Spring 2006

Design & Develop A Satellite Telemetry Display Application

Kelly J. Stuhlsatz
Regis University

Regis University
School for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

Design & Develop a Satellite Telemetry Display Application

Prepared By Kelly J Stuhlsatz

Regis University Master of Science in Computer Information Technology

February 26, 2006

Project Paper Revision/Change History Tracking

Date	Author/Modifier	Description
1/8/06	KJS	Initial version of front matter and chapters 1,2
1/15/06	KJS	Added Chapter 3, added research list to chapter 2
1/30/06	KJS	Added Chapter 4
2/12/06	KJS	Added Chapter 5
2/18/06	KJS	Removed references to first-person. Added chapter introduction information. Added information to research product section. Added references
2/25/06	KJS	Fixed redlines
2/26/06	KJS	Fixed redlines

Abstract

The purpose of this thesis project is to design and implement an extensible and adaptable GUI for viewing satellite telemetry. The software available today to view satellite telemetry is very crude, hard to understand, and is not very reliable. Software is usually built from scratch for each new satellite system. This project will build a GUI that can be used to display all types of satellite telemetry. The GUI will be built in Java, have an easy to use interface, and be very customizable. The GUI will allow users to run the software on Windows and UNIX based operating systems. It will allow the user to set up the telemetry viewing pages as he or she fits. The goals for this project are to design and build the software application.

Table of Contents

List of Figures

1. Chapter 1: Introduction

Chapter 1 will give an in depth look at the project thesis as well as why this type of GUI development is necessary in the satellite industry today. This chapter will go over the project thesis as well as show the need for doing a project such as this. In addition, a description of the project goals and a brief summary of the project scope will be discussed. This chapter will also describe the goals of the project and what is the project scope. The scope described is not all-inclusive, but does provide a high-level description of what the project was expected to complete. As every project is expected to have, there are constraints on this project, which are listed to show the limitations that were predefined for this project.

1.1. Introduction

At the time this paper was written every satellite system that was built needed to have its own software to view the satellite telemetry values that were beamed down from the spacecraft. Software was usually built from scratch for each new system that came online. This customized approach was largely due to the complexities from satellite to satellite. The problem was that rebuilding the software was very expensive for the customer. However, if the developer had access to a generic display GUI it would allow the customer to only have to budget money for simple customizing of the software for the application to work. In the end, the savings would be substantial for the customer.

Another key problem with developing a reusable GUI was that each satellite operator had very distinct preferences on how he or she wished to see the telemetry values. Therefore, the generic software needed to be flexible enough so that the operator could build pages of telemetry values as he or she saw fit. At the time this project began these customized pages took a software developer many hours to build and compile into the software application. This was a primary reason why software was rebuilt for each new satellite system.

For this project, software was designed and built that allowed users to view telemetry data in a fully customizable GUI. The author of this paper was also the project manager, architect, and software engineer. “Project manager,” “developer” and “the team” will be used as references to the author throughout the paper. The author was required to do all the managing of the schedule for the project. He also did research on the different telemetry display tools available on the market at the time of the project to determine the features that were included in the final software application. The one identified constraint on the system was that a real world data source for the GUI was not provided. For testing the software application, a test data source had to be provided.

This project entailed designing and developing a Java-based program to view real-time satellite telemetry. A complete software lifecycle was used on this project. The phases included planning, analysis, design, implementation and testing. Work in the analysis phase included research into other software applications similar to the one this project focused on. The design phase included using UML design tools. The software that was built has a user interface that allows the user to set up customized

pages of telemetry values. The telemetry input is designed to be streamed in over a socket connection from an outside source.

1.2. Thesis statement

This project will design and develop a Java based program to view real-time satellite telemetry. Research will be conducted to validate that a product can be built that will satisfy customers' needs. The design will use standard UML design framework and industry standardize patterns. The software being built will have a user interface that will allow the user to set up customized pages of telemetry values. The input values for the telemetry will be streamed in over a socket connection from an outside source.

1.3. Need for the project

Every satellite system that is built today needs to have its own software to view satellite telemetry values that are beamed down from the spacecraft. This software is usually built from scratch for each new system. This is largely due to the complexities from satellite to satellite. This rebuilding of software can be very expensive for the customer. If a generic display GUI were built then the customer would only have to provide money for simple customizing of the software for it to work.

Another problem is that each satellite operator has very distinct preferences on how he or she wishes to see the telemetry values. If the software were flexible

enough then the operator could build pages of telemetry values as he or she sees fit. In order to build these customized pages in today's technology, it usually takes the skills of a software developer to create and compile the source code into the software application. This is a big reason why the software is rebuilt for each new satellite system. If an application could be built that would allow the customer to build his or her own display pages instead of hiring software developers to do it, it would be of great value.

1.4. Project Scope

The scope of this project was to design and build a software program for viewing satellite telemetry. The project entailed design documents that needed to be built as artifacts as well as software source code. This project included the following deliverables:

- Design artifacts, specifically use cases, class diagrams, sequence diagrams, and test procedures.
- Research to analyze current tools developed by other vendors.
- Software design documented in UML and making use of software design patterns.
- A software program with an intuitive, user-friendly interface and these requirements:
    - Telemetry data points should be easy to find and group together.

- The operator will be able to select telemetry points and create pages of data to display.
- The data being displayed will have different colors to denote the state of the data. A green color will denote a good state whereas a red color will denote a bad state.
- The GUI will allow the user to create pages or views of data on the screen.
- The GUI will allow multiple pages to be shown at once.
- Pages of data can be saved and reloaded so that the operator does not have to redo the setup created from a previous session.

In order to show the full effect of the software, test data or a test data source was provided to simulate new data arriving from the spacecraft. This data source was not an official artifact of the project, but merely a test tool.

1.5. Project Goals

The project was centered on building software usability and functionality. The primary goal was that the resulting application was easy to use and operate. The software program should not require a great deal of user training to operate. The software needed to have common features or similar capabilities of other software applications that the operator is used to working with. For example, drag and drop features needed to be consistent with how other software programs work

The secondary goal was that the software be flexible and generic when it was designed and built. Standardized design patterns needed to be used so that other developers could easily understand the design and extend or modify it, if necessary. One of the problems attempting to be solved was the reusability of the software across multiple satellite ground systems. The software needed to be flexible enough so that only minor modifications were needed for it to work with another satellite ground system vendor.

This project was also about exploring new concepts and getting feedback from the software and satellite community on how software applications can be developed better. This is only the first stepping-stone to beginning communication on software reuse technology in the satellite application area.

1.6. Constraints on the Project

One of the essential components of the satellite ground system for processing telemetry is the input data source. For this project the input source was a simulated source. The source can vary by each ground system provider. For this project the team developed a generic interface that could be used for the input data source. This generic interface could then be fitted with an adapter of a specific source to input the data. On this project, the team developed two different adapters that could be utilized. The first was a pure socket interface into the system. The second adapter was an object-oriented class that can be used to send the data through.

For testing the software, a random number generator was used to send data through the class interface. This format worked out well, as the data could be easily modified to change the test data that was used.

1.7. Summary of Chapter 1

This project was about going through the software lifecycle and process to design and build a generic application that can be used to view satellite telemetry. There is a current need in the marketplace for a low cost flexible solution to view the thousands of telemetry points that come down from the satellite. The current products on the market today either don't provide the functionality that is needed or it is a custom-made product that is extremely expensive to build. This product uses a generic and reusable interface that it can be easily modified for all types of satellite ground systems in the future.

The goals of this project were to first make the application easy to use and flexible; and second, to come up with new concepts and GUI designs that are currently not being used by similar applications. The project is as much about exploring new concepts as it is about getting a real software application built. After the completion of this project, the end product will be shown to a variety of people in the satellite and software developing arenas. This project is just the first step into a larger expanse of software applications.

2. Chapter 2: Review of Literature / Research

Chapter 2 describes the research into similar telemetry display products and what they can provide. It describes how each of the products found differs from the software that this project is hoping to offer. It also describes why this project will make an impact in the software world today with its new innovative concepts.

2.1. Review of Research Conducted

2.1.1. Review of similar applications on the market today

Most of the research examined was done by running searches on the Internet. Some of this information also comes from the author's previous knowledge in this area. Thirteen viable products were found that were similar to what this project entails. In addition, there are other products available today by software companies that are not advertised on the Internet. These other products are supplied (or provided) by large defense companies that sell ground station software and hardware. Therefore, these products were difficult to gather information on. The larger corporations are not selling a simple COTS solution for telemetry display software; they are in business to sell large amounts of hardware or the complete software ground station system. The software is usually custom made with a price tag that can range in the millions of dollars. Not much research was prepared on these custom

made products as this project is more about a generic interface that can be used for all types of telemetry display systems.

The other systems available in the community can be divided into two categories. The first category can be grouped into satellite tracking software. These applications usually provide a nice graphical display that tries to simulate the current orientation and position of the spacecraft as it relates to the orbit of the earth. The products range in quality from basic freeware type of software to very expensive software that has nice quality pictures of the earth and geographic features. Most of these products were used with a real time view of the satellite.

The second category of applications is used to create satellite ephemeris and orbit prediction. These products help in the planning of satellite orbits and to make corrections if need be. There was usually analysis features that helped to playback orbits and do future orbit planning. This second category of software products did not provide any real-time information. Information was usually supplied to it through the use of data files.

