconsistency database after the reports come in.

1.3 Operational Environment

Our product will operate on the backend of Kingland's system and will be automated to detect inconsistencies on incoming reports. Thus, the product will need to be able to operate with minimal user input and will need to generate results that can be integrated into Kingland's existing infrastructure.

1.4 Intended Users and Uses

This project will supply Kingland's analysts with information on inconsistencies within client data. The product will be on the backend of Kingland's system and its processes will be automated. As such, no users will directly interact with our system on a day to day passes as Kingland will display the results of our output using their own user interface. However, if Kingland wishes to improve the system in the future or needs to fix something their developers will need access to the code and documentation of the project. Thus, it is important to provide material for future developers on this project.

1.5 Assumptions and Limitations

1.5.1 Assumptions

- There will be an inconsistency database containing more than 100 million historical reports
- The end product shall not require a user interface
- The product will only need to detect equality comparisons
- The inconsistency database will be periodically updated with new data

1.5.2 Limitations

- The program will not be able to be tested on the full sized dataset
- The program cannot be tested with all possible configurations
- Program will be deployed on a machine with less than 64 GB of RAM

1.6 Expected End Product and Deliverables

1.6.1 System architecture of proprietary solution

Delivery Date: 01/20/2018

This deliverable will encompass the design of the proprietary solution that will be developed to solve this problem. This deliverable will be expanded on in the design document and will involve the overall system block diagram, UML class diagrams, and class documentation.

1.6.2 System implementation of proprietary solution

Delivery Date: 02/20/2018

In addition to the architecture of the proprietary solution an implementation of the solution will be developed in java. This implementation will be provided to the client for use in their daily inconsistency checking.

1.6.3 Analysis of proprietary solution

Delivery Date: 03/20/2018

There are many standard industry solutions that could be utilized to solve this problem. Following the implementation of the proprietary solution the team will test the solution to determine its average runtime along with the detection rate of inconsistencies. The team will perform similar analysis of standard industry solutions and provide the findings to the client so they might evaluate which solution is best for their needs.

1.6.4 Test cases of solutions

Delivery Date: 04/02/2018

All test cases that are used for the solution will be provided for the client so they might verify the solution is valid using the test cases. They will also be able to utilize the test case if further development is needed for the project.

1.6.5 User manual

Delivery Date: 04/02/2018

A user manual that will discuss how to set up the application and automate its processes. This will include documentation on how to set up the configuration file for their needs. It will also include how they can incorporate the outputs of the application with their user interface.

2 Design

2.1 Previous Work/Literature

One consideration for our project is previous work done in detection of inconsistencies in large data sets. In order for our solution to provide value for our client it will need to suit their needs better than other existing solutions. We have looked at a few different systems that are similar to ours. The first of which is proposed in the paper “An Efficient Method of Data Inconsistency Check for

Very Large Relations." The solution proposes the utilization of functional dependencies and applying an association finding algorithm on the data set. This solution works well with smaller number of rules and when looking for very specific types of inconsistencies. However, for our project we will have a large number of rules and will be checking for many different types of inconsistencies both intra-record and inter-record. This would lead to many types of associations in the data set. Our proposed solution will be better at handling a variety of inconsistency types. Another issue with this solution is it does not handle the issue of swapping the table in and out of main memory. Our solution helps to reduce the size of the table using hashing and thus all or most of the table will be able to load into main memory at once and we can avoid costly disk access. Another potential issue here is in the paper "Inconsistencies in big data" by Zhang. In this paper he discusses four types of inconsistencies. These types cover one type of inconsistency we have with missing data, however it fails to highlight inconsistency between two data sets. The paper proposes the use of a machine learning system for detecting inconsistencies. While this system is good for learning how inconsistencies are caused and working to avoid them this is not an issue Kingland needs solved. Since all of Kingland's data is sent to it by its clients it cannot avoid inconsistencies, so strategies for this are not relevant to their problem. Another reason this solution might not be practical is that Kingland will require their data analysts to check on inconsistencies to determine the best course of action. This would further limit the abilities of any machine learning system deployed. As such our solution is more practical and better suited to Kingland's needs for quick data detection and reporting.

Another consideration on our project is parsing large XML files very quickly. Since the daily reports of records received by Kingland can be in excess of 100 GB we need to have an XML parser that can handle this. For this we looked at the article by Haufler on parsing XML files in Java. This article highlight the benefits of SAX for our project. Specifically the consideration of memory management with an XML parser. Using a parser that loads the entire DOM into memory at once would be costly since it can often take about three times the storage of the XML file itself. Thus, SAX seems to be a better choice for processing the large XML files we will receive. According to testing done by Staveley, there is also a performance benefit in terms of time when using SAX on large files compared to other Java XML parsers. This provides further justification for the use of the SAX parser in our project.

2.2 Proposed Solution

Our proposed solution to this problem is to create a proprietary system that will utilize hashing to speed up comparisons and to reduce the memory required by the database. Hashing will improve the speed of comparisons as we only need to do equality checking. Therefore, values can simply be compared after they are hashed to see if they are equal. This allows us to eliminate the current use of SQL inner join statements in favor of lookups on indexed columns. By indexing the entries in the database by the columns that are important for equality comparison we can quickly lookup information. Our solution also reduces the space of the table as we will only need to store the hash values which can be smaller than the original values and only stores those attributes that are necessary for inconsistency detection. This reduces the table size and will allow more or all of it to be loaded into main memory at once.

First, we will create a configuration file format. This file will specify how we should process the daily reports received. We will then create a configuration parser that will turn this file into a configuration object. The configuration object will be utilized by the raw data parser to accept the daily reports that Kingland receives. As each item in a report is parsed it will be sent to the inconsistency matcher. By sending the parsed data one element at a time we will not have to bring all of them into main memory at once, which will improve the performance of our program. The inconsistency matcher will check the element against the inconsistency database. If there is an inconsistency, then the element will be sent to an inconsistency report. Otherwise it will be sent to the exporter which will add it to the inconsistency database. An overview of the different modules used in the project can be found in *Figure 2.1*.

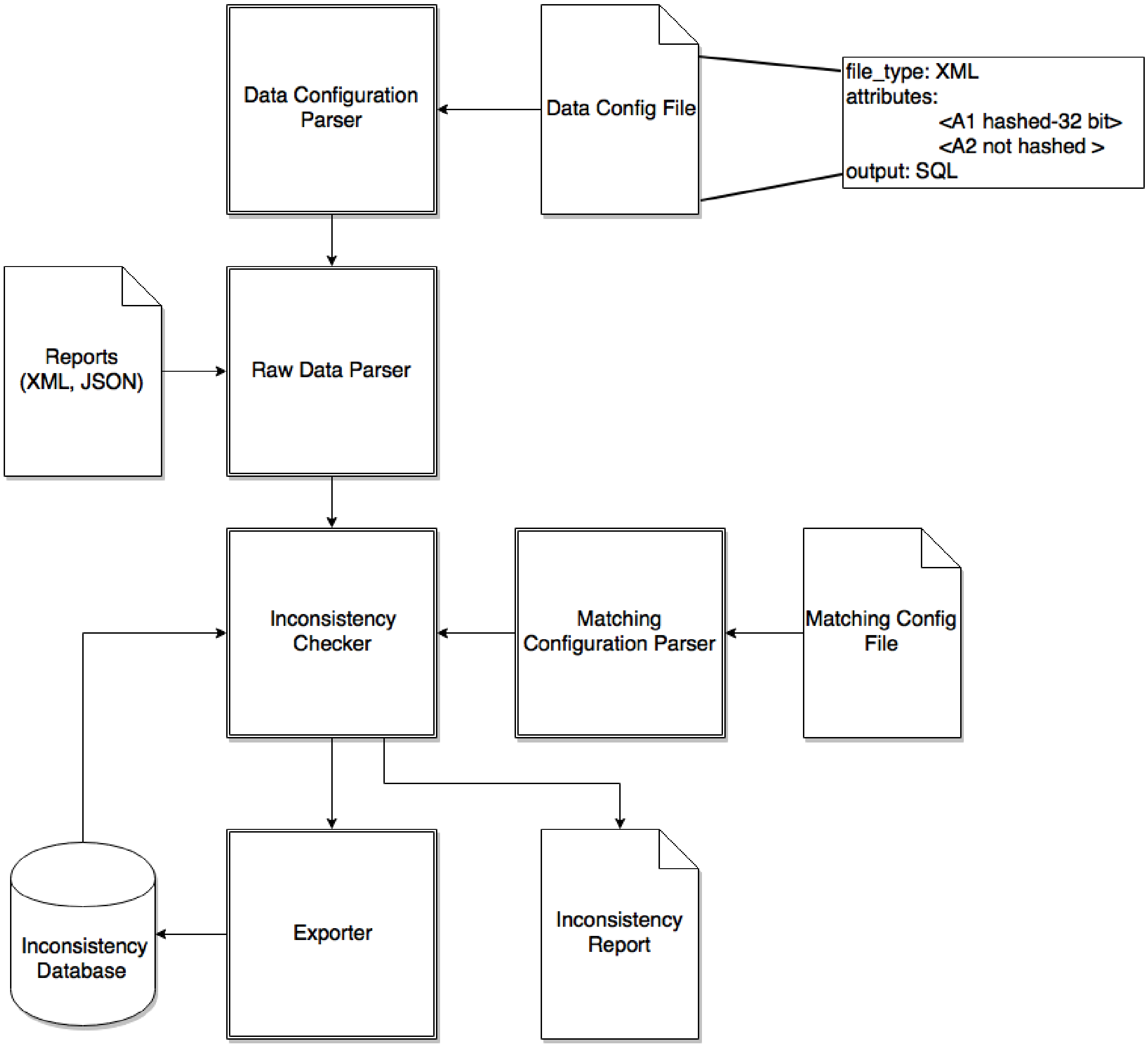

Figure 2.1: Block diagram of high-level system architecture.

2.3 Assessment of Proposed Methods

2.3.1 Technical Approach

The design laid out above fulfills the specifications laid out in section 2.1 through a variety of methods. The use of a configuration file allows us to handle various forms of input and to utilize only the relevant information by any given scan by allowing Kingland to specify those things before the program runs. Creating a hashed database allows us to get rid of SQL inner-join statements because we can just do simple equality comparisons. It also lets us do inconsistency checking in less than one hour using the speed of equality checks. Also, by using a good hashing function will produce few conflicts and therefore only a few false positive marks. Finally, our design allows ours programs to easily access the inconsistency database used in inconsistency detection.

2.3.2 Strengths

This solution has several strengths that make it an appropriate choice for this problem. The first strength is the ease of implementation. This solution is built on a few small parts that can be implemented with relative ease. This will give more time to analyze the proposed solution against the existing solution and other industry solutions to determine its viability. Another strength of this solution is it is very modular. The solution is separated out into several distinct parts and changing one part will not require changes to the entire design. The solution also allows for us to quickly adapt to changes in the problem statement. The design of the configuration file allows for us to quickly add in new inconsistencies if needed and modify the settings of current ones to adapt to any changes we encounter. Finally, the configuration file also allows us to testing much easier as it can be utilized to output to multiple different types of database or files so they can be tested against each other.

2.3.3 Weaknesses

This solution comes with a few tradeoffs including having to duplicate data into another database and the possibility of not being able to detect every type of inter-record inconsistency. Both of these shortcomings are small issues. While usually duplicating data isn’t a great solution, in this case the duplicated data will actually take up less space than the original and only needs to updated as often as the original data. Also, not being able to solve every inconsistency isn’t a huge issue as even if we can only solve a large amount of these issues it would still reduce the time needed to run a inconsistency scan significantly.

A more significant issue with this solution is the potential for false positive detections of inconsistencies. Since the values we compare will be hashed, there is a possibility of collisions. This is a tradeoff of speeding up the system that is deemed acceptable. As an analyst will need to go through the flagged inconsistencies, to determine appropriate actions, it will be a simple matter for them to mark it as a false positive and move on.

The biggest shortcoming of our proposed solution would be that a third party solution may be able to do this job almost as well. In this case it may be easier for Kingland to use this third party solution as it would have better support from a full development team, and could be used in some of Kingland's other solutions. In order to address this we will be comparing the performance of our solution with several standard industry solutions. This will allow us to report our opinion to Kingland on which solution will be best for their needs.

