Data Verity Time Machine

Kyle Cooper, Krista Dunlap,
HaoXin Luo, Damien Churchwell

June 18th, 2019

1 Introduction

Data Verity, Inc. is a technological solutions company that provides cloud-based tools to meet the business needs of financial institutions. They provide data warehousing services including business intelligence, relationship management, knowledge training, and problem solution. These solutions grant their clients financial insight and customer data analysis. They were founded in 2000 by two brothers: Gordon and David Flammer. Throughout the summer field session our main contact was David, and he guided us through the project, providing assistance and teaching us the inner workings of Data Verity's database.

Data Verity stores its clients' data within a production database where it is often modified by its users. Maintaining the integrity of the database becomes a critical part of the design. Their servers store a wide variety of data types from customers in different industries, and changes are frequently made to the data. The primary goal of this field session was to provide Data Verity with a means to track and revert users' changes to the production database. Our client essentially wanted a backup software application for their database implemented in their system to help them achieve that goal.

Our product Database Time Machine offers our client a solution to track and visualize changes to their database within a graphical user interface (GUI) and revert those changes if necessary. In particular, it is important to know who made any given change, when the change was made, and what data was altered. Since some sections of the data are lengthy, such as large blocks of code or text, it was also necessary to create a visual differ to allow an organized display of data with syntax coloring and highlighting of changes made.

2 Requirements

The first thing we took into consideration when starting the project was the general requirements that the client had put in place. Thus, we talked with the client and were able to come up with a list of functional and nonfunctional requirements. These requirements are listed below. However, it is important to point out that, due to the short period of time we had to complete the project, the client was willing to be somewhat flexible when it came to these requirements.

2.1 Functional Requirements

The two main components of the product are the central repository table and the user interface.

- Central Repository Table (CRT)
  - Show the state of the record before and after users' editing changes
  - Track who inserts or deletes data and when/where

  - Track differences between versions
  - Track differences using CodeMirror for large text strings comparison
- User interface
  - Display the changes tracked by the CRT
  - Display these changes in an easy to understand way
  - Give the options to revert changes between different editing versions
  - Implement buttons to toggle between editing versions
  - Allow fullscreen editing of large text fields being changed

2.2 Nonfunctional Requirements

- The program must run on Data Verity's server
- The program must be written using JavaScript and PHP
- The program must be written using MySQL
- The program must utilize Data Verity's database
- The program must be uploaded to a Subversion repository on Data Verity's servers
- The program should have thorough documentation
- Unit test cases should be included to ensure consistent functionality

3 System Architecture

Another important aspect of the project is the system architecture and technical design. These are the essential building blocks of the project and are crucial parts of understanding its construction and functionality. In particular, we will talk about the overall functions and interactions of the application and its two main components: the Text Differ and Form Differ.

The app performs two major tasks: tracking and displaying changes made to the Data Verity database, and reverting those changes. That is, it must be able to undo changes made and restore previous versions of the database. The app consists of a graphical user interface (GUI) that serves as a way for a user to review and revert the changes to various repositories as well as a backend that provides the functionality. The backend is mainly comprised of a table trigger, which tracks the database changes, and a controller, which is able to revert the database back to a previous configuration.

Most of the technologies and software used were predetermined by the client, Data Verity, Inc. They have expressed that we should use PHP, MySQL and JavaScript. The GUI is

mainly comprised of PHP and JavaScript and constructed in ExtJS, which is a specialized JavaScript library. Data Verity's web site was built using ExtJS, so they wanted us to implement our application in a similar design that would be compatible with other system applications. ExtJS is designed for object oriented JavaScript that delivers cross-platform web applications. ExtJS is based on components working together, and implements other techniques like DOM scripting and asynchronous loading. ExtJS was difficult to use immediately because we all had limited experience in JavaScript and learning a library is difficult when unfamiliar with the base language. We have also elected to use CodeMirror as part of the GUI, which is another JavaScript library that helps with text editing and difference checking. CodeMirror was not a required library, but was suggested by our client.

Figure 1 depicts the high-level system architecture. The system was constructed from multiple distinct components:

- Data Verity Database: This is the MySQL database that the company already uses. Our product is built on this foundation, and is dependent on the underlying form structure and objects.
- Three-way differ: When conflicting changes are made, a three-way differ is opened to compare your changes with the conflicting changes and the original version.
- Two-way differ: This is a simpler version of the differ that can be manually opened to compare your changes to the original version of the form. This will also be accessible from the CRT in the future, but this feature is unimplemented.
- ExtJS Table Layout: The ExtJS library provides a table JavaScript structure which we used to organize the differ display on the webpage. Each row is an entry in the form, and each column is a different version of that entry.
- CodeMirror Merge View: Also labeled Text Differ in our code and documentation, the merge view object spans multiple columns and displays all versions of a text or code field. This object contains CodeMirror text editors that provide a number of useful features.