Out of all the products viewed, the first category was closest to what this project was trying to accomplish. It did provide some real-time update information to the user, but it was almost always limited to orientation of the spacecraft. One of the goals of this project was to provide a display that showed health and status of all the telemetry data points on the spacecraft. Most of these products were trying to only work with spacecraft control and not to display specific sensor or hardware information to the operator.

All of these products gave the team ideas as to the layout of the GUI and functionality that could be provided. It was realized that this application could use more user-friendly shortcut buttons to achieve the functions that the operator needs on a daily basis. A lot of the applications assume the operator has had extensive training and can understand its concepts. Many of the other applications viewed had simplistic interfaces that at times seemed redundant and a waste of screen real-estate; but it is now apparent that it was put in to make it as simple as possible and more user friendly. None of the products provided the complete system that this project was looking for. The products could not be used as an add-on or enhancement to the application that this project needed to produce. Below is a list of products that was found during the research. They are grouped into products in which a small license fee was collected vs. freeware/shareware products. The listing also shows the platform the product runs on, as well as the company licensing the product and the license fee.

The following products include a license that must be acquired and a fee paid for each computer it is installed on.

Pay for Products:

- **MacDoppler Pro** – Satellite tracking and ground track analysis software
  - *Platform*: OSX, Mac OS 9.1 *Licensed by*: Dog Park Software *Price*: $100 (MacDoppler Pro X)
- **Nova for Windows** – Real-time satellite tracking software

  - *Platform*: Windows *Licensed by*: Northern Lights Software Assoc. *Price*: $60 (Nova for Windows)

- **SatPC32** – Satellite ground tracking software (German made product)
  - *Platform*: Windows *Licensed by*: DK1TB *Price*: $50 (AMSAT)
- **SCRAP** – Satellite tracking and analysis tool
  - *Platform*: Windows *Licensed by:* Bytheway SDL *Price:*$60 (SCRAP)
- **InstantTrack –** Satellite tracking program and ephemeris generator
  - *Platform*: DOS *Licensed by*: AMSAT *Price*: Donation to AMSAT org (InstantTrack)

The following products usually require no license or the license is included in the installation.

Freeware/shareware/ limited license products:

- **Orbitron –** Satellite tracking and prediction tool set.
  - *Platform*: Windows *Licensed* by: Sebastian Stoff Cardware *License*: none (Orbitron)
- **SatScape –** Satellite tracker and telemetry display.
  - *Platform*: Windows *Licensed by*: Scott Hather *License*: Freeware (SatScape)
- **Predict –** Orbital prediction software
  - Platform: Linux, SunOS and Windows *Licensed by*: John A. Magliacane, KD2BD *License*: GPL (Predict)
- **Portable Predict+ -** Portable version of Predict product

- *Platform**:*** Linux, SunOSand Windows *Licensed by*: John A. Magliacane *License*: GPL (Portable Predict +)

- **PetitTrack –** Satellite tracking software
  - *Platform*: Embedded Linux *Licensed by*: Edson Pereira**,** N1VTN *License*: GPL (PetitTrack)
- **PocketSat+ -** Satellite tracking software for PalmOS and Pocket PCs
  - *License*: PalmOS, WindowsCE 3.0 *Licensed by*: Big Fat Tail Productions *License*: Shareware (PocketSat)
- **Tracksat Web Software –** Satellite tracking software for mobile devices
  - *Platform*: Windows (Tracksat)
- **SIT60** – Provides information in a raw table like format. Used to analyze more historical data and not real-time telemetry display.
  - *Platform*: Hardware based system (SIT60)

2.1.2. The contribution this project will make to the field of satellite applications

This project demonstrates that a common GUI application can be produced to analyze telemetry. Most of the applications existing today center around two concepts: satellite tracking and ephemeris prediction. Neither of these types of programs is well suited to working with all the different types of satellite telemetry. The custom made applications that are available are very expensive and spacecraft specific. This project shows that a low cost alternative can be produced that is generic enough to be reused on several different satellite systems. This will give

customers more options when working with ground system software. Some of the new concepts and features that this application provides will be someday reused on other software applications.

The design patterns used on this project were not exceptionally innovative, but they do provide a nice, easy to use framework for extending the design for future program needs. The design is not satellite specific and can be reused with other data types that need a GUI in order to display them.

2.2. Summary of Chapter 2

This chapter examined satellite telemetry applications that are available on the market today. There was no clear product that could produce what this project was trying to build. The applications existing on the market were very simplistic satellite tracking software or ephemeris generation software that are not sufficient for analyzing all types of telemetry points on the spacecraft. These products ranged from open source software to purchase by use products. Most of the products did not have very flexible features that could be modified easily.

Other products are custom-made that are usually sold as a package of other ground system software by large aerospace companies. They did not provide a COTS solution for just this one type of product. The bundle of software that they provide can range in the millions of dollars to customize, install, and support.

In essence, this product will provide commercial off the shelf (COTS) software that can be modified easily and at a low cost to the customer. It has features and

functionality that is currently not available to the market without great expense for a custom made product.

3. Chapter 3: Project Methodology

This chapter will describe in detail the software methodology that was used for this project. The methodology closely resembles a waterfall model. The phases consist of planning, analysis, design, implementation, and test phases. For each phase, there is a description of what was accomplished for each segment and the purpose for doing that particular phase. There is also a review and description of all the deliverables in this chapter. In addition, most of the important deliverables are shown as well.

3.1. *Software Methodology*

3.1.1. Hybrid waterfall and agile methodology

For this project there was no one software methodology used. This is mainly due to the project only having one team member and the goals of the project were very broad in scope. While there was a definite purpose to the project, it was a learning experience as the project progressed. The project started out as a basic waterfall methodology; however, as the project advanced the team had to keep going back and redoing certain phases as more information was gathered. As a result, the methodology used is better termed a hybrid type of the waterfall method. In order to carry out the planning and analysis of the project, there had to be some prototyping of

the software ideas. If these ideas were not feasible or turned out to not be the best concept, the analysis then had to be redirected.

This methodology was fairly easy to implement since the team was comprised of only one member. As a result, there wasn't a lot of time spent communicating with other teams members. One conclusion that could be made based upon the team structure, is that this project completed in more of a free form fashion. However, this is not the case. The team worked diligently to adhere to the waterfall methodology as much as possible.

Given that the project was more research based, it required the team to follow a strict structure or form. Initially, the team had to allow the project to go where it wanted because new ideas were constantly being fed back into the design and implementation. Once the project was completed, the project manager assumed that the next evolution of this application would have a more structured form. This project allowed a base foundation to be laid out and give the application a clear direction in which it is to follow.

3.1.2. Planning phase

During the planning phase, the basic schedule was created. The project manager also developed a plan for completing the project with the various deadlines in place. At first the project manager drastically underestimated the time needed to complete this project; it turned out to be far greater than anticipated. It can be assumed that this was due to not having a good understanding of what the application needed to accomplish. As the team began to prototype the concepts, the larger the

application got. The initial plan was for the project to be completed in six months, only having to work approximately 20 hours per month. The project manager created a high-level task list and scheduled out estimates for each task.

During this portion of the phase, the scope of the project was defined. The task list helped define how large the project was going to be. The scope of the project was to follow the software process and produce a software application.

The planning phase was actually very short. The majority of the work was the creation of the tasks and their estimates.

3.1.3. Analysis phase

In the analysis phase the background research was completed. The analysis phase also included the documenting and ironing out of the requirements. Research was done using the Internet and by talking with experienced professionals who knew a lot about how an application like this would be used. Searching on the Internet turned up good results, but not exactly what the team was looking for. Most of the products researched ended up only being short sales pitches and advertisements for their products. This was a good start, although only so much could be gathered without actually attempting to purchase their product.

Conversely, the discussions and interviews held with colleagues in the satellite telemetry field turned up excellent results. The team introduced several ideas and designs to the colleagues and received their feedback for how things could be improved. This aided tremendously in defining the requirements for what the application should do. The team then used this knowledge to shape the design of the

application. Some ideas that originally were to be included in the application were now abandoned with new ones being added.

A number of high-level GUI diagrams were drawn out on paper to assist in showing the concepts to other colleagues. These original GUI designs were used in building the prototype GUI.

3.1.4. Design phase

In the design phase the use cases were defined and put together. First a use case listing was made. Then details of each use case were put in place. After looking at all the use cases, the project manager concluded that there was too many use cases to implement in this project. Some would have to wait until the next evolution of the project. Several use cases were then eliminated to establish what the essential functionality needed to accomplish the project goals. Detail descriptions were made for each use case. This provided a road map for starting the design of individual classes that the software will use.

After the use case analysis was completed, it was time for the sequence diagrams to be built and to realize the use cases. For each use case a sequence diagram was built. It gave more detail as to how the software was going to work. It provided a lower level of detail that use cases normally do not provide. In some cases, individual classes were identified in the sequence diagrams.

A high-level class diagram was also produced to help show how classes would interact with each other. This was especially helpful in figuring out where improvements could be made to the design. For example, at first the diagram showed

that a lot of redundancy was being created in the data object layer of the code. This led to changing the design. By using object-oriented polymorphism techniques, it helped in reducing code size and simplifying the design. This also made the code easier to modify and maintain in the future when bugs in the software were found. The class diagram did not detail every class in the system. Instead it showed the major functional areas of the system and how messages and data would be passed between them.

