**Master Thesis**

**VISUALIZATION OF FUND DATA RECONCILIATION**



Submitted by:

Rohit Misra

1025275

Master of Engineering (IT)

Under the guidance of:

Prof. Dr. Andreas Pech

Prof. Dr.-Ing., Dr. rer. pol. Herbert Nosko

# Acknowledgement

I am very thankful to our supervisors Prof. Dr. Pech, Prof. Nosko, Mr. Ripoll and Mr. Wranik for his guidance and patience at every step of this thesis. Without their support and invaluable feedback, I could not be able to complete it. I acknowledge his contributions to enhance our knowledge on the subject.

I am also thankful to some of my colleagues for sparing their time to review my thesis, and their moral support.

Also, I cannot forget to thank my family and close friends, who always support me for my success. Their love always remains the key source of my motivation. I dedicate this thesis them.

Contents

[Acknowledgement 2](#_Toc410893233)

[1. Introduction 5](#_Toc410893234)

[2. Keywords 6](#_Toc410893235)

[3. Background 8](#_Toc410893236)

[4. Requirements Gathering/Analysis 11](#_Toc410893237)

[4.1 Usability 11](#_Toc410893238)

[4.2 Rapid development 11](#_Toc410893239)

[4.3 Compatibility 12](#_Toc410893240)

[4.4 Scalability 12](#_Toc410893241)

[4.5 Performance 12](#_Toc410893242)

[5. Architecture 13](#_Toc410893243)

[5.1 Database architecture 13](#_Toc410893244)

[5.2 Backend Architecture 15](#_Toc410893245)

[5.3 Frontend Architecture 18](#_Toc410893246)

[5.4 Framework architecture 19](#_Toc410893247)

[6. Design 20](#_Toc410893248)

[a. Selection of Technologies/frameworks 20](#_Toc410893249)

[6. Prototypes 23](#_Toc410893250)

[1. Recon Grid 23](#_Toc410893251)

[2. Recon Graph 24](#_Toc410893252)

[6. Development 25](#_Toc410893253)

[6.1 Selection of development environments 25](#_Toc410893254)

[6.2 Selection of a development methodology 28](#_Toc410893255)

[6.3 Principles of Agile methodology 29](#_Toc410893256)

[7. Implementation 31](#_Toc410893257)

[7.1 Workflow 31](#_Toc410893258)

[7.1 Frond End development: 32](#_Toc410893259)

[7.3 Backend Development: Spring Tool Suite 36](#_Toc410893260)

[7.4 Database (Oracle) Development: 38](#_Toc410893261)

[8. Testing 40](#_Toc410893262)

[8.1 Automated Testing 41](#_Toc410893263)

[8.2 Manual Testing 42](#_Toc410893264)

[9. User Acceptance 43](#_Toc410893265)

[10. Deployment 44](#_Toc410893266)

[11. Difficulties 45](#_Toc410893267)

[12. Future Development 46](#_Toc410893268)

[13. Conclusion 47](#_Toc410893269)

[References 48](#_Toc410893270)

# Introduction

This Master Thesis is aimed at exploring and describing the approach to resolve complex visualizations related to reconciliation of financial data and migration processes. This solution is to be meant to be an upgrade over the current reporting and data analysis provision, which is the usage of multiple excel sheets.

Background:

AMICON is acarda’s Migration & reconciliation of fund administration system solution.   
Its key features are:

* Automated migration & reconciliation is the key to successfully importing or exporting current fund administration systems, or integrating new ones.
* In addition, there is also an ongoing need for IT solutions for fund takeovers and mergers.

Other solutions developed at acarda also implement similar methodologies for fund transfer/billing or other financial applications across various clients. Hence the goal for this solution is to be compatible across all other platforms that might use financial reconciliation reporting. These applications use a variety of frameworks and architectures. So making the reconView solution compatible across all of them is taken as a challenge.

# Keywords

For the purpose of easy understanding of the general terms used in the document, the following keywords are explained here:

1. Solution: It is the resulting software, which will be delivered to the client and includes the Application bundled together with the changes made in the Target Applications.
2. Application: It is the new product, which is to be delivered to the client according to the specified requirements.
3. Target Applications: These are the main applications where the solution application created in the thesis is deployed.
4. AMICON: It is one of the target application which deals with fund data migration.
5. ABPM: It is another target application which takes care of real estate fund management.
6. Backend: The layer behind the Frontend layer which is essentially a Java program acting as a mediator between the database and the view.
7. Frontend: It is the user interface which the client or a user of the application sees to get further access to the software.
8. UI: It is also known as User Interface, and is the same as Frontend.
9. Modules: These are the executable output from the java code, which are deployed on the server. There are a total of two modules developed in this thesis.
10. Application Server: It is the environment, which runs the software that has been developed.
11. End user: Is anybody using the application through the user interface who possibly has a minimum knowledge about the technical details of the software.
12. Recon reports: After a fund migration or real estate payment schedule, a reconciliation is executed to compare the Source/Target or Expected/Actual values of certain fund attributes. Eg, fund price, or real estate payments.
13. Recon data: It is the raw reconciliation data which is stored as a pseudo data warehouse in the RE\_REC\_DATA table in the Database. In short, it is the comparative result of the reconciliation process
14. JSON format: Also known as JavaScript Object Notation, is a standard format for data transfer between the server and web application.
15. DOM: Also known as Document Object Model, it is the standard object model providing programming interface for HTML.
16. Recon GRID: The tabular format representation of the reconciliation result.
17. Recon Graph: The graphical representation of the reconciliation result. (Bar chart)
18. Recon UI: This is the application (as mentioned in point 3). It consists of two components Recon GRID and Recon Graph.

# Background

Usability engineering [[1]](#footnote-1) is a field that is concerned generally with human-computer interaction and specifically with making human-computer interfaces that have high usability or user friendliness. In effect, a user-friendly interface is one that allows users to effectively and efficiently accomplish the tasks for which it was designed and one that users rate positively on opinion or emotional scales.

Financial Industry Users differ such that they prefer dense visualization[[2]](#footnote-2) of data in a total view. Examples are e.g. trading applications, whereas securities traders prefer to use multiple displays simultaneously instead of paging on a single display. Financial Risk software or fund accounting software also displays many information, thus average user usability concepts are not fully applicable.

Acarda’s migration and reconciliation software (AMICON) is a solution to migrate fund data like positions, accounts, bookings and transactions from a source fund administration system to a target fund administration system. Examples of such systems are Simcorp Dimension or Multifonds.

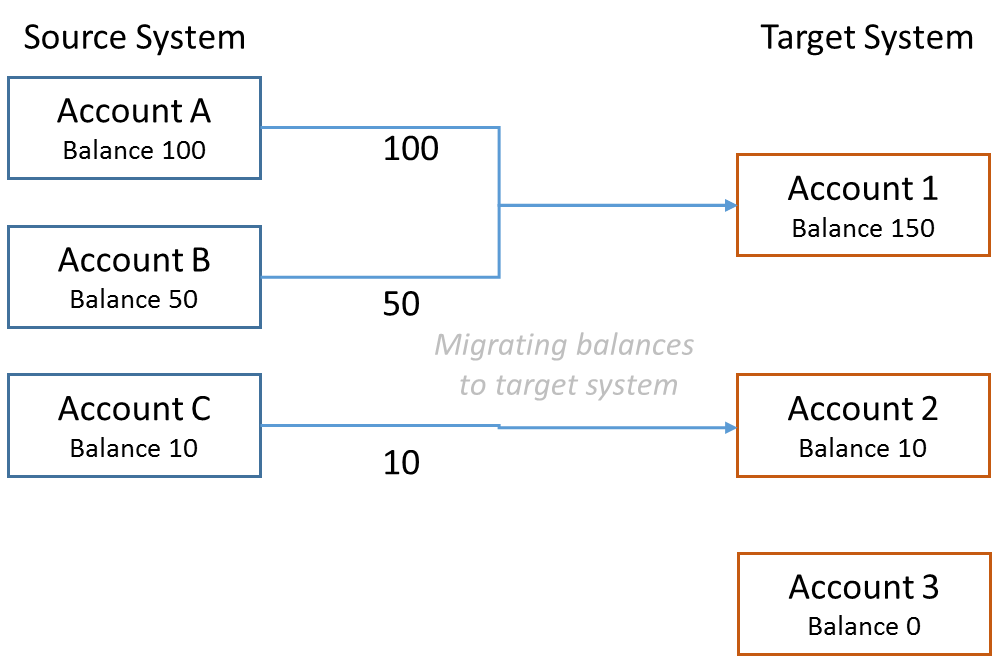


Figure 1 Example of account balance migration

The software provides a reconciliation function which performs post-migration data verification between source and target system. Differences may occur e.g. due to different valuation of assets or rounding issues.

Additionally, the reconciliation function is also applied in the so called “parallel run”, where after migration, both the source and target systems process new transactions and bookings and calculate the net asset value of the fund. The mapping rules from the migration phase (see e.g. aggregation in Figure 1) are incorporated into the reconciliation. This reveals, if differences from migration occur or change over time.

