|  |
| --- |
|  |
| Software Design and Documentation Final Exam |

Daniel Campos

April 15th, 2013

Table Of Contents

Question A: Analysis of a Open Source Software 2

About YACS 3

Technical Overview 3

New Feature: Display all Courses that could fit in schedule 3

Advantages and Benefits 4

User Scenario 4

Class Diagram 6

Sequence Diagram 7

Class Responsibility and Collaborator Models 7

Design Decisions 8

Open-Closed Principle 9

The Liskov Substitution Principle 9

Ease of Change Test: 9

New Feature: Add Sports 10

Advantages and Benefits 11

User Scenario 11

Class Diagram 11

Sequence Diagram 13

Class Responsibility and Collaborator Models 14

Design Decisions 15

Open-Closed Principle 16

The Liskov Substitution Principle 16

Ease of Change Test: 17

Additional Feature: Display All Available Courses 18

Question B: Implementation Plan for Open Source Software 18

Methodology Overview 18

Developer Roles 19

Schedule 19

Essential Team Activities 21

Communication Tools 21

Question C: Agile Development and Mythical Man Month. 22

Works Cited 25

# Question A: Analysis of a Open Source Software

## About YACS

YACS is a free and Open Source RPI class scheduler application. It has numerous features but the most important amongst these is the ability to generate a class schedule based upon a user determined course selection of the up to date RPI catalog. More About YACS can be found on its website http://yacs.me

## Technical Overview

YACS is a web browser based application that is written in Python, JavaScript and displayed using HTML on its website. The Python and JavaScript are used for the creation of courses catalog, then, based upon the input and course selection of the user a set of schedules are created and displayed. This creation of schedules happens because the system and scheduler takes all the courses the user has created and every combination of schedule and just displays the ones where no two classes overlap.

## New Feature: Display all Courses that could fit in schedule

YACS current has a pretty simple and straightforward system for searching for courses and adding them to the user’s prospective schedule. A user can search for a course based on one of its identifiers (Name, Course ID, Category, Time Hours, etc.) or the user can search for courses based on what subject matter they fall into (e.g. ECON,CSCI,etc.) The user’s schedule visualizer continuously updates as a user adds courses to their prospective schedule. As these courses are added the system shows the user their possible schedules. As collisions (courses with the same class time) occur, YACS does not allow the user to add that course to their schedule. This allows YACS to make sure the schedules generated never have a conflict.

Building on this, I believe it would be beneficial to for a user to be able to view what courses they could take in the time slots they have left in their schedule. The user would input their courses that they consider a “necessity” (courses they are sure they want to take) for the semester. Then the user would use this feature and be add all the course materials they are interested in(ECON,CSCI,ETC).Then YACS will present the user with a list of the classes they could take organized alphabetically. The user then can comb over these options and decide which courses they would like to take.

This feature would allow users to decide what other courses they would like to take based on what they actually can take.

### Advantages and Benefits

1. The user is now able to search for courses amongst what courses are possible due to their time blocks.
2. The user is now able to build their schedules quickly and efficiently since they can see only what courses are open based upon their schedule/time blocks.
3. The user can also easily identify which classes are at the same time because as they select one other with the same timeframe immediately are no longer options. Based upon this the user can now see what trade offs they have of taking one course vs. another. The old method would require you to add a class and search through the other categories to see what courses the selected ruled out.