3 Project Requirements/Specifications

3.1 Functional Requirements

- Solution must not use SQL inner-join statements
    - Kingland's current solution to this problem is to use SQL inner-join statements which can take a long time. Thus, our solution should eliminate these statements to save time.
- Solution must utilize only relevant information
    - Our solution needs to run using only the small amount of fields needed to actually detect an inconsistency. This will reduce memory utilize and speed up the detection.
- Solution must compare current records to previous records as well as other current records
    - Our solution needs to be able to compare inconsistencies between records found in new reports and also between new reports and previously saved records.
- Solution must validate all fields are present in data
    - Our solution needs to ensure that all required fields are in each received record.
- Solution must handle various forms of input
    - Our solution should be able to handle data input in multiple formats including XML and JSON.
- Solution must update inconsistency database after analysis
    - Our solution should update an inconsistency database so that future checks will work with the latest version of the database.

3.2 Non-functional Requirements

- Solution must perform inconsistency check in less than 1 hour for daily reports
    - Our solution should be able to detect and report all inconsistencies in a new report in an hour or less.
- Solution must be able to analyze 100 million or more records at a time
    - New reports can have 100 million records or more. Therefore, our solution should be able to handle this size of input.
- Solution must run on Kingland's system
    - Our solution is a proprietary solution for Kingland and therefore must be able to run on their hardware.
- Solution reports less than 5% false positives
    - Our solution should have less than 5% false positive inconsistencies to reduce the time spent fixing these.

3.3 Standards

3.3.1 Testing Protocols

First, a protocol that will be implemented in the test environment is a list of items and features to be tested. This will be maintained in the design document as well as the pass/fail criteria for each of these items. The purpose of this protocol is to verify that the function of each feature has been fully developed and that there is a method of determining how successful the feature is. This protocol is also outlined by *IEEE 829* as a good practice for writing test documentation.

Second, we will use a protocol to utilize a variety of testing techniques to test all facets of the product and ensure it meets the standards outlined by Kingland. These testing techniques are outlined in great deal in *ISO/IEC/IEEE 29119-4* and will serve to group similar functions into actionable test sets.

The third protocol our tests will follow is to use a third-party logging software to assess the execution time of each function in the product to determine bottlenecks in the xml processing and inconsistency detection and provide insight on how those processes can be more efficient and timely. This is not specifically outlined by the IEEE, but does fall under the broad category of performance testing.

The final protocol to be implemented in the testing of this product is to have complete code coverage in the unit tests. This doesn't ensure complete error detection, but it does ensure that the behavior of the SUT (software under test) is thoroughly understood by the developer and that any future changes to the SUT will not affect the current functionality of the software.

3.3.2 Ethics

None of these practices should be considered unethical by ISO, IEC or IEEE because they are primarily gathered from standards outlined by these organizations. If any of these practices are deemed unethical by the team at a later date, they will be revised or removed so that they are no longer unethical and will adhere to principles and criteria outlined by the ISO/IEC/IEEE.

3.3.3 Project Applications

These standards are very applicable to this project because they outline how the team will perform testing and will give a guide to follow when writing and executing tests so that the tests will all have a common level of detail and structure so that a level of risk can be removed from the product. This is important because the less risk associated with the product, the more accurate the schedule will be and unforeseen costs due to risk will be mitigated.

4 Test Plan

4.1 Validation and Acceptance Testing

Requirement	Validation/Acceptance test
Solution must not use SQL inner-join statements	A Style Checker will be used to ensure that SQL inner-join statements do not appear in the production code.
Solution must utilize only relevant information	The size of the Inconsistency database created by this solution shall be compared to Kingland's central database to determine if this requirement is satisfied by our solution.

Solution must compare current records to previous records as well as other current records	Using Kingland's current solution as an Oracle to determine if our solution detects the same inconsistencies.
Solution must validate all fields are present in data	A unit test will be used to confirm that any missing fields in the raw data are flagged as such.
Solution must handle various forms of input	Unit tests will be used with multiple forms of sample data to determine how well the parser can handle different configurations of input data.
Solution must update inconsistency database after analysis	Kingland's main database will be checked to verify that it has been updated with the information that has been verified as consistent.
Solution must perform inconsistency check in less than 1 hour for daily reports	We will compare performance log files gathered in Log4J to determine the success of this.
Solution must be able to analyze 100 million or more records at a time	This will be validated using actual data Kingland receives on a daily basis and success will be determined based on whether or not our solution can perform faster and with the same accuracy as Kingland's system.
Solution must run on Kingland's system	We will work with Kingland to deploy our solution on their machine to test its performance. We will also attempt to test on machines with similar specifications to Kingland's.
Solution reports less than 5% false positives	This will be tested by processing inconsistency files and determining an average number of false positives.

