# MITRE Baseline Configuration System

## Implementation Plan

FINAL REVISION, October 8, 2008
Purdue University, CS 307, Fall 2008

Team MITRE:
Catherine Brown
Michael Dunn
Mark Nowicki
David Tittle

# TABLE OF CONTENTS

## 1. Introduction

### *1.1. Purpose*

The project will be a running script that will retain, compare, and parse information form provided XML files. The benefit for the stakeholders is to remove the need for tedious manual comparisons.

### *1.2. Background Information*

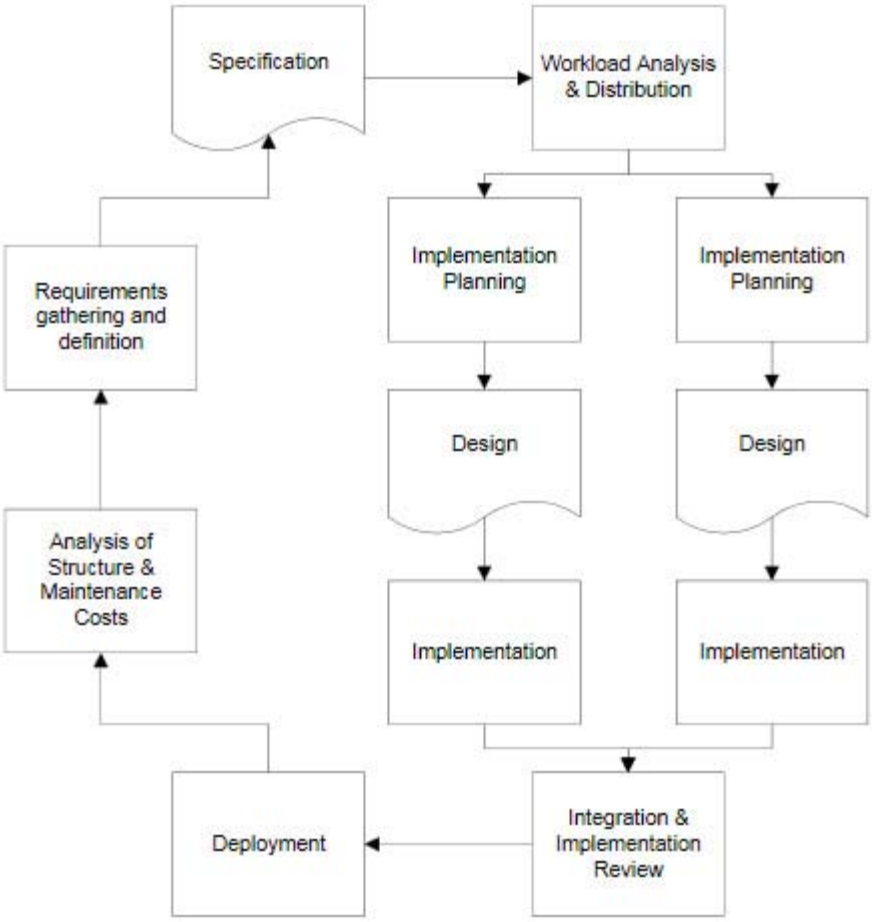
The design for the program is well underway, as described in the earlier Design Document and Requirements Document. The stakeholders are MITRE, who will be using the program, and the course instructors, who will be grading the project in its entirety.

### *1.3. Processes*

The software process we will use is a hybrid of the spiral model, phased release model and concurrent engineering model. Each iteration through the loop represents a phase, starting with "Requirements Gathering & Definition." From this process, we will get a "Specification" document. As a team, we will analyze the "Specification" and estimate "Workload Analysis and Distribution." From this stage, we will break into sub-groups. Each sub-group will do "Implementation Planning." Prototyping would occur at this stage as well. The "Implementation Planning" will yield a "Design" document for that part of the project. The "Design" will be used to aid in "Implementation." Once both sub-groups have completed their implementation, the sub-groups will reunite for the "Integration & Implementation Review." During this process, the group will test and review all code that has been committed to our system. Once code has been integrated and tested properly, it will be approved by all group members for