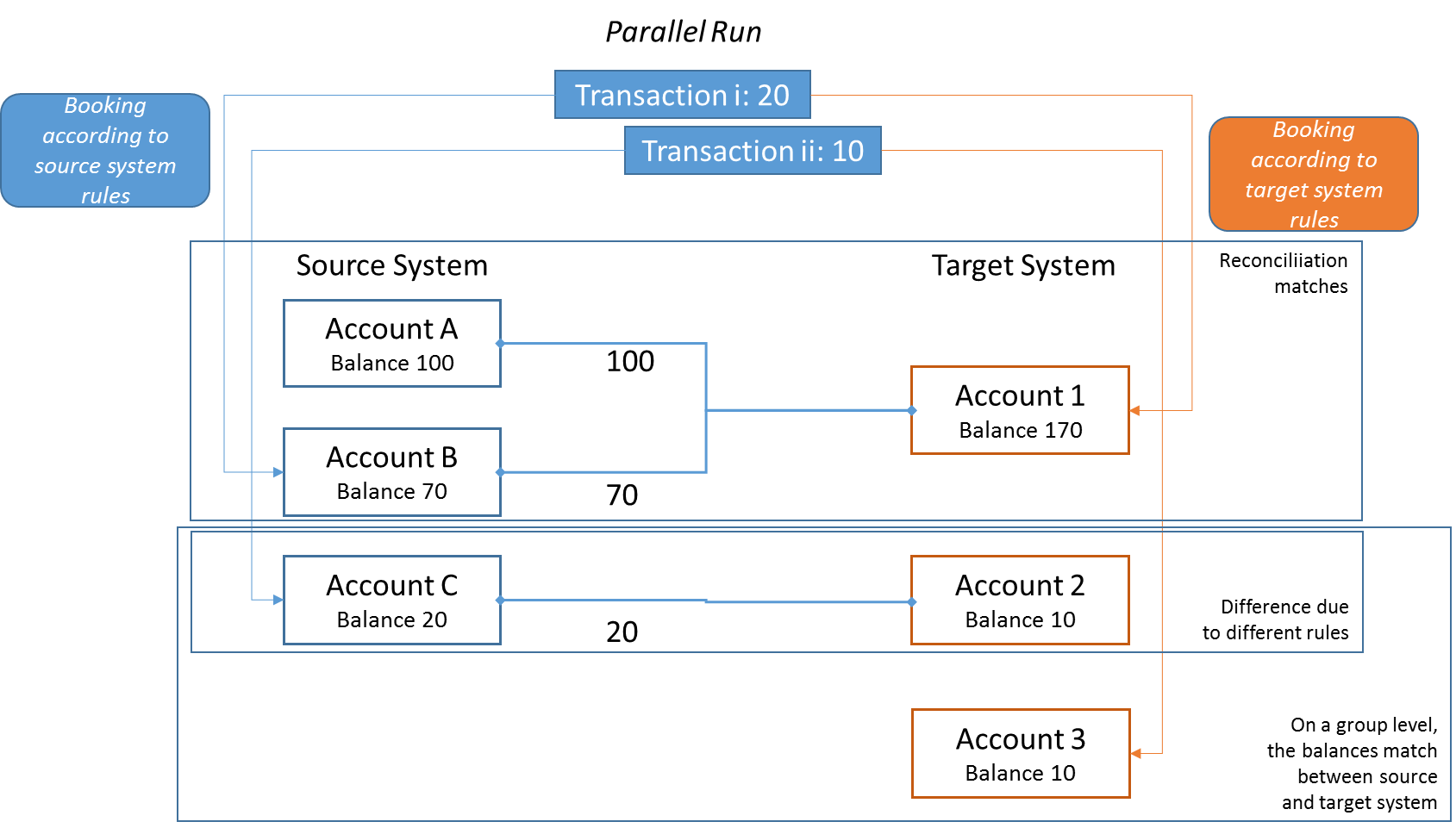


Figure 2 Example situation during parallel run

The reconciliation functionality thus uses logical consistency criteria and can be seen as an instance of semantic data integration[[3]](#footnote-3). The semantic rules and relations are built-in into the tool and not explicitly modeled as an ontology.

The User Interface for the creation and maintenance of the consistency criteria is limited such that SQL query parts need to be edited. A more business-user friendly modelling of criteria is a future extension.

Currently, the only visualization of the reconciliation results is done via an exported excel spreadsheet with highlighting of differences. Acceptance of the differences (see e.g. Figure 2, where there is no difference on aggregate level, thus difference is explained) has then to be done in the AMICON dashboard, a graphical user interface based on web technology, with additional functionality out of the scope of this thesis like workflow monitoring or account mapping.

The aim of Acarda is to incorporate the visualization of the reconciliation results into the AMICON dashboard in a financial user-friendly manner.

The goals of the thesis are:

* To provide a design specification for the visualization funded on a review of state-of-the-art techniques and approaches
  + The visualization design needs to be selected under the criteria of
    - integration into the Acarda tool
    - compatibility with existing software components
    - Maintenance effort
  + The constraints are
    - Oracle Database (views or PL/SQL are possible)
    - Java spring framework (used for the middleware of the User Interface)
    - JavaScript on the browser visualization side
  + Within the constraints, a degree of flexibility is given, e.g. data can be processed in a Database View, within a spring Java controller using a library or on the JavaScript side
* To develop the graphical user interface to the reconciliation data within the AMICON GUI (which is based on spring tool suite and dhtmlx-ui as JavaScript library) based on the developed design documentation

As for the visualization, client feedback can be used as a starting point. Here, clients asked for functions as usually available within a data warehouse cube, e.g. flexible perspectives or dimension change.

Clients wanted to see results per group, per instrument type or based on total tracking error (reconciliation difference).

Also, overview functions (e.g. using graphical visualization techniques) can be utilized.

# Requirements Gathering/Analysis

The client is looking to significantly increase its business efficiency and reduce its operational costs and sidewise reduce the time taken to generate reconciliation summaries and reports. The ReconUI will not only allow the customer to view the summary of the fund migration reconciliation faster and easier but also in a better meaningful and understandable manner.

Acarda GmbH strongly believes that the “ReconUI” has the potential to revolutionize the process of visualization of fund migration reconciliation.

The ReconUI is aimed at holding the following key features:

1. Having a grid view on demand for selected Accounts/Financial Instruments
2. The feature rich grid view (ReconGrid) enables end users to see a comprehensive overview of matches/mismatches and filter the data according to desired parameters.
3. Summary view as a graphical representation of reconciled results.
4. Faster loading of reconciliation data into the view.
5. User dependent configuration of the view.

The most important points which were not to be compromised while the research and development of this product were:

## Usability

The proposed solution must be high in usability as it targets business consultants and financial experts as the end users. Hence, there shall be a low level of complicacy in the design and structure. This implies, that they should be able to use the software with ease with minimal training and supervision.

## Rapid development

The development methodology decided for this product is SCRUM. This way of iterative and incremental agile software development framework which defines a flexible, holistic product development strategy where a development team works as a unit to reach a common goal.

## Compatibility

A key requirement of the ReconView is that it should be compatible with other applications which have different architectures.

## Scalability

The solution should be scalable to handle huge amounts of reconciled data and their substantial information and transform this data into the view.

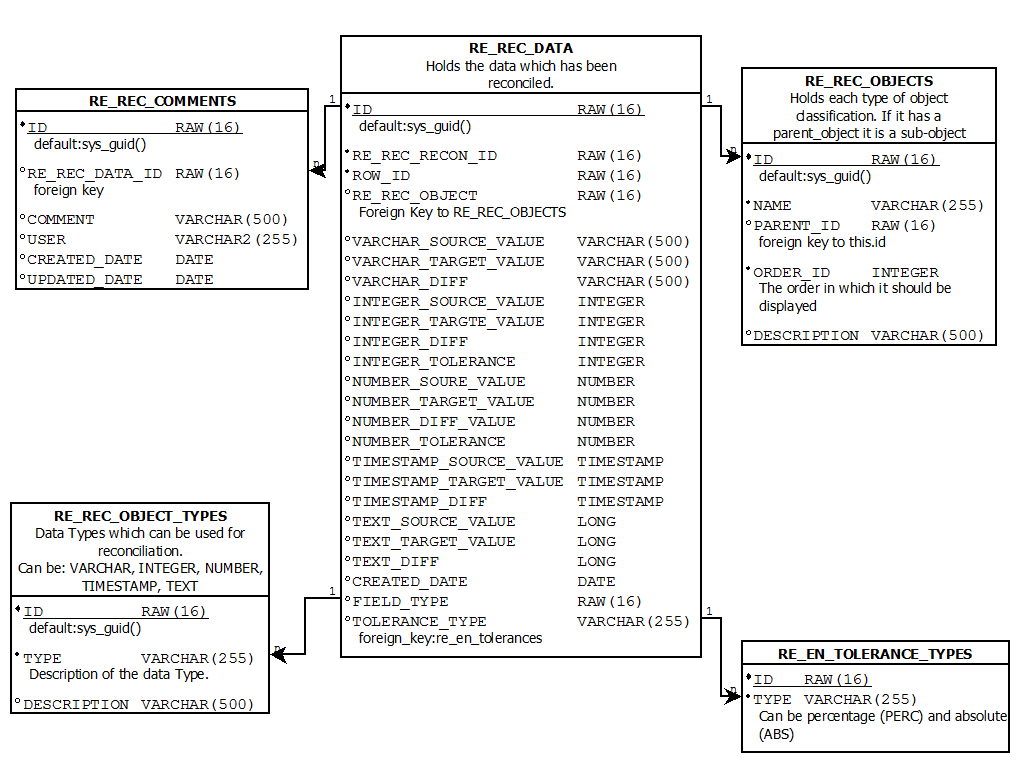
## Performance

Performance is an uncompromisable factor of the requirement for the requirement of this product.

# Architecture

The first consideration in the architecture was decided to make the solution independent of the application, i.e. it could be used seamlessly across the wide range of AMICON variants and also other applications such as ABPM.

## 5.1 Database architecture



After the fund migration process, the recon engine populates the reconciliation result as raw data into the RE\_REC\_DATA table.

