Software Development Process for CS 429

As more complicated requirements are presented and proposed, as well as increasingly more powerful computing hardware resources like CPU, memory and graphic cards were developed, software systems are getting even more and more convoluted. According to the record of *informationisbeautiful.com*, a modern operation system has already reached 40-90 million of lines of code, and the most used search engine and web service platform -- Google, surprisingly, has the incredible amount of code base that has 2 billion of lines of code in total.

However, software developing is not like building a great wall, which is the repetition of limited number of measures and techniques. To build a software that has high performance as well as being robust, scalable and reusable, developers need to excellently cooperate with each other and manage the project to mitigate the risks and avoid losses in profits and resources. That's exactly why software engineering management methods are needed when developing high quality software systems.

In our projects for this course, development process will be followed in order that our application can stay healthy and can be easily scaled along the developing progresses towards increasing extent of code base and that possible delay of schedule can be avoided so that we can implement what we need in time.

The start of the project is to form a team with members that can and are willing to efficiently work with each other. The first step is to bring up a proposal that is attractive and aspirational enough to bring talented people together. However, it is not a goal that can be easily reached, since there are chances that people don't like the specific idea and reject it as a result. When this happens, a full set of analyses can be applied to make it better.

As it was mentioned, projects without a proper scale would be hard to be accepted. A project that is too easy may not be worth spending much time and human resources, while projects that are too large may not be finished in time. At the beginning, the baseline that contains things that are really necessary are the most appropriate amount of tasks. In the world of software development, done is usually better than perfect. We can always follow an iterative and incremental developing process to enrich the projects with high-level or more challenging features and functions.

On the other hand, if a project proposed is too similar to other products that are already available, there is no reason why resources should be spent on it as well. Building a new project, though sometimes following is necessary, is more about innovation. During the rewrite of the project proposal, I should make careful and systematic comparison in order to exam the pros and cons of each existing competitors, and features of our own project needs to grow from the flaws determined in these analyses.

After overcoming this difficulty of getting the proposal accepted, my proposal of project may fail to attract enough skilled members to work on the project which may cause the failure of the project. For various reasons, people may refuse to join a project, including technic requirements that are too hard, