Spring 2006

Object-Oriented Design of an Automated Calibration System for an Analog I/O Process Control Device

Craig N. Rogers
Regis University

Running Head: OO DESIGN OF AN AUTOMATED CALIBRATION SYSTEM

Object-Oriented Design of an Automated Calibration System for an Analog I/O Process Control Device

Craig N. Rogers

Regis University

School for Professional Studies

Master of Science in Computer Information Technology

Abstract

The goal of this project was to deliver the design of object-oriented software for the control of a custom calibration/test system. This calibration/test system is to be used for the production testing and calibration of an Analog Input device used in a process control system. Software features include a GUI (Graphical User Interface), Analog Input device troubleshooting tools, calibration/test system calibration and troubleshooting tools, and report printing capability. This project followed the methodology defined by the Unified Process Model and delivered design documentation through the Inception and Elaboration phases. This paper discusses the planning, documentation and testing differences between a large software project and a small software project.

Table of Contents

List of Figures

Object-Oriented Design of an Automated Calibration System for an Analog I/O Process Control Device

The primary objective of this project was to develop high quality software for the control of a custom test/calibration system. The purpose of the test/calibration system was the production test and calibration of a product called an Analog Input (AI) Module. The software developed for this project runs on a PC, called an AI Test PC, which controls all of the calibration steps and functional tests necessary to prepare a given AI module for customer use.

The AI module is a precision measurement device that reads voltage and resistance values and is used by customers within a Distributed Process Control System (DPCS) (See Figure 1) as part of an industrial process (ex. Petroleum refining, power generation etc.). It is one of a family of I/O modules providing a variety of input and output capabilities to the DPCS system.

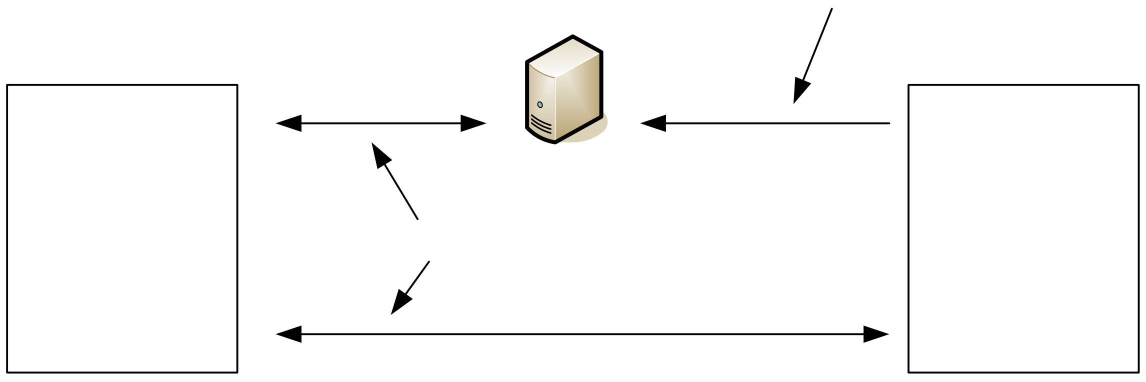

Figure 1 - Distributed Process Control System

Project Need

To achieve the high precision measurement capabilities required of the AI module, a series of functional tests and calibration steps must be performed. These tests and calibration steps are performed using a custom designed and built AI Test System (See Figure 2 for a simplified view). This test system contains instrumentation capable of providing all of the different voltage and resistance values required as inputs to the AI module.

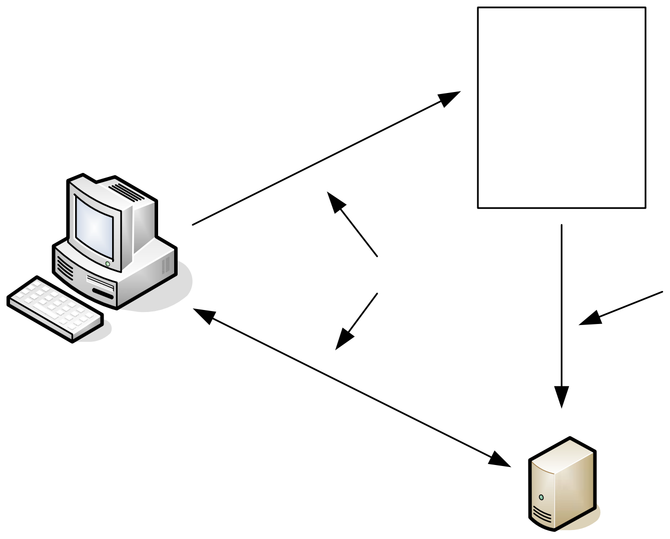

Figure 2 - AI Test System

The AI Test System operates as follows to calibrate an AI Module: The software running on the AI Test PC communicates with the test system instrumentation to supply an input value to the AI Module. At the same time the software sends a message directly to the AI Module with information on what input value it should expect to read. The AI Module reads the input value supplied by the Test System instrumentation, compares it to the expected value and generates and stores the value it read and the difference (called an offset) in memory.

The new AI Test System replaces an aging, obsolete test system. It will increase production yields, reduce testing time and be more easily maintained than the old test system. The focus of this project, for the purposes of this Professional Project, was limited to the design of the software and an examination of the design process to assure a high quality result.

Scope

From an organizational viewpoint, the complete project included the completed hardware and software needed to perform both the calibration of the AI Module and the test system. For the purposes of this Professional Project, the focus was reduced to only the software portion of this project. It was assumed the hardware would be operational and available. As the project unfolded, the test equipment and some key hardware design documentation and C++ libraries needed for coding were not completed or available as needed. Therefore, the project was completed only through the deliverables defined (see below) for the Unified Process Models' Inception and Elaboration Phases.

Included in this project are both a system analysis using Object-Oriented development tools and a Software Test Plan based on the Testing Maturity Model (Burnstein, 2003). Deliverables included the following: A full set of Expanded Use Cases and Design diagrams, Screenshots of major screens, and a Software Test Plan.

This was considered to be a small development project with one test engineer assigned to complete both hardware and software. Current organizational quality processes are targeted at large development projects and require significant documentation, inspections and testing. These organizational burdens would have added significant time delays to the project. To avoid these delays and still achieve the goal of high quality software the development process was adapted to fit the needs of a one person development project. These adaptations are noted and discussed throughout this paper.

Methodology

One of the goals for any design project is to achieve a high quality level for the final product. The development process to meet this goal requires a well defined methodology, detailed project plan, quality plan, test plan and design documentation to support the design and decisions made. Significant documentation would be required to cover all areas in detail. This level of rigor would overwhelm a small design project. This paper will examine where and how to reduce this burden and still achieve high quality.

The design and development of this software was performed by one individual. To be effective, a methodology was needed that was flexible, provided a well-defined path and documentation requirements that were thorough but not obsessive. The Unified Process Model is a scalable methodology which meets those requirements. This approach provides four iterative phases for the definition and development of the software. As adapted for this project these phases include the following:

1. *Inception Phase* – This phase defines the basic requirements/objectives, activities and evaluation criteria for the project. Deliverables for this phase include: a project vision statement, identification of primary actors, significant use cases and descriptions of additional requirements.
2. *Elaboration Phase* – This phase expands on the Inception phase. Requirements are fully developed, test requirements clearly defined, risks detailed, project plan expanded and a design model with some prototypes available. Deliverables for this phase include: a requirements document, analysis model with use cases and domain model, design model, user interface screenshots, test plan and construction phase plan.

3. *Construction Phase* – As the name implies this phase builds the actual product. Executable code is developed, preliminary user documentation is written, design documents completed, testing is fully defined, unit testing and integration testing completed. Deliverables for this phase include: executable (and installable) software, preliminary user manual, test report (for Unit and Integration testing).
4. *Transition Phase* – This phase delivers the complete software package. System level testing is performed, user documentation completed, design documentation completed. Deliverables for this phase include: deliverable software, user manual, and test report (system level and beta testing).

Once again, this project was limited to the inception and elaboration phase by the following constraints: test equipment hardware was not completed, critical C++ libraries were not available, and test equipment design specifications were not completed.

Definition of Terms

The terms defined below are significant to the understanding of this project and paper**.**

- <u>AI</u> - Analog Input Module. One of a family of modules providing a variety of input and output capabilities to the system. The AI provides for sixteen (16) precision inputs of three possible types (RTD, millivolts, Volt).
- <u>Blackbox testing</u> – Also called “Functional Testing”. This test technique focuses only on the input and output specifications of the item under test. The tester does not have any visibility into the inner workings of the system or code.
- <u>CMMI</u> - Capability Maturity Model–Integrated - A process improvement model developed by Carnegie Mellon University’s Software Engineering Institute. This

model provides process requirements and methodology necessary for the assessing and improving an organization's processes.

- GPIB - General Purpose Instrumentation Bus (IEEE 488.1) that allows up to 15 intelligent devices to share a single bus by daisy-chaining the devices.
- RTD - Resistance Temperature Detector. Measures temperature based on the principle that temperature changes in pure metals results in an almost linear positive change in resistance.
- TMM - Testing Maturity Model - A process improvement model, focused on software testing, developed by the Illinois Institute of Technology. This model applies, and expands, CMMI principles to an organizations software testing processes.
- USB – Universal Serial Bus – A serial bus standard used for communications between PC's and peripheral devices.
- UUT - Unit Under Test - The subject module to be tested and calibrated.
- VME - VERSAmodule Eurocard Bus - A computer bus standard.
- VXI - VME eXtensions for Instrumentation. An open standard bus architecture used for communications with test and measurement equipment.
- Whitebox testing - Also called "Glass Box Testing". This test technique uses detailed knowledge of the programming code of the item being tested to select the input test data and define the expected outputs. This type of testing may be applied to a complete application or to components of an application. Frequently it will require additional programming (called a "test harness") to provide the inputs and monitor the outputs.

- xAPI - Application Programming Interface. Library of functions that provide software application programs access to the company proprietary management and control system information.

Summary

The primary objective of this project was to develop software for the in-house production testing of an Analog Input Module. These goals were modified due to time constraints and the unavailability of the automated test/calibration system hardware and key deliverables from the test hardware designer. The reduced objectives for this project paper are limited to the deliverables associated with the Unified Process Models Inception and Elaboration Phases. The project will be completed through the Construction and Transition phases at a later date.