This raw data is a collection of the values of each objects joined by a common row identifier. Each record in the RE\_REC\_DATA table is linked to an object identifier and a row identifier. Simply put, an object is a column in the final data representation in the UI and a row identifier groups values together to form a row.

Furthermore, each object is linked to a column in the RE\_UI\_COLUMNS table. This gives us the various view related configuration parameters for a column in the Grid to be shown in the UI/View, for example, the column name, the column width if the columns should be visible or not, etc.

The data required to construct the view was observed to be broken down into two major parts:

1. Grid information
   1. Column information
   2. Rows data

A hint of hierarchical data structure was sensed here and the ways to efficiently represent this hierarchical data was to be decided. After some research, the resolution was boiled down to two options.

1. The commonly sought of process to deal with this data is to have java classes mapped as database entities in the backend with suitable relationships among them to have a hierarchical relationship.[[4]](#footnote-4)

An example would be having a GridData class which has an ArrayList of Strings arrays for Row Data and an ArrayList of ColumnData class.

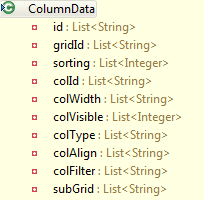


Figure 1 Example of a pojo storing Column data

1. The alternative and more innovative method of retrieving data is to have a PL/SQL procedure which joins the required tables and aggregates the data into a JSON object with can be received directly by the backend and with suitable pre-checks and additional parameters, can be directly passed on to the view where the JSON data is used for the configuration and data representation.

## 5.2 Backend Architecture

First of all the backend design was to be decided based upon the programming language and the server-side framework. With an array of choices, the research was done.

With the programming language, the available options were:

1. .NET
2. PHP
3. JAVA
4. node.js

Due to business and licensing reasons, Java was already a more preferred language.

After having chosen java, extensive use of the full features of Java Enterprise Edition was made to tackle the requirements. Even java has a wide range of offerings with the frameworks to be chosen.

Owing to known issues with the traditional framework and development methodology of J2EE applications, the following points were brought into picture according to [[[5]](#footnote-5)] and based on industrial experience:

1. According to several code reviews, J2EE applications have a tendency to be comprised of disproportionate volume of "plumbing" code, which is a code that does not do anything. Examples of such codes include JNDI lookup code, try/catch blocks used to acquire and release JDBC resources etc. These codes prove to be a major drawback for the resources that are used to write and maintain such code, which should instead be focusing on the business domain of the application.
2. Another drawback is the use of distributed object model which leads to code duplication and excessive amounts of code. It also depends on the business requirements as if the requirements suggest a distributed architecture, then the trade-off between complexity and performance needs to be accepted (Spring offers features for such scenarios).
3. The aim to introduce EJB while actualizing business logic in J2EE applications was to reduce the complexity instead it is unnecessarily complicated.
4. Overuse of EJB in every transactional application leading it use a distributed component model. The original motive of EJB was to be used in internally distributed and transactional applications.
5. Technology constraints are masked using the concept of design patterns. Many of the "J2EE design patterns" are not indeed design patterns but are used to cover the design flaws. There is an alarming need to investigate these ambiguous design patterns and search for uncomplicated, more rewarding approaches.
6. Unit testing of J2EE applications is tough, especially of the J2EE APIs and EJB component model. Unit testing is needed to accomplish high-test coverage and to emulate different failure scenarios, for e.g. connectivity to database. It is also important to establish short run time of tests while development or maintenance phases, thus reducing time for redeployment.
7. Entity beans, another component of J2EE technology have floundered due to restraints on object orientation and harmful for productivity.

In response to these issues, Spring framework was chosen to counter these underlying problems of classical J2EE programming techniques [5]. This is because Spring Framework enables Plain Old Java Objects to inherit the benefits of enterprise services. These POJOs are injected into java classes at runtime and can be re used in the runtime environment. This is termed as dependency injection.

Based on Java language constructs, dependency injection use the framework-specific interfaces. Through the use of methods or constructors that framework can access at runtime, the dependencies such as configuration parameters and collaborating objects are disclosed by application classes.

These POJOs will contain essential functionalities as database connection, user authentication etc. They will be injected into a java class serving as the main function of the program. This main function receives the request for a reconciliation report, fetches the data from the database, and transmits it to the UI. The transmission of data was decided to be JSON, and not XML[[[6]](#footnote-6)] for performance and parsing friendly reasons.

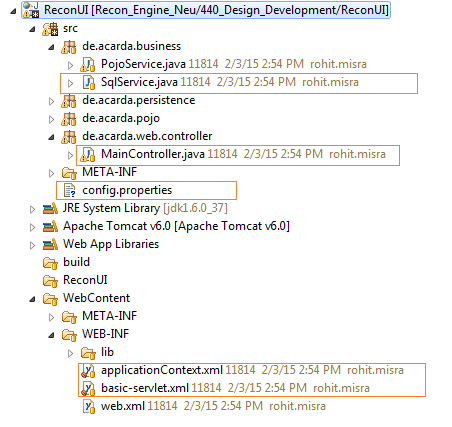


Figure 2 Entire Backend Structure

Now, the uncompromisable requirement of compatibility was to be tackled. Since, all the target applications which will inherit the ReconUI as an additional feature, are of different frameworks, authentication mechanisms and possibly database setups, this was a matter of debate.

1. The common way of dealing with this issue was to ignore it and customize the reconUI separately for the different applications. This was, the complete modules (both frontend and backend) of the reconUI was to be included into the target applications and in certain cases, separate database connections was to be maintained.
2. The innovative approach of dealing with the compatibility requirement is to have the reconUI as an independent module on the application server. The end-user interacts with the target application and in the scenario of viewing reconciliation reports, the target application, in the background communicates with the ReconUI module to retrieve reconciliation data in the final JSON format, and delivers it to the UI. The end user would have no perception of communicating with the ReconUI.

## 5.3 Frontend Architecture

The frontend application was decided to be designed in JavaScript using suitable libraries to support the additional and extended functionalities other than which JavaScript provides.

JavaScript was chosen over various other scripting languages available today because of several features that set it apart. [[[7]](#footnote-7)]

1. The asynchronous programming model enables events to be handles and suitable action be performed when the user interacts with the User Interface.
2. JavaScript makes extensive use of DOM API calls to access or manipulate web components or data stored in the browser. This gives the opportunity to change the contents of the page programmatically, without reloading the page.

After having a look at the requirements and the client’s expectations, it was evident that the tabular and graphical reporting views would change with respect to different scenarios, business requirements and also with time. Hence, this called for a fully customizable UI, which based on different parameters, would change the experience, appearance and also the behavior of the UI.

The intelligent way of constructing such a UI is to have a JavaScript function which acts as a class. Upon instantiating the JS function with suitable parameters, the recon grid or the recon graph is rendered[[[8]](#footnote-8)].

This gives the frontend an Object Oriented (OOP) feel. As it is in classical object oriented programming, the ReconUI class and ReconGraph class will hold in them certain members to describe the way the object is and its behavior:

1. State variables, to hold the component’s state[[[9]](#footnote-9)] (as default and parameterized) as certain variables which will be the base of the construction of the UI components. For example, size, location, position, identification of the report to be displayed, etc.
2. Functions, or as it is referred to in OOP nomenclature, behavior or methods, which enable the object to interact with the user and the backend. For example, load(), refresh(), delete(), close() etc.

These two components will be scripted into two JavaScript files and will be included in the application. The point of inclusion into the solution was another point of debate. Two, most significant methods were thought of:

1. Dynamic fetch of the JavaScript files from the ReconUI module on demand would enable the target application, which is already independent of the ReconUI in the backend scenario, also to be independent of the ReconUI frontend. This method, would obviously implement cross browser scripting, and was hence was declined due to the serious security issues and permissions required to be addressed. [citation for cross browser scripting]

The other method thought of in this regard, was the merging of the ReconUI frontend (two JS files) directly into the frontend of the target application. This solution was safer to handle and easier to implement and deploy.[[[10]](#footnote-10)]

## Framework architecture

The way the data is transferred across the applications and the server was very vital for deciding the architecture of the application. Two commonly used APIs were put to discussion as to which would be suitable for the current application, keeping simplicity, reusability and deployability in mind.

1. SOAP API

SOAP API, or Simple Object Access Protocol, relies on XML defined by schemas to construct and maintained a strongly typed and rule based messaging structure. Data elements are enclosed within strictly named tags and their structure is governed by a schema definition called WSDL (Web Service Definition Language), which is optional.

If, the size of the transmitted doesn’t matter, and an effective client server and function driven communication is more important, SOAP is a better option. But owing to its data heavy XML tags, after careful consideration, this option was not considered.

1. REST API

Unlike SOAP which is a standardized protocol, REST API is merely a framework or a methodology. It makes user of HTTP and commonly JSON. Owing to the JSON format, this way of data transmission doesn’t follow strict typing and definitions.

This is a better option for this particular solution because, it is more performance format friendly.

# Design

## Selection of Technologies/frameworks

A wide range of technologies and platforms were considered for this application. These technologies were compared based on multiple parameters explained below.

1. Performance

The selected architecture, framework or methodology should not compromise on performance which would lead to slow and poor customer experience, and put high cost on hardware.

In the assessment on performance, **Oracle** database was found to be the most favorable selection. MongoDB, on the other hand being a NoSQL database, was brought down on performance on retrieval and insertion of large amounts of data at once.

**Java**, on the other hand was more performance oriented than the rest of the possible offerings owing to the highly agile Spring framework, faster compiler and interpreted bytecode.

1. Scalability

The solution should be scalable to a higher number of users, data sets and a larger network model.

**Oracle** and MS-SQL Server promised greater scalable opportunities with possibilities of clustering, partitioning and warehousing technologies.