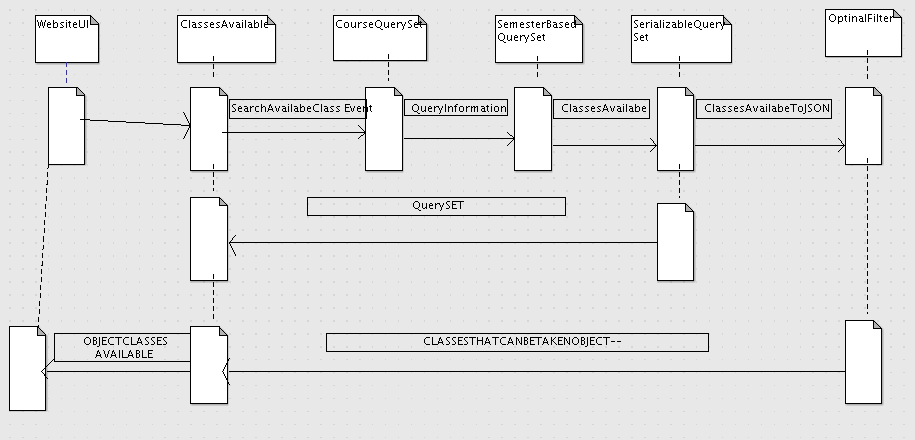
### User Scenario

Daniel Fields is a second semester sophomore studying computer science at RPI. He is due to register for classes soon and has a 3 classes that he needs to take, and wants to take another. Daniel has finished most of his core requirements so for his other classes he is unrestricted. He likes to be challenged by strange and new subject matters and because of this is comfortable taking courses in all subject matters. Daniel has used many course schedulers and has always been frustrated by the lack of ability to see the complete selection of what classes would fit into his free time. He does not like searching through many pages to then have to check and see if the course his attention is drawn to fits in his schedule.

One day Daniel logs on to YACS. He adds the courses he has to take: MATH 2400 Intro to Differential Equations, CSCI 4430 Machine Learning and Computation and CSCI 4150 Intro To Artificial Intelligence. He is planning on taking 2 more courses and they have no restrictions. He adds those courses to his schedule and then clicks on the “see viable classes” button. He chooses the ECON, IHSS, MGMT, PHIL and STSS departments. He then is presented with about 50 options. He then chooses MGMT 4530 Starting a new Venture and IHSS4520 Drugs in History and checks out his schedule options. Liking what he sees he then registers these classes and is signed up to take said classes for the Fall 2013 term.

### Class DiagramMacintosh HD:Users:Rob:Desktop:Screen Shot 2013-04-14 at 5.23.22 PM.png

### Sequence Diagram

The Sequence diagram represents how a user would check to see what courses they could take.

### Class Responsibility and Collaborator Models

|  |  |
| --- | --- |
| Website UI | |
| Responsibilities:   1. Displays the User Interface 2. Handle User Input | Collaborators:  None |

|  |  |
| --- | --- |
| ClassesAvailable | |
| Responsibilities:   1. Search all the categories that the user wants to try 2. Make sure to merge the objects to create one large object. | Collaborators:   1. CourseQuerySet |

### Design Decisions

One of the goals of this design was to keep coupling minimal and extensibility high. In order to accomplish this, simplicity, modularity and expandability were though of extensively. Furthermore relevant design patterns were implemented in the app as they allow change to be easily incorporated into the code.

While creating this additional feature I tried to use the Abstract Factory design pattern. I did so by trying to turn the already existing search method into its factory and then creating an abstract factory that would handle the different searches. I did so to minimize the amount of code that needed to be added. I also did this to make this new method be encapsulated. I chose this method because by using it, YACS is able to keep good Data encapsulation since the artifact itself never interacts with the data. The class will merely call up the other class in the correct way and then merge its results. This encapsulation was also a little weaker than I would have wished because the language used, Python, does not have the concept of data hiding. I did use it because encapsulation also refers to encapsulating the data by obscuring data through the use of classes. I did that with my feature because it does not allow anything to access the data outside of the abstract factory and its handler.

Cohesion within the design is quite strong, as objects within a package work together to get a task completed, but rarely have dependencies on external classes. All the classes connect with each other and function as a unit to produce the final result, an object containing all the classes. By doing so I have the least data exposure and had good cohesion.

### Open-Closed Principle

The open-closed principle states that *“software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification”* (Meyer, 1998). The addition described in this document follows this principle to the edge. The see available classes feature is just an extension of what is happening before and its search style. It cannot be modified because of the nature of abstract factory; its modification would be something difficult. If it were to be extended it would be simple since the information is easy to understand and access as long as the abstract factory is used correctly.

### The Liskov Substitution Principle

The Liskov Substitution Principle states “If for each object o1 of type S there is an object o2 of type T such that for all programs Pdefined in terms of T, the behavior of P is unchanged when o1 is substituted for o2 then S is a subtype of T” (Liskov, 1988).

My developed features follow this principle. It follows it because the abstract factory can access and use all the feature of the factory but the factory cannot use or modify anything in the abstract factory. The factory does not know how many other entities (Subject Matters being searched) are happening at once and does not deal with any of it. Since the factory can only be used when the abstract factory was called prior then it works.

### Ease of Change Test:

1. How many classes did you have to add to support to use the new class?

I only had to add one class, which would become an abstract factory for all the different clients.

1. How many classes did you have to change to support this new class?

No classes needed to be changed because the abstract factory takes advantage of the preexisting class of search and then merges the results to be displayed.

1. Suppose you decided that you wanted to start displaying the courses by time period classes would you have to change to support this new information?

I would have to change at most one class that sorts the JSON array of classes on the time instead of the name. Everything else takes in objects and therefore would not need to be modified.

1. Suppose that you wanted a new property that displays only at most 50 courses, how many classes would you have to change to support this?

All I would have to change would be the abstract factory since it takes in all the answers and it would need to sort through all the classes and choose 50. All the other classes would not need to be changed.

## 

## New Feature: Add Sports

These days a large percentage of people in college are involved in some sort of athletics. For most involved in these events skipping a practice is not an option. YACS could benefit greatly by being able to have the sports practices as “courses” much as those in ROTC programs are listed as if normal courses. Athletes would be able to select their sport and this would include all of their team events: practices, gym time, film time etc. After they selected their sport they would be able to see what classes they are not able to take because of practice. If the user has a conflict with a necessary class then they can talk to their coach early to reschedule their practice. This feature would also allow athletes to view what their day may look like including all their sports practices, not just with their courses.

### Advantages and Benefits

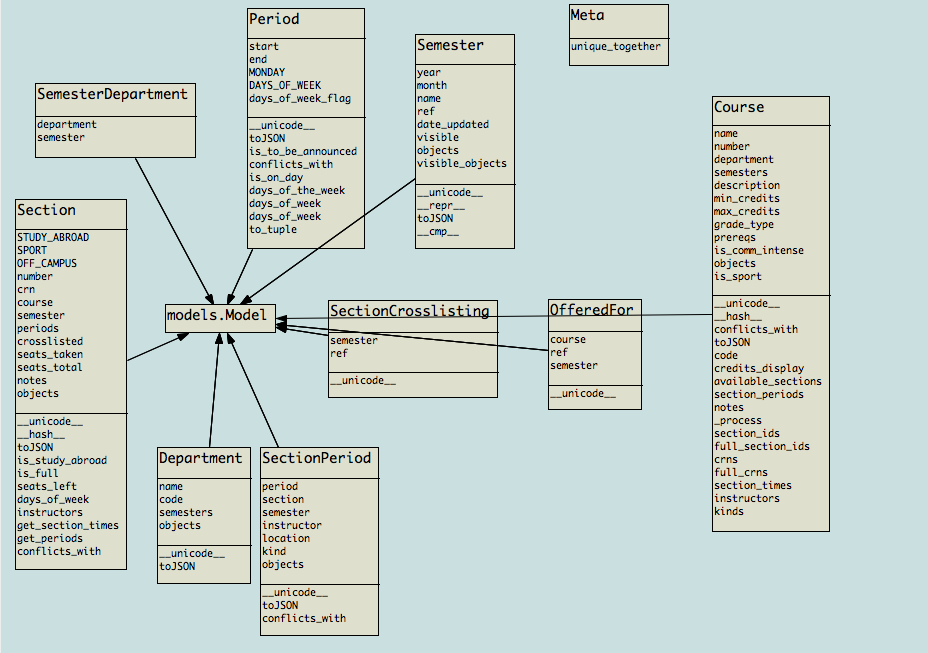
1. Allow people to figure out what classes they can take once they have inputted when they have sports practice or workout sessions.
2. Allow people with sports practices to schedule their semester with all of their activities much as those in ROTC programs.
3. Allow coaches to be notified early if their players cannot make practice. If a lot of players are unable to make practice then he will know early that he needs to change practice time.

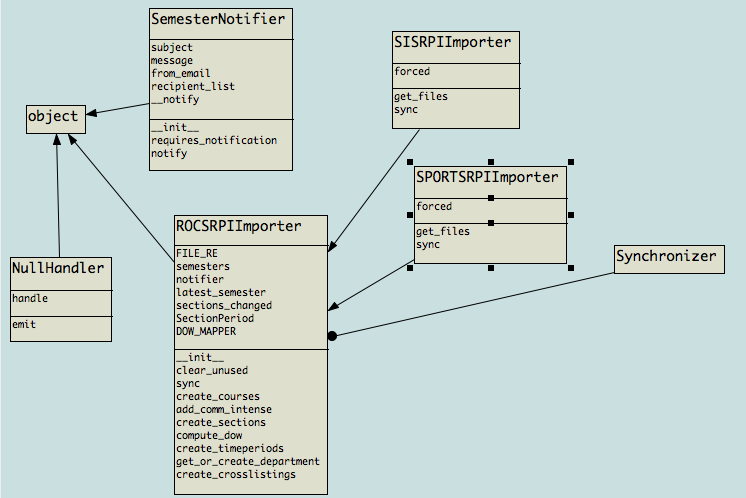
### User Scenario

John Mctavis is the starting Strong Safety for the RPI Football team. Before he even has to register for classes he knows exactly what times he will have practice, gym sessions, film, etc. and would like to be able to schedule his class selection around these events.

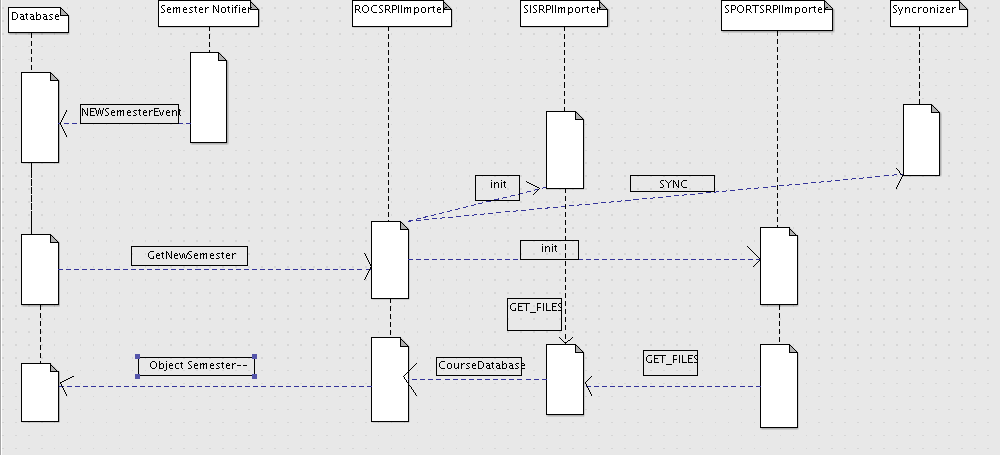
John logs onto YACS and scrolls down to the Sports Section. He then scrolls down to the Football section and clicks the checks the box. It adds it to his schedule and then he adds his other courses around what wont miss any practices. He then registers and has no schedule conflicts with sports causing him never to miss practice.

### Class Diagram

The following Class Diagram represents the section dealing with the creation of each course and its information. It has been modified to have information about whether or not it is a sport and related information

This Class Diagram represents the changes done to the course importer that sets ups the courses for visualization and usage.

### Sequence Diagram

The sequence diagram shows the interaction of the system to generate a new semester

### Class Responsibility and Collaborator Models

|  |  |
| --- | --- |
| SportsRPIImporter | |
| Responsibilities:   1. Import and update all the sports in RPI based on the open source importer. | Collaborators:   1. None |

|  |  |
| --- | --- |
| SISRPIImporter | |
| Responsibilities:   1. Import and update all the courses in RPI based on the open source importer. | Collaborators:   1. None |

|  |  |
| --- | --- |
| ROCSRPIImporter | |
| Responsibilities:   1. To import all the courses and sports to a database. | Collaborators:   1. SISRPIImporter 2. SPORTSRPIImporter |

|  |  |
| --- | --- |
| Semester Notifier | |
| Responsibilities:   1. Inform the system of when a new semester is beginning and the information is up | Collaborators:   1. None |

### Open-Closed Principle

The open-closed principle states that *“software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification”* (Meyer, 1998). This feature follows this principle since the new features are open for extension but closed for modification. The factories and the extractors are something that are set and already built and to modify them would mean to change the purpose of the function. It works for extension since if someone wants to use this importer in another way then it is simple and possible.