A secondary objective was to find methods to adapt the organizations development processes, designed for large development projects, to fit the needs of a one person development project while maintaining the requirement for high quality.

Chapter 2: Research

This project was a software development project with a narrow focus; the test and calibration of a hardware product using a custom calibration/test system using a development process adapted to a one-person development project. In this case, the primary research for the software design was limited to acquiring the published documentation for all the hardware components of the system and User interviews. Additional research was performed on alternate programming languages and on the achievement of high-quality software through the design of software development processes.

The software design documentation acquired included such company generated materials as the Product Engineering Specifications, Product Instruction Manuals, Factory Calibration Instructions and system schematics. These documents provided the actual performance requirements for the AI Module and the necessary technical data to communicate and control all Company devices used within the calibration/test system. Operator and Programming manuals for the test and measurement instruments used in the test system provided similar information on the communication and control of all programmable devices in the system.

Beyond the technical information needed to design the basic test and calibration functions is the need to provide an adequate user interface and tool set. The interface and tool set needed to be easy to use and provide the ability to perform product testing and troubleshooting, as well as, testing of the calibration/test system itself. The author’s 20 plus years of experience in the design of test equipment, combined with interviews with technicians experienced in product troubleshooting provided the inputs for this segment of the project.

Programming Languages

The selection of a programming language required some analysis. The choices considered were Visual C++, Java, and a commercial graphical development environment software package aimed at developing automated test and measurement applications. There are several such packages on the market which provide graphic programming capability for programmable instrumentation using any standard communication interfaces (ex. GPIB, VXI, serial, USB etc.). These products provide device drivers and command libraries contained in easy to use graphic blocks. These products are aimed at test equipment engineers and provide a very fast and flexible method to develop test and measurement application software. Agilent Technologies, Kiethley Instruments and National Instruments are some of the most popular vendors.

The possibility of using one of these graphical tools to develop the test software was examined. Product literature at each company's website was reviewed and two packages emerged as good candidates. National Instruments' LabWindows (price range $1,200 to $4,300) and Agilent Technologies' T & M Toolkit with Test Automation (price starting at $1,500) were examined in more detail because they both allow the direct use of C++ code. This was a critical requirement because the Company's proprietary C++ libraries provide the only method of communicating with and programming the system Controller module.

LabWindows is a highly intuitive graphical environment that uses device drivers and command libraries contained in easy to use graphic blocks. Graphic blocks are dropped on a drawing and connected as necessary and the required specifications entered. Any required C++ code is entered into blocks and the whole application code can be compiled.

Agilent Technologies primary offering is named T&M Toolkit with Test Automation, retail price starting at $1,500. This package is suite of tools and libraries which integrates directly with Microsoft's Visual Studio C++ or Visual BASIC. T&M Toolkit does not provide all the graphical tools supplied by National Instruments' offerings, but it greatly eases the development of pure C++ code by automating many of the common test and measurement tasks.

These graphical tools are excellent for controlling programmable test and measurement instruments and general test development. However, there are both direct and indirect costs associated with the use of these tools. Although the graphical nature of these packages is intuitive to a degree, achieving proficiency still requires ascending a learning curve. For this one-time project it was felt that the potential development time savings of such a tool did not justify the purchase cost and the learning curve delays. Another point against these tools was the difficulty of supporting future maintenance of the test software.

Java was considered as an option. Java is frequently used in situations requiring the reuse of legacy C or C++ code. Another common usage is to directly access hardware run with C or C++ code (Foote, 1996), which is the situation in this project. Java has some advantages over C++. In general terms it is easier to develop graphical user interfaces and coding is more portable that C++ (Dietel & Deitel, 2003). At a more technical level, Java has superior memory management and avoids the difficulty of multiple inheritances (Martin, 1997). The author had training in Java programming but no practical experience. The prospect of using Java for this project was attractive for the technical challenge. However, it was felt that the advantages of Java were not sufficient to justify the move from C++.

A program written entirely in C++ is far more supportable by this organization than a program written partially in a previously unused development language interfaced to C++ code blocks. This decision might have been completely different if there was any expectation of future test equipment development projects.

The choice of programming language was limited to C++ for the following reasons:

- The proprietary xAPI Library, required for communications and control of the System Controller, consists of C++ classes and functions.
- All the programmable test and measurement equipment used in the test system come with C/C++ compatible command libraries.
- Assigned Programmer is most experienced in C/C++
- High costs of procuring and using a test & measurement application development package.

Development Processes

Quality is a characteristic that products and services possess. However, quality levels (poor, good, high), as used in general conversation, are more perceptions than an exact definition (Russell, 2004). Several factors are considered when discussing software quality. A 40 year old, unattributed definition of quality, cited in several articles and books (Glass, 2001), refers to quality as a collection of "ilities"; Reliability, Modifiability, Understandability, Efficiency, Usability, Testability, and Portability. The weighting given to each of these quality factors varies from project to project, company to company and manager to manager. With all of these variables, defining software quality for even a small project is extremely difficult (Russell, 2004).

The phrase high quality is really a misnomer; *acceptable* quality for a product is probably a better description. The achievement of acceptable quality is a balance of quality factors and resource management. A large development project would require the establishment of metrics and quantification of the desired levels for these metrics and a method for collecting and analyzing these metrics (Russell, 2004). Small projects can be less stringent since fewer people and documentation is involved. This paper will not attempt to tackle all of possible parameters and metrics used to describe software quality. Much has been written and is available on these subjects.

On one point most authors agree: quality is not just passing a test at the end of the project. Quality is built into the product throughout the development process. True quality improvement is systemic. An organization begins with simple processes and goals and slowly works to improve these processes through a continuous improvement program (Humphrey, 1989). To define this process improvement process the Software Engineering Institute (SEI) of Carnegie Mellon University was formed. The SEI developed the Capability Maturity Model (CMM) as a model for software development processes. This model has been refined over the years and has evolved into the Capability Maturity Model Integration (CMMI) (Kasse, 2004).

The basic concept behind the CMMI is that an organization passes through a series of Maturity levels as it improves its development processes. An organization cannot be expected to adapt the detailed processes and tools of a high maturity level organization overnight. There is a significant learning process involved and an organization must be comfortable at one level before taking the next step. An environment of continuous improvement must exist within the organization. Several process areas are addressed by CMMI. Among these are: requirement specifications, project planning, project management, quality planning, configuration management, coding, testing and software maintenance.

Requirements Development

“Requirements are the foundation for any development project” (Christensen & Thayer, 2001, pg. 63). A general requirements specification leaves too much to be interpreted by the designer. The end product of such a specification would be a product that does not meet the customers’ needs. Requirements specifications should describe the functions, performance, external interfaces and design constraints (Thayer, 2002). All requirements should be clearly defined and verified prior to starting the design (Thayer & Dorfman, 2002). For this project the Unified Process Model provides for this activity as part of the Inception Phase. UML use cases were used to develop the specifications. The iterative nature of this process fulfilled the verification requirement by reviewing the requirements with the main stakeholders. This activity, as applied to this project, is described in more completely in Chapter 3.

The importance of the requirements specification is not only for the designer. It is also the crucial element for quality assurance. All design, code, test and implementation documents need to be traceable to the requirements specifications (Palmer, 1997).

Testing

A test plan is frequently overlooked or glossed over by most developers. However, as pointed out by Ilene Burnstein (2003), testing is a key part to achieving a high quality product. Test plans need to address testing during various phases throughout the project. Who develops and performs the testing will vary depending on the size of the organization, the project and the Testing Maturity Model (TMM) maturity level of the organization.

A small project, or an organization at a low maturity level, may use only developers to perform testing. Testing for a large project may require a completely independent team of testers for testing the completed product or release candidates. In either case, detailed test cases need to be developed and documented. Maintaining these documented test cases is of importance for future maintenance of the software. Changes made by new programmers can be analyzed for their potential impact on the software and regression testing.

Test plans also need to include required test resources, both people and equipment, and a project schedule. For this project, a formal test plan was written (see Appendix B for the complete software test plan). Due to the limited size of the project, and the fact that the complete development project was being done by only the author, testing will also be performed by the author.

There are many types of testing that can be performed at different points of the software development. Some of these tests require knowledge of the underlying code (Whitebox Testing) while others are performed from a User's perspective with no knowledge of the code required (Blackbox testing). There is also a concept of testing software before it is actually coded (Patton, 2006). This technique actually uses paper prototypes of menus and other user interface screens to allow an end user to walk through the program. This activity is actually performed as part of the requirements identification. If used with the Unified Process Model this activity would fall into a later iteration of the Elaboration phase.

Within the two basic test types of testing, Whitebox and Blackbox, are a number of sub categories of tests. Whitebox tests are designed to find flaws in the internal logic of the code (Parekh, 2005), (Burnstein, 2006). This requires an in-depth knowledge of the code. As such it requires a highly skilled tester to design and perform the tests. Generally this activity is performed by the programmers themselves, either by the code writer or by a peer. Two of the most popular types of Whitebox testing are unit testing and statement coverage. Unit testing verifies the functionality of a section (or chunk) of code dedicated to a particular action or set of actions. There are no hard and fast size requirements for this chunk of testable code and the definition is left up to the tester. Statement coverage tests uses tests designed to exercise every statement in the code at least once.

Blackbox testing designs functional tests based more on the User's point of view. The tester first must identify the various actions to be performed by the software. This includes both outputs and outcomes (Bolton, 2006). *Outputs* are the immediate and observable results of performing an action. An *Outcome* is a look at the state of the system after performing an action. Checking outcomes may require a deeper analysis of the system than a casual user would attempt. Examples of these outcomes could include file creation, registry changes, driver updates etc.

Test cases are then designed to perform the various actions and confirm the corresponding outputs and outcomes. These test cases should include not only standard actions but also test the limits of the software by performing actions and entering values that are outside the accepted range. This is done to assure that the software is capable of handling these out of range actions without a catastrophic failure. This is a crucial requirement for testing since users will frequently attempt to perform inappropriate actions.

The test plan designed for a project cannot address all test types nor can it be comprehensive for all possible test cases. The amount of testing to validate all aspects of a program is simply too time consuming and complex to be practical. Instead each software project requires an individual test plan unique to that project. The selection of testing types and the test cases performed must be done by an analysis of the project (Burnstein, 2002). Two of the main questions to be asked are: (1) What is the purpose of the software? (2) What are the worst case consequences of a failure of the software? Software with minimal consequences would require less testing than software with higher cost consequences. Where these limits are depends on the Company's judgment.