"Deployment." After deployment, any unforeseen bugs will be tracked, logged and prioritized in our "Analysis of Structure & Maintenance Cost." During this phase, we will also review the project, from a top down approach, looking at such aspects as: technology used during implementation, usability concerns, security and stability. From this analysis, we will perform "Requirements Gathering & Definition" regarding our next phase, which starts the process over again. This software process places emphasis on implementation review and cooperation between group members. It also encourages an extra level of quality assurance, so that all changes are thoroughly scrutinized before being considered for a release. After reviewing the code from the bottom up during the "Integration & Implementation Review" and following the "Deployment" the software process takes a different approach to review, a top down approach in the "Analysis of Structure & Maintenance Costs." This assures that the design as a whole is also scrutinized.

*1.3.1.*




*1.4. Builds*

Sentinel is a large project to accomplish in just a few short weeks. In order to complete the project, Build 1 will comprise of the Rules Manager and the Database. The most urgent need is for the ability to compare XML files with the flexibility of the user-defined rules and xml schemas. This is why the Rules Manager needs to be the top priority – it is the component that will fulfill that need. Even if no other component is completed, this one will be able to do the automatic testing required. The database follows closely behind in priority as an area to store all the information needed by the Rules Manager component. These two components can meet the most basic defined requirements for the project and can be demonstrated to the project partner.

