PAETEC Product Catalog Editor
Team Spy Penguins
Mark Chadbourne Robert Marinaro Joseph Plourde
Phil Schmitte Tom Sheppard

Project Sponsors
Erika Mesh and Neil Hauser

Faculty Mentor
Nathan Sarr

Table of Contents:

Project Overview .................................................................................................................. 2
Basic Requirements ............................................................................................................... 2
Constraints ............................................................................................................................. 4
Development Process .......................................................................................................... 5
Project Schedule: Planned and Actual .................................................................................. 7
Key activities and milestones............................................................................................... 7
System Design ...................................................................................................................... 8
Process and Product Metrics ............................................................................................... 10
Use Cases Satisfied: ............................................................................................................ 11
Estimation Accuracy: ........................................................................................................... 12
Product State at Time of Delivery ...................................................................................... 13
Project Reflection ................................................................................................................ 14
References .......................................................................................................................... 16

Table of Figures:

Figure 1 – This entity relationship diagram is a graphic representation of the requirements................................................................. 4
Figure 2 – Development Cycle flow chart ........................................................................... 5
Figure 3 – Team Roles .......................................................................................................... 6
Figure 4 – Architectural Layered View ................................................................................ 8
Figure 5 – Effort tracking results ........................................................................................ 10
Figure 6 – Use Cases Satisfied ............................................................................................ 11
Figure 7 – Estimation accuracy by task .............................................................................. 12
Figure 8 – Estimates versus Actual times .......................................................................... 13
Project Overview
PAETEC currently uses spreadsheets and manual processes to specify their products and product catalogs. This system is prone to human error and is difficult to maintain, since all work is done through several ad-hoc processes. PAETEC hopes to create a new unified definition and system to define products. The deliverables of this project will act as the first steps in reorganizing the way product catalogs and defined.

The proposed system will be a web-based tool, used to view and edit PAETEC's product catalog using a new master definition. This master definition needs to be able to describe any existing and new products PAETEC may create. It will be designed to enable non-IT users to specify and edit products, and define business rules for entities. Product versioning will be managed by allowing comparison and grouping. The system originally was designed to support the exporting product definitions to XML and SQL. Exporting functionality was removed from the requirements early in the project in order to reduce the scope of the project to fit the time frame. Each definition entity will have a unique ID to allow regression to legacy formats. The master definition also needed to be designed in such a way as to allow reverse engineering from legacy definitions to be possible.

Basic Requirements
Originally the sponsors did not have a large list of formal requirements. The team had several discussions with the sponsor to elicit requirements from them. As these meetings continued, the team eventually had enough information to start work on a Requirements Specification Document. It was strongly recommended that the team use Blueprint Requirements Center 2009. The use of this tool allows for ease of incorporation of all requirements into PAETEC’s requirements archive. Blueprint allowed for creation of use cases and the storage of textual business and software (functional and non functional) requirements in one place. Below is a list of basic requirements of the system.

- A new master data definition must be designed to describe all of PAETEC’s products in general. These basic elements include:
  - Products
  - Features
  - Attributes
  - Connection Numbers
  - Business Rules
  - Product Catalogs
- Product Catalogs will be made up of one or more products
- Product Catalogs will have ids representing an id and version number
- Products can be a composite of other products
- Product can contain multiple features
• Products can contain any number of attribute values
• Products can contain a list of connections numbers
• Products can contain business rules
• Products will have ids representing the catalog id and product id
• A feature is composite of attributes, business rules and other features. As a rule, a feature cannot contain products or connection numbers
• A feature can be made up of other features
• A feature can be made up of multiple attributes
• A feature can contain multiple business rules
• An attribute is a value associated with another element
• An attribute can represent a value for a product catalog
• An attribute can represent a value for a product
• An attribute can represent a value for a feature
• An attribute can be defined to represent the pricing of a product
• An attribute will contain a field for waivablility
• Attributes must reflect all possible values. This can be done by:
  • Lower and upper limits for a value
  • An initial standard value
  • A collection of valid values
• Attributes can be numbers, strings, or a list of values
• A conditional can be added to an attribute using nested boolean logic based on the values of dependent attributes to determine it's value
• A list of connection numbers or circuit ID and number type
• Business Rules will place a business constraint upon another element(s)
• Product Catalogs can be versioned. Versions of a product catalogs can be created in the system that will be linked to other versions
• Versions of a product catalog will be able to be compared for similarities and differences
• The user interface must be accessible to both IT and non-IT users
• The system must be able to be used by more than one user at once
• The system will lock a Product Catalog to prevent two users from editing the same entity at the same time. The system will remove a lock on an entity when the user is finished editing it
The System will allow for deleting of entities. When an entity is deleted so will the children.