Some prototyping of the GUI was done during the design phase; however, this became the launch of the implementation phase. The prototypes soon turned into real code that was going to be used in the application.

3.1.5. Implementation phase

The implementation phase is where most of the time was spent on the project. This project required learning about new GUI concepts and features using the Java programming language. The concepts of the design were used, but lots of time was spent in trying out new features of the language that was learned from others or through books.

The initial start of this phase was to create the environment setup to do the work and get the system running. An editor and compiler had to be installed and setup on the development hardware. There was some thought put into setting up a simple software version control system. But since the project only had a limited number of people on it, it was then decided that a simple backup system would work to handle any issues in loss of software. This proved to be a very good idea, as

halfway through the project there was a hardware failure and all information was lost. Because there was a backup system in place, only the environment had to be reloaded and setup again.

After the environment was setup, the classes were put in place to do the functionality that was identified by the use cases. The software development was done in an iterative type of fashion, where small units of functionality were done and then tested. After each software unit was working it was then evaluated as to whether or not this is what the team wanted. If it wasn't quite what was needed then more ideas were tried out. It was estimated that each part of the software was rewritten a couple of different times. Each time the team moved forward with a new unit of code, it was identified that there was a better way to build the GUI and some of the concepts were reworked on the fly.

The exit criteria for the implementation phase were mostly due to time constraints. The point finally came when the team decided that the GUI was satisfactory and ready to be demoed to other people. Some time could have been devoted to clean up during this phase, but it was determined that since the use cases and goals of the project were complete it was time to mark this milestone as complete.

3.1.6. Test Phase

Initially, the test phase did not go as well as originally planned. This was chiefly due to time constraints on the project and that much of the unit testing had already been completed throughout the implementation phase. The preliminary plan

was to carry out individual performance timing on the software. The project was to take measurements from the different input sources and compare which input interface was better and how improvements could be made to the software in order to make if faster. Much of the testing that was done on the software was low-level unit testing and bug fixing. Multiple input data source interfaces were not completely worked through. The test phase turned into cleanup and tweaking of the system in preparation for the demos.

3.2. Review of Deliverables

For this project there were several deliverables that were proposed and created. Each phase of the project had different deliverables and artifacts that were then developed. These artifacts document the progress of the project as well as build on top of each other to make a complete design. Below is a list of the artifacts that were created and how they were utilized. Following the list are figures that represent the most significant artifacts for the project.

1. **Schedule** – Schedule detailing the tasks to be accomplished as well as duration of those tasks.
2. **Scope Document** – Documents the scope of the project as well as goals the project is attempting to accomplish.
3. **Requirements list** – List of the requirements that the project is trying to accomplish.

4. **Use case Diagram** – Diagram of all the use cases and how they interact with each other. See Figure 1.
5. **Use Case Details** – Detailed step-by-step sequences for each use case.
6. **Class Diagram -** Diagram of the high level classes and the interaction with each other. See Figure 2.
7. **Sequence Diagrams -** Class level details that represent the steps in the use cases. See Figure 3.
8. **Software -** The software used to build the application. Screenshots were taken to show functionality that was built. See Figures 4, 5, 6, and 7.

3.2.1. Use Case Diagram

This use case diagram shows the major functional concepts of the project. The majority of the low-level GUI details are not shown here but accounted for in the Organize Data use case. The operator initiates the use cases and typically interacts with the main GUI display. Some input and output XML files are also used.

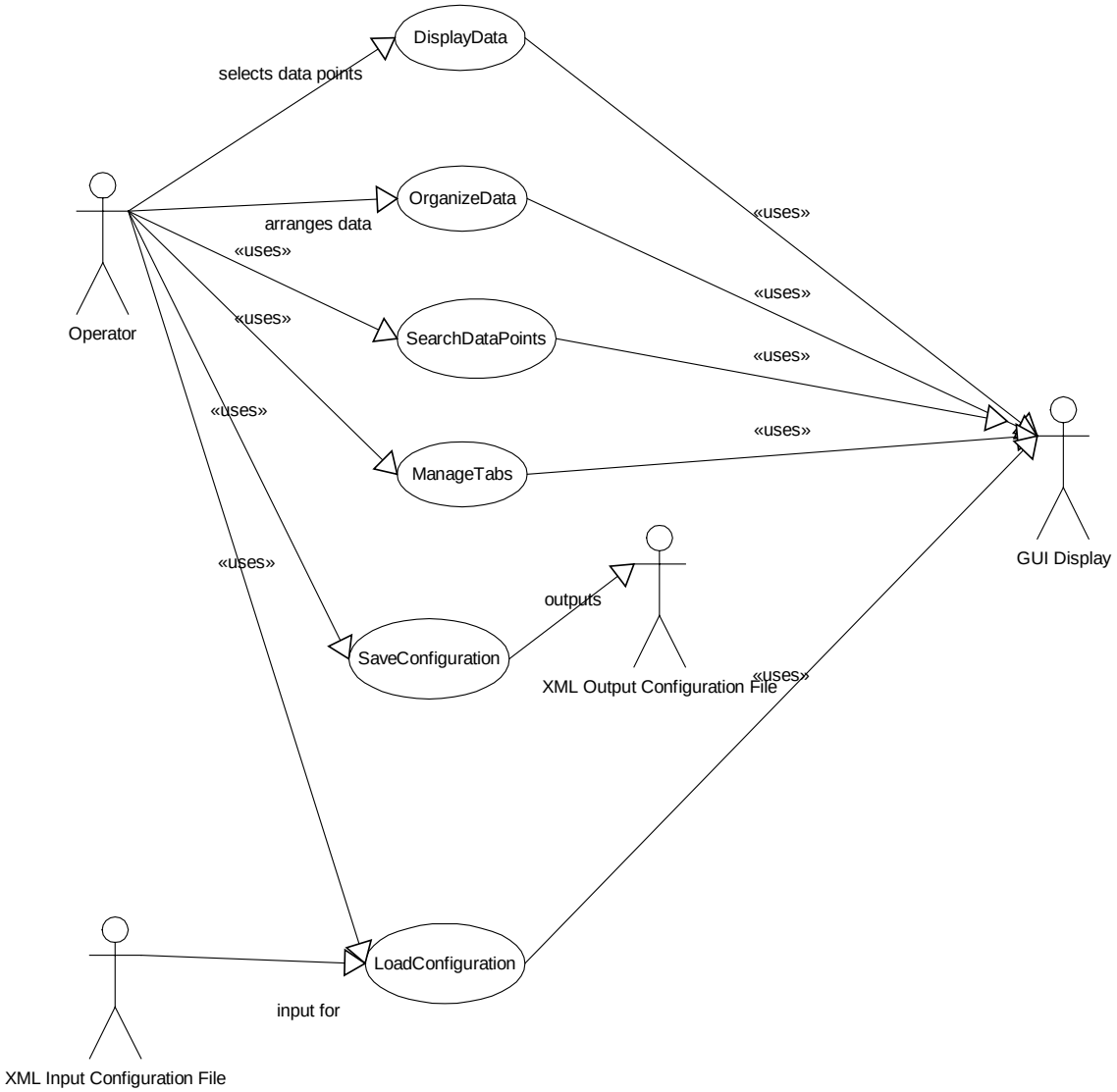

Figure 1 Use Case Diagram

3.2.2. Class Diagram

This is a high-level class diagram that represents the functionality identified in the use cases. It shows the data objects such as the DataGroup, DataAND, and DataTable objects. It also shows the most important part of project, the update mechanism. This uses the proxy pattern and is implemented by the Updatable interface.