### The Liskov Substitution Principle

The Liskov Substitution Principle states “If for each object o1 of type S there is an object o2 of type T such that for all programs Pdefined in terms of T, the behavior of P is unchanged when o1 is substituted for o2 then S is a subtype of T” (Liskov, 1988).My developed features follow this principle. This is true since though both the SISImporter and the SPORTSImporter are derived from the same type of thing( much the way a rectangle and square are from a polygon) they do not get their information from each other and both inherit from the higher section, the ROCSImporter.

### Ease of Change Test:

1. How many classed were changed to add this support?

One class was changed to add this support. The ROCSImporter since it now needs to call up the SISImporter and the SportsImporter and merge the two

1. How many classes were changed to add this support?

One class was added since it was needed to import the sports.

1. Suppose this were to be modified to import intramural games. How many classes would have to be changed to support this new information?

One, the sports importer. It would just have to go to the website that displays the usage of field and now import the things that were not varsity sports, the intramural sections.

1. What would have to be changed to support a new property such as having information about what type of practice it was (conditioning, film, etc).

None, if information was present it would just need to be placed in the description or type of event. This piggybacks of the usage of Lecture and Recitation kind of thing.

## Additional Feature: Display All Available Courses

This feature would be much like display available courses talked about as my first feature with a twist. The difference between these two is that this one would not require the user to input the subject areas they were interested in. Due to the nature of the search coded before a search of no items (for all intents and purposes it is a search of “ “) will return all items. This can be done without modifying the backend because the abstract factory (the code that I added) is able to take no handlers as an input and when it receives that It searches for null and by result will get all the available classes.

# 

# Question B: Implementation Plan for Open Source Software

## Methodology Overview

The methodology that I selected for developing my features was SCRUM. Scrum is an Agile methodology that seeks to be “a flexible, holistic product development strategy where a development team works as a unit to reach a common goal"(New New Product Development Game, 1986). The goal of this methodology is to increase the speed and flexibility that a team can have while it is developing a project. The name comes from Rugby, where a scrum is set of overlapping players that come together to push. This is applied to software since Scrum is a technique with overlapping cycles that come together in a hope to produce the best results quickly. This technique is special because it has a few very specific set roles for the developers: Product Owner, Development Team, and the ScrumMaster. The product owner represents the voice of the customer and is responsible for making sure that the team delivers a product of value to the customer. The Development team is responsible for creating a shippable product at the end of each Sprint. The ScrumMaster is accountable for removing all the risks and impediments to the team that may keep them from delivering their sprint goals/deliverables.

Scrum is also different in the way that it makes sure that everyone is well informed about what is happing around the project. Scrum makes sure that every day, the project team members communicate: “What have they done since yesterday”, “What are they doing today”, “What might prevent them from reaching their goal today”. This is done to allow the ScrumMaster to know what roadblocks he may remove and what large roadblocks may be coming up. This daily exercise also helps every member of the team understand how the project is advancing and know where his or her piece of the development will join in.

Besides these daily meetings Scrum has set meetings that the group must do at certain stages of the development cycle to specific guidelines but I will not get into that because it is a lot of information.

## Developer Roles

Since I only have 4 people I would in this case merge the roles of the Product Owner and the ScrumMaster since neither job will probably require a lot of work. The other 3 people will work on the product development.

## Schedule

Due to the methodology of Scrum and how it works I have broken down my feature development into two sprints and separated them as so.