Summary

This project required the adaptation of a development process designed to support large projects to a one person project. Decisions were made that allowed a balance between the established processes and the limited resources (human, monetary and time) available. Programming language, methodology, requirements specifications, design tools and testing plan were all selected to minimize documentation time and still meet the primary quality requirements set by the organization. These quality requirements are the establishment of solid product requirement specifications and traceability of all design and test documents back these specifications. The remainder of this paper shows the selection and development of these documents and process choices made for a small development project.

Chapter 3: Methodology

As stated earlier, the methodology selected for this project was the Unified Process Model. Originally developed by Grady Booch, Ivar Jacobson and James Rumbaugh (referred to by popular literature as “The Three Amigos”) for Rational Software, it is now under the umbrella of IBM with their purchase of Rational Software. IBM markets a series of products and tools based on this development process. This project used the methodology without any of these sophisticated tools. The Unified Modeling Language (UML), as implemented by Microsoft’s Visio, was used to develop all the software models for this project.

Unified Process Model

The Unified Process model is a configurable, scalable process suitable for use in both small and large development organizations. In the simplest terms, the Unified Process provides a standardized process to develop and maintain models of a software system. It breaks development into four logical and well defined phases. Several iterations within each phase may be performed to refine the deliverables generated for the active phase. It was developed by the creators of UML to make the most effective use of these modeling tools.

This project was only completed through the Inception and Elaboration Phases. Construction and Transition Phases will be performed at a later date.

The Inception Phase of the project is used to both identify the basic requirements for the most significant functions of the software and reach concurrence among the stakeholders on the scope of the project. Deliverables typically developed for this phase include:

- Project Vision Statement to provide a general overview of the core requirements and features.
- Use Case Models for the most significant use cases.

- Supplementary Specifications to describe those non-functional requirements that do not easily fit within a use case.
- Identification of primary actors.
- Project Plan showing the completion dates for the major tasks within in each development phase.
- Initial Business Case and Risk Assessment. This project was for the development of a test system to support a new product. In this situation the business case and risk assessment were performed for the product and is not applicable to this project.

The Elaboration Phase is the most critical of the four phases (Brugge, 2002). It is during this phase that a fully developed vision of the software is created. The primary work product is a detailed design model of the software. This model is presented through the use of various UML diagrams. For this project the design model includes Use Cases, Sequence Diagrams and Class Diagram. These diagrams were selected because they provide the greatest amount of information in a compact form. They are easily understood and can be used to trace the design back through to the product requirements, a key step in a quality inspection.

The deliverables developed for each phase of this project are presented below and discussed in more detail later in the chapter.

Deliverables

The deliverables created for the Inception Phase were:

- A *Project Vision Statement*. This consisted of a short description of the core requirements for the software.
- *Use Case Models* for those activities identified as significant to the software. These were developed as a UML use case diagram.

- *Supplementary Specifications* were presented as a list of descriptions for requirements not easily shown in a diagram.
- The *identification of Primary Actors* was done using a brief description of the titles and functions of the various users of the software.
- The *Project Schedule* was presented as a Pert Chart showing the major tasks with estimated task time, start and completion dates and the dependencies between these tasks.

The deliverables created for the Elaboration Phase were:

- *Expanded Use Cases*. These Use Cases included both UML Use Case diagrams and a detailed text description for each of the most significant use cases.
- *Sequence Diagrams* consisted of a series of UML diagrams for major tasks showing the timing of various activities.
- *A Class Diagram* showing the relationship between the various identified classes.
- *A Test Plan* was developed as a separate document identifying the major test requirements, tools and schedule for testing.
- *The Project Schedule* created during the Inception Phase was expanded to include new information.
- *Screenshots* were created for several displays to demonstrate the User Interface.

Where appropriate, Unified Modeling Language techniques were used to present the requirements in a consistent manner. As described by Charles Richter, UML "is an object-oriented analysis and design notation." (1999, p.9). UML is not a methodology but a language capable of describing a system from different viewpoints and to various levels of detail. These features provide excellent tools for the Unified Process Model. A more detailed description of the UML diagrams and examples of them are included later in this chapter in the discussion of the project deliverables.

Distribute Process Control System

In order to develop this software it was first necessary to understand the basics of the product and test equipment involved. Armed with this understanding the scope of the project could be more clearly defined and the definition of the project more fully developed.

As previously mentioned, the AI module is a component of a Distributed Process Control System (DPCS). It is one of a family of I/O modules providing a variety of input and output capabilities to the system. A basic DPCS system (see Figure 3) requires an Operator Control Station (PC or Workstation, running proprietary control applications), communication interface, Control module and several different types of I/O modules. DPCS systems are designed for a specific customer application using various quantities and types of Controller and I/O modules.

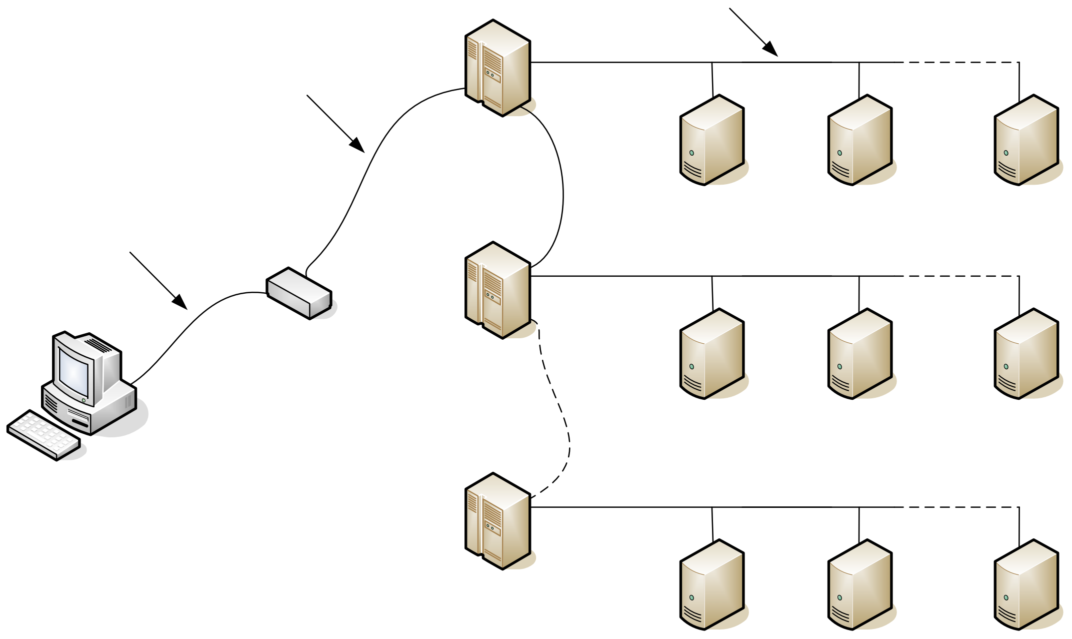

Figure 3 - Distributed Process Control System

In a DPCS system, an Operator Control Station sends a custom configuration (a program written in a proprietary language) to the Controller through the communication interface. The Controller is then free to directly control the process. It communicates with I/O modules over a proprietary module bus, sending the required action commands, monitoring module status and I/O conditions and adjusting values as needed to maintain the process. The Controller reports any errors directly to the Operator Control Station and takes any corrective actions allowed within its' programming. The human Operator can then use the Operator Control Station to take appropriate action.

AI Module

The AI module is a highly precise instrument capable of reading RTD (*Resistance Temperature Detector* uses resistance values to represent temperature values), millivolts and Volts on each of its' sixteen (16) independent input channels. In order to achieve the high precision required for this module each channel must be calibrated in each operational mode. A calibration step requires supplying the precise value to each input and sending a calibration command through the Controller to lock the readings into a table in a non-volatile memory device located on the AI module. The sixteen channels can be calibrated either simultaneously or individually. The equipment used to perform this calibration, and also perform a variety of required functional tests, prior to shipping to a customer, is called an AI Test System.

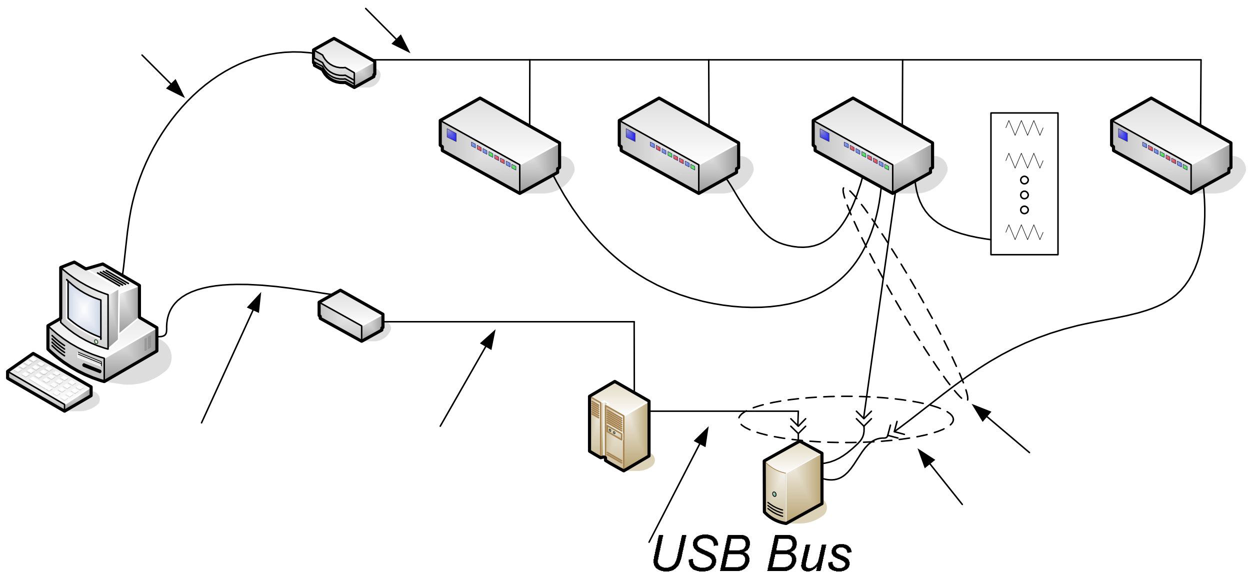

Figure 4 - AI Test System

AI Test System

The AI Test System (see Figure 4) consists of a PC, a USB/GPIB interface (this device converts a standard PC Universal Serial Bus (USB) into an industry standard General Purpose Instrument Bus (GPIB)), four programmable GPIB based instruments, serial communication interface (proprietary device), a custom precision resistor board (containing the resistors required for calibration and testing), controller, a subject AI module and a connection adapter (to allow quick connect/disconnect of the subject AI module to the test system). The GPIB devices include a digital multimeter (DMM), power supply (this power supply is programmable because it must be varied during functional testing), voltage source (with ranges of millivolts and Volts), and a switch matrix (containing dozens of relay switches). The DMM, voltage source, precision resistor board and subject AI module are connected into the switch matrix. The power supply is connected to the AI module to supply operational power and the controller communicates to the AI module over the module bus.