The Interface includes a way to captures product specific business rules (i.e. making some values conditional on others).

The system will also manages existing product definitions by allowing them to be versioned and grouped into catalogs and enforced by the data definition.

Figure 1 – This entity relationship diagram is a graphic representation of the requirements.

Constraints

There were several constraints given to the team by the sponsors. The system had to be implemented in Java. Subversion also had to be used whenever version control is required. This applied to team artifacts, as well as if used as part of the project. After weighing the pros and cons of using a web application or a desktop application, the sponsors decided that the team should create a web application. The system had to be compatible with Internet Explorer 6 and had to run on Tomcat. If the team decided to make the application use a database it had to be done in Oracle. The team consisted of five members and two project sponsors, as a result scheduling and due dates needed to take into account multiple people’s commitments. The team also had a time constraint of six months to complete the project. As a result the team spoke with the sponsors about cutting exporting from the final system. It was also required that all major deliverables need to be reviewed with the project sponsors and a set of PAETEC stakeholders.
Development Process
The team had discussed any mandated requirements with the sponsor before beginning planning and discussion of the development process. No direct process was required; however the sponsor did want a heavy focus on documentation, as well three major pieces of functionality. Based on a suggestion to use a formal and non-agile methodology the team decided to use a modified waterfall, with staged releases. A diagram of the chosen process is displayed below.

![Development Cycle flow chart](image)

This approach worked well with the project because the functionality could be easily partitioned into three sections. It was also valuable because the release could be completed in order of priority and the most important functionality would be in place in the event that the third release could not be realized. Release 1 would originally result in the backend system, allowing for the creation and editing of entities. The user interface was to be developed in Release 2, since the sponsors had ranked this as their second primary goal. The development of an export feature was to be developed in Release 3, but as requirements gathering continued this functionality was dropped. The sponsors did not have any major issues with the process, and it was approved. During architecture and system design phase the team changed how implementation would occur over the releases. Release 1 resulted in the definition of the master product definition, the ability
to view and edit product catalogs and products, add new products to product catalogs, and the ability to view basic product information. Stage 2 would finish user interface implementation, allow for the creation/editing/viewing of all entities, and allow for versioning and a comparison of those versions. The team decided to reintroduce the third release as a time set aside for bug fixes and testing, as well as clean up the user interface to ensure the system would meet the sponsors’ standards.

Although the process did not address communications, the team and sponsors agreed to weekly meetings from 4-6 on Tuesdays. This would allow for any feedback, discussion, and decisions that needed to be communicated. The team and sponsors would also use email to discuss more urgent matters, and to present deliverables. The roles of the team were addressed during the creation of the project plan. The roles identified, the person responsible, and a description can be found on the following table.

<table>
<thead>
<tr>
<th>Role</th>
<th>Responsibility</th>
<th>Staff Member</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Manager</td>
<td>Primary contact with sponsors and coordinating with team members. Also responsible for quality assurance.</td>
<td>Phil Schmitte</td>
</tr>
<tr>
<td>Requirements Lead</td>
<td>Oversee requirements solicitation and verification.</td>
<td>Rob Marinaro</td>
</tr>
<tr>
<td>Design Lead</td>
<td>Oversee system design and ensure system complies with specifications.</td>
<td>Joe Plourde</td>
</tr>
<tr>
<td>Implementation Lead</td>
<td>Manages code integration.</td>
<td>Tom Sheppard</td>
</tr>
<tr>
<td>QA Lead</td>
<td>Oversees testing plan and ensure proper test coverage.</td>
<td>Mark Chadbourne</td>
</tr>
<tr>
<td>Risk Management Lead</td>
<td>Tracks risks and develops mitigation strategies.</td>
<td>Phil Schmitte</td>
</tr>
<tr>
<td>Configuration Management Lead</td>
<td>Oversees the versioning and maintenance of the project artifact and code repositories.</td>
<td>Joe Plourde</td>
</tr>
<tr>
<td>Webmaster</td>
<td>Keeps the website up to date</td>
<td>Tom Sheppard</td>
</tr>
<tr>
<td>Note Taker</td>
<td>Take notes. If the note taker is unavailable to take notes, another team member will take the responsibility for that meeting.</td>
<td>Rob Marinaro</td>
</tr>
<tr>
<td>Development Engineer</td>
<td>Works on design and implementation.</td>
<td>All</td>
</tr>
</tbody>
</table>

**Figure 3 – Team Roles**

These roles were filled in by interest and by fairness to team members. They were fulfilled without any problems, and no person switched roles during the project’s development.
Project Schedule: Planned and Actual

In order to develop the project’s schedule, the team incorporated the Wideband Delphi Process. This method is a consensus-based estimation technique for estimating effort. The process for Wideband Delphi was as follows:

1. All artifacts and documents needed for the project were listed.
2. Each team member wrote down how long they thought it would take to complete each milestone.
3. The team would discuss discrepancies to get each member’s view point.
4. After discussion the team would write down new estimates. This was repeated until there was a general consensus.

Key activities and milestones

- Implementation of each stage
- Testing of each stage
- Business Requirements Specification
- Software Requirements Specification
- Research of new tools
- Software Engineering department documents and deliverables
  - Presentations
  - Post mortem
  - Technical report.

The original estimates were too low, and resulted in a scheduled completion date that was about six weeks earlier than the deadline. After discussing our schedule with our sponsor and faculty coach the team discussed informally how to adjust our estimates. When a consensus was made the team revised the scheduled task. The schedule was created using Microsoft Project.

The actual schedule was very different from the original plan. These discrepancies occurred for several reasons. The team did not have any real experience estimating the length of time it would take to complete tasks. Originally the team did not plan appropriately for revision time, or for time between submitting documents and getting sponsor feedback. It was assumed that the documents would be worked on, completed, and then accepted by the sponsors. The team soon realized this was not the case and adjusted the schedule to incorporate “revisions” time to take sponsor feedback into account and adjust these documents. A good example of this revision time can be seen in the amount of time it took to complete the project plan and requirements documents.

Another issue that occurred with the schedule was a difference in team availability and sponsor availability. There were times were the team would have to wait for feedback, before entering the revision phase, as well as times the team could not get documents to the sponsors in time for them to review them. This was mainly a result of the team getting
most of their work done over weekends, and the sponsors working a standard work-week. Again, this issue occurred in the earlier stages of the project and was taken into account when creating new end dates for tasks and milestones. When originally estimating time, there was a misinterpretation of estimating for group work, due to the fact that the team did not count an hour for each member’s effort. For example, if the entire team spent two hours discussing a section of the design, that needed to be counted as ten hours of work rather than two.

As mentioned earlier the project’s schedule was originally created and maintained in Microsoft Project. As the project continued the team began encounter issues with the program. This would result in projections of excess hours per weeks, moved end dates, or inaccurately displaying how long a milestone took to accomplish. The team researched alternatives for storing the schedule information, and eventually moved the project into an excel spreadsheet. Estimation accuracy was one of the team’s metrics. By tracking this we were able to see where our inconsistencies lie.

**System Design**

![System Design Diagram](image)

*Figure 4 – Architectural Layered View*
The system design that the team created was based upon the requirement that the system should be as decoupled from the user interface as possible. Our design lends well to this idea, as there is only one connection from the UI to the model and that connection provides only data request and data transfer. The UI can only contact the server and ask for information, but the internal model is never touched directly by the user interface. The design of the system follows the Model-View-Presenter pattern, using the application of the pattern that was modeled by Microsoft. There are actually several versions of this pattern, each with slightly different interpretations of how this pattern should be utilized, however the Microsoft version seemed to fit our needs more by isolating the UI from the actual internal model. Model-View-Presenter was retired in 1996 by Martin Fowler and separated into two separate patterns: Passive View and Supervising controller; however, this did not affect our decision to utilize this pattern in our design since it fit our needs well. GWT also allows for developers to use either a Model-View-Presenter or Model-View-Controller pattern when developing web applications, and since the Model-View-Presenter pattern worked with what we were doing, and allowed for complete isolation of the UI and the model it, we chose it.

The system is separated into four subsystems: UI View, UI/System Manager, Entity Management, and XML Persistence. The UI View is completely composed of GWT and GWT-Ext UI components that form to create an idea of pluggable views. When a user requests to view or manipulate a particular entity, the ScreenManager determines what view should be created and plugged into our view socket.