In the programming consideration, Java appeared to be more scalable because, the intended use of Spring Framework and POJOs gave way to separation of concerns. Keeping the parts dealing with the database and the frontend independent was possible.

1. Convenience of license:

Licenses are a crucial consideration while the development of software. Frameworks/technologies with high licensing price and less support framework leads to a high cost addition to the development process. Hence, preference was given to those tools whose license was already available to the firm. **Oracle** and **Java** being on the list.

1. Ease of installation:

The development process, the testing and subsequently, the deployment process should be easy to describe and perform.

* + 1. Database:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Database | Performance | Scalability | License | Installation |
| MS-SQL |  |  | **** | **** |
| MYSQL |  |  | **✓** | **✓** |
| Oracle **✓** |  |  | **✓** | **** |
| Mongo DB |  |  | **** | **** |
| DB2 | **** | **** | **** | **** |

* + 1. Server side framework:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Framework | Performance | Scalability | License | Installation |
| JAVA **✓** |  |  | **✓** | **✓** |
| .NET |  |  | **** | **** |
| PHP |  |  | **✓** | **✓** |
| Node.js |  |  | **✓** | **** |

* + 1. Client side framework:

Two of the best Java Script Frameworks were chosen to design the interactive user interface like the data grid and the charts needed to represent the data in a tabular and graphical form.

* 1. DHTMLX UI

The DHTMLX library is a powerful JavaScript framework which generates User Interface components on simple function calls such as a popup window, a grid table with various in built functions to help developers build feature-rich, interactive web interfaces faster.

2. Raphael.js

This is a lightweight and fast performing JavaScript framework which allows the developers to draw visual and vector graphics in the user interface, for example, circles, squares, or any custom shapes.

These shapes or visualizations are provided with DOM access and proper API to enable them to have behavior and functionalities like regular HTML components. This means interactive shapes and graphics can be designed which respond to user input and also have their attributes modifiable in runtime.

## 6.2 . Prototypes

Following are the wireframe prototypes decided by the development team in collaboration with the client and keeping rich design considerations in mind. The client laid down their required design aspects which helped the prototype to be prepared. Broadly speaking, there are two concrete divisions of the frontend:

### 6.2.1 Recon Grid

The recon Grid is the tabular representation of the reconciliation report with respect to an attribute. E.g. Fund ID, Bloomberg ID, Immobiliennr. etc.

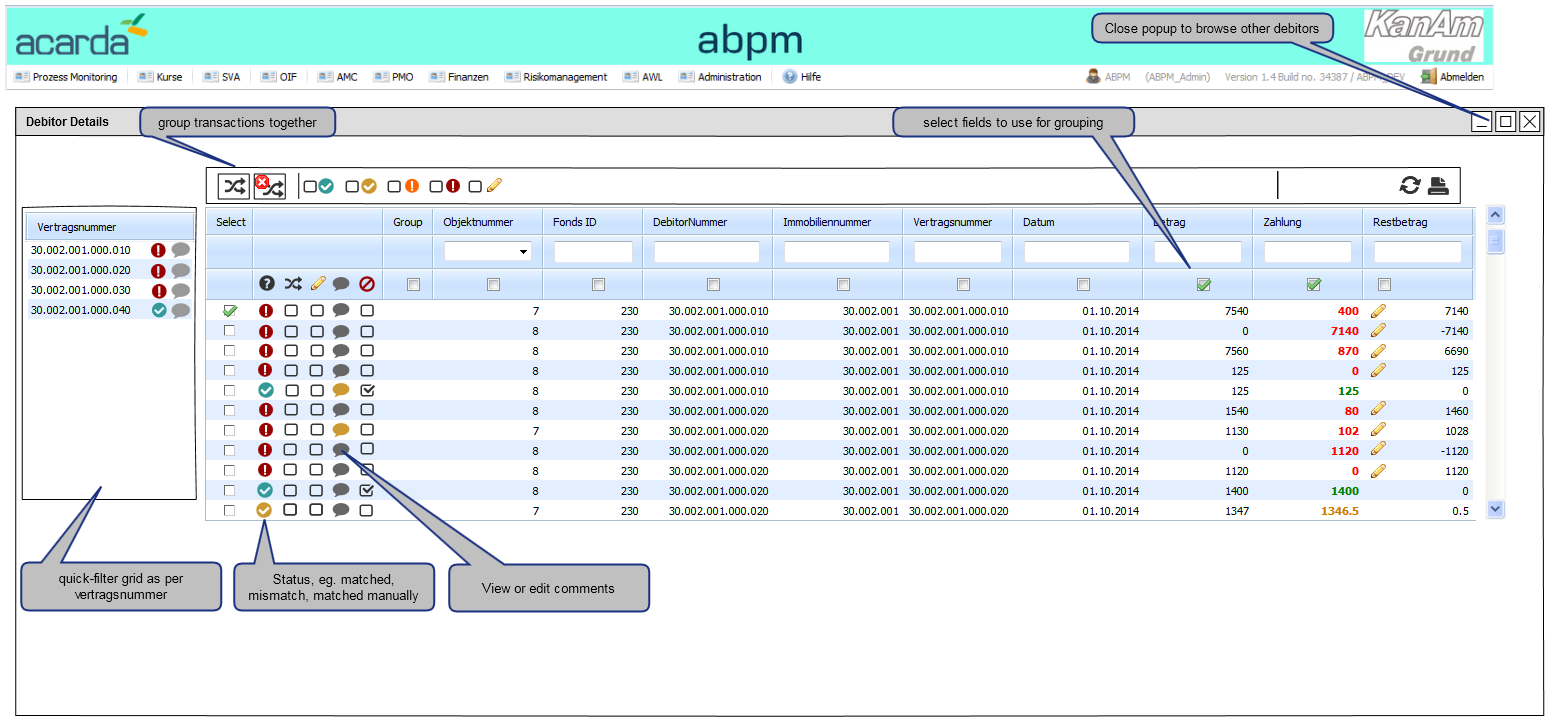


Figure 3 Prototype of ReconGrid

The following information are to be represented in the grid:

* 1. Primary attribute of a record. (Fund id, etc)
  2. Secondary attributes for description or meta information (date, account number, customer name, etc.)
  3. Comparable value pairs (Source-Target, Expected-Actual)
  4. Difference between the two values in absolute and/or percentage
  5. Acceptable tolerance value of the reconciliation

The grid should be able to have the following behavior:

1. Ability to filter the values as per the desired attribute. Eg. by customer name = “John Doe”, by city = Frankfurt.
2. The provision to insert comments for manual editing or bookings.

The grid should also make the following indications:

1. Whether a particular record is matched, or tolerated or unmatched with appropriate pictorial indicators.
2. Appropriate information as to why a particular record is unmatched.

### 6.6.2 Recon Graph

The Recon Graph is a graphical representation of the matches and mismatches in a graphical format, preferably bar chart. Since, the graph is intended to be kept interactive. It should handle user’s mouse clicks and redirect to the graphical view of the selected record to show the detailed information about an attribute (Fund, Mietvertrag, etc).

It should be noted that each row of the Recon Grid corresponds to each “Bar” of the Recon Graph

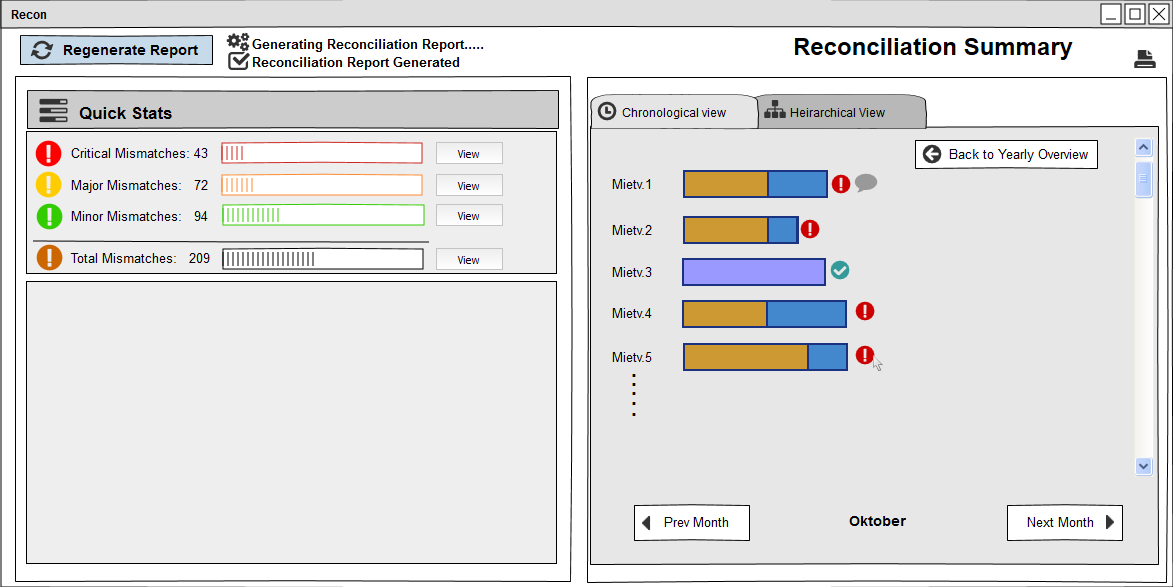


Figure 4 Prototype of ReconGraph

# Development

## 7.1 Selection of development environments

Development of the various stages of the product was achieved using appropriate Integrated Development Environments (IDEs) for each application.

1. Frond End development: WebStorm IDE  
   WebStorm is a lightweight yet powerful IDE, perfectly equipped for complex client-side development.

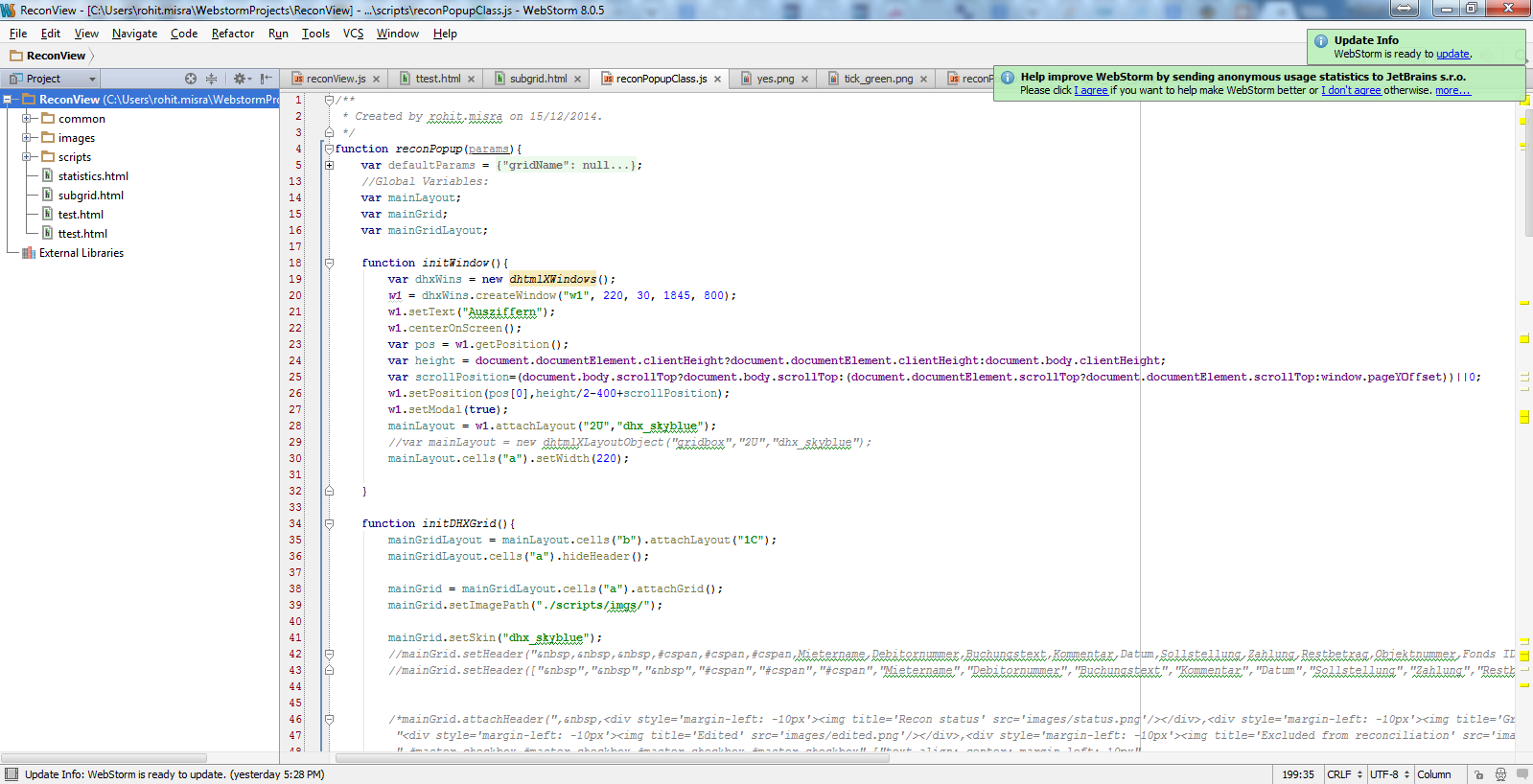


Figure 5 WebStorm IDE

1. Backend Development: Spring Tool Suite (Based on Ecplise IDE)

Spring tool suite is a customized all-in-one Eclipse based distribution that makes application development easy. The tool suites provide ready-to-use combinations of language support, framework support, and runtime support, and combine them with the existing Java, Web and Java EE tooling from Eclipse. Database

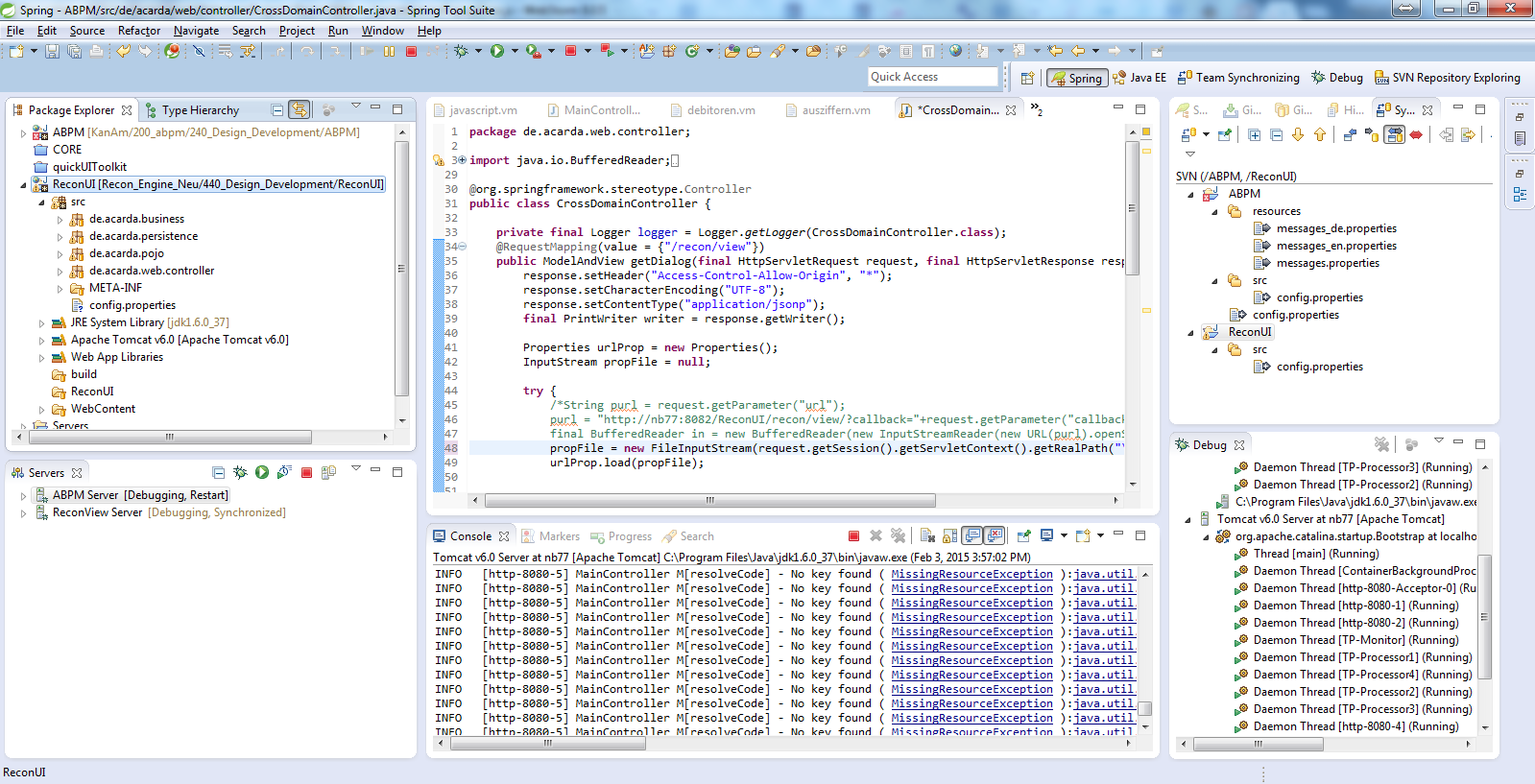


Figure 6 Eclipse IDE

1. (Oracle) Development: TOAD for Oracle IDE

Toad for Oracle is a powerful productivity solution for Oracle database development and administration. These Oracle DB tools combine extensive automation with intuitive workflows to provide the developer with deep functionality.

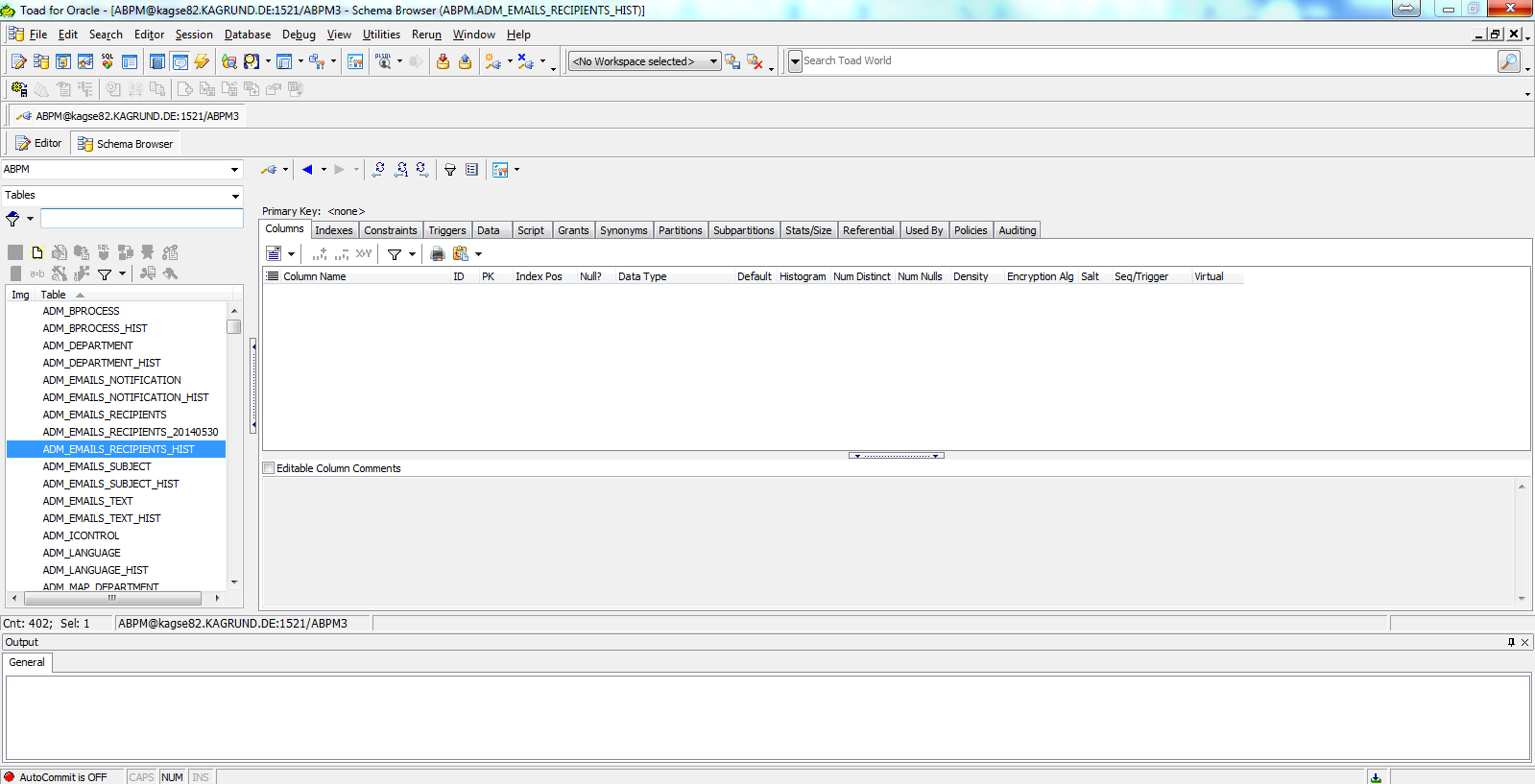


Figure 7 TOAD For Oracle IDE

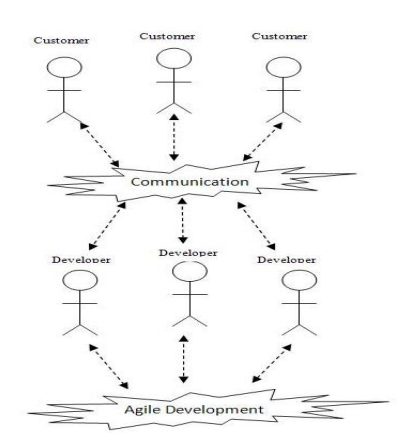
## 7.2 Selection of a development methodology

Regarding the development methodology involved, agile methodology of development was preferred. In contrast to the conventional and rigid development methods, agile development methodology is an iterative development process which allows changes throughout the development cycle and puts more focus on the close co-ordination between the developer and the customer.

In the agile manifesto, as shown below [[[11]](#footnote-11)], the elements on the left side are given more preference over the elements on the right side.

|  |  |
| --- | --- |
| **Left Side** | **Right Side** |
| Individuals and interactions | Processes and tools |
| Working Software | Comprehensive documentation |
| Customer collaboration | Contract negotiation |
| Responding to change | Follow a plan |

According to [[[12]](#footnote-12)], Agile gives preferences to the individuals and the interactions among them over processes and tools. Agile methodologies are formed on a concept that the individuals working in the organization are the most important part of the project. There should be proper communication between the team members. Because, if the communication among the team members will be regular then they will be able to overcome some of the important problems and there will be more chances for individuals to learn from the experiences of their senior members. Just because of the close coordination among them they can make more efficient systems and can share their issues with one another. They believe in a piece of working software instead of a comprehensive documentation. The working system will be more beneficial for customer as compare to that bunch of documentation in order to provide development team with feedback. They think that customer collaboration is more important than contract negotiation because close coordination of customer is also a quality assurance and defect detection activity. In Agile, customer actively participates in the whole development process and guides the team about the system’s requirements. Agile methodologies prefer a quick response to change over following some predefined plan. Because today’s market is dynamic, it is not a stagnant market, so processes and projects should be flexible to accommodate a change.



## Principles of Agile methodology

1. Satisfy the customer through the early and quick delivery.
2. Welcome change in requirements even in the late in the project.
3. Keep delivery cycle short (weeks).
4. Business and development people should work together.
5. Build project around some motivated people.
6. Place emphasis on face to face communication.
7. Working software is primary measure of progress.
8. Promote substantial development pace.
9. Continuous attention to the good design and technical excellence.6
10. Simplicity is essential.
11. Best result come from self-organizing team.
12. Teams discuss regularly that where and how to improve.

Some key practices of agile methods are: scheduling according to prioritization, delivery of software in increments, regular feedback from expert customers, special stress laid upon face-to-face communication, pair programming, test-driven development, automated regression testing, regular integration, self-organizing teams and periodic tuning of the methods. A working piece of software is the primary measure of success [[[13]](#footnote-13)].

# Implementation

## 8.1 Workflow

Target application Database

Recon UI   
Database

Target Application Backend

Recon UI Backend

Target Application Frontend

Recon UI Frontend

|  |  |
| --- | --- |
|  | JDBC Communication with the database to retrieve data |
|  | HTTP data transfer |

## 8.2 Frond End development:

As established in the architecture, two JavaScript functions have to be developed to render a grid and a graph/chart to represent the Reconciliation data. Loosely speaking, both components are different representations of the same data.

In the target application, there would be, in the user interface, different ways to invoke and display these reports in the components. For the sake of simplicity during the implementation phase, a button was used to invoke the report.

On clicking on this button, a popup window was developed to be displayed and within it either the grid or the graphical representation of the data was to be shown.

In technical terms, on the click of the button the following code was executed:

|  |
| --- |
| reconPopup({  gridName: null,  gridType: null,  filtering: null,  showHeader: true,  footer: false,  sortBy: null  }); |

The reconPopup function is essentially a JavaScript class which is instantiated with passing the parameters.

Speaking about the function in detail, the following is the outline/skeleton of the variables and functions:



Figure 8 The outline of the ReconPopup JavaScript Class

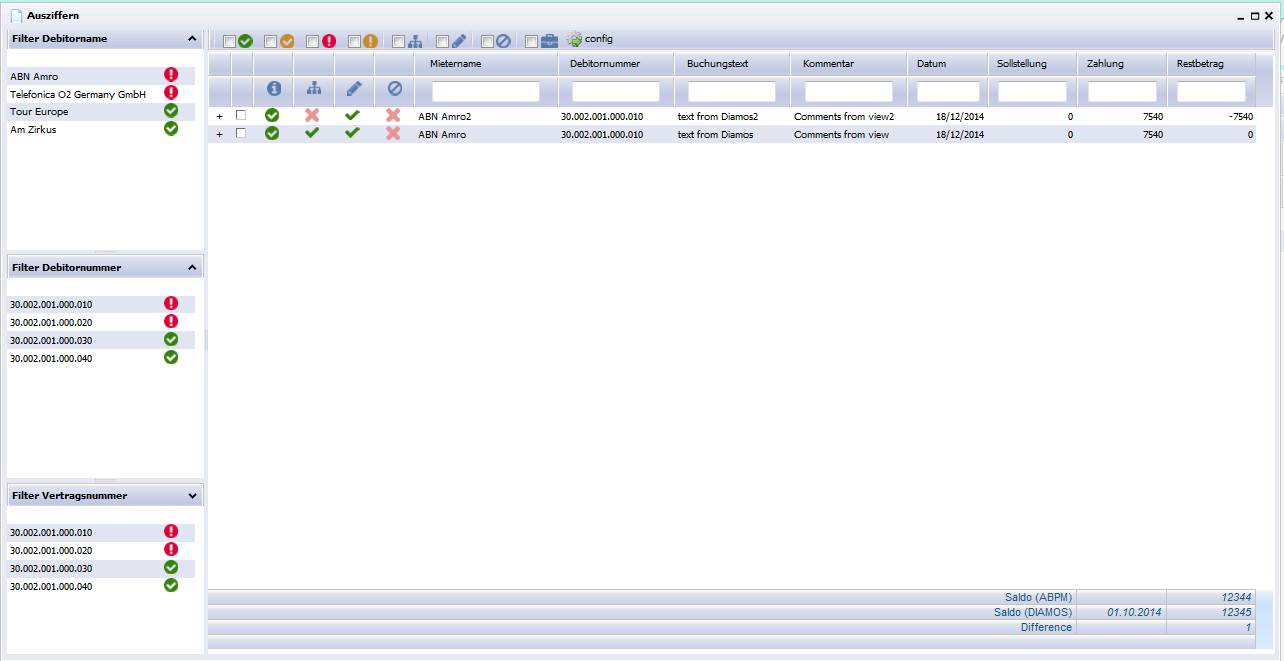


Figure 9 ReconPopup Testing snapshot

The defaultParams are the parameters which are to be passed to the class, much like constructor arguments, which decide the appearance and behavior of the component.

The popup window has three major components:

* 1. Grid

The recon grid gives a tabular representation of the Recon Data. The physical properties of this grid is kept dynamic and can be changed in the config[figure 11] menu button on the toolbar.