The test software controls the inputs to the subject AI module through the switch matrix. Selected switches are closed to connect the desired input to the chosen input channel of the AI Module. The AI Module generates a value for this input. A calibration command, containing the actual value of this input (as read by a DMM), is then sent through the Controller to the AI Module. The AI Module compares the value it generated to the value contained within the calibration command and saves the offset value (the difference between the two values) along with the generated value. This process is repeated for a total of three input values (low, medium and high) across the allowed range and for all 16 channels. The module uses these values to compute an offset curve for each channel. During real world operation the AI module will report values modified by this curve to the Controller.

Review of Deliverables

The deliverables for each product phase are presented and discusses for the remainder of this chapter.

Project Vision Statement

The primary purpose of the software developed for this project was to provide the Operator with the ability to test and calibrate an Analog Input Module, and verify this calibration, with minimal Operator intervention. Secondary features included module troubleshooting tools, automated test system calibration, test system troubleshooting tools and report printing.

Examples of such secondary features are: (1) the ability to manually set input values to each channel and read the module's input response, (2) perform each functional test separately, (3) print module calibration report, (4) test communications with each system device.

Operators interact with the system through graphical user interfaces compatible with Microsoft's Windows XP operating system. Menu options available are based on the allowed access of the Operator. Denied options are grayed out in the affected menus.

Primary Actors

One of the first steps in the development of Use Cases is the identification of the Primary Actors. These are the people who interact with the system. For the Analog Module Calibration/Test System there are four categories of actors, the higher level users also have the ability to perform all lower level functions:

1) *Supplier Tester:* This is the person who performing production testing. The Supplier Tester performs only the automated test and calibration with little technical knowledge of the product.

2) *Supplier Technician:* This is the person who performs troubleshooting and repairs of any modules that fail during the automated test and calibration process. The Supplier Technician has a higher level of technical training and requires the ability to use the test system to set individual input values, verify AI module readings to set GPIB device parameters individually.

3) *Supplier Test Engineer:* This is the person who performs troubleshooting and calibration of the test system. The Supplier Test Engineer has a high level of technical training and needs to have the ability to control all devices individually.

4) *Owner Test Engineer:* This is the person who performs system modifications and upgrades of the test system. The Owner Test Engineer has overall responsibility for the test system.

5) *Operator:* This term is used as a generic descriptor for all types of users.

Use Cases

One of the most important deliverables of the Inception Phase is the identification of the basic Use Cases in a Use Case Diagram. These diagrams provide an easily understood, albeit generalized, look at the high level requirements for this software. For this project several Use Cases were initially identified and documented in a Use Case Diagram (see Figure 5). From these Use Cases only two were identified as “key” to the project. Selected Use Cases are “Module Calibration” (the calibration and testing of the subject module) and “System Test & Calibration” (the calibration and testing of the test system – See Appendix A). Basic requirements for these “key” Use Cases are described over the next few pages.

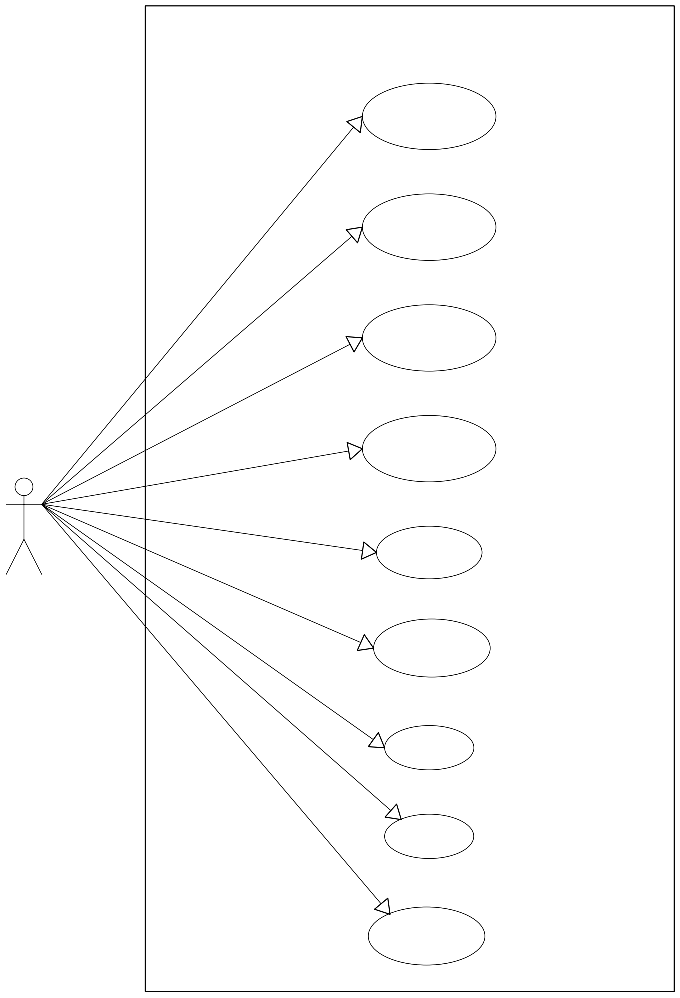

Figure 5 - Basic Use Cases

Use Case Description – Module Calibration

The Use Case descriptions at this phase are general. Greater detail is provided during the Elaboration phase.

The basic requirements, as defined by the hardware design group, include several defined actions and tests. What is left open to the test developer is how to implement these requirements. The full suite of required actions for the AI module must include:

- Calibration of the Cold Junction Resistor (CJR). The CJR is a component on the AI module used to compensate for temperature errors when using a thermocouple (a temperature measurement device commonly used in industrial processes) to measure temperature. The CJR must be calibrated to achieve the greatest possible accuracy for temperature measurement.
- Verification of the CJR calibration
- Calibration of all channels in RTD mode
- Verification of the RTD calibration
- Calibration of all channels in Voltage mode
- Verification of the Voltage calibration
- Calibration of all channels in millivolts mode
- Verification of the millivolts calibration
- Open Thermocouple Test
- 5 Volt Variation Test

Other requirements include:

- System start-up: Verifying operation of the test system components

- System shutdown: Set all equipment to default states and terminate communications.
- User Sign-on: Allow operator to enter name and password and be given access rights for allowed functions.
- User Sign-off: Terminates access to the test system.
- Print Product Calibration Report: Print a report for a specific product serial number.

Use Case Description – System Test & Calibration

The basic system test and calibration use cases are developed from the requirement specified by the test system hardware designer. These requirements include:

- System Calibration: Measure and storage of all resistor values used for product calibration of CJR and RTD.
- Verify proper operation of all Test & Measurement instrumentation by setting all input types and verifying these values.

Supplementary Specifications

In addition to the basic functional requirements described in the use cases other requirements include:

1) Software needs to be installable from both a CD and from a network drive.
2) User interfaces shall use Microsoft Windows XP compatible windows and menu structures.
3) Software Quality Factors:
    a. Maintainability – The software must be easy to repair or modify.
    b. Expandability – The software must provide an easy method to add functionality at a future time.

c. Survivability – In the event of a failure of the AI module or test system, the software must place the system into a default state and allow operation of the system from this point.

d. Usability - The software must be easy to install and use.

Expanded Use Cases

Expanded Use Cases are a deliverable identified with the Elaboration Phase. The primary purpose of this project was the calibration of an Analog Input Module and the verification of that calibration. The specifics of how to apply the inputs for each input mode are slightly different but the basic ideas are the same. Therefore, only one input mode will be used as example of the use case description.

The expanded use case diagram shown in Figure 6 breaks the “Calibrate Module” use case into the various activities required to accomplish this primary task. A use case description follows this diagram and gives a full description of the steps involved. These descriptions provide a major piece of the requirements detail required by a programmer to develop the software.

The CJR/RTD input mode will be used as an example for this paper. This mode was selected because it is the most complex of the AI calibration modes, requiring the switching in and out of 85 different resistors.

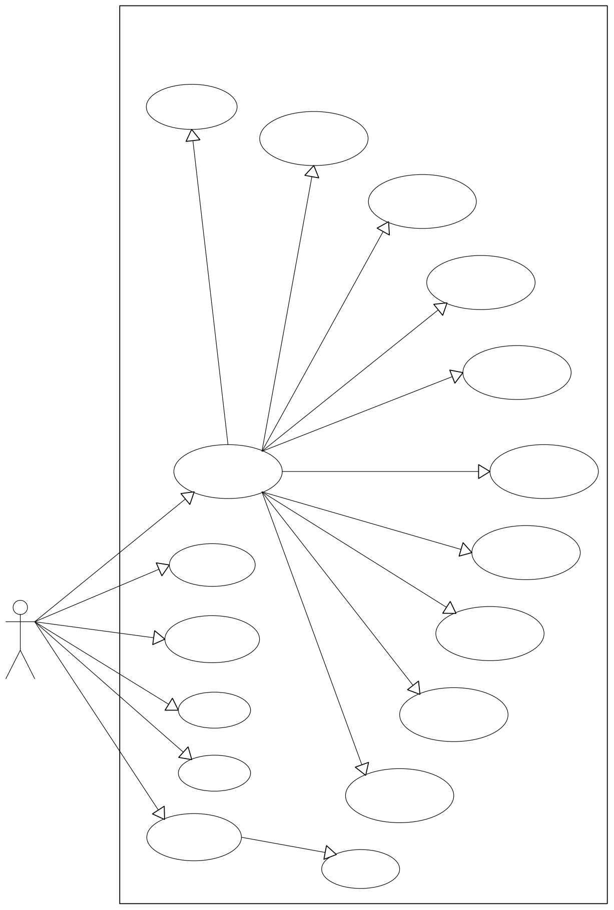

Figure 6 - Factory Calibration Use Case Diagram

Use Case Descriptions – Example.