The UI/System Manager is responsible for handling all user requests and figuring out where the request should go on the server side. This provides the isolation of the UI from the internal model by controlling the traffic that each uses. The ServerManager is the major component in this subsystem and provides access to the EntityManagement subsystem through a series of locks. It was important to handling locking through both the ScreenManager and the ServerManager, as user authentication does not exist. To take care of this, the ScreenManager requests to the ServerManager to view or edit an entity, and the ScreenManager then determines if the entity can be viewed or edit based upon a locked property. If the entity is unlocked the request can be completed, otherwise it cannot be viewed. The alternative would have been to use some form of authentication and attached an entity's lock to that user id; however, since there was no requirement and the team was told to just make a single user, we had to come up with our own way to lock.

The Entity Management and the XML Persistence subsystems handle the actual data that is stored within the system. The Entity Management maintains an active and current list of all entities in the system by product catalog. The internal hierarchy is enforced and maintained by the EntityManager, which oversees all queries and changes to the model. When a change is made, it reflects that change in the file systems by telling the XML Persistence layer that changes to the model need to be saved. The XML Persistence maintains the representation of the internal model by utilizing a form of persistence. This subsystem does not rely on a single form of persistence; however for our implementation we wrote directly to the file system and stored data into XML files based upon a schema we developed. The major goal of this subsystem however, was to provide future revisions by decoupling what form of persistence was being used,
therefore something like SVN could be used to store the XML, or a database could be used in place of XML all together. The idea is by keeping the form of persistence flexible and separate from the rest of the system, it allows for future expansion and redesign as needed.

Other than scrapping SVN, we have not had any major changes to the design of our system, and even that change affected the actual technologies used, rather than the design directly. Instead of creating a connector to SVN, we just implemented a connector to the file system and saved to that instead. Again, the goal of our entire system was to be as flexible on the front end and back end as possible. We had talks about different technologies that would slightly modify our design such possibilities would be the use of Icefaces over GWT, or just making a desktop application. In the end, Icefaces seemed to be too difficult and a desktop application seemed impractical for distribution, therefore we decided to make a web application using GWT. Using GWT allowed us to create the UI without having to worry about writing JavaScript, as GWT is similar to using Java Swing, and it allowed us provide ease of access for the users of the system.

**Process and Product Metrics**

The team employed three metrics during the course of the project. The first metric was team effort, which was tracked by a weekly timesheet in a spreadsheet required by the Software Engineering department. These took place over the twenty weeks of the project and there was one sheet for each week. The results of the first term showed that the team estimated longer hours, than actual hours working. On the individual level this was mostly the same. The same trend persisted in the second term. Again the team estimated more work per week then actually performed. Over the course of the entire project the team had a tendency to estimate more work per week then they actually preformed. This could have occurred for several reasons. Several weeks the team did not explicitly separate work before the time sheets were due. As result the team would have to guess what aspects of the product they were working on before fully knowing what they would be doing that week.

<table>
<thead>
<tr>
<th></th>
<th>Effort Tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg</td>
</tr>
<tr>
<td>Mark Chadbourne</td>
<td>12.30</td>
</tr>
<tr>
<td>Robert Marinaro</td>
<td>11.43</td>
</tr>
<tr>
<td>Joseph Plourde</td>
<td>17.50</td>
</tr>
<tr>
<td>Phillip Schmitte</td>
<td>13.23</td>
</tr>
<tr>
<td>Tom Sheppard</td>
<td>14.00</td>
</tr>
<tr>
<td>Team totals</td>
<td>68.45</td>
</tr>
</tbody>
</table>

**Figure 5 – Effort tracking results**

RIT also requires two other process metrics of our own choosing. These metrics were use cases satisfied and estimation accuracy.
Use Cases Satisfied:

Use cases and requirements satisfied provided a method for tracking overall project completion. The number of use cases and requirements were specified in the Requirement Analysis phase. The use cases can be found in the team’s Requirement’s Specification Document¹, which was created using Blueprint Requirements Center 2009. This metric was chosen because it would be a way to show progress of the project. A use case will be considered satisfied when all steps are completed in implementation. Below are the results of use cases satisfied for release 1 and release 2.