This means the user can configure, as per their choice, the order of the columns, the Header texts, the widths of the columns, etc.

The grid also shows pictorially, as icons the status of the reconciliation as matched, unmatched or tolerated, etc.

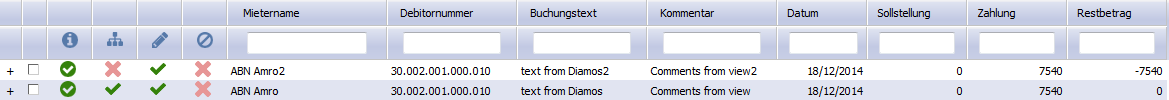


Figure 10 Recon Grid

* 1. Toolbar

The grid toolbar carries essential components to enhance the experience of the user. It has several check boxes to narrow the data as per choice, eg. show only matched results.

The toolbar can also be configured to have buttons for additional operations, such as add a record, manually edit a record, etc.

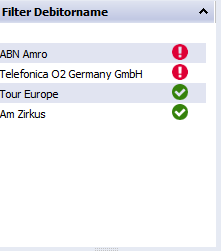


Figure 11 Toolbar over the Recon Grid

* 1. Filters

The filters lie on the left side of the panel. They display the unique values of desired columns to help the user filter the grid as per the selected value.

There is also an indication of the overall status of the grid.



The initWindow() function generates a popup window where the report has to be shown. This takes into consideration essential parameters, such as height, width and position of the window as predefined default values if no user defined values are supplied in the parameter array. This function also creates the layout where the grid, the toolbar and the filters will be rendered.

The initDHXGrid() function creates and initializes a DHTMLX Grid, which is a component to display tabular data. This requires design parameters to construct the grid, such as, the column widths, the column header names, the column filters, the column types, etc. It also after the construction needs the data to be displayed in the grid.

|  |
| --- |
| "data": {  "rows": ["+, 0, green, no, yes, no, ABN Amro2, 30.002.001.000.010, ....",  "+, 0, green, yes, yes, no, ABN Amro, 30.002.001.000.010, ...."],  "gridMeta": {  "colId": "Spacer, checkbox, img1, img2, Debitornummer, ....",  "colHeaders": "#blank, #blank, Mietername, Debitornummer,....",  "colWidth": "25, 25, 45, 45, 45, 45, 160, 130, 130, 100, 130...",  "colAlign": "center, center, center, center, left, left, left, ....",  "colTypes": "ro, ch, acardaFlag, string, string,....",  "colVisible": "1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0"  }  } |

Fig. Sample Grid Data received from the server.

## 8.3 Backend Development: Spring Tool Suite

The backend was developed entirely in Java as an Enterprise Web Application. The significant sections of the project is:

1. Java Source files

These files contain the source code to achieve the programming requirements. The source files are compiled by the Java Compiler during runtime and class files are generated. They are stored in different packages with respect to their usage.

The most important java files written are:

* + 1. main controller (ReconUI backend)
    2. CrossDomainController (target application backend)

The CrossDomainController is a function integrated into the target application which receives the AJAX request call from the UI and transfers the relays the request to the ReconUI. This also checks if the user is authenticated or not. The reason to use such a relaying and not directly calling the controller in the ReconUI is because different target applications have different Authentication mechanisms and different application frameworks. Hence it is not easy for the ReconUI part residing in the Target application UI to directly communicate with the ReconUI. We can force it to directly communicate with the ReconUI, but much of the authentication logic has to be exposed on the JavaScript code on the frontend and also lead to CrossDomain Scripting on the Frontend, which is hence an unsafe solution.

The concern of “How will the CrossDomainController find the ReconUI backend’s main controller” will be dealt in the configuration files part.

The MainController in the ReconUI Backend, on receiving the request from the CrossDomainController, after certain pre-checks, invokes a PLSQL procedure, which returns the data in the form of row data and column data in a JSON hierarchical format. This JSON Data is then passed on back to the Crossdomain controller and subsequently to the AJAX calling function in the ReconUI Frontend.

1. External JAR files included

To enhance the features of Java and add additional functionalities, some JAR Files are added, for example:

1. JSON for parsing and rendering data into JSON format

2. JDBC for communicating with the database.

1. Configuration files

There are many configuration files in the application.

1. Server and application configuration.

These files explain:

1. The database configuration (location, port, username, password, schema), so that the application use this information to connect to the database.
2. Properties file to store the address of the ReconUI backend. This is picked up by the CrossDomain Controller to communicate with the ReconUI backend.
3. The servlet mappings. These point to certain functions in the java code and can be reached by URLs. For example, the cross domain controller can be reached by <http://localhost:8080/ABPM/ReconView>. Where, localhost:8080 is the address and port of the deployed application, ABPM is the application name as it is deployed on the application server, and ReconView is the name of the mapping to that function which is invoked when such a URL is requested.

## 8.4 Database (Oracle) Development:

The database stores various tables for the configuration of the ReconUI and also the appropriate data to be displayed. Since the data format has to be kept generic, the data is not stored in the database as it is displayed in the ReconUI Grid.

Rather, data is spread across a pool of raw information, where they can be related with a ReconID, which is an identifier to link report data for a single reconciliation together.

The reconciliation report is requested based on a ReconID. This ReconID is used to identify what objects are required to describe each record in the Recon Data. For example in Fig., ID is the reconID of a particular Recon.



Figure 12 RE\_EN\_RECON table

Using this ReconID, the RE\_REC\_DATA Table is joined to get the individual values of each cell of the data grid in the ReconUI. These cells are joined together by a common Row ID. As seen in the figure. , the data has a common ROW\_ID, hence it will appear in a single row in the resulting data in the UI.

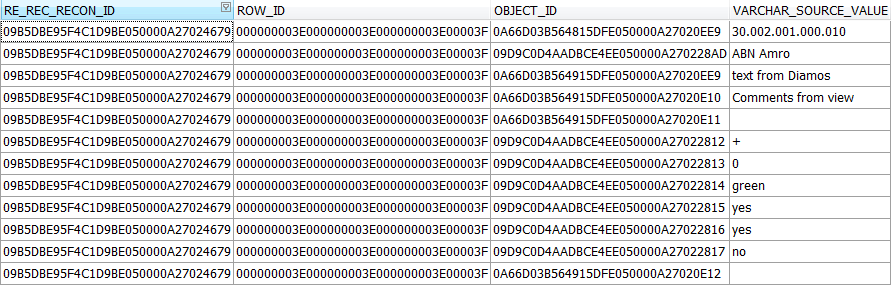
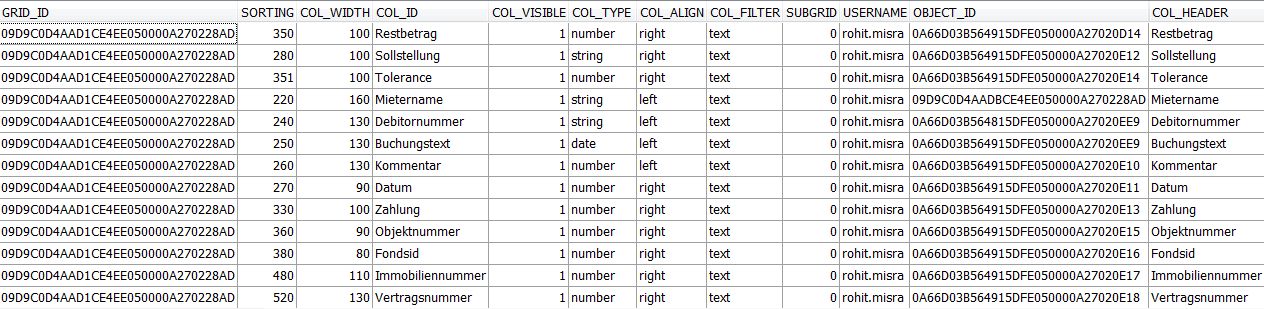


Figure 13 RE\_REC\_DATA Table

The OBJECT\_ID against each data loosely denotes the column to which the data belongs. As seen in the Fig. , all the OBJECT\_IDs give out the column information as to what should be the width of the column, is it visible or not, etc. Even, the configuration can be different depending upon the user who is using the UI.



All these data are grouped together into a single JSON Object Array passed on to the ReconUI Backend. (Refer to Fig. on Pg. 34)

# Testing

Software Testing Process in Agile

With the increase in the criticality of the software systems, the need for quality products has increased. Customers are always looking for quality products. Companies are investing a lot of money to achieve quality in software products. Software testing is a quality assurance activity. It is an important part of any project. Agile software development focuses on individuals and interaction, strong collaboration with customers, and finally with short and frequent deliveries of valuable working software. If we look at these activities, they are useful from testing and quality assurance point of view, but if we compare Agile with other conventional methods, then we will come to know that from testing perspective, Agile methods have lacked in different important aspects of software testing process [[[14]](#footnote-14)].

If we look into the quality assurance practices which are being followed in four Agile methods, then we would come to know that these methods have greater emphasis on building constructive quality practices. The software testing process has a destructive attitude whereas fewer Agile method practices attribute this behavior. These methods are based on iterative and incremental development (IID) that uses short, often time-boxed development cycles. Customer satisfaction is the main task of Agile methods. There is less emphasis on tools, processes, documentation, and following a specific plan, which are traditionally most important in achieving quality assurance and testing practices. [[[15]](#footnote-15)]

Quality assurance is one of the most important and crucial aspect in today‟s software system, due to the criticality of the systems. Quality assurance consists of all activities and practices which are used to ensure software product quality as a part of the development process. Software Testing is a quality assurance activity, it is a process to detect the differences between developed product and required conditions, and to evaluate that either the application has all of the desired features or not [11].