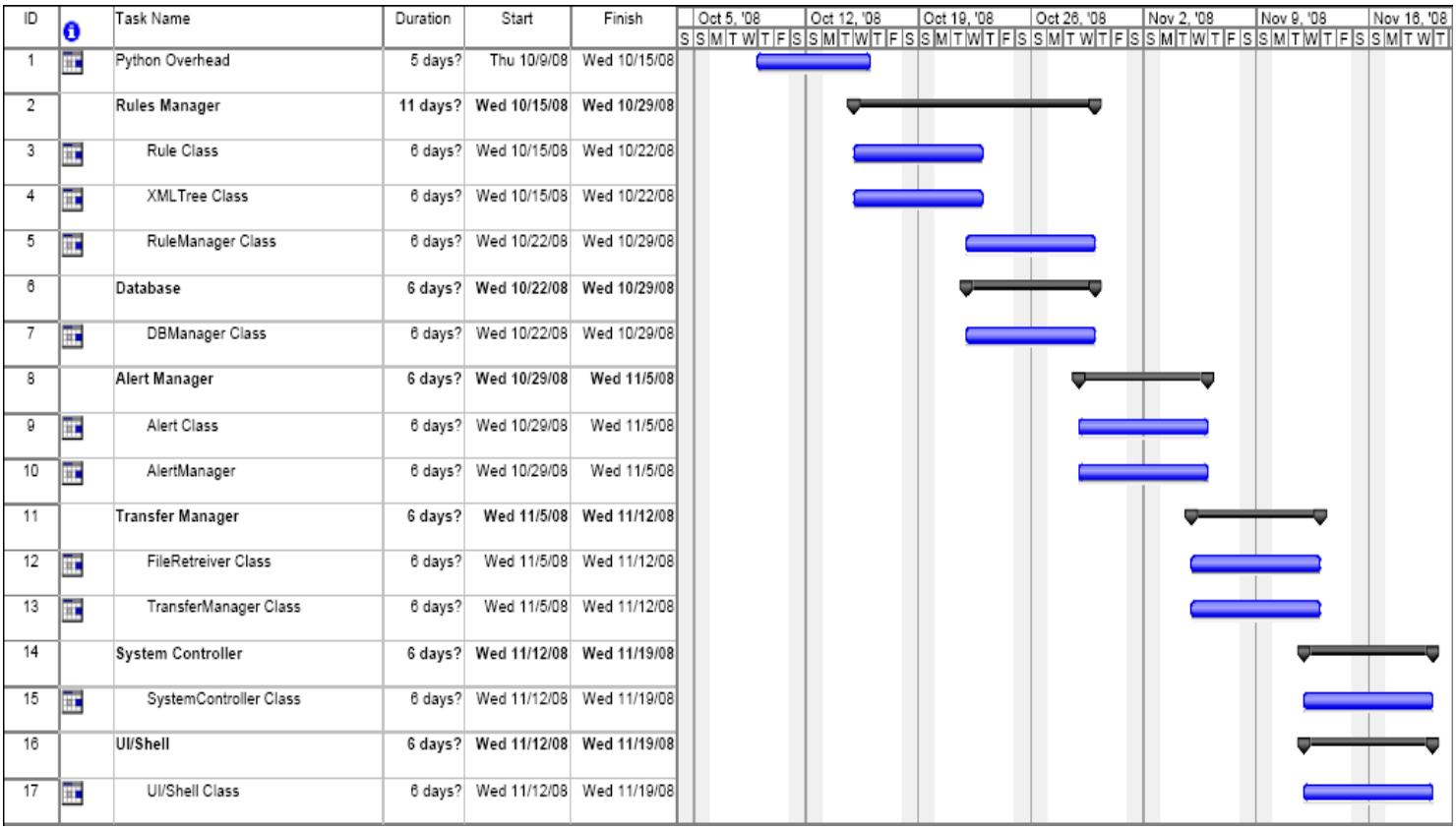
### *1.4.1. Build I*

Once Build 1's objectives are accomplished, the focus changes to the features that need implementation. The Alert Manager and the System Controller are the next priorities. The Alert component will be able to gather information from the database and send alerts as necessary. The System controller provides a single component that can direct and organize the other components into a fully functioning program, rather than a series of processes.

### *1.4.2. Build II*

To wrap up Build 2, the Transfer Manager and the UI/Shell need to be realized. The Transfer Manager will control the process of retrieving the XML files from across the network, and the UI/Shell provides an access point for the user.

## *1.5. Gantt Chart*



| ID | Task Name | Duration | Start | Finish |
|---|---|---|---|---|
| 1 | Python Overhead | 5 days? | Thu 10/9/08 | Wed 10/15/08 |
| 2 | **Rules Manager** | **11 days?** | **Wed 10/15/08** | **Wed 10/29/08** |
| 3 | Rule Class | 6 days? | Wed 10/15/08 | Wed 10/22/08 |
| 4 | XMLTree Class | 6 days? | Wed 10/15/08 | Wed 10/22/08 |
| 5 | RuleManager Class | 6 days? | Wed 10/22/08 | Wed 10/29/08 |
| 6 | **Database** | **6 days?** | **Wed 10/22/08** | **Wed 10/29/08** |
| 7 | DBManager Class | 6 days? | Wed 10/22/08 | Wed 10/29/08 |
| 8 | **Alert Manager** | **6 days?** | **Wed 10/29/08** | **Wed 11/5/08** |
| 9 | Alert Class | 6 days? | Wed 10/29/08 | Wed 11/5/08 |
| 10 | AlertManager | 6 days? | Wed 10/29/08 | Wed 11/5/08 |
| 11 | **Transfer Manager** | **6 days?** | **Wed 11/5/08** | **Wed 11/12/08** |
| 12 | FileRetreiver Class | 6 days? | Wed 11/5/08 | Wed 11/12/08 |
| 13 | TransferManager Class | 6 days? | Wed 11/5/08 | Wed 11/12/08 |
| 14 | **System Controller** | **6 days?** | **Wed 11/12/08** | **Wed 11/19/08** |
| 15 | SystemController Class | 6 days? | Wed 11/12/08 | Wed 11/19/08 |
| 16 | **UI/Shell** | **6 days?** | **Wed 11/12/08** | **Wed 11/19/08** |
| 17 | UI/Shell Class | 6 days? | Wed 11/12/08 | Wed 11/19/08 |