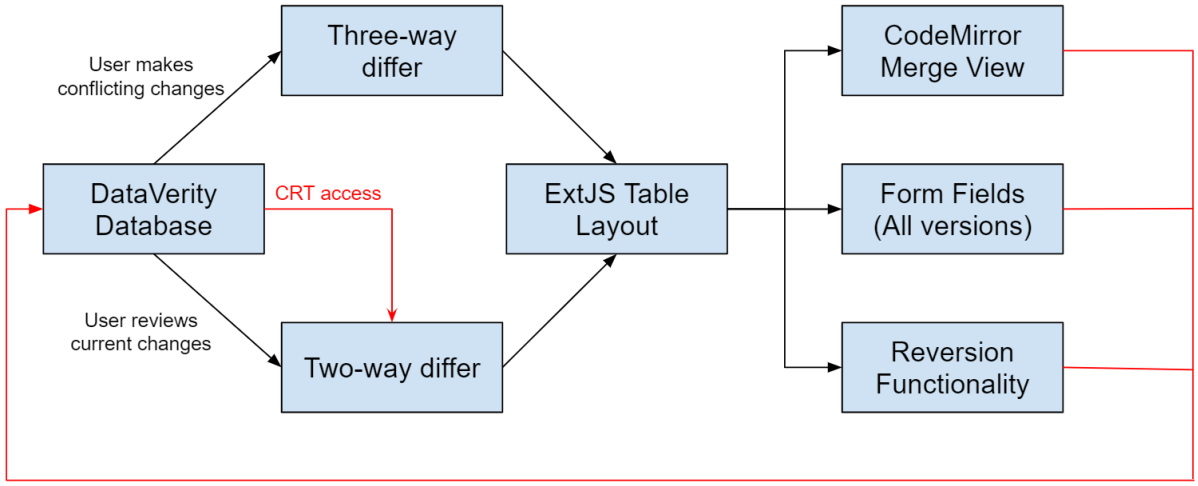

Figure 1: High-Level System Architecture

The final version of our product does not connect to the CRT. Therefore, changes made are currently only stored in the differ form, and not relayed to the database. When implemented by the client, the CRT will be able to open a separate two-way differ that behaves the same way, but displays historical changes rather than changes currently being made.

The Text Differ object, as seen in Figure 2, below, takes multiple strings as parameters, either 2 or 3, depending on the panel mode. It then creates a CodeMirror Merge View object that presents the differences of each string and allows for editing of the main/current string. An arrow button is included for each difference with functionality from the CodeMirror library. The arrow button allows the user to revert the changes from the current version back to the previous or conflicting version. The scrollbar also has a scroll lock that allows the user to scroll the 2 or 3 panels all together or separately. Furthermore, we implemented a key listener that allows the user to fullscreen the text editor by pressing the F11 key, allowing more efficient editing of large text strings. The object can also take sections of code that are labeled as PHP, SQL, or HTML in the database and apply syntax highlighting, an important feature for viewing large sections of code.

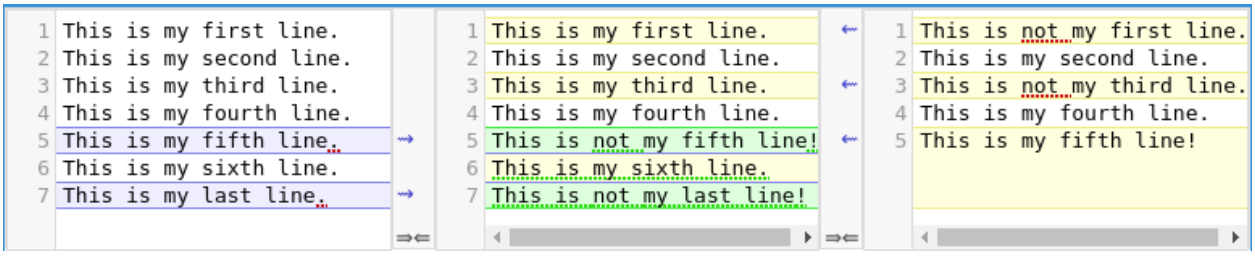

Figure 2: CodeMirror Merge View (3-Panel Instance)

The Form Differ (displayed in Figure 3, below) implements the Text Differ and provides a

more robust GUI for the user. It essentially looks through a submitted form and compares it to the original form. If there is a third-party form conflict (meaning that a third-party has edited the form while you also have the form open), then it also compares that form along with the two others. Form Differ implements the Text Differ for large text areas as well as sections of code, or it otherwise displays smaller sections of text and other fields as they are originally shown in their forms. In order to make up for the lack of functionality when displaying other fields as normal, we have added two main buttons. These buttons are arrow buttons and a current revert button. The arrow button functionality is very similar to the arrow buttons in the CodeMirror object. They allow the user to change their current changes to the previous version or conflicting version. The current revert button allows the user to revert their current changes field back to what the user had first submitted, in case they have tried to merge the field and messed up or want to see what they originally submitted.