## 8.1 Automated Testing

Automated testing can be defined as a testing in which no human intervention is involved. Test automation depends upon the importance of scenario, if the scenario is not so important then manual is suitable. And also another reason to automate the test is: when there are hundreds of lines of code to test or there is repetition then automated testing is more suitable to save the time.

There are some pros and cons of automated testing. Some of pros are as:

* Automation is best to run repeatedly tests.
* It gives the ability to main stream scenario and run automation against code that frequently changes to catch regressions in a timely manner.
* Automated tests can be run at the same time on different machines, whereas the manual tests would have to be run sequentially.
* Cost is low when long term testing.

Following can be the cons of automated testing:

* The cost of automation is more, especially when you writing the tests or configuring the automate framework.
* The visual reference cannot be automated e.g. if the font color or size can’t be defined via code then it is manual test.
* If the tool has limitations then those tests are manual.
* The big con is that it does not find a new bug.
* The cost of test automation is high when its short term testing.

Other than these there are many factors involved in testing. It mostly depends upon the project type and company’s management, who decide that when to automate the test. Test automation depends upon scenarios, in some cases manual testing is not possible, for example when there are a lot of repetition, there is thousand lines of codes or when the testing is long term then only automated testing is suitable to save the time and cost.

## 8.2 Manual Testing

Manual testing is vice versa of automated testing, manual testing is a testing in which human interventions are involved. In this testing, test engineers test the code their self. For manual testing a good knowledge of software tester is required and a software tester possess a certain set of qualities i.e. patient, observant, speculative, creative, innovative, open-minded, resourceful, un-opinionated, and skillful. Having all these qualities a software tester can find more bugs, but complete testing is not possible. Manual testing is suitable if a program is not so important. Some of pros and cons of manual testing are:

* If a test case runs only twice then it should be manual to save the cost.
* Ad-hoc or random testing is possible for tester.
* More bugs can be found via ad-hoc technique as compare to automation.
* There are no limitations in manual testing.
* You can find new bugs using manual testing.
* Manual testing is more time consuming.
* After every new build code tester must rerun all required tests again, this will be a huge at the end.

Since, the scope of the product was limited to a single screen, with minimum amount of components, manual testing was a simpler and faster method to test the software.

Various test cases were formed in tabular manner while development and were passed on to the testing team which consisted of peers of the same hierarchy and also business consultants. The testing process yielded multiple bugs in the development process and helped to fine tune the software.

# User Acceptance

The user acceptance phase is the most critical step and process of this development process. The high level design document presented by the client served as a reference for the acceptability for the software being developed.

Following the in-house acceptance testing, the product was deployed in a UAT(User acceptance Testing) environment where the end users checked for the various requirements feasibility.

The acceptance test was run against supplied market data and the results obtained were compared to the expected results.

# Deployment

Since maintaining an independent state of the application was essential, the application will be deployed as a separate module on the application/web server. The parent application (AMICON/ABPM) will place calls to the application and retrieve data without the comprehension by the end user.

Authorization to access the ReconUI will be dealt as per authentication situation in the calling application. That is, being authorized to AMICON/ABPM is sufficient.

A number of application servers were compared for this purpose. Amongst Tomcat, JBOSS, Glassfish and IBM Websphere, Tomcat was selected since it is very lightweight as compared to the others in question and still provides sufficient scalability for small applications like ReconUI.

# Difficulties

The following were the difficulties faces while analysis of the solution and also the software development process.

1. Incompatibility of various JavaScript Frameworks with Internet Explorer 8.

AngularJS , D3.JS and Ext.JS were found to be incompatible with the old versions of internet explorer which is still supported by some of our clients.

1. Frequent memory leaks were a development glitch which was solved using a memory profiler and detecting the memory loopholes.
2. Inefficient SQL queries caused major performance degradation at the database level. This caused the UI to timeout while waiting for the response from the server.
3. Multiple unmanaged JavaScript variables and function slowed down the web browser and ultimately crashed it many times.
4. In the backend level, issues arising from threading and synchronization resulted in substantial side effects. Proper code management solved this issue to a greater extent.

# Future Development

A few aspects of t7his solution has been kept in mind and left out for future development.

* 1. Usage of a NO-SQL database, like MongoDB, since the data selected is in JSON Format. Highly scalable and RAID servers might increase the performance on MongoDB
  2. KnockoutJS is thought of to replace the traditional JavaScript data model at the UI end. this would make the data more handle able and performance better.
  3. Tomcat and Java are being discussed to be entirely replaced by Node.JS servers with better performance and threading enabled programming. This would make the system more lightweight

# Conclusion

In the process of this thesis work, I realized that it is important to make technology decisions at the right time and for the right reasons. Good business decisions provide companies with appropriate supporting tools so they can produce good products. When it comes to software development, dealing with tough design issues head-on is one requirement for today’s visionary software developer. When combined with other software engineering considerations, a healthy research and analysis can support the development of cost-effective software systems that, in turn, provide valuable, reliable business support.

# References

en.wikipedia.org/wiki/Usability\_engineering November 16, 2014

http://www.interactivedata.de/uploads/File/Claryfying%20Financial%20Information%20with%20visualization%20Tools.pdf, Nov.16, 2014

<http://commit.wim.uni-mannheim.de/uploads/media/commitWorkshop_Stuckenschmidt.pdf>

http://docs.jboss.org/hibernate/orm/4.2/manual/en-US/html/ch05.html#mapping-declaration-class December 2, 2014

http://www.wrox.com/WileyCDA/Section/Why-Use-the-Spring-Framework-.id-130098.html December 6, 2014

http://www-01.ibm.com/support/docview.wss?uid=swg21639597 January, 2, 2015

An Empirical Study of Client-Side JavaScript Bugs, Ocariza, Bajaj, Pattabiraman, Mesbah, 2013 ACM, Pg 1,

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction\_to\_Object-Oriented\_JavaScript Jan 4 2015

http://www.webmonkey.com/2010/02/make\_oop\_classes\_in\_javascript/ Jan 4 2015

http://phrogz.net/JS/classes/OOPinJS.html Jan 4 2015

Abrahamsson, P., “Agile software development: Introduction, current status and future”, VTT Publications, November 2005.

Jim highsmith, “Agile Project Management: Creating Innovative Products”, Publisher: AddisonWesley Professional, April, 2004, ISBN-10: 0321219775.

Tichy, W. F., “Agile Development: Evaluation and Experience”, In Proceedings of the 26th international Conference on Software Engineering, May 2004, IEEE Computer Society, ISSN: 0270- 5257

Juha Itkonen, Kristian Rautiainen, and Casper Lassenius, “Towards Understanding Quality Assurance in Agile Software Development”, International Conference on Agility Management (ICAM 2005), Helsinki, July 2005.

Fowler, M., J. Highsmith, “The Agile Manifesto” Software Development Magazine. August

2001, URL: http://www.sdmagazine.com/documents/s=844/sdm0108a/0108a.htm (last checked: Jan 01, 2015).

1. en.wikipedia.org/wiki/Usability\_engineering November 16, 2014 [↑](#footnote-ref-1)
2. http://www.interactivedata.de/uploads/File/Claryfying%20Financial%20Information%20with%20visualization%20Tools.pdf, Nov.16, 2014 [↑](#footnote-ref-2)
3. http://commit.wim.uni-mannheim.de/uploads/media/commitWorkshop\_Stuckenschmidt.pdf [↑](#footnote-ref-3)
4. http://docs.jboss.org/hibernate/orm/4.2/manual/en-US/html/ch05.html#mapping-declaration-class December 2, 2014 [↑](#footnote-ref-4)
5. http://www.wrox.com/WileyCDA/Section/Why-Use-the-Spring-Framework-.id-130098.html December 6, 2014 [↑](#footnote-ref-5)
6. http://www-01.ibm.com/support/docview.wss?uid=swg21639597 January, 2, 2015 [↑](#footnote-ref-6)
7. An Empirical Study of Client-Side JavaScript Bugs, Ocariza, Bajaj, Pattabiraman, Mesbah, 2013 ACM, Pg 1, [↑](#footnote-ref-7)
8. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction\_to\_Object-Oriented\_JavaScript Jan 4 2015 [↑](#footnote-ref-8)
9. http://www.webmonkey.com/2010/02/make\_oop\_classes\_in\_javascript/ Jan 4 2015 [↑](#footnote-ref-9)
10. http://phrogz.net/JS/classes/OOPinJS.html Jan 4 2015 [↑](#footnote-ref-10)
11. Abrahamsson, P., “Agile software development: Introduction, current status and future”, VTT Publications, November 2005. [↑](#footnote-ref-11)
12. Jim highsmith, “Agile Project Management: Creating Innovative Products”, Publisher: AddisonWesley Professional, April, 2004, ISBN-10: 0321219775. [↑](#footnote-ref-12)
13. Tichy, W. F., “Agile Development: Evaluation and Experience”, In Proceedings of the 26th international Conference on Software Engineering, May 2004, IEEE Computer Society, ISSN: 0270- 5257 [↑](#footnote-ref-13)
14. Juha Itkonen, Kristian Rautiainen, and Casper Lassenius, “Towards Understanding Quality Assurance in Agile Software Development”, International Conference on Agility Management (ICAM 2005), Helsinki, July 2005. [↑](#footnote-ref-14)
15. Fowler, M., J. Highsmith, “The Agile Manifesto” Software Development Magazine. August

    2001, URL: http://www.sdmagazine.com/documents/s=844/sdm0108a/0108a.htm (last checked: Jan

    01, 2015). [↑](#footnote-ref-15)