<table>
<thead>
<tr>
<th>Totals</th>
<th>Release 1</th>
<th>Release 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satisfied</td>
<td>16</td>
<td>27</td>
</tr>
<tr>
<td>Partially Satisfied</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Satisfied with Errors</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Not Satisfied</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>Dropped</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Total</td>
<td>35</td>
<td>35</td>
</tr>
</tbody>
</table>

Figure 6 – Use Cases Satisfied

The results were what the team expected. At the end of release 1 the team had fulfilled its goals of setting up the file system and creating part of the UI to fulfill the related use cases of Products and Product Catalogs. The only related use cases that were unresolved were that of deleting Products and Product Catalogs. Release 2 continued work on the use cases completed of the use cases unimplemented as well as those that were completed with errors. Overall this metric was useful for implementation, as well as for writing release notes to the sponsor, to notify them of unresolved and unimplemented issues with each release. The results also show that implementation went well for both the first and second release, as the team did not have any missing functionality for each release. This metric however was not useful until implementation began. For the first twelve weeks, before implementation this metric could not have been tracked since no code was written. As mentioned in the Process section, the project was originally designed to have backend functionality written for the first release, and then the user interface would be developed for release two. As the team was finalizing design, the decision to split up the functionality of the system was made. This decision allowed the use cases satisfied metric to be more easily tracked.
Estimation Accuracy:

The second metric chosen by the team was estimation accuracy. The information collected from the team’s first metric, the individual reports, were compiled and compared with the project scheduling estimations. This metric was chosen to help the team track schedule slippage. It was also the team’s intention that this would allow us to improve our time estimation accuracy. The team also planned that the differences between estimates and actual time spent was to be used for determining a more accurate estimation for other tasks that were not implemented. Originally, estimated times for tasks and our complied actual time information was kept in an excel spreadsheet. When the project’s schedule was converted from Microsoft Project to Microsoft Excel this information was also added. The team also added estimated start and end dates, as well as actual start and end dates. The amount of time worked on a task was updated every week, based on the information on our trackers. Below is a graph showing our estimation accuracy, as well a graph showing the actual time vs. time of our estimations.

![Graph showing estimation accuracy](image)

**Figure 7 – Estimation accuracy by task**

Each line represents a task. The Tasks are not labeled because the graph became too cluttered when this information was on the chart.
This chart is Effort estimates vs. Actual times. The light blue is Estimates and the purple is Actual time. Overall the accuracy is very sporadic. Again the titles of the task were left unlabeled in order to make sure the graph did not become too cluttered.

Overall our estimation accuracy was one of the biggest problems the team had early on. As the project increased our estimation accuracy became closer to our actual times, with several exceptions. One can infer from the data, which as a team did not become more accurate as development of the project continued. The data shows that our estimation accuracy remained sporadic throughout the project. One of the biggest problems this metric provided was that during the first quarter there were few documents and milestones that were completed. As a result the metric was distorted due to lack of data.

**Product State at Time of Delivery**

The state of the project is close to completion. The team is finishing its final release, and adding minor improvements where possible. The Export functionality of the system is missing. During requirements gathering and scheduling it became obvious the team was not going to have enough time to implement everything the sponsors had originally listed in their Project Proposal. The requirements of this functionality would have allowed IT users with knowledge of the legacy applications to map from the product definition to formats used by various legacy applications. The export feature would have also supported adding new mappings and external formats without editing the underlying software. The system would have also allowed reverse engineering from existing legacy application product catalogs into the new master product catalog based on existing
mappings. The sponsors ranked this feature the lowest priority, so it was left unimplemented; however the design took this into account when it was created. Originally the team planned on using SVN within the system for management of the team’s file structure. Time constraints and several issues arose during planning. In the planned system each user had to check out a local “sandbox copy” of the data, this would have resulted in a large amount of data to be transferred and stored on system. Due to these factors the sponsors suggested removing the SVN requirement of the system and store the product catalogs in a flat file system hosted on the server. The team discussed the SVN issues and agreed to remove SVN. This resulted in a new design, in which the team had to implement locking from scratch. The team most likely saved time during implementation, as well as prevented performance issues that may have arose by not implementing SVN.

There were not written requirements for locking, but the team knew there had to be some file management to support multiple users at once. This was originally going to be done through SVN and it’s locking system. This would have insured only one user would have access to a file at a time, but complexity and performance issues would have arose. The team removed the SVN requirement from the system, so we had to develop our own system for locking and unlocking files.