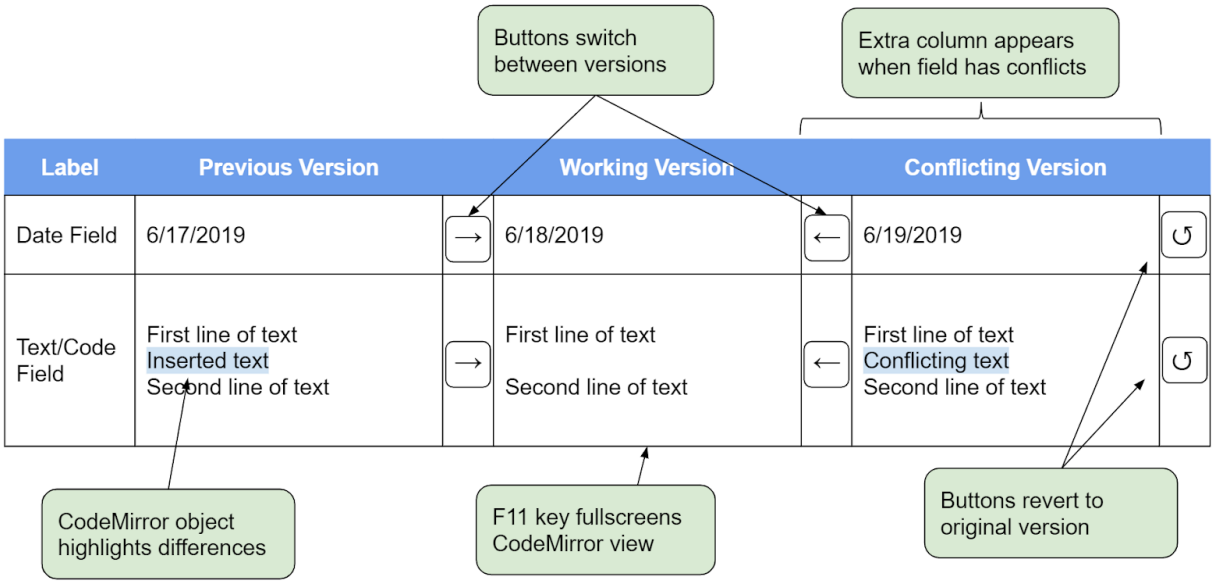

Figure 3: Representation of the Graphical User Interface (GUI)

4 Quality Assurance

The purpose of this section is to outline the various activities we took to ensure the quality of our software and which aspects they contributed to. Software quality is an important part of creating any sort of software, especially professional software. Quality control is crucial for a number of reasons. The first is that ensuring quality allows for a happier, more satisfied client because the end product has a higher grade and usability. The second is that it becomes much easier to maintain quality along the way as the project increases in size. This is because when building on top of something that is already excellent, it's much easier to finish with an excellent product.

In the next sections, we will describe each step of the Quality Assurance Plan and how it relates to establishing each aspect of software quality.

4.1 Architecture and Design

In the beginning stages of our process, we created an architecture diagram to plan out the structure of our product. We then had the design approved by the client to ensure that it met all of the functional and nonfunctional requirements of the project. This diagram provides guidelines for our project, allowing us to handle each section in a more organized fashion. Having this plan allows us to ensure we meet all the client's needs and wants. Having this plan approved by the client also makes sure we are contributing to the maintainability of the software because it follows an approved layout that the client can understand and maintain after it is complete. It also allowed us to improve the design so the size is limited and not growing exponentially or out of control.

4.2 Software Engineering Process

We used the Agile software engineering process. This flexible workstyle allows engineers to reactively create and program features as the client desires them. We participated in daily standups to inform one another on our progress. This helped us meet or modify the requirements laid out by the client. In order to stay updated on the week by week feature demands, we used Trello to outline client stories and specify complexity. Agile allowed us to get as much work done as possible in a limited time frame. This is particularly important as we had a slight time crunch, with only 6 weeks to finish the project.