Use Case UC1: Calibrate Module

Primary Actor: Supplier Tester

Trigger Condition: Selection of activity by Supplier Tester

Preconditions: Logon by Tester

Post conditions: Test System instrumentation returned to default conditions.

Primary Flow:

1) Calibrate Channel 0 in CJR Mode.
2) Calibrate Channel 1 – 16 in RTD Mode.
3) Open Thermocouple Test.
4) Calibrate Channel 1 – 16 in mV Mode.
5) 5V Variation Test.
6) Calibrate Channel 1 – 16 in Voltage Mode.
7) Display Pass/Fail Message

Alternate Flows:

None. Failure flows are handled by the specific test or calibration mode use cases.

Notes:

- For each input mode a module must have each channel calibrated at three input values and verified at two other values.
- All calibrations and tests performed with prompting of Operator to make appropriate jumper and switch settings.

Class Diagram

The Class Diagram shows the main classes needed for the system and include the attributes and operations required to achieve the necessary functionality. Each class represents an object within the test system. Class diagrams provide the basis for the creation of object-oriented code. A description of the various classes follows the diagram.

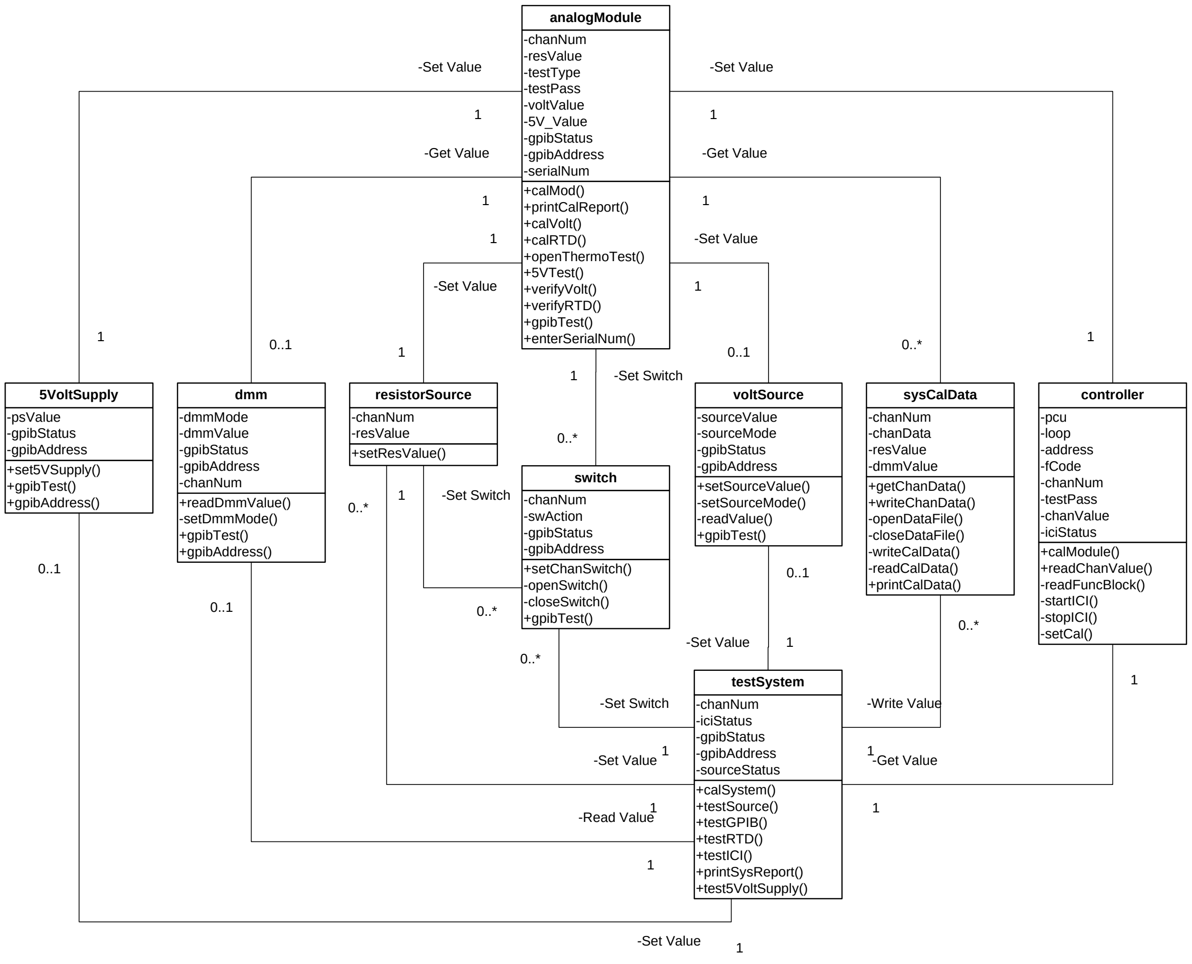

Figure 7 - Class Diagram

The main classes and the objects they represent are:

analogModule – The AI Module itself.

5VoltSupply – The programmable 5 Volt Power Supply used to power the AI Module.

dmm – The Digital Multimeter.

resistorSource – The resistors used to provide inputs to the AI Module.

switch – The Switch Matrix used to connect inputs to the AI Module.

voltSource – The voltage source used to provide inputs to the AI Module.

controller – The Controller device used to communicate with the AI Module.

testSystem – Contains the basic operations to calibrate and test the Test System itself.

SysCalData – The data base file containing the Test System calibration data.

Sequence Diagram

Sequence diagrams are a dynamic modeling technique. They provide a visual model of the logic flow through the system. The Sequence Diagram shown in Figure 4 is one of several developed for this project. For this example the flow of logic shows a complete pass through the Use Case for calibration of the AI module in CJR/RTD mode. Various classes and class operations are captured on this diagram.

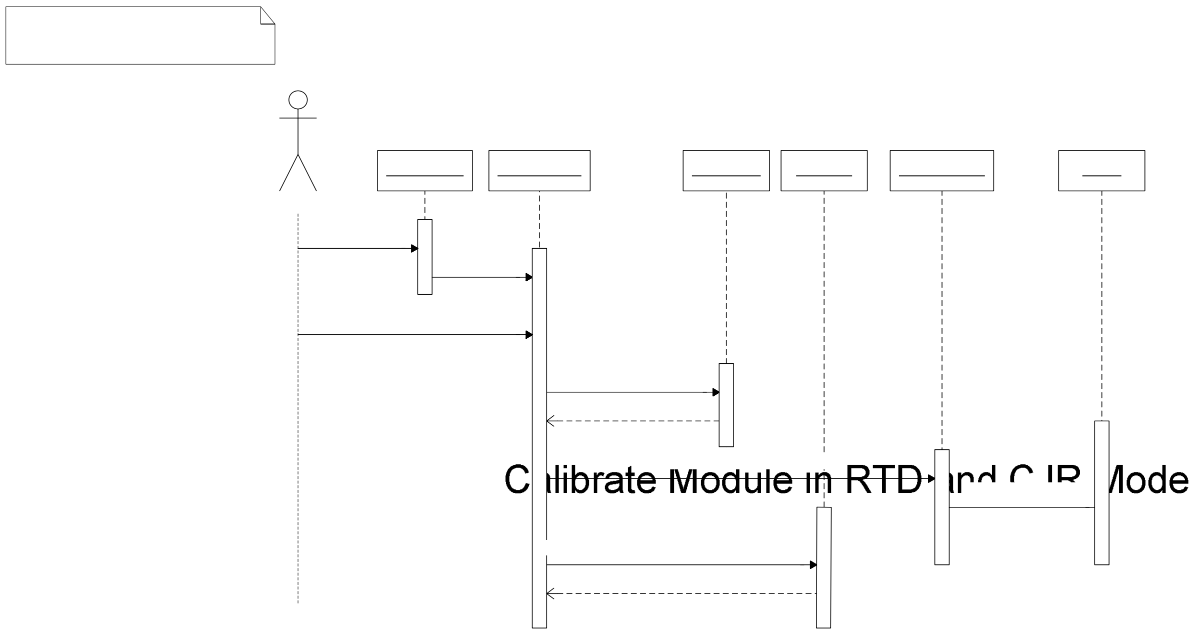

Figure 8 - Calibration Sequence Diagram

Test Plan

For this project, testing consists of both Whitebox and Blackbox tests (See Appendix B for the Software Test Plan). The consequences of a malfunction of the software are small. The user is not placed in physical danger in event of a failure and damage to property is slight. A system crash would be an inconvenience and would cause some production delays but is easily recoverable. Therefore the testing focuses only on the main actions and did not try to be comprehensive. Whitebox testing was limited to unit testing to verify proper operation of selected classes. Blackbox testing was used verify proper presentation of all allowed inputs to the AI module, the proper response by the AI module and the ability to communicate with all hardware devices. All testing was designed for manual operation. These test selections are part of the adaptations to the development process to achieve a quality product with limited resources.

Operat

re

1. Request to calibrate module in RTD and/or CJR mode.

2. Enter channels to be calibrated.

3. System retrieves channel calibration data.

4. System sets channel inputs.

Blackbox Testing.

- Validate communication/control for each GPIB device.
- Validate communications/control of the Controller
- Validate control of AI module (through the Controller)
- Validate correct settings of each GPIB device at each AI module calibration point.
- Validate the operation of the AI module "calibrate" command.
- Validate correct settings of each GPIB device at each Test System calibration point.
- Validate correct file creation of Test System calibration data file.
- Validate correct opening and reading of Test System calibration data file.

Whitebox Testing.

- Validate operation of "AI calibrate" class (Class contains all commands required to cause AI to save data).
- Validate operation of "Test System Calibration Data" class (Class contains all commands for opening & closing data file, reading & writing data)
- Validate operation of "Controller communication" class. (Class contains all commands to read and write data to Controller).
- Validate operation of all three "GPIB" classes. (Each class contains all commands for operation of the specific GPIB instrument.)