Table 4.1: Table of validation and acceptance tests.

4.2 Documentation

Documentation is an important part of software testing and development because it allows developers to understand tests, categorize bugs, track the progress of a bug fix and keep a history of encountered bugs. Because of this, defect reports will be used to track and record bugs found during development and testing and will be recorded as an Issue in GitLab using a template.

4.3 Testing Procedure

The testing process for this project revolves around test driven development. Unit tests will be written to capture the functionality of a module and then the amount of code necessary to make the test pass will be written. A test suite will be developed in parallel to the production code and will be run with continuous integration on GitLab. This will allow the developers to have constant feedback as to whether or not their code changes currently implemented functionality of the code or if the changes are in-line with the functional requirements of the module.

If a developer finds a bug, they will attempt to recreate the bug and record those steps in an bug report Issue in GitLab. The bugs will be assigned to developers to fix at our weekly meetings if the severity and importance are “Low”, otherwise more immediate action will be needed and the Quality Assurance Lead will assign the issue to the developer deemed most qualified to fix it.

5 Challenges

5.1 Feasibility

This project is feasible because every member of our team has been exposed to the various components of our solution, such as Java, hashing, and SQL. While the project may prove difficult to integrate since there are a lot of modular components, we have a firm grasp of the underlying technologies required to implement a solution. Setting up a system where we can test our code on a very large dataset is required and after consulting with our faculty advisor we believe we will be able to do so using university resources. For these reasons, we are confident in our ability to create a proprietary solution. The feasibility of implementing existing big data solutions is more variable since none of our group members have experience in this area. However, we have worked research time into our schedule and this step involves adapting an existing solution, not development of a new solution. Because of this, we do not foresee implementing existing solutions reducing project feasibility.

5.2 Cost Estimate

We do not expect to incur a cost for developing our proprietary solution. This is due to the project being entirely software based, us not receiving payment for our work, and each member having access to the necessary equipment through Iowa state or personal ownership to complete this project. After we finish, the machine at Kingland where we deploy our solution will have associated running costs. However, they already perform this task with a dedicated machine and our solution will end up saving them money since it should be running less often. Third party solutions that we will look into usually have a per usage hour cost associated with them, typically less than a dollar per hour. Thus, this cost would remain relatively small in the development phase and only become substantial if Kindland decides to pursue one of these options permanently. We anticipate working on this project approximately six to ten hours a week for the entirety of two semesters.

6 Timeline

6.1 First Semester

The first semester will be focused on design of the proprietary solution. In the course of the semester a prototype of the proprietary solution will be created. This prototype will be demonstrated to Kingland to confirm that it functions as desired and to present the time estimates of the final application. The time estimates provided are based on the complexity of each part and the date it will need to be delivered by. Table 6.1 below shows the timeline of the semester indicating when deliverables should be completed.

Deliverable	Description	Start Date	Due Date
Project Plan V1	Initial draft of the project plan	09/15/2017	09/24/2017
Team Website V1	Initial version of the team website	09/15/2017	09/24/2017
Project Prototype	Prototype version of the application	09/27/2017	12/01/2017

Config File Prototype	Prototype of configuration file for report parser	09/27/2017	10/06/2017
Design Document V1	Initial version of the design document	10/06/2017	10/13/2017
Configuration Parser Prototype	Prototype of the data configuration parser	10/06/2017	10/23/2017
Raw Data Parser Prototype	Prototype of the raw data parser	10/10/2017	11/06/2017
Project Plan V2	Revised project plan	10/20/2017	10/27/2017
Exporter Prototype	Prototype of the exporter with basic hashing	11/01/2017	11/16/2017
Inconsistency Detection Prototype	Prototype of inconsistency detection	11/07/2017	12/01/2017
Final Project Plan	Final version of the project plan	11/27/2017	12/01/2017
Design Document V2	Revised Design Document	11/27/2017	12/04/2017

Table 6.1: First semester project timeline.