## 2. Management Tools, Risks, and Challenges

### *2.1. Management Tools*

In order to schedule the implementation phase of the software a Gantt chart is shown below. This management tool was chosen because it allows for larger tasks to be broken down into smaller components, and each of these components can then be given an estimated duration. This hierarchy is also displayed in an aesthetically pleasing way with blue bars showing the duration of each class being worked on with black lines overlaying these bars to show how the classes with their durations add into the total duration for a particular component. Considering all of the management tools made available to us, this aesthetically pleasing quality of the Gantt chart, which allows for software developers to most easily portray the scheduling of the project to a business partner, was a major factor in deciding which management tool to use. On top of this, out of all the management tools brought to our attention, some members of the team already had some experience with creating Gantt charts. In regards to source control and bug management, we decided on using Assembla which is a web-oriented project management center. We chose this because it was a centralized location that can accomplish both tasks, source control and bug management. Assembla includes a standard code repository with source control for source management. On top of this Assembla includes a tracking system that will allow us to label code that is to be committed with a date and information regarding the changes or additions that were made to the code. We plan to use this tracking feature to view when certain bugs arise and make note of this on Assembla to accomplish the task of bug management.

### *2.2. Risks and Mitigation*

In the course of scheduling the builds for this software, three risks were identified regarding the implementation phase. The first, and most general, risk is the problem of incorrect duration estimates. More specifically, this risk involves the chance that we may not have given ourselves enough time to complete one or more of our scheduled tasks. The iterative design approach is our mitigation strategy for this particular risk. In using an iterative design approach, the project will be able to be presented in some working form to the business partner after any particular step (task) in the implementation process. In this way, if a particular task begins to take too long, a meeting can be arranged to explain why the task that is taking an extended period of time. The project can then be shown with the current functionality so that the project partner can see what functionality has been implemented and what functionality may not be required for the finished product. Another more specific risk identified, was the risk of underestimating how much code will have to be written. In an effort to reduce the amount of code

that is written, we will encourage the use of open source libraries that have already been created in this project. Especially in the case of the XML Parser and the transfer protocol, using open-source Python XML Parsing and transfer libraries will greatly reduce the amount of code needed to be written. The third risk identified in the course of scheduling the building of this application is the risk of unidentified overhead. The only planned overhead as far as the current schedule is concerned is the overhead for learning the Python programming language. It has become apparent that there may be some overhead involved with learning and setting up a MySQL database, as well as other unforeseen overhead that may arise as the implementation of this software takes place. The only reasonable mitigation strategy for this was to plan to include the learning regarding all known overhead in the time allocated for the Python overhead. So by learning both Python and MySQL, as well as any other overhead that we can predict before the application implementation begins, before the actual coding begins we hope to cover all overhead that will be involved in creating this piece of software from beginning to end. The aforementioned mitigation strategies were devised in hopes of reducing the amount of unforeseen problems that may occur throughout the duration of the implementation phase of this project, but we understand that we still need to expect that problems will arise that we are for which we are not completely prepared.

## 3. Tasks, Cost Requirements, and Schedules

### *3.1 Tasks*

When developing the XML Baseline Configuration System, the workload will be split up into smaller subsections as defined in the Gantt chart in Section 1.5. Each task will be developed by two developers at a time, with the Python Overhead being a strictly individual task. The order of the breakdown is done by the level of urgency or importance to the system.

#### *3.1.1 Python Overhead*

This task can be broken down into multiple sections, but still are within the constraints of the learning curve of the system. In order for the system to be developed with quality, robustness, and maintainability, the development team will need to get acquainted with the Python language, which is the chosen development platform for this system. However, learning the Python language will not be the only requirement. Other requirements such as learning accessible, open source Python libraries and researching and making use of existing Python Testing Suites are all considered to be subsections to “Python Overhead”.

#### *3.1.4 Rules Manager*

The Rules Manager task is considered to be one of the most important components of the system, thus a higher priority task. This component is used for the actual parsing and comparing of the collected XML files. The task can be broken down into subsections as follows:

##### *3.1.4.1 Rule Class*

The Rule Class is a class which contains a String which will contain an XPath query. This XPath query will be used against the parsed DOM Trees ( Explained in the following section ) in order to extract important information for comparing the gathered XML file and the baseline XML file. Will be instantiated by the RulesManager Class.

##### *3.1.4.2 XMLTree Class*

The XMLTree Class will be used to store the parsed XML files into a DOM tree, which will consist of this class. Using the Rule Class, the system will be able to simply query this class to extract needed information for the comparison functionality.