Items out of the scope of the project.

- Operation with different programmable test & measurement instruments.
- Validation of hardware system (although the act of software testing will validate some hardware functions by default).

Pass/Fail Criteria

- A test was considered to pass only if all expected results (as specified in the test case) are observed. All other observed results would be considered failures.

Project Schedule

The project schedule is shown below, in Gantt chart form. This schedule identifies the four phase and major tasks involved in completing the software project. This schedule assumed the availability of the hardware component by the start of the Construction phase.

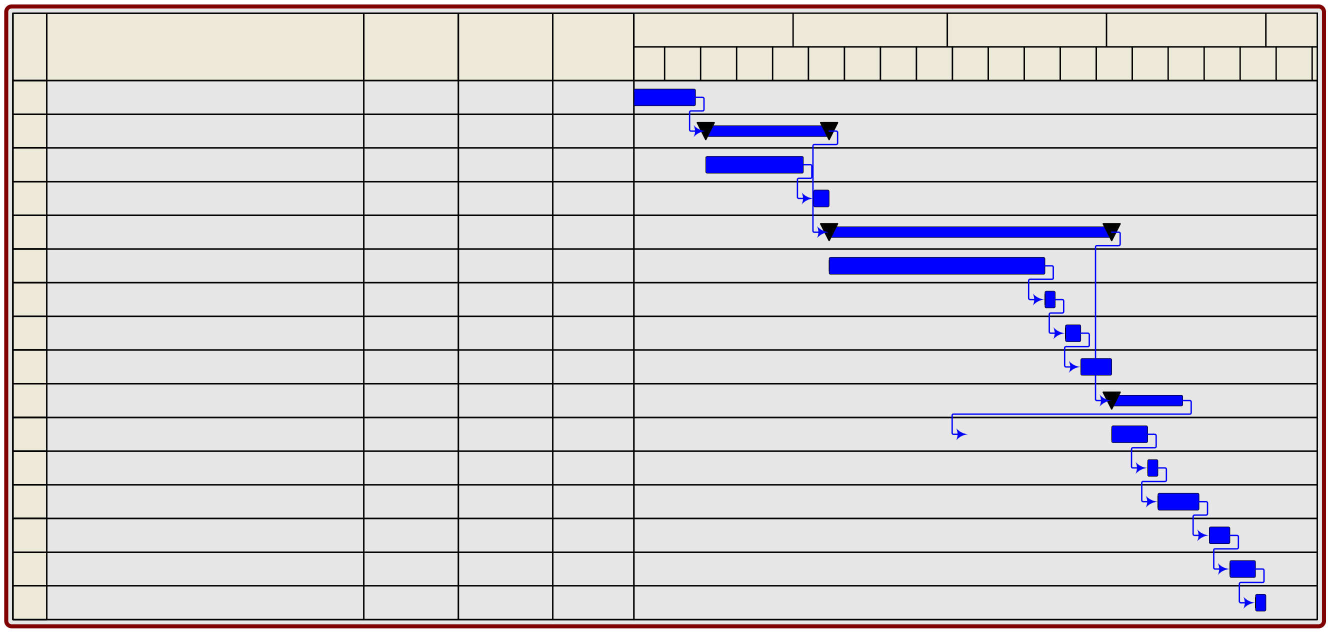

Figure 9 - Project Schedule

Summary

The Unified Process Model was the methodology selected for this project. It fit the requirements of being a flexible and scalable methodology while incorporating the use of UML tools. These characteristics fit the needs for this small development project.

All project deliverables (UML diagrams, project schedules and test plans and other documentation) were selected to provide a simple and solid design, traceable to the product requirements without excessive paperwork. The examples included here show the type of documents appropriate for a small project. These selections provide the framework for the author to develop the software without providing the detail that would have been necessary if the coding was to be done by a group of programmers. However, these documents provide sufficient linkage for a Quality Inspector to verify that all requirements are being addressed, an important aspect of software quality.

Developing a test plan is another important aspect to the delivery of a high quality product. Different types of testing were selected to balance the needs for quality against the time allowed and the resources available.

Chapter 4: Project History

Background

During a manufacturing outsourcing project a weakness was uncovered in the manufacturing/test process for an Analog Input (AI) module. The existing equipment was deficient in many respects. After having seen continuous use for nearly 10 years the relays, interface connectors and wiring were worn and fragile. The computer was a 486 based PC running Windows 3.1 and the GPIB interface card was an ISA standard. All of these hardware/software components are considered obsolete – none of them is still supported by the manufacturers. The automated switches were custom designed and all other instrumentation was obsolete. The test software was structured programming written in C using obsolete function libraries. Although still working, a failure of any component in the system could result in long production delays of several weeks before repairs could be made. Further problems arose from a lack of available tools to troubleshoot failed products and for the test and calibration of the test system. This created training problems with the new operators and technicians.

The author proposed a complete redesign of the test/calibration system using modern hardware and software. The new design would include automated and semi-automated troubleshooting tools for both the product and the test system. Some additional tests, previously done on separate test systems would be included in the new system and would be performed automatically. Tentative management approval was given to begin the design work in April 2006.

Project Management

One Test Engineer (the author) was assigned to perform all aspects of the project. The Organization follows a product development process that is designed to achieve a high quality level for all products, with a heavy emphasis on software development processes. Although not normally applied to internal test equipment projects it was felt that this test system was of sufficient importance to warrant a close adherence to these processes. However, these processes were designed to support large scale projects with 10 or more programmers, multiple testers, system architect, quality assurance engineers, project manager, product manager, development manager and potentially two or more development sites. The amount of documentation developed is considerable and inspections were called for at several points in the process. Clearly this was not an acceptable condition for a one person project. The process requirements were adjusted throughout the Inception and Elaboration Phases to reach a balance between product quality and the resources available.

Significant Events

A complete product requirements specification is a key to a successful development process. The two phases completed for this project, Inception and Elaboration, both addressed the clear identification of requirements. As part of the Inception Phase, basic research was performed as described in Chapter 2, to define the basic requirements of the system.

During this initial investigation it first appeared that the requirements were very complete. The designers of the AI module established very clear specifications and procedures required to perform the necessary product calibrations. To a lesser degree the test system hardware requirements were identified. These requirements were much more general than those provided by AI module designers. However, they were clear in the types of activities that needed to be done. Left open was how they should be implemented. Much of this was dependent on technical information contained within the operator/user manuals for the GPIB instrumentation and the schematic diagrams for the complete test/calibration system. In addition to these technical requirements the users' needs also had to be addressed.

The discovery of user needs was done by the author writing a list of desirable functions. Discussions with the primary users, the test floor production manager and two test technicians, were held to review this list and identify other possible functions and characteristics. A basic use case diagram was created and reviewed again. Modifications were made, the full set of Inception phase deliverables was completed and the project moved to the Elaboration phase.

The Elaboration phase entailed a more detailed look at all of the requirements and a deeper investigation into the technical specifications. The deliverables for the Elaboration phase were created and reviewed with the primary users and with the hardware designers. Each document went through either two or three separate iterations. Development of the screen snapshots, expanded Use Cases, Use Case descriptions and Sequence Diagrams revealed flaws in the original Use Cases and resulted in several changes. This was also true during the development of the Class Diagrams. Developing the Class Diagrams led to redefining and combining Classes. This, in turn, required reworking the Sequence Diagrams. The end result was a more cohesive model.

Hardware design was then begun and a full set of drawings and parts lists generated. A capital justification to build two new test systems was written and submitted for management approval. However, Management chose to reject the proposal for two systems. After additional discussion with various executive stakeholders it was agreed to resubmit the request for one test system. This was done and approval is still in limbo as of September 1, 2006. For budget purposes this project is not expected to be formally approved until January 2007.

After completion of the Elaboration Phase the project was placed “on hold” pending final approval. Coding of certain classes could begin but critical C++ libraries and functions supplied with the test and measurement equipment will not be available until equipment is ordered and delivered. The management decision is not to proceed with the coding until final project approval is received and hardware is available. For the purposes of this project paper the project was redefined to cover only the Inception and Elaboration Phases. This change was justified by the Elaboration Phase being the design phase. The Construction and Transition Phases are where the actual product is created from the design documents, but the Elaboration Phase is where the requirements are fully developed and the design documents created.

Evaluation of Project

The project goals to complete all deliverables identified for the Inception and Elaborations Phases were successfully met. These deliverables were selected for their ability to maximize the amount of information in the most compact form. A review of the project deliverables required for each phase shows the following results.

Deliverable Status

Inception Phase:

• Vision Statement	Complete
• Use Case Diagram	Complete
• Supplementary Specifications	Complete
• Identification of Primary Actors	Complete
• Preliminary Project Schedule	Complete

Elaboration Phase:

• Expanded Use Cases.	Complete
• Sequence Diagrams	Complete
• Class Diagram	Complete
• Test Plan	Complete

• Revised Project Schedule	Complete
• Screen Snapshots	Complete

The project goal of adjusting the development process to achieve a quality product was successful to the point covered by this paper. Final results will not be known until the AI Test System is operational in a factory test environment. What can be said is the deliverables created provide an easily verified path to the product requirements. The quality of these product requirements are verified through the iterative methodology of the Unified Process Model. (This process model is basically self inspecting since the requirements go through several reviews during the project.).

Problems

Clearly it was disappointing to have to redefine the project when the hardware was delayed, but the design phase was unaffected. For the Inception and Elaboration phases of the project no significant problems were encountered. The changes to the requirement and design documentation were part of the iterative nature of the Unified Process Model methodology. Still there were process areas that should have been addressed more fully and process steps taken that could have improved the quality of the end product. These will be discussed in Chapter 5.

Summary

The original project was the complete hardware, software and construction of an AI Test System. For the purposes of this Professional Project the focus was limited to the development of the software. An unfortunate management decision to place the hardware portion of the project on hold resulted in a redefinition for this Professional Project to include only the design of the software. In terms of the selected methodology, the Unified Process Model, the project was limited to the Inception and Elaboration Phases. These phases cover the requirements definition through to the design documents, project schedules and test plans.

Further impacting this project was the need to cover the process steps required by the organization as part of its’ quality assurance program. The design and test documents were all developed fulfill the process requirements.