4.3 Code Reviews

In order to assure that our code meets the clients standards, our group met with Data Verity at least once a week to discuss plans and weekly goals. These meetings were conducted over GoToMeeting, a webinar service. The purpose of these reviews were to ensure that our application meets the client's requirements and eliminate much of the potential for defects and bugs. The client's wealth of knowledge also helped us to avoid any roadblocks that might occur due to our lack of experience. Code reviews were also key in preventing code smell, to produce programs with reasonable size and efficiency. We also pair programmed as a form of code review to ensure that the code we produced was high quality and bug-free.

4.4 Coding Standards and Practices

When it comes to coding standards, we used two main practices: the SOLID principles and pair programming. In particular, this practice ensured that our application runs well in Data Verity's modular application environment. Pair programming has a driver and a navigator that synchronously write and analyze code. This practice ensured that tiny mistakes were caught for efficiency, and in the big picture, performance analyzation is done proactively. Thus, both of these ensured efficient and clean code as we developed our product.

4.5 Version Control

During the development phrase, Data Verity helped us to create a demo database for each team member, so that members on the team could work on their own version of the database rather than the production database. This effectively prevented any unintentional or undesired changes made to the production database. Our group also used Apache Subversion to maintain the current and historical versions of our code and the database. It speeds up the synchronization of pair-programming and team-oriented programming. Using these two techniques, we were able to minimize the chances of any mistakes or ill-intended acts that could be made on our database. Version control ensured the security level of our software all along the development phrase.

4.6 Testing

Our testing process involved active human testing of the functionality of the user interface. In the simplest form, this involved pressing implemented buttons to ensure that they produce the expected functionality. However, this also included acting as a malicious user to attempt to damage the program or the integrity of the data, and acting as an inexperienced user by interacting with elements of the interface randomly. This was intended to minimize the risk of such users breaking the program or exploiting any potential defects or bugs. Additionally, we documented our testing process to allow for future extension of the features and functionality that had previously been tested. This documentation process also provided an organized testing structure for us to reference, preventing both missed tests and redundant tests. The testing allowed us to ensure the performance of the software as well as allowed us to test for any possible security risks.

5 Results

Our Database Time Machine allows Data Verity to track and review changes in their database and provide a GUI for their clients and employees to use. We implemented a form differ that scans through a form and finds all areas that differ from the original information on the form. This form differ displays all the differences in a table for the user to compare. It also can display a third version of the form if a different user has modified the form recently. Using this form differ, the user it able to settle merge conflicts, revert changes, or edit the fields manually before a final submission. With assistance from the client, we were also able to set up a table (the Central Repository Table) that tracks changes within their database.

While we were able to make a good amount of progress, we were unable to link the differs and GUI with the CRT changes. However, with a little work, the progress we have made can be easily modified to be used for the CRT changes. The project could be adapted to any uses in which the client needs to revert or see differences with many, if not all, fields in their database.

In terms of our testing, we were only able to check that the GUI was displaying the correct

items and formats in the correct way. To test this, we made changes to a variety of fields (a short text field, a large text area, a date field, etc) and ensure the form selected the correct display. We also tested to make sure that the 2-panel and 3-panel modes were chosen correctly. As we mentioned before, 3-panel mode was chosen when a third-party also edited a document, creating a conflict. We also implemented two buttons to revert changes; one for the user's changes, and one for the conflicting changes. To test these we pressed each button after various different scenarios including without editing anything, after pressing a different button, and after manually editing the fields. With each of these tests and scenarios, we were able to confirm that the GUI and buttons performed well and correctly. We tested on all checkouts as well as on both the Google Chrome and Firefox browsers. The Data Verity team reviewed our features as we created them, and have approved the functionality of the final product.

5.1 Lessons Learned

One of the lessons we learned was that while a base language, such as JavaScript, might be simple to implement in the project, the client may require a language library be used, such as ExtJS or CodeMirror. Such a library may be more complicated and take more time to get familiar with. In particular, working with ExtJS panels and windows proved to be a more difficult aspect to grasp and work with. We learned to rely on our clients expertise, our other team members, documentation and StackOverflow.

Another lesson was to stay flexible when it came to adhering to our product's initial design. In particular, we found that we often had to redo code because of company-specific functions, formats and architecture design requirements even when our code worked. We would often work hard to get functional code before the client would review it, and explain that it was not what they wanted. They would then help guide us towards a better format or implementation for them. In the long run, getting familiar with clients' preferences and maintaining effective communication is critical to achieve a successful project delivery.

We also found that very small changes and mistakes could cause the whole website (in our case, personal checkouts) to stop working. This made it difficult to see which parts of the website were causing it to crash. To combat this, we learned to use the browser's debuggers efficiently as well as coding in very small intervals. Pair programming also proved very effective for helping us catch the mistakes and bugs before running our code in the test environment.