#### *3.1.4.3 RulesManager Class*

The RulesManager Class will be one of the largest and most complex classes in the entire system. The RulesManager Class consists of several functionalities including: parsing into DOM trees, comparison of two DOM trees, and sending alert messages to the AlertManager Class. To insure quick and easy implementation, a developer who implemented the Rule Class, and a developer who implemented the XMLTree Class will implement this section together.

### *3.1.3 Database*

The Database task only consists of one task, the DBManager Class. Essentially this class will be used for other classes to interface for the database. This task has a high ranking due to the fact that in order to have a fully automatic system, there needs to be a database in which a system can quickly reference specific information. For example, if the system wants to determine what rules apply to a specific XML file, then the system can query the database for that object, and then use that object to find any information it needs quickly, and reliably.

### *3.1.4 Alert Manager*

The Alert Manager task is comprised of two subtasks. This task is ranked as such do to the need of an automatic messaging service. This service will be used when a change is found and someone needs to be alerted.

#### *3.1.4.1 Alert Class*

This class will be very simple, containing a simple String, and a client id which will be used to identify which client server the change belonged to. Will be used in the AlertManager Class.

#### *3.1.4.2 AlertManager Class*

This class will be used to manage all of the Alerts that were created, and form messages from them. This will have the functionality to gather information required and send the messages accordingly.

### *3.1.5 System Controller*

The System Controller task consists of just one main class, the SystemController class. This class will be used to create the automation for the system. Since it is the main controller, this is a very critical task for the system. Using all of the components that exist, this class will bring

them together for full unity and automation of the system.

### *3.1.1 Transfer Manager*

The completion of this task solely depends on the completion of these two subtasks:

#### *3.1.6.1 FileRetriever Class*

This class will be used to create a connection using SCP to gather required XML files from client servers.

#### *3.1.6.2 TransferManager Class*

This class will take the XML file retrieved from the FileRetriever class and save it onto the file system.

### *3.1.7 UI/Shell*

This task will be the final and least important component that will be worked on in the project. However, it is still undetermined how/when this will be developed. Since there is a chance for not fully completing the system, it must be taken account for. For example, if 70% of the project is complete by the end date, a UI/Shell must be developed for that 70%. A system cannot be developed without having any methods of interaction. With that stated, if the system goes as planned, any additional features such as a web interface will be completed at the end of the project.

## *3.2 Cost Requirements*

When determining the cost requirements for software projects such as this, one has to take into account the best case scenario, and a worst case scenario. Outlined in this section are both scenarios.

### *3.2.1 Best Case Scenario*

When viewing the development plan in the best case scenario, the system should be able to be completed in a total of six weeks. This best case scenario can be observed by referencing the given Gantt chart in section 1.5. In this case, each task has allotted one week for completion, and using a concurrent development method with four developers, it can be done in six weeks. The concurrent development method allows for two tasks to be worked on simultaneously by two groups of two developers.

### *3.2.1 Worst Case Scenario*

When viewing the development plan in the worst case scenario, there is no way to identify when and where a problem may or may not occur. However if an issue does arise that will delay development over the time allotted ( one week ), special attention will be given to that portion and handled efficiently. If a situation does arise where the deadline is approaching and some tasks have not been completed, then those tasks will be dropped and developing a UI/Shell for completed components will take focus for all developers. Ensuring interactivity between the system and the users is a high priority; however it cannot be done until the developers have decided which components can be implemented in the time allotted.

## *3.4 Schedule and Milestones*

The schedule (timeline) of our project can be referenced from the Gantt chart shown in section 1. The system is broken down into eleven different tasks and should take approximately six weeks for completion. Milestones in the system would be Build 1 and Build 2. Build 1 is a big milestone because three main tasks will have been completed: Python Overhead, Rules Manager, and the Database. Arguably, these could be the most difficult tasks to complete. The second milestone will be Build 2. By Build 2 the entire system should be completed and functioning.

## *3.5 Team*

The team will divide into two pairs for each part of the project. After the completion of a task, the pairs will split and reform such that one member from each previous team combines to form a new team. In this way, each team member will be able to share their experience on every part of the project with their partner as well as take responsibility for general requirements.
The first week of work is dedicated to learning Python, and it is expected that each member will continue to improve their python skills throughout the project.