Chapter 5: Conclusion

Lessons Learned

This project tied together the use of several tools and processes; Unified Process Model, the use of UML tools, development of a test plan and following a development process designed to improve product quality. All of these different areas needed to be scaled down to a one person development project without negatively impacting the quality of the product.

The iterative process followed by the Unified Process Model provides a formal mechanism for modifying and refining the design. In this project, the actions involved in modeling the system provided new insights into the requirements and how to improve the design. The design is substantially improved from the first iteration.

The core of a successful design is the establishment of clear and complete requirements. Vague requirements can result in the creation of software that does not pass acceptance testing. Correcting the problems at that point can result in long project delays and high project costs (Roberts, 2006). Using UML modeling techniques to develop Use Case diagrams helped to clarify the requirements. The act of developing the Use Cases and discussing them with stakeholders was a great help to clarify the requirements for both the stakeholders and the designer. The iterative approach used by the Unified Process provided the mechanism for easily reviewing and changing the requirements as the project evolved.

This project also showed that at some point the requirements have to be frozen. Feature creep, even on a small project like this one, can cripple a project. Another point is that UML diagrams should not be used to attempt to detail every piece of the design. The Unified Process Model promotes the use of UML for only the most important parts of the design and the author concurs. Modeling becomes a case of diminishing returns between the efforts required to document details versus just getting the code written. Where the breakdown point is will differ for each project. For a project like this, the UML documentation covers only the major activities and leaves the minor Use Cases underdeveloped. This was justified for several reasons:

1. The project was very focused on a one-of-a-kind, proprietary, test system.
2. In terms of software development this was a fairly small project.
3. The developer and programmer were the same person, with several years experience developing similar test equipment and software.

A different set of circumstances for this project might have resulted in the need for more UML documentation and detail. For example; if the developer and programmer were separate people and the programmer was inexperienced in writing test software. In such a situation more expanded Use Cases for some of the more minor Use Cases would be an advantage.

Developing a test plan early in the project was an important step. Not only does an early test plan provide better scheduling data but it also helps to identify product requirements. For this project, knowing that unit testing would take place, some Class functions were made public that would have been made private in order to allow easier testing.

Test plans are living documents during the project and are subject to the same iterative process as all other project deliverables. The initial test plan identifies the main test objectives, test process, scheduling estimates and high level deliverables. As the project progresses the test plan should be revisited and revised.

Status and Improvements

Although the project was successful at establishing a complete design, there were some project activities that could have been done better. The development process should have been more closely examined at the beginning of the project. The documentation and process steps that would be required should have been identified early instead of examining and selecting documents and procedures throughout the phases.

An area that was not directly covered by the project plan was the development of an actual Software Quality Assurance Plan (SQAP). An SQAP is part of the normal development process but was left out of this project because it was felt to be unneeded due to the size and relative simplicity of the project. Some aspects of Software Quality Assurance were indirectly addressed by the iterative nature of the Unified Process Model methodology and the Test Plan. In the case of this project the documents were generally short and easily reviewed. No formal, documented, reviews were performed. Instead, for the two phases described in this paper, informal discussions by the author, test technicians and hardware designers were performed as part of the iterative nature of the Unified Process Model. These essentially resulted in a review of the requirements and design documentation by the major stakeholders. However, having a written SQAP would allow for a more formal review of the project and give better assurance that no important activities were missed. A formal SQA Plan should be written for all projects. An SQAP would include the identification of the work products to be produced, the type of review to be performed and a schedule of these reviews.

Another area not addressed by the Project plan was Configuration Management (CM). Configuration Management for a large project requires a formal Configuration Management plan. This helps to assure that the final system configuration is complete and used the correct revisions of each component. In this case the department’s normal operation is to compress all the component files into a zip format and store in a secure documentation system. This was not directly addressed for several reasons: (1) The small size of the project, (2) Only one programmer was involved (3) Only a few separate components are required and (4) normal department protocol already covered the requirement. Although it was felt that the needs of configuration management were met, a formal CM document describing these activities would provide evidence that the subject was addressed.

Next Steps

The next phases to be performed are the Construction and Transition phases. The next steps in the Construction phase would be to complete the hardware assembly and then move to coding. However, Management chose to reject the proposal for two systems. After additional discussion with various executive stakeholders it was agreed to resubmit the request for one test system. This was done and approval is still in limbo as of September 1, 2006. For budget purposes this project is not expected to be formally approved until January 2007.

At this point the project is “on hold” pending final approval. Coding of certain classes could begin now but critical C++ libraries and functions supplied with the test and measurement equipment will not be available until equipment is ordered and delivered. The management decision at this point is to not proceed with the coding until final project approval is received and hardware is available.

The Construction phase will require the primary tasks of coding and testing (unit and integration only). Also included in this phase is the development of preliminary operating instructions and test reports. For this project the operating instructions must cover the installation of the software, the installation of the hardware, the care and operation of the hardware, description of the operation of the software, and the steps required to perform the full test and calibration of the AI module.

The tasks performed during the Integration phase are the finalization of the user documentation and the performance of a full system test. This testing will be based on the requirements documents and the user documentation. Two or three iterations will probably need to be performed as errors are discovered and corrected. The deliverables from this phase will be the final releases of the software, user documentations and system test report.

Looking ahead to the Construction and Transition phases the scheduling of a code review of key Classes would be a good idea. In the same way a review of the test plan and test cases would help to improve the quality of the software. Additionally, a formal audit of the test reports would help to assure the acceptable quality of the test system. The review of the operating instruction (user manual) by a third party, with formal document approval is required by the Organization's standard practices.

Summary

All software designers want to deliver high quality products. The quality achieved has been shown to be dependent on good design processes. These processes frequently carry with them the burden of documentation, quality inspections and testing. There is a price to implementing these processes. The labor cost and the scheduling impact. A development process aimed at a large project encompassing several developers can absorb this overhead and see substantial benefits.

A small project consisting of only one or two developers would be severely impacted by the quality overhead in both project cost and schedule delays. Even then the benefits may be negligible. Small projects need to have the processes tailored to them. Documentation requirements should be reduced and process steps scaled back or eliminated. However, this scaling back needs to be done with the quality goals maintained.

This project is an example of a small software development project. The key point is that project tasks and documentation can and will vary with the size of the project. All areas of a project, such as methodology, schedule, quality assurance, configuration management, etc., need to be considered. However, how they are addressed is dependent on the size of the project. A large project, with many people involved, will require formal documentation for all aspects of the project. Without this documentation and procedures the quality of the software product will fall short and the project will probably suffer delays. Smaller projects, such as this one, with only one member would suffer greater delays in the development of the full, detailed documentation required for a large project. However, random cutting of documentation requirements or process steps would lead to an unacceptable product. To achieve acceptable quality all areas of the development process needs to be address regardless of the size of the project. The differences lie primarily in the documentation choices and the detail presented.

Works Cited

National Instruments Inc. Retrieved August 5, 2006, from National Instruments Website: http://www.ni.com

Agilent Technologies Inc. Retrieved August 5, 2006, from Agilent Technologies Website: http://www.agilent.com

What is CMMI®?. Retrieved August 15, 2006, from Carnegie Mellon University, Software Engineering Institute Website http://www.sei.cmu.edu/cmmi/general/general.html

Ambler, Scott W.. *UML 2 Class Diagrams*. Retrieved September 11, 2006, from Ambysoft Inc. Website http://www.agilemodeling.com/artifacts/classDiagram.htm

Ambler, Scott W.. *UML 2 Sequence Diagrams*. Retrieved September 11, 2006, from Ambysoft Inc. Website http://www.agilemodeling.com/artifacts/sequenceDiagram.htm

Ambler, Scott W. (2005). *A Manager's Introduction to The Rational Unified Process (RUP).* Retrieved October 6, 2006, from Ambysoft Inc. Website http://www.ambysoft.com/unifiedprocess/rupIntroduction.html

Bolton, Michael. (2006). *The Factors of Functional Testing*. Retrieved October 6, 2006, from StickyMinds.com. Website http://www.stickyminds.com/pop_print.asp?ObjectId=11006&ObjectType=ART

Brugge, Bernd. (2002). *The Unified Process*. Retrieved July 6, 2006, from Techische Universitat Munchen. Website http://wwwbruegge.in.tum.de/teaching/ss02/SE/07UnifiedProcess.pdf

Burnstein, Illene. (2003). *Practical Software Testing.* New York: Springer-Verlag.

Christensen, Mark J., Thayer, Richard H. (2001). *The Project Manager's Guide to Software Engineering's Best Practices.* The Institute of Electrical and Electronics Engineers, Inc.

Deitel, H. M., Deitel, P.J. (2003). *Java How to Program*. (5th ed.). New Jersey: Prentice Hall.

Foote, Bill. (1996). *Java Tip 17: Integrating Java with C++*. Retrieved November 9, 2006 from Java World. Website http://www.javaworld.com/cgi-bin/mailto/x_java.cgi

Glass, Robert L. (2001). *Revisiting the Definition of Software Quality*. Retrieved November 9, 2006 from Stickyminds.com. Website http://www.stickyminds.com

Humphrey, Watts S. (1989). *Managing the Software Process*. Addison-Wesley Publishing Company Inc.

Kasse, Tim (2004). *Practical Insight into CMMI®*. Artech House.

Martin, Robert C. (1997). *Java and C++ A critical comparison*. Retrieved November 10, 2006, from Object Mentor Inc. Website http://www.objectmentor.com/resources/articles/javacpp.pdf

Palmer, James D. (1997). *Traceability*. *Software Engineering*, M. Dorfman and R.H. Thayer, eds., IEEE Computer Society Press, Los Alamitos, Calif.

Parekh, Nilesh (2005). *Software Testing – White Box Testing Strategy*. Retrieved October 6, 2006, from Buzzle.com. Website http://www.buzzle.com/editorials/4-10-2005-68350.asp

Patton, Jeff (2006). Test Software Before You Code. Retrieved October 6, 2006, from StickyMinds.com. Website://www.stickyminds.com/pop_print.asp?ObjectID=11104&ObjectType=COL

Richter, Charles (1999). *Designing Flexible Object-Oriented Systems with UML*. Macmillan Technical Publishing.