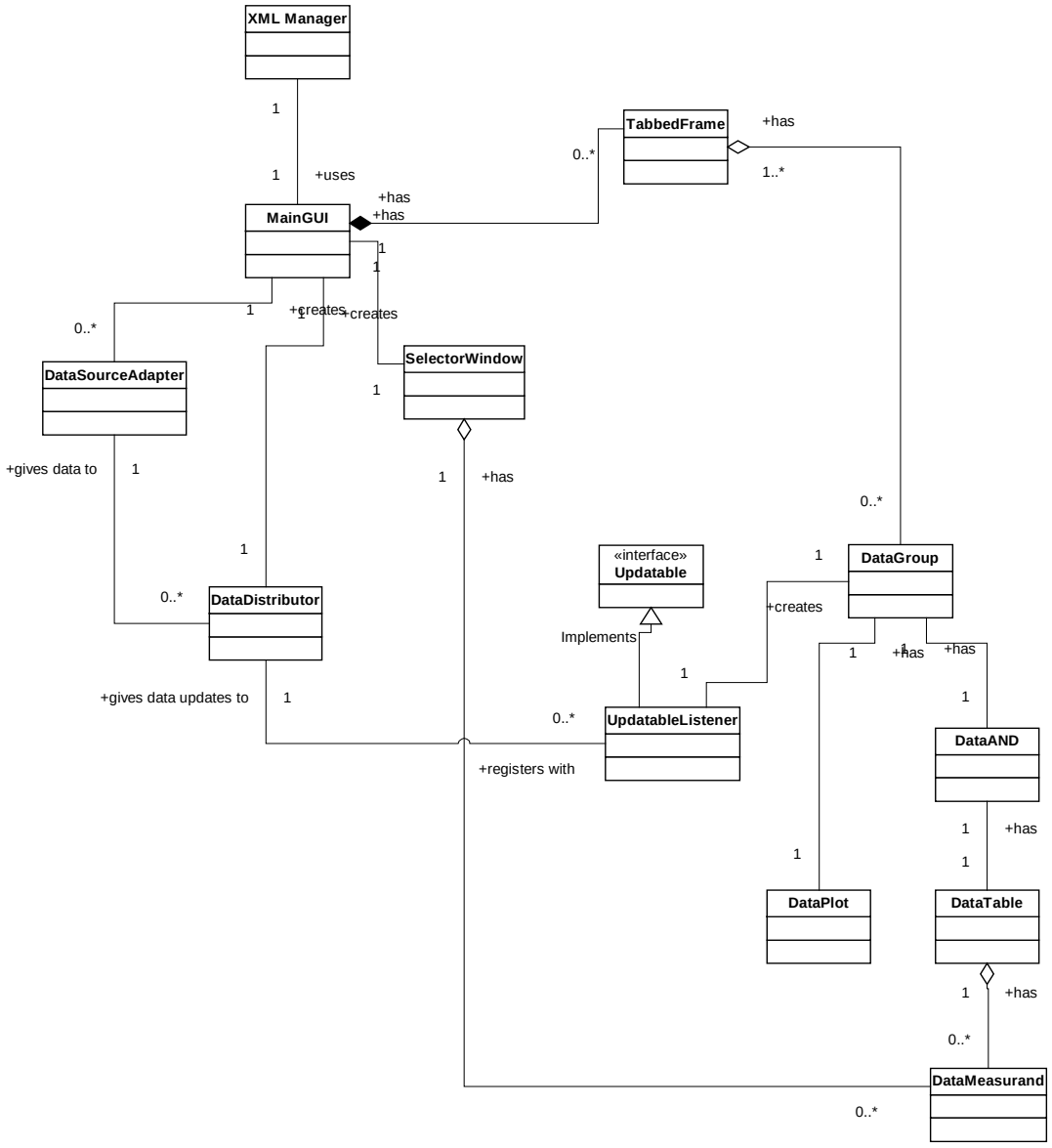

Figure 2 Class Diagram

3.2.3. Sequence Diagram

This is a representative sequence diagram for showing how the GUI can be saved off as a XML configuration file. There were several sequence diagrams created for this project but only one is shown here as an example. In this sequence diagram the XML file could be loaded back again at a later time for use.

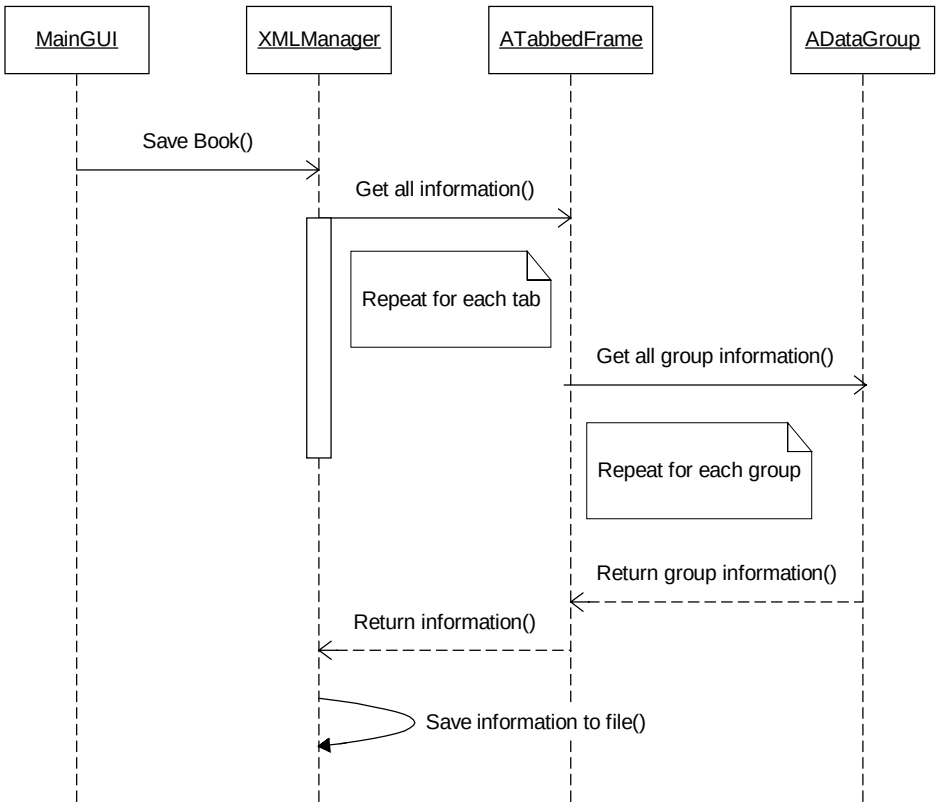

Figure 3 Sequence Diagram

3.2.4. GUI Screenshot #1

This screenshot shows the three parts of the display. On the upper left is the selector section. This is used to select the data points that the operator wants to analyze. The lower left is the filter section to aid in the selection of data points. The right side of main display is the work area where data points and plots are displayed.

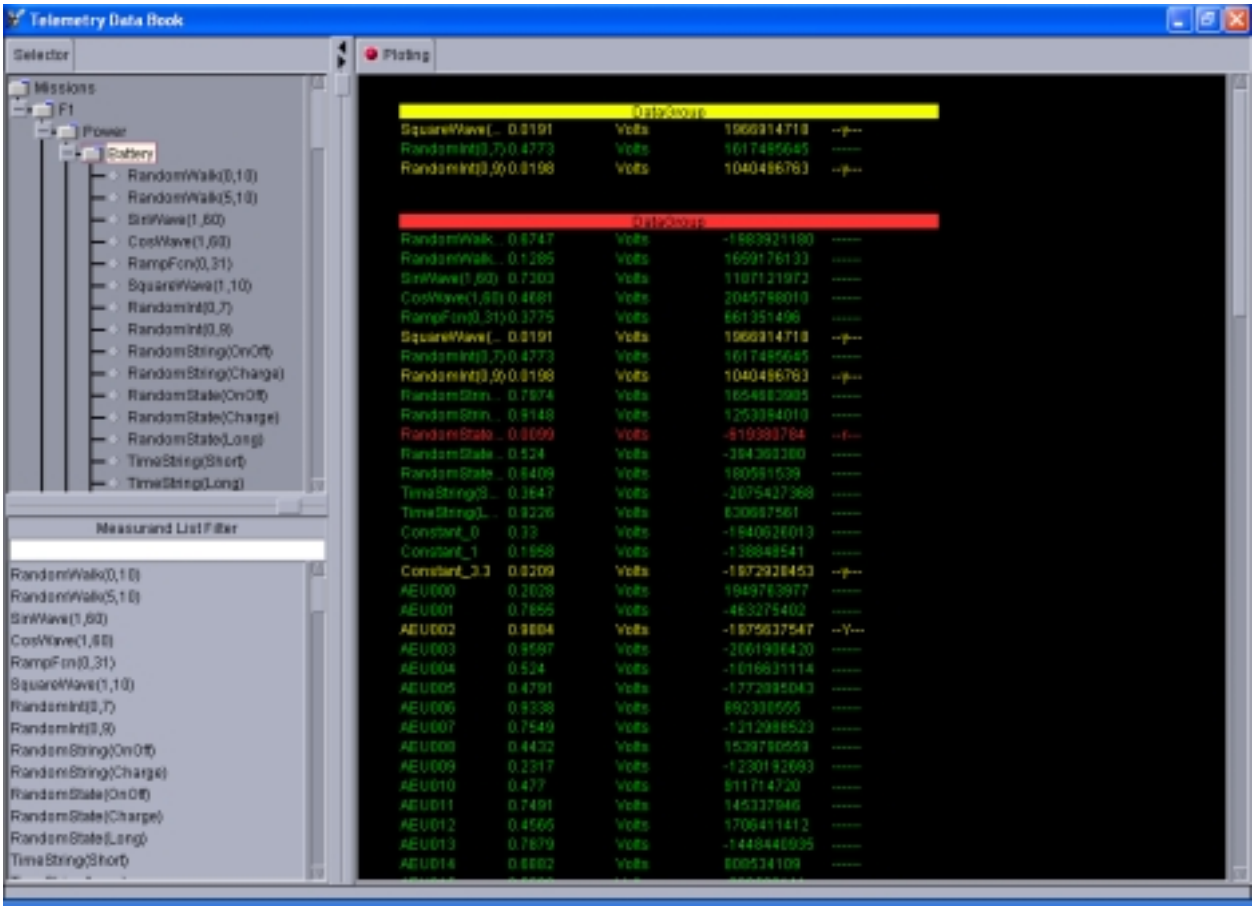

Figure 4 Main GUI Screenshot

3.2.5. GUI Screenshot #2

This GUI represents some of the editing features that are available on the GUI. The GUI data group tables allow each group to be customized to only show certain columns of data and can be configured in a variety of ways. These columns can then be shown or hidden. They can also be removed completely from view for the operator.

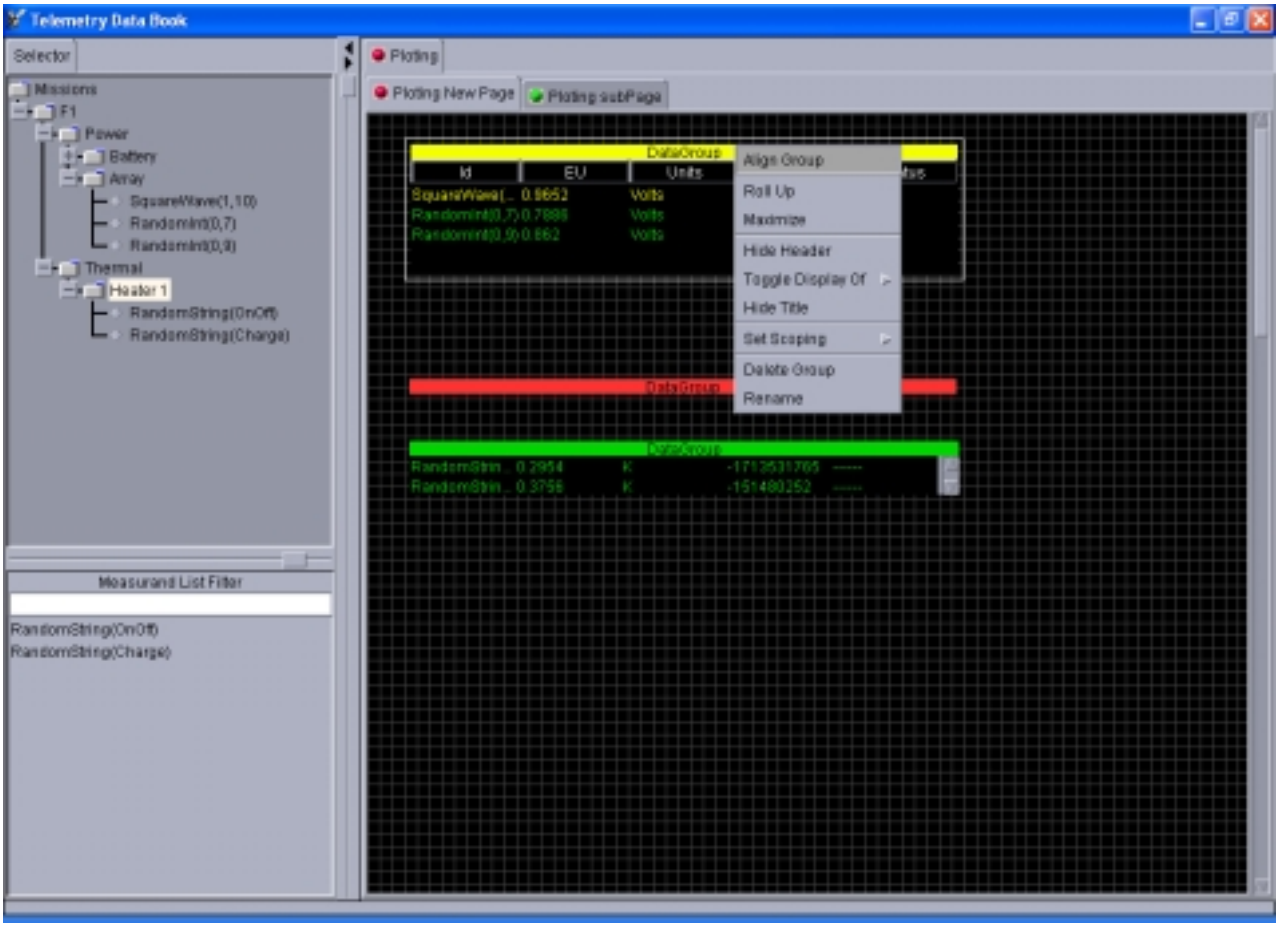

Figure 5 GUI Being Edited

3.2.6. GUI Screenshot #3

This screenshot shows how the GUI display can be simplified for the operator to display only small subsets of information. Figure 6 shows how a telemetry data page can be torn off from the main GUI and be treated as a separate window. This window can then be iconified on its own or deleted as the operator wishes. The main data window is still used to work with and display data; yet this torn off page acts independent of the main application.

Figure 6 Torn Off Window

3.2.7. GUI Screenshot #4

This screenshot is a good example of how once the data displays are setup; all the other editing features such as the selector can be hidden. This allows the operator to have certain selection features turned on for editing and maintenance, but also giving the operator use of the same GUI for operation environments. This allows the GUI to be more flexible so that the customer does not have to purchase two types of software applications; one for creating the displays and another for viewing them.

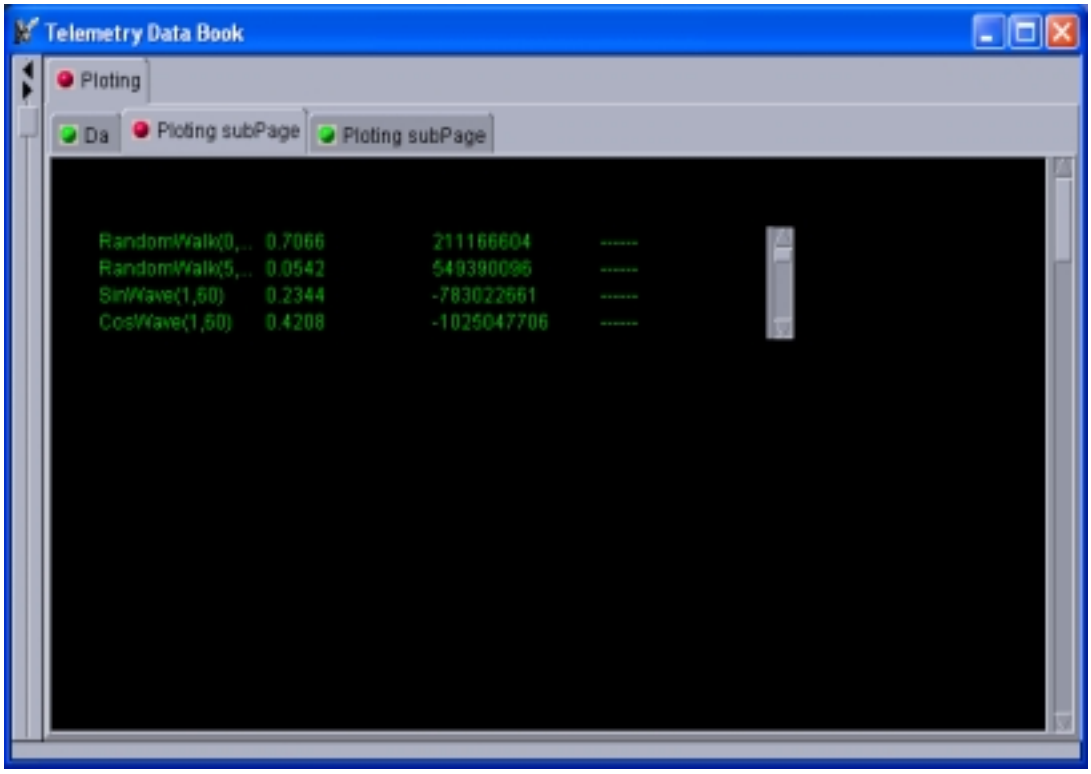

Figure 7 Selector Window Folded Down

3.3. Summary of Chapter 3

Chapter 3 discussed the details of each phase of this project. The project's methodology followed a hybrid or modification of a waterfall type of software model. Although there were distinct phases, there were times in which the project had to go back to previous phases due to new information that was discovered. The phases of the project included a planning phase, analysis phase, design phase, implementation phase, and test phase. Each phase had distinct milestones with associated artifacts. Each artifact was then used in the next phase of the software development lifecycle.

The planning phase was short and mainly used to workout the tasks and durations for the project.

In the analysis phase, research was done on other products available on the market today. This research helped in the guidance that the project took. The project goal was to create a software GUI that was different than other products on the market today. Many of the products found did not satisfy the functional need that the project was trying to accomplish. Most of the products were only ephemeris generation software or satellite tracking software.

The design phase clearly defined and listed the use cases. The list of use cases grew rather large and had to be scaled back. Once scaled back, the detail descriptions for each use case were created. This phase also started to produce GUI prototypes and rough GUI concepts. Class and sequence diagrams were also done to represent the use cases. The class diagram ended up being a high-level diagram and more

detailed classes were implemented in the implementation phase. The sequence diagrams were done to represent each use case.

The implementation phase was where the software was constructed and unit tested. This was the longest phase to complete due to the complexity of the project. GUI concepts were also coded and integrated together. Several times the ideas and software were scrapped for new ideas that were better and would make the software a more integrated and consistent GUI. To ensure that each piece of software ran correctly, unit tests were conducted at several intervals. Accordingly, every time a new piece of software was written, most of the existing functionality was regression tested.

The test phase was the last phase of the project. It was cut short due to a majority of testing was executed in the implementation phase and due to time constraints on the project.

4. Chapter 4: Project History

Chapter 4 will discuss how the project originated, the project milestones, and schedule. The project did not just begin as an idea; but was born from a need that was first identified at a company. The project background will be detailed later in this chapter. This chapter will also describe the project plan and what the milestones were. The project plan set out to accomplish certain milestones which will not only be identified, but also elaborated upon to show which ones were met or discarded. When a project ends, it typically is analyzed to conclude which segments were successful and which segments failed. This chapter helps capture those success and failures.

4.1. Project Background

The foundation for this project started 5 years ago when a team of software engineers decided they were fed up with the constant maintenance of an old and antiquated software product that was used to display satellite telemetry. At the time, it was standard company practice to spend hundreds of hours modifying a baseline software product to fit the customer's needs. Since these software products were not off the shelf products, it was expected to charge the end consumer or buyer of the software to modify it. The company made money on each hour spent to modify the baseline software. This was both good and bad for the company.

This practice had several advantages to it. It kept a steady stream of software engineers employed in order to make the necessary modifications to the software. Then, when the engineers were not working on the new software products they were fixing up the baseline software for the next customer. It was also advantageous due to the fact that once the product was sold; it required maintenance and support services in order to keep it up-to-date. Therefore, a company was not only purchasing software, but also had to purchase maintenance and support in order for the software to be of any value to the company. These maintenance contracts could be very lucrative for the seller company and locked the customer into that product. Once the initial product was purchased, it was very difficult to switch to another company's product because of the their large investment.

The disadvantages to this practice were found in the maintenance and versioning of the software product. Each customer wanted customized modifications to the product. This resulted in several different versions of the product being delivered. It became more difficult when one customer would not accept the software that had features that another customer wanted. Due to the plethora of changes, the software code ended up in a state of what software developers call spaghetti code. The code did not follow any practiced disciplined software design. It had been quickly modified so many times, for so many different customers, that it was starting to become impossible to maintain. The same changes to the software had to be made across the board to each different baseline version of the software. On top of all that, there was a constant change in personnel. The newest developers that began working for the company was always put on this type of maintenance work. Many times the

programmers did not realize the impact of the changes made to the software, which only made things worse.

A pinnacle was finally reached and something drastic had to be done. A decision was made to develop a new software product that would allow for many of the design changes being requested by the customers to be solely configuration changes to the baseline product. The more configurable the product was, the easier it would then be to maintain configuration files instead of constantly changing the source code. The product could also be sold with a cheaper license fee due to the low cost to maintain the product. This would allow for the product to be sold to more customers and hopefully bring in new business due to the company name being more prevalent in the marketplace. A good product name goes a long way in the satellite industry.

A small team of engineers was formed to brainstorm and develop the new product design and features. After several brainstorming sessions, the initial project ideas were laid out and planned. However, soon after the original project began, it stopped. Money for research and development was halted due to changes in the marketplace. The company wanted to focus its money on maintaining and completing existing problems. This was the commencement of this project. The good ideas brainstormed in those meetings were expanded to become the basis for this project.

4.2. Project Management Process

4.2.1. Building the plan

The plan for this project was not to finish in a predetermined amount of time, but rather to complete the research and prototypes of the GUI. At first the project plan was setup as a series of tasks that needed to be completed. Individual durations were given to each task as more of a guideline on how long it was going to take in order to implement the project. If the project were deemed to be too long or large, then the tasks would have to be de-scoped or removed from the project. After reviewing the tasks, it was determined that certain functionality would need to be cut out of the project. While the functionality was necessary, it was a task that could be developed in another phase of the software. The project manager's focus was to have smaller iterations so that future phases of the project could be changed if the previous phases turned up unexpected results.

The necessary tasks in the plan were then lined up in an order to be completed. A rough schedule was created to aide in the direction of the project. Unfortunately the project was stalled due to outside influences. In the end, the schedule did not have much affect on the project. The tasks were still followed and given status but the schedule dates became irrelevant.

The following tasks and durations were created for this project:

- **Planning Phase** – 2 days
  - Define scope - 1 day
  - Create detailed schedule – 1 day

- **Analysis Phase** – 12 days
  - Research telemetry information – 2 days
  - Research other telemetry display products – 4 days
  - Define requirements – 1 days
  - Build use case diagrams – 5 days
- **Design Phase** – 10 days
  - Build GUI prototypes – 5 days
  - Realize use cases – 5 days
- **Implementation Phase** – 18 days
  - Build GUI's – 5 days
  - Build telemetry data engine for testing – 3days
  - Build backend connection of telemetry values – 5 days
  - Integrate all code together – 5 days
  - Write test plans – 3 days
- **Test Phase** – 2 days
  - Setup test environment - 1 day
  - Evaluate software test procedures – 1 day
- **Final Report Write-up** – 2 days
  - Gather results – 1 day
  - Write report – 1 day

4.2.2. Collecting status during the project

The project plan was maintained in Microsoft Project. Tasks and durations were entered into the scheduling tool. Tasks listed on the schedule were followed in the order that they were originally assigned. Status was given to each task to ensure proper management of the milestones. The majority of the tasks on the schedule had an exit criteria that had to be achieved before the milestone completion could be deemed finished. As a task was completed, it was then flagged on the schedule as 100% done. Multiple tasks were worked on at the same time during various phases of the project. Due to this, care had to be given in order to not get too far ahead in other phases of the project. For example, the team did not wish to get too far into prototyping the GUIs without all the research first being completed. If it seemed as though one task was too far ahead then that part of the project was slowed up until other tasks could be completed first.

4.3. Project Milestones

4.3.1. Milestone details

The milestones for this project determined when each phase was complete. Many of the milestones corresponded to a deliverable being created. This was the simplest method to track the progress of the project. The milestones and phases are listed as follows:

- **Planning Phase** –
  - Milestone - Scope document created
- **Analysis Phase** –
  - Milestone - Use case diagrams and descriptions fully documented
  - Milestone – Requirements documented
  - Milestone – Document research on other telemetry display products
- **Design Phase** –
  - Milestone - Class diagrams completed
  - Milestone - Sequence diagrams completed
- **Implementation Phase** –
  - Milestone - Software complete
  - Milestone -User's manual complete
- **Test Phase** –
  - Milestone - Test completed and results documented
- **Final Report Write-up** –
  - Milestone – Final report document

4.3.2. Which milestones were met or canceled

Most of the milestones listed on the schedule were achieved. There were only two milestones not met: one in the test phase and one in the latter part of the implementation phase.

The first was the milestone to create a user's manual was canceled. It was determined that since this the first phase of the project and only a prototype of the real system, there was little need for a user's manual at this time. Also, due to time

constraints, the user's manual was determined to be a nonessential part of the project. For the next evolution of the software product, a user's manual should be produced.

The second milestone not met was the documentation of test procedures and test results. The testing of the software turned into a low-level unit test of the software. The outcome being that a large full system test of the application was not completed. Instead, testing the bugs in the system was the first priority. The timing and performance of the GUI was also not tested. However, this type of performance testing should be done in future evolutions of the application.

4.4. Changes from the original project plan

The initial portion of the project began smoothly. Progress was being made at the predictable rate initially planned. Then unforeseen events occurred. The computer used for the project encountered hard drive problems and resulted in a reinstallation of the software environment used to build the design and source code. There was down time for approximately 3 weeks in order to get the computer fixed and up and running again. Some data was lost and had to be reworked.

The implementation phase of the project took far longer than anticipated. Due to scope creep on the requirements of the project, more time was given to get the software working. New functionality kept being added to the project, which only caused more bugs in the software. With the increase of bugs in the software, the performance testing part of the project had to be removed. Time had to be factored into getting the project done. It was determined to hold the performance testing and metrics collection for a future phase of the program.

4.5. Project Success and Failures

Overall, the project was a big success. It accomplished its major goals and the purpose of what it was trying to do. The size of the project turned out to be far greater than originally anticipated. Looking back, the project manager realized that the project should have had a team of people working on it or at least scoped it down into smaller pieces for just the one individual.

The project also brought about good discussion points with peers in the industry. This was the groundwork for a new out-of-the-box way of thinking about GUI design and reusability. The prototype that was built, served as the discussion point to other uses that this type of design and implementation could potentially have. The project produced a lot of deliverables and documentation. A solid design was created for the software application. The diagrams and use cases were documented in the Microsoft Visio case tool. The project also produced about 5000 lines of source code.

As for what failed on the project, the project was too large to begin with. The functionality that was sought after was just too much to complete in the first phase. Because some of the Java GUI concepts were new, it created lots of bugs in the software. As the bugs were found and fixed, new implementation was also being done. The project should have paused in the implementation phase and reevaluated as to how much of the GUI functionality was going to be done for this project.

Time constraints did not allow for as much testing as originally planned. The test phase was intended to include performance metrics gathering and analysis. When

the test phase began however, it was apparent that the software was in no real shape to put out performance metrics. There was not ample time to create a test environment to collect the metrics or to code a good software driver to act as the data source for testing. Testing of the software could also be considered a failed part of the project.

4.6. Project Metrics

Metrics on the project were collected as part of keeping track of status on the tasks within the schedule. No other metrics were collected. Theoretically, there was to be a large amount of metrics collected during the test phase of the project. In the end this did not happen due the test phase being scaled back in order to fix bugs in the system. Originally, metrics were going to be collected on the speed and data rate that the program could handle. It was important for this type of application to be fast and efficient. Standard Java features and tools were going to be used to collect this data. There are products available on the market to also collect and analyze program performance; but, they were not considered during the planning phase of the project.

Status on the tasks was collected when milestones or deliverables were completed. The schedule was updated with percent complete for all other tasks not finished yet.

4.7. Summary of Chapter 4

The project went very well and was a great success. It accomplished what it set out to do and had good results. From the beginning, the project plan was setup and followed as outlined. Tasks were created with durations and then laid out on a schedule. The project had five phases: a planning phase, an analysis phase, a design phase, an implementation phase, and a test phase.