6.2 Second Semester

The second semester will be focused on the complete implementation of the proprietary solution and analysis of industry solutions. The beginning of the semester will be spent making the prototypes, developed in the first semester, fully functional. Once this is completed the solution will need to be analysed to ensure that it meets Kingland's expectations. Our group will also begin research into industry standard solutions to see if these can provide more benefit to Kingland or if they can be integrated with the proprietary solution. A final analysis of the solutions will need to be developed with our recommendation to Kingland as to what solution will best fit their needs. The time estimates provided in the table are based on the complexity of each deliverable along with the team knowledge in the areas related to each deliverable. Table 6.2 below provides the overall schedule of the semester.

Deliverable	Description	Start Date	Due Date
Implementation	Implementation of proprietary solution	01/08/2018	02/22/2018
Data Configuration File	Create final format of the data configuration file	01/08/2018	01/15/2018
Matching Configuration File	Create final format of the matching configuration file	01/08/2018	01/15/2018
Data Configuration Parser	Create final version of data configuration parser with all functionality	01/16/2018	01/23/2018
Matching Configuration Parser	Create final version of the matching configuration parser with all functionality	01/16/2018	01/23/2018
Raw Data Parser	Create final version of the raw data parser with all functionality	01/24/2018	02/07/2018
Inconsistency Checker	Create final version of the inconsistency checker with all functionality	01/24/2018	02/07/2018
Exporter	Create final version of the exporter with all functionality	02/08/2018	02/22/2018
Analysis	Analysis of proprietary solution against industry standard solutions	02/23/2018	03/23/2018

Runtime analysis of proprietary solution	Analysis of the runtime needed for the proprietary solution on large datasets	02/23/2018	03/01/2018
Analysis of Apache Spark	Analysis of utilizing Apache Spark for inconsistency detection	02/23/2018	03/16/2018
Analysis of Azure Data Lake	Analysis of utilizing Azure Data Lake for inconsistency detection	02/23/2018	03/16/2018
Analysis of Hadoop	Analysis of utilizing Hadoop for inconsistency detection	02/23/2018	03/16/2018
Final Analysis of Proprietary and Industry Solutions	Analysis of the best option for Kingland to utilize in inconsistency detection	03/16/2018	03/23/2018
User Manual	User manual for proprietary solution	03/23/2018	04/06/2018
Final Report	Final report of project outcomes and analysis	03/23/2018	04/20/2018

Table 6.2: Second semester project timeline.

7 Closing Material

7.1 Conclusion

Kingland is in need of a product that can detect inconsistencies in large data sets in an efficient manner so that they can reduce the resources necessary to run these daily detections. Our product will solve this problem by using hash functions to reduce the size of the information that is being compared as well as only comparing specific information that is sensitive to individuals and companies to ensure the consistency of that information. This product will execute faster than the current method of using SQL inner join statements and will allow a larger amount of data to be processed concurrently because of the smaller overall data footprint. Our product will save Kingland time and money in their inconsistency detection.

7.2 References

Haufler, Andreas. “Conveniently Processing Large XML Files with Java.” *Dzone.com*, 10 Jan. 2012, dzone.com/articles/conveniently-processing-large.

Murnane, Tafline. “ISO/IEC/IEEE 29119 Software Testing.” *ISO/IEC/IEEE 29119 Software Testing Standard*, softwaretestingstandard.org, 24 Oct. 2013, www.softwaretestingstandard.org/part4.php.

Smrcka, Ales I., Ph.D. "TEST PLAN OUTLINE (IEEE 829 Format)." *IEEE 829 - Standard for Test Documentation Overview*. Brno University of Technology, n.d. Web.

Staveley, Alex. “JAXB, SAX, DOM Performance.” *Dzone.com*, 31 Dec. 2011, dzone.com/articles/jaxb-sax-dom-performance.

Sug, Hyontai. "An Efficient Method of Data Inconsistency Check for Very Large Relations." S International Journal of Computer Science and Network Security 7.10 (2007): 166-69. Web. 22 Sept. 2017.

Zhang, Du. (2013). Inconsistencies in big data. Proceedings of the 12th IEEE International Conference on Cognitive Informatics and Cognitive

Computing, ICCI*CC 2013. 61-67. 10.1109/ICCI-CC.2013.6622226.

7.3 Appendices

7.3.1 List of Figures

7.3.2 List of Tables