Roberts, Clare (2006). *Do Your IT Projects Suffer from Requirements Clarity Issues?*. Retrieved October 6, 2006, from StickyMinds.com. Website http://www.stickyminds.com/pop_print.asp?ObjectId=11392&ObjectType=ART

Russell, Richard G. (2004). *Defining Software Quality – An Essay*. Retrieved November 9, 2006, from IO.com. Website http://www.io.com/~richardr

Schach, Stephen R. (2002). Object-*Oriented and Classical Software Engineering* (5th ed.). McGraw-Hill.

Stevens, Perdita & Pooley, Rob (2000). *Using UML.* Pearson Education Limited

Thayer, Richard H. & Dorfman, Merlin (2002). *Software Engineering Volume 1: The Development Process* (2nd ed.) The Institute of Electrical and Electronic Engineers, Inc.

Appendix A – Use Case

Use Case UC3: Calibrate CJR/RTD Mode

Primary Actor: Supplier Test Technician

Trigger Condition:
1) Calibrate Module use case
2) Selection from menu by Supplier Test Technician

Preconditions: Logon by Operator

Post conditions: Test System instrumentation set to default conditions.

Primary Flow:

1) Prompt Tester to set jumpers and address switches on module.
2) Prompt Tester to insert in RTD test slot of test system.
3) Set Input Resistor Values by activating relays in GPIB switch.
   a. For sixteen (16) channel calibration: Set input resistor value to first calibration value for each channel. Read average resistor value from Calibration Data File.
   b. For individual channel calibration: Read actual resistor values for each calibration resistor from Calibration Data File.
4) Send calibration command to Controller.
5) Read calibration confirmation message from Controller.
6) Repeat steps 3 through 5 for the remaining two calibration values.
7) Verify CJR/RTD Mode Calibration.
8) Recalibrate any out of tolerance channels.

Alternate Flow:

1) Calibration failure message from Controller (first time occurrence).
   a. Re-send calibration command.
2) Calibration failure message from Controller (second time occurrence)

a. Display error message.

b. Return to main menu.

3) Channel out of tolerance (After two calibration attempts)

a. Display error message.

b. Return to main menu.

Appendix B – Software Test Plan

logo

Software Test Plan

Number:	TP-001
Revision:	A
Date:	24 February 2006

Test Plan for Analog Input Module Calibration Test System Software

---

Table of Contents:

1.0 Subject Software

Software ID No.	*Software Title*	*Revision Level*	*Author*
PS-MTCA-0030	Analog Input Module Calibration	A	C.N. Rogers

2.0 Reference Documents

Document No:	*Document Title*	*Revision Level*	*Revision Date*
XXXUI240774A2	Analog Input Module Product Instruction Manual	A2	11 March 2005
XXX0382.007.50	Analog Input Module Calibration Engineering Instruction	0	15 Sep 1997
XXXI350254C0	API Developer Manual	2.1	1999
XXXI350254C0	API Developer Interface User Manual	2.1	1998
370428C-01	National Instruments NI 488.2 User Manual	2.0	Feb 2005
370430C-01	National Instruments NI 488.2 Software Reference Manual	2.0	Feb 2005

3.0 Document Control

Rev.	*Date*	*Revised By*	*Approved By*	*Change Description*
A	24 Feb 2006	C.N. Rogers		New Release

4.0 Introduction

The Analog Input (AI) Module is a component of a Distributed Process Control System. It is one of a family of I/O modules providing a variety of input and output capabilities to the system. A basic system requires a communication interface, Control module and several different types of I/O modules.

Simply put, an Operator Control Station (PC or Workstation, running proprietary control applications), sends a custom configuration (a program written in a proprietary language) to the Controller through the communication interface. The Controller is then free to directly control the process. It communicates with I/O modules sending the required actions and monitors their status. The Controller reports any errors directly to the Operator Control Station and takes any corrective actions allowed within its' programming. The Operator can then use the Operator Control Station to take appropriate action.

The AI module is a highly precise instrument capable of reading RTD (resistance values representing temperature values), milliVolts and Volts on each of its' sixteen (16) independent input channels. In order to achieve the high precision required for this module each channel must be calibrated in each operational mode. A calibration step requires supplying the precise value to each input and sending a calibration command through the Controller to lock the readings into a table in a non-volatile memory device located on the AI module. The sixteen channels can be calibrated either simultaneously or individually.

On a macro level the test system consists of a PC containing a GPIB card, three GPIB based instruments, serial communication interface (proprietary device), Controller and a subject AI module. The GPIB devices include a Digital MultiMeter, Programmable Voltage Source (with ranges of millivolts and Volts), and a programmable switch matrix (precision resistors will be connected within the switch to provide RTD calibration points). Communications and programming of the Controller requires a library of proprietary classes.

This project includes the hardware and software design to perform these calibrations and to calibrate the test system itself. For the purposes of these tests it will be assumed the hardware is operational. Only the software tests are included in this test plan.

5.0 Test Description

Testing will consist of both Whitebox and Blackbox tests. All testing will be performed manually. Whitebox testing will be used to verify proper operation of selected classes. Blackbox testing will be used verify proper presentation of all allowed inputs to the Analog Input module, the proper response by the AI module and the ability to communicate with all hardware devices. Documentation control and defects reporting will use networked tools described in section 11.0 entitled “Support Tools”.

<u>Blackbox Testing:</u>

- Validate communication/control for each GPIB device.
- Validate communications/control of the Controller
- Validate control of AI module (through the Controller)
- Validate correct settings of each GPIB device at each AI module calibration point.

- Validate the operation of the AI module “calibrate” command.
- Validate correct settings of each GPIB device at each Test System calibration point.
- Validate correct file creation of Test System calibration data file.
- Validate correct opening and reading of Test System calibration data file.

Whitebox Testing:

- Validate operation of “AI calibrate” class (Class contains all commands required to cause AI to save data).
- Validate operation of “Test System Calibration Data” class (Class contains all commands for opening & closing data file, reading & writing data)
- Validate operation of “Controller communication” class. (Class contains all commands to read and write data to Controller).
- Validate operation of all three “GPIB” classes. (Each class contains all commands for operation of the specific GPIB instrument.)

Items out of the scope of the project:

- Operation with different GPIB instruments.
- Validation of hardware system (although the act of software testing will validate some hardware functions by default).

6.0 Test Environment

6.1 Analog Input Module Calibration System

6.2 PC

6.2.1 Windows XP SP 2.0

6.2.2 Visual Studios C++ Ver. 6.0

6.2.3 National Instruments GPIB Card, Model ####

6.2.4 National Instruments GBIP Software, Model ####, Ver. ###

6.2.5 xAPI Runtime Application (Internal, proprietary)

6.2.6 xAPI Developer (Internal, proprietary)

6.3 Analog Input Module

7.0 Pass/Fail Criteria

7.1 A test is considered to pass only if all expected results (as specified in the test case) are observed. All other observed results will be considered failures.

7.2 All failures will be documented in the defect tracking tool (see Section 11.0 Support Tools).

7.3 Defects will be analyzed by the Project Team for further action.

8.0 Test Suspension and Restart Criteria

8.1 None of the testing requires extended or overnight runs. All testing can be suspended at the end of day and resumed the following day.

8.2 Failure of any "Whitebox" test will suspend further testing on that class until it has been repaired. Testing will resume at the beginning of the class test.

8.3 Failure of any "Blackbox" test will suspend further testing until it has been repaired. Testing will resume at the failed test, regression testing will be performed as deemed necessary.

9.0 Risks and Contingencies

9.1 Hardware Test System not available. Contingencies: The individual GPIB instruments are available and several tests regarding GPIB communications & control can be performed with them. Communication Interface and Controller are available in several test beds and communications tests with the Controller and AI module can be performed.

9.2 Software not available. Contingencies: Some number of Classes and modules will be available. Testing can be performed at this level on those chunks that are available.

10.0 Staffing: Training Needs and Responsibilities

10.1 One tester/developer will be assigned to the project.

10.2 Tester requires knowledge of C++, GPIB communication interfaces, GPIB devices, Controller operation, xAPI commands, Analog Input Module operation.

10.3 Training on Support Tools.

11.0 Support Tools

11.1 *PBSystems Test Case Manager (TCM).*

This tool is focused exclusively on the creation and management of test cases. This fits for use on this project because it is a simple to use tool providing the basics: (1) a method for creating the test cases, (2) maintaining revision control on the test cases, (3) record test case status, (4) database allowing tracking test case status (5) report generation.

11.2 *Mozilla's Bugzilla*.

This is an open-source defect tracking system that allows individuals or groups of developers to track outstanding defects. It contains all the basic features needed and includes several extras found in many of the full featured commercial tools. Use of this tool forces the user to be disciplined and consciously record defects and changes, instead of merely writing them down and losing track.

12.0 Test Deliverables

12.1 Test Report

12.2 Test Cases

12.3 Test Harnesses

13.0 Scheduling

13.1 Test case development is estimated at 5 days.

13.1.1 Two days for development of Blackbox tests. This work to begin immediately after release of final requirements specifications.

13.1.2 Three days for development of Whitebox tests. This work to begin upon receipt of class designs.

13.2 Training on Support Tools is estimated at 2 days. Training to begin within one week of test plan approval.

13.3 Testing time is estimated at 6 days.

13.4 Debug and Repair time is estimated at 4 days.

13.5 Regression & Retest time is estimated at 3 days.

13.6 Testing, debug & repair (11.2, 11.3, 11.4) will be performed consecutively after receipt of completed software and availability of completed calibration system hardware.

14.0 Testing Costs

Estimated Labor Costs:	18 days X 8 hr/day X $50/hr	=	$ 7,200
Estimated Training Costs:	2 days X 8 hr/day X $50/hr	=	$ 800
Estimated Facilities Costs:			$ 1,000
	TOTAL		**$ 9,000**

15.0 Appendix

15.1 Glossary

- AI – Analog Input Module. One of a family of I/O modules providing a variety of input and output capabilities to the system. The AI provides for sixteen (16) precision inputs of three possible types (RTD, milliVolt, Volt).

- GPIB - General Purpose Instrumentation Bus (IEEE 488.1) that allows up to 15 intelligent devices to share a single bus by daisy-chaining the devices.

- xAPI – Application Programming Interface. Library of functions that provide software application programs access to the Company Proprietary management and control system information.

- RTD - Resistance Temperature Detector. Measures temperature based on the principle that temperature changes in pure metals results in an almost linear positive change in resistance.