Each phase had milestones and deliverables that needed to be completed. The planning phase was where the schedule was created. In the analysis phase, research was done and a project scope document was created. In the design phase there were several deliverables. Use cases were created. Class and sequence diagrams were created that mapped directly to the use cases. The implementation phase is where most of the software was developed. The source code itself is a deliverable for the project. The test phase was the only phase that did not go as intended. Performance tests that were scheduled to be run, did not get done. Most of the test phase was used for fixing software bugs in the implementation phase.

Overall, there were very few changes from the original plan for the project. The changes that did occur were in the test phase and creation of end user documentation, such as a user's manual on how to operate the software. The user's manual was taken out of the project, as it did not provide much use for the goals of this project. The software will require more work to be a production application; so documentation of the final product should be held off until a more final phase of the software is written.

The metrics that were collected on the project were mainly schedule status. There were plans originally to collect metrics on the performance of the software, but that was pushed out of the project due to time constraints. It was decided that the software was not ready for performance data collection in this phase.

5. Chapter 5: Lessons Learned and Next Evolution of the Project

Now that the project has been described and explained, it is time to look to the future and analyze what could be next for the software application. Chapter 5 will not only focus on lessons that were learned from this project, but how to apply them to future phases of the project. In addition to lessons learned, this chapter explores new ideas and concepts that future versions of the software may want to consider. This project was only the beginning and this chapter will look at what is next.

5.1. Project lessons learned

The project was a valuable learning experience. There were so many lessons learned that it is hard to capture them all. Therefore, the project team has grouped the most important lessons learned into categories. The categories are:

- Time management
- Managing user expectations
- Using interviews as a vital source of information
- How color and GUI white space are essential to an easy-to-use GUI
- The importance of a good software infrastructure

5.1.1. Time Management

Time management is as important for a software project, as it is in day-to-day activities. Creating a detailed work/plan schedule is central, but so is the constant evaluation of the time to complete each task being worked on, as well as handling issues that come up along the way. When working on the project, there was several times in which conflicts of available time resources appeared. There never seemed to be enough time to start and complete a phase of the project in a short time period. It was as though every time the project was running at full speed and accomplishments were being made; a higher priority task would come along and bump the current task to the end of the schedule. Several times the newer task turned out to not be as important after all. In most cases, it would have been more productive to have completely evaluated the importance of the new task first before just jumping over to it. In evaluating the project, it would have been more efficient to finish up the current task before moving onto the next one. A lesson learned: time management not only includes creating a detailed schedule of tasks and resources; but also having the strength to stick to the schedule even when minor glitches pop-up.

5.1.2. Managing User Expectations

Managing user expectations is a large part of a software project. The project may be going perfectly as planned. The tasks could be on schedule and milestones being reached without incident. Nevertheless, if the end user of the software application is not satisfied with the product being produced then it is all for not. Throughout the project, there must be constant interaction with the customer (either

working directly with one or receiving feedback from potential customers). This interaction helps to validate that the project is still on the right track and will be a success. If along the way the project receives feedback that the software application is confusing or is not user friendly then one of two things may have to happen. One, the project may have to refocus on design of the GUIs and change them to make them more user friendly. Or two, more effort needs to be put into managing the customer. This managing of the customer will require spending more time with the customer to help them perceive what value the software application will have when it is finished.

On this project, not enough time was spent managing user expectations. In the analysis phase much time was spent with a potential customer to ascertain ideas and opinions on what the customer wanted and how the GUI could be made user friendly. Most of these discussions revolved around basic requirements and GUI look and feel. This was an excellent example of how to interact with a customer. Unfortunately, this is the only phase in which this interaction occurred. In the design and implementation phases there was a loss of contact with that current customer. Time was also a factor in the ending of communication with the customer. Unfortunately, the project became behind schedule and there never was ample time to reestablish follow up meetings with the customer. Since the completion of the project, there is some concern that the end product will not live up to user expectations. This will undoubtedly result in some reworking of the user interface. Rework can always be a costly effort on a software project.

5.1.3. End User Interviews

As was discussed in the managing user expectations section above, the art of interviewing the end users to discover information is vital to the success of a software project. The end users (customers) of the software typically have a preset idea of what they want or would like to see. However, in some case this is not true. Sometimes the customer wants to see the future. They desire software to be new and different than other applications they have seen before. They want the project team to brainstorm and present new ideas. Even in these cases it is also good to get customer feedback. It helps in focusing the new and good ideas on the project. Not every idea can be a slam-dunk on the first try. Most good ideas get thrown in the trash once the end user sees them. This is why it is imperative to have constant feedback and interviews with the end users.

5.1.4. How color and GUI white space are essential to a user friendly GUI

One item that tends to get overlooked in software applications is the look and feel of the GUI. Human machine interface or HMI is the study of how humans interact with machines. An important concept today is how humans interact with computers. Computers today do not just have straight command line interfaces. Instead, they are very graphical in nature. Some software applications do not even require a person to use the keyboard at all. The mouse and the use of menu buttons on the application allow the user to do almost everything the software is capable of. Because of this simplified interface, it is more important than ever to create a GUI

that the operator can easily navigate in order to get the maximum use out of it. For easy navigation, the GUI cannot be overloaded with buttons and information. There needs to be an equal amount of white space as well has information on the GUI.

For this project, there was lots of information being displayed on the screen. The screen showed possible telemetry points that could be selected, as well as the grouping they belonged in. The other side of the screen showed data coming in as well as status of each telemetry data point. There potentially could be as many as 2000 data points being displayed on the screen and those data points could be getting updated every few seconds. With this much data being presented to the user, careful consideration had to be taken to not allow the user to get overloaded with information. The operator had to be able to quickly sort out the important information from the irrelevant data.

It was promptly learned that the use of color on the GUI was key in helping the user see the data. Based on feedback from the customer, it was determined that not all the information could be displayed at once. A certain amount of white space needed to be added to the GUI. With more white space and the use of different colors to denote status of the data points, it allowed for a more user friendly GUI. Because some of the data was now not being displayed, there had to be some new design work done to organize the data points and flow up status to the main display the user interacts with.

5.1.5. The importance of a good software infrastructure

The software application developed in this project started from scratch. There was no base software product or application framework that was reused. This had both good and bad implications. It was good in that there was no bad software that this product was based off of. There were also no inherent assumptions of how the software was to be designed or built that could possibly conflict with what the new application was trying to accomplish. It was a clean slate to start building the application from. It was also bad in the sense that much of the basic software functionality had to be created from scratch. Some basic tools, like configuration handling, error handling and reporting, xml input and out, all had to be created for this project.

This project spent a lot of time recreating the basic software components that has been built many times over in the software industry. The Java software language does provide an extensive set of basic infrastructure set of libraries, but more complex pieces were needed. For this reason, the project used the Internet and freeware/shareware products. It did take some time to search out these products and get them installed and working; but in the end it made the software baseline more robust and usable. The lessoned learned in this portion is that the project would have been completed faster if a known set of infrastructure products were already available to use. The ramp up time to find, install, and learn the interfaces of the imported software cost the project lots of time. This needs to be taken into consideration when planning out future schedules. If a baseline is not already setup, then more time

needs to be added to the implementation phase of the schedule. This environment establishment and infrastructure setup could also be another iteration in the schedule.

5.2. Initial project expectations

The expectations of the team coming into the project were set very high. There was a great need that had arisen for this project and there was a lot of enthusiasm on what could be accomplished. The software application was going to change the way that satellite telemetry was analyzed and managed. While, these expectations were set very high, they were not unrealistic. Expectations for this project included:

- Development of reusable and flexible design
- Extensive research on products out in the market today
- Innovative GUI design and layout
- Base software library set that could be easily be extensible

This project was expected to create a design that was flexible and reusable for future versions of the application. Research for this project was anticipated to be done on all types of telemetry display software available today. Freeware/shareware versions as well as customized applications were expected to be investigated. The GUI design and layout were to be unique in that the old antiquated GUI standards did not have to be followed. New and fresh ideas were encouraged and needed to be

used. Lastly, the software built was to use object-oriented techniques and patterns that make the software flexible and reusable.

5.3. Evolution of the Project

In this section the possibilities of the project's future will be discussed. In most cases the hooks to extend the software to meet these future considerations were already put in place. The following sections illustrate the vision of this software project.

5.3.1. Future Revisions to the Application

This project was only the first phase of the software application. It was the concept test and analysis phase. The future of the application is to develop a marketable software application. Although, in order for it to be a marketable application, a lot of work still needs to be done. The range of possibilities is endless, but here is a list of immediate considerations for enhancements:

- Multiple data input interfaces
- Plotting capability
- DataViews type schematics displays
- Enhanced data source scoping capability
- Data range limit setting
- External error reporting and messaging capability

- Database connectivity to store information

Multiple Data Sources

In order to make this software application truly expandable their needs to be a way in which multiple data sources could feed data to the display. This would require developing several different template interfaces. The list of data sources include:

- Raw socket interface
- CORBA interface
- Standard Java class API interface
- XML input streaming
- Database input source

All these interfaces have advantages and disadvantages. The reason the application needs to implement different interfaces is that it will allow for a more flexible installation at customer sites. Some customers do not want to change the input source, so they expect the new software product to easily connect to their legacy system.