Although we had told the sponsors we would look into live search for the product it remained unimplemented. A Live Search is a search in which the results are dynamically displayed and changed. There were too many other problems that needed to be fixed, and this feature was described as “nice to have, but not important.” Another discrepancy that existed between what was discussed and what was delivered was an issue dealing with labeling. Currently an entities’ basic information is contained in a sub-pane. The labels of this sub-pane simply state that entities’ type. The sponsors had expressed interest in having this label contain more information, such as the name of the entity, so that a specific entity could be selected when its information was minimized. Again the team was unable to add this feature due to other issues that needed to be addressed. Another feature that was planned is a confirmation dialog stating that the user is moving away from the page without saving. This was not explicitly promised to the sponsor, and was left out in order to complete other promised features.

**Project Reflection**

Throughout the past two quarters, many situations have arisen that have served as learning experiences for our team, both positive and negative. Developing a project of this type and magnitude, and in this sort of atmosphere (no direct instruction, large and variable scope, few department-enforced deadlines) was an experience that despite several co-ops between us, few on the team had dealt with before. The following is a brief overview of what went well, what went poorly, and what we would have changed if we did it all again.

There are several aspects of our project that we feel went successfully, some so much so that they stand out above the rest. The requirements gathering phase went well. Although
we spent perhaps significantly longer than other teams on this phase of the project, we feel that in the long run, the time invested was well worth it. The complexity of the system we were tasked with creating demanded a substantial level of detail in documenting its feature set. Because of this, we focused on ensuring our vision of the project meshed with that of our project sponsors, achieving this mainly through lengthy requirements gathering and documenting session, utilizing the Blueprint Requirements Center tool. After documenting thoroughly, we sent our artifacts to the sponsors for review and made several iterative revisions until all parties were satisfied with the product. Putting in this level of effort simplified and shortened the design and development phases.

Other aspects of the project that went particularly well included unit testing (in contrast to our integration testing, which will be discussed shortly). Each team member created unit tests for their own code and the subsystems functioned as intended as individual components. Unit-level bugs were caught and fixed utilizing these tests. Bug fixing in general also went fairly well. The team was able to track defects and keep up to date on their status by using a simple tracking spreadsheet that kept everyone aware of the current state of the system and focused on the issues to be resolved.

Some parts of the project that did not go quite as well included, among other things, the poor accuracy of our initial schedule. In the beginning of the project, we grossly underestimated the time it would take to gather requirements, complete design and even define the project scope. Although we consciously added more time in for requirements in order to create a detailed specification, our failure to account for time to process and act upon feedback from sponsors caused significant discrepancies in the early stages of our project between our estimates and the actual amount of time required to complete artifacts. Having never really worked with live stakeholders before, we did not have the foresight to account for revision times as we should have. Partly because of this, and the inevitable schedule slip that accompanied it, we were forced to eliminate some features from the product, including the third release, initially meant to allow for an export of the product’s data definition to other formats early on. We also made some concessions later in development, opting to remove SVN integration for the use of a simple file system for the time being. Integration of our code, while executing smoothly, did produce a fair amount of bugs that needed to be fixed, most of which were of greater complexity than the bugs found in unit testing.

If our team were to go through the senior project experience again, we would certainly make some adjustments. We understand now that our estimation techniques were based on unrealistic scenarios that we had previously dealt with in a more strictly classroom setting. Knowing that these situations do not apply to the real world, we would account for a greater amount of time to complete documents, making allowances for customer feedback and revisions. The integration of our subsystems into a complete product also could have been improved. The process would likely have benefitted from a more gradual integration of components, fixing bugs as we pieced the system together one by one, rather than the way in which we approached it. This was similar to our original plan for implementation. However, when our planned releases changed to a vertical
implementation of the system the team had to develop all aspects of the system at once. This was probably too ambitious for a project of this complexity. During the design phase the team believed GWT would allow the system to send a fully populated panel across to the client. This would have reduced the complexity of communication, but GWT required data to be passed through a serializable object, which a panel was not. This resulted in the team having to redefine our approach during implementation and communication problems surfaced because the data structures were not well defined.

References
1. Requirements Specification Document
2. Team Project Plan
3. Team’s Excel Schedule
4. System Design Document
5. Architecture Design Document