Sprint I(Search for Free Classes)

* Backlog grooming/Story time: At this period of time the developers will play around with the system so they can understand how it works and how the users will use it. They will then decide how to break up the work and learn about the existing code frame.
* Phase I: in this phase the team would modify the existing search class to become a concrete factory and to be able to return all the classes.
* Phase II: this phase would deal with the creation of the abstract factory that handles the previously created concrete factory. The developers would make sure that the abstract factory can create as many concrete factories as it will need (different amount of subject areas can be selected).
* Phase III: the developers will merge what they had created into the old website and then create the HTML methodologies to be able to call up their newly created abstract factory.
* Phase IV: Testing. The developers will test the features to make sure that they merged properly, didn’t affect other functionality, and are doing their supposed task.
* Shipment

Sprint II(Add Sports/Testing Previous)

* Backlog grooming/Story time: At this period of time the developers will play around with the system so they can understand how it works and how the users will use it. They will then decide how to break up the work and learn about the existing code frame. While doing so they are also testing and tweaking their previously released code and making it work better and deciding what modifications are necessary for this improved use.
* Phase I: In this phase the team will find the correct information source for all the practices and turn it into a format that is comparable to the format the course importer takes, so that the merging of the two causes no issues.
* Phase II: with the SISImporter as an example the team will create a Sports importer that creates the courses from the sports practices and can create a database in the form of an object and a JSON to help merge.
* Phase III: the team will modify the ROCSImporter to call up their newly created importer and then after that is done merge the two sets of course lists to create the database of the semesters classes and sports practices.
* Phase IV: Testing. The developers will test the features to make sure that they merged properly, didn’t affect other functionality, and are doing their supposed task.
* The team ships.

## Essential Team Activities

While developing, the team is constantly updating their Sprint Backlog. A Sprint backlog is a visual aide, usually a whiteboard, that shows distributed by each use case, items that are: to be done, in progress, testing, completed. Our team would be constantly updating this board. They would do so at each Daily Scrum. They are also doing their Beginning of Sprint scrums and End of Cycle scrums, which, are in depth meetings to define goals and conclude how the team has reached them. The team would not have to do a Scrum of Scrums much because it is a relatively small team and therefore would have no sub teams.

## Communication Tools

As mentioned in previous parts, the teams will have daily meetings to inform each other of their progress and impediments. They will also use their Sprint Backlog to see what is in progress and what is already done, which is useful for the short deadlines. They will also have a Burn Down Chart, which is a visualization of how much work has been done/how much work remains and how close they are to product release. This way the can see how close they are to schedule and about how many more hours need to be done before releasing.

# Question C: Agile Development and Mythical Man Month.

Fredrick Brooks wrote the Mythical Man Month back in 1975 about an industry that was still in its infancy, software development. The main topic of Fredrick’s book was the quicksand that development projects get into once they start to think of people and time required to develop as interchangeable things. He wrote about five different misconceptions that can cause software development to fail. Since that time, many people have come up with different ways to mitigate these misconceptions. The methodology that best handles these misconceptions is Agile Extreme Programing also known as XP. XP is based on four core values: communication, simplicity, feedback and courage. By applying these values, and the 12 practices that come from them all of Brooks’ misconceptions can be stopped.

One misconception is the misallocation of time when scheduling a software task. Due to how people usually develop code, nowhere close to enough time is ever given to debugging and to system testing. XP mitigates it through a few of its practices mainly: assumed simplicity, incremental change, and teach learning. By having assumed simplicity the code is kept simple. When the code is kept simple, it only handles the situations that are present at the moment and is trimmed down. This allows for quicker and more effective debugging since there is less “fluff” to go through. When a team has incremental changes, it makes sure that each stage has debugging and system testing. Since the team only moves on to the next area when the last one is done, each stage has efficient debugging and therefore when the product is getting close to be done it has been constantly debugged. Teach learning is the concept that programmers should focus on learning the amount of testing, design, and retrofitting required for their fixes instead of just making statements about testing. By using this, programmers become much better at debugging and can do so much quicker and flexibly.

A second misconception is schedule slippage. This is the idea that if a product was behind at the beginning of the project I can just add more workers at the end of the project. This is not the case since by the time an issue or schedule slip reaches the end the problem will be so large that it cannot be fixed. XP deals with this by: simple design, less complexity, small release and embracing change. As a team embraces change they are able to keep all of their options open which allows them to have a more open and lenient schedule. By having an open and lenient schedule, the team can have room to accommodate for schedule slippage. By having simple design schedule, the scheduler is less likely to slip because any extra complexity is immediately removed. Using less complexity causes the implementation of anything becomes more manageable and adding extra manpower is no longer a necessity. Lastly, by having small releases schedule slippage is much less likely to occur because the team knows when a group is behind schedule and fixing this slippage is much easier since less will need to be done.

The third misconception is that each task will only take as long as it ought to. This means that to complete each task it will take less time that it actually will. Programmer will compliment this by swearing that something is the “last thing” or “ it will work after I fix this one section”. XP mitigates this by using its practices of incremental change, concrete experiments, and assume simplicity. Incremental change is useful because when there is a problem it is know sooner and it is much easier to fix a small problem than a big one. This practice is also helpful because it is less likely that a task will take extra time when each task has been broken down into a deliverable and all the items are understood making it less likely to be late. By using assume simplicity we follow XP belief of creating a solution that solves only today’s problems and not the more complex tomorrow’s. This makes the tasks not go over the planned amount because no feature is caught up in adding extra features or making sure it can handle the systems that will be using it in the future. Since it is only handling the present there is no need to add extra features and miss deadlines.

The fourth misconception is poorly monitored schedule progress. In older development since a waterfall technique was followed, it was only really known at the end of each cycle, which was already completed, the progress that had happened. XP avoids that through Rapid feedback, incremental change, and accepted responsibility. Rapid feedback allows all people to understand quickly and simply how their piece of the project is going. They can understand how their piece is fitting into the larger scheme and if it isn’t up to standard or isn’t working right they can immediately know so they can fix it. Incremental change breaks up big problems into small cycles and therefore allows the development team to know much more frequently how they are progressing on their project. Since they have shorter and more viewable milestones they can immediately understand how their development is progressing. By using accepting responsibility people spend less time trying to hide their lack of work or falling behind schedule, they take responsibility for what they were supposed to do and because it is understood what each person of the team has to do everyone can know their progress and intervene if their progress isn’t up to date or to standard.

The fifth and final misconception is the mythical man month and the thesis of Brooks’ book. This misconception is the idea that men and months are interchangeable entities. Brook states this is wrong because when you add more people you complicate the channels of communication and slow people down. XP mitigates this issue by the use of rapid feedback, quality work and honest communication. Rapid feedback allows the team to understand how well the project is progressing, what needs to be done and the general pace of things. By doing so it is easier to have everyone understand what is happening and therefore decrease the amount of channels of communication, therefore speeding things up. By using quality work everything stays on schedule and if the project did seek to get speed up by adding extra people, then replacing bad parts and having the new workers not understand would not slow it down. The most important is honest communication, since it increases individual ownership and keeps everyone informed and up to date. Honest communication is the straightforward informing of what everyone is doing in the grand scheme of things. There is no hiding what has been done or making a accomplishment seem greater than it was. If someone is behind schedule it is made know and how to remedy it. By doing so the amount of channels of communication required to fix these issues is decreased and therefore so is its cost.

When XP combines its four core values and twelve methodologies it does a lot to solve many of the pitfalls of software development. While it, like any other agile methodology, does a lot to solve the issues, it is important to know that even with these practices many issues can arise. The way that a project gets late is one day at a time, regardless of its development technique.

# Works Cited

Liskov, B. (1988, May). Data Abstraction and Hierarchy. *SIGPLAN Notices* , 23,5.

Meyer, B. (1998). *Object-Oriented Software Construction.* Prentice Hall.

Beck, K., & Et., A. (2001, February 01). *Manifestor for Agile Software Development*. Retrieved 11 19, 2011, from http://agilemanifesto.org

Boehm, B., & Turner, R. (2003). Balancing Agility and Discipline: A Guide for the Perplexed. Boston, MA: Addision-Wesley Professional.

Brooks, F. P. (1975). *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley.

Liskov, B. (1988, May). Data Abstraction and Hierarchy. *SIGPLAN Notices* , 23,5.

Meyer, B. (1998). *Object-Oriented Software Construction.* Prentice Hall.

Norfolk, D. (2010, January). *Avoiding the Integration Tar Pit.* Retrieved from http://www.pervasiveintegration.com/Downloads/Documents/Pervasive-Beye-IntegrationTarPit.pdf

Schwaber, K., & Beedle, M. (2001). *Agile Software Development with Scrum.* Prentice Hall.

Zannier, C., Erdogmus, H., & Lindstrom, L. (2004). *Extreme Programming and Agile Methods - XP/Agile Universe.* Springer.