Plotting Capability

The ability to plot historical information is a key concept in today's telemetry display software. Most applications allow for the user to select a data point and graph out the history of that data point over a period of time. This allows the operator to do trend analysis of the data. The GUI would then have several different pages of graph data as well as live data points. The operator would be able to mix and match graphs

and data tables as needed. The current application does not have this capability, but the hooks are in place to easily add this enhancement. Some freeware plotting packages were installed and used on this project. Problems were found using the freeware product and so it was removed from the baseline. In the future, their needs to be some trade studies done to find a first-rate plotting package. As the research showed, there are freeware plotting packages that could be used as well as some pay for products.

DataViews Type Schematic Displays

Another common component of products that are available in the marketplace today is the use of DataViews schematic displays (also called mimics). These displays show a schematic view of the satellite system that the operator is trying to analyze. The incoming data changes the schematic in a way that graphical represents what is going on with the satellite system. The software add-on product that makes this type of display is called DataViews. DataViews has become somewhat of a standard in the viewing of telemetry display data. Many customers are very familiar with how the products work and ask for it by name.

Based upon the popularity of DataViews, and that there is not a noticeable replacement for it, this software application would do well if it also had the schematic capability. This application could set out to create its own version of schematics, but the cost of creating them from scratch may not be justified. For future versions of this telemetry GUI, those types of cost considerations need to be weighed. It would be a build versus buy trade.

Enhanced Data Source Scoping Capability

For this project, there were several places where consideration was made as to the type of source data needed to be provided. These types include scoping for: real-time data, playback data, spacecraft specific data, and site-specific data. In the software, there are several places where the operator may choose to select (scope) where the input source data is originating. This allows setting up views for playing back data at a later date and analyzing the historical data. These are quite useful tools for the operator. This project did not complete all the scoping functionality that was initially thought of. For future work, more functionality could be added to fully implement scoping of source data.

Data Range Limit Setting

One of the features of this project was to design a way for the operator to easily see when there was a problem with the spacecraft. This was done through the use of color and data range limits. Each data point has a minimum and maximum range that the data could be set in. If the incoming data went outside the minimum or maximum allowed values, the color would change from green to yellow to red, depending on the severity of the value outside the range. This project demonstrated this concept, but there is no real good way to set these limits or adjust them. For future phases, consideration should be made for an editor to adjust these minimum and maximum values. It would also be beneficial to have several different levels of out of bounds values. For example, depending on the satellite mode, the range of

values might be set one way. Then, if the satellite changed to a different mode, say a more critical mode, then the limit may be different. This allows for the operator to easily change what he or she sees depending on the criticality of the situation.

Currently, this software application hard coded these range values into the software. This was mainly done to display the concept rather than to get the real system working. All data points in the system have the same minimum and maximum values. In the real system, each data point should be unique. A graphical editor to alter these ranges would be preferred, but a standard configuration file would also work.

External Error Reporting and Messaging Capability

Something that every great software application needs is an external API to report errors and messages. Most customers in the satellite world have large expensive legacy systems that they use to do the work. The last thing the customer wants to do is buy a new product that does not work well with their old system. Most of the time, the companies have a system or process for reporting errors and messages to other part of the systems. Typically the legacy system has another GUI just for reporting errors. It would be ideal if this software application could easily be customized to report errors in the legacy format and backbone. Some customers use a publish/subscribe CORBA type of messaging while others are more socket interface driven. If the design of this software is done correctly, it should be an easy conversion to report errors and messages that follow the legacy system standards.

Database Connectivity to Store Information

For the past couple of years, when people hear the word database they immediately cringe. They think of the skyrocketing costs associated with buying and then maintaining these databases. Then they also think of all the high priced hardware they are going to need to run it. Luckily, there are new products out there today that are breaking the trends of the classic database molds. An open source product such as Cloudscape make the big products such as Oracle and Sybase seem unneeded. Cloudscape offers database-like persistence without the high priced overhead.

This software project could benefit from a database such as Cloudscape. There are several different areas that could use database-like persistence. Areas such as:

- Persisting user built telemetry pages
- Persisting data minimum and maximum range values
- Storage of the data points and configuration of each point
- Persisting of scoping configurations
- Storage and retrieval of historical data for later analysis

For some persistent items, a database either locally or remotely will be essential to a good software product. The design could be remote access to a large database to hold this information, but alternatives such as local database access needs to also be addressed. Currently, the software uses XML file storage to store and

retrieve configuration information. This was an intermediate solution that needs to be addressed in future phases of the project.

5.3.2. Possible marketing and selling the application

From the very beginning of this project, it was never about only doing one phase of the application and then quitting. The intent was to create an application that could be produced commercially and sold to satellite vendors. The project was only to be the first step of a proof of concept to sell the application. This project was a good first step. It is a long way away from any kind of marketing or promotion to a real customer. Many future enhancements that were discussed in previous sections would first need to be accomplished. This project has terrific potential for a company to expand it and move forward. Mostly it will require some time and money to further advance the project's scope. Plans currently are to show this demo application to as many people as possible within the satellite industry. With a little luck and good selling techniques, a company may establish some funding for research and development to expand the project.

5.4. Project Conclusions

This project was a software development project. Its purpose was to go through the software development lifecycle and create a software application that was new and different from other satellite telemetry display products on the market today. The project accomplished everything it set out to do and more. The project had lofty

goals and turned into a very large effort for the size of the team. The team was comprised of just one person to do the research, design, and development of the project. The project lasted nearly 2 years because of various delays in the implementation phase.

Scores of research was done on this project, including interviewing customers and gathering information on what the GUI designs and requirements should be. This proved to pay off during the implementation phase, as the GUIs were completed in the analysis and design phase of the project.

The design and implementation phases of the project are where the majority of the valuable work was done. A good design was created that ensured flexibility for future enhancements. The implementation phase however ran into problems with software bugs. This was mostly due to too much functionality that was being developed at the same time. If the functionality would have been developed in smaller pieces and tested more precisely after each piece, it would have created a more stable environment with fewer bugs.

Performance testing on the project was canceled due to how long the project lasted. Original plans for the project included time for testing, but due to the implementation phase lasting longer than projected and because of software bugs still in the code, the performance testing was canceled. On the whole, the project overall was deemed great success.

5.5. Summary of chapter 5

This chapter was about the reflection of what was learned on the project and how the project could be evolved in the future. The project was a great experience for all that participated and there were many lessons learned on the project. Some of which include: time management skills, setting user expectations, using interviews as a good source of information, HMI usage, and the importance of a good starting software baseline.

Time management is an art, and the more a project manager practices it the better he or she becomes. Time management can be comprised of a good set of tools for estimating and managing a project. Setting user expectations is also very important on a software project. Being in constant contact with the end user or customer allows for a project to safely assume that the project will be a success in the end because the customer will be happy with the product. Interviewing customers and end users allows designers on the project to get a feel for how the software will be used. This is important for addressing the customer's needs. Another skill that designers can acquire is in HMI. The look and feel of a GUI can drastically make or break a software application. A GUI that no one wants to use can force customers to be unhappy and abandon the software product all together. The last lesson learned is the importance of having a good software baseline to start with. Having a good tool set allows developers to spend more time on the important parts of the application rather than waste their time developing a baseline.

There are many enhancements that can be made to the software application in the future. Some of these include: multiple data input interfaces, plotting capability,

DataViews type schematics displays, enhanced data source scoping capability, data range limit setting, external error reporting and messaging capability, and database connectivity to store information. Any combination or all of these enhancements are future project expansions that can be done. Each enhancement could be a separate phase in the project or all could be combined if the project was large enough. This project was only the first phase in a large scope of what potentially could be accomplished. With the added enhancements, the software application could be marketed and sold to customers.

In conclusion, the project turned into a large-scale software project. Many developers could have been added, and the project could have had an official testing phase. Overall, the project was a success and served as a springboard for new development on the application down the road.

References

MacDopplerPRO X. Last Retrieved February 18, 2006
From http://www.dogparksoftware.com/MacDopplerPRO.html

Nova for Windows. Last Retrieved February 18, 2006
From http://www.nlsa.com/nfw.html

AMSAT The Radio Amateur Satellite Corporation. Last Retrieved February 18, 2006
From http://www.amsat.org/amsat-new/index.php

SCRAP Satellite Contact Report Analysis and Prediction. Last Retrieved February 18, 2006
From http://207-207-72-42.ip.theriver.com/SCRAP/SCRAP_files/frame.htm

Instant Track. Last Retrieved February 18, 2006
From http://www.amsat.org/amsat/instanttrack/

Orbitron – Satellite Tracking System. Last Retrieved February 18, 2006
From http://www.stoff.pl/

SatScape. Last Retrieved February 18, 2006
From http://www.satscape.co.uk/

Predict. Last Retrieved February 18, 2006
From http://www.qsl.net/kd2bd/predict.html

Portable Predict +. Last Retrieved February 18, 2006
From http://www.qsl.net/kd2bd/predict.html

Petit Track. Last Retrieved February 18, 2006
From http://www.qsl.net/n1vtn/petittrack.html

PocketSat +. Last Retrieved February 18, 2006
From http://bigfattail.com/software/pocketsatplus/

TrackSat. Last Retrieved February 18, 2006
From http://www.qsl.net/zl3ad/tracksat.htm

SIT60 SATELLITE INTERNET TELEMETRY. Last Retrieved February 18, 2006
From http://www.globalw.com/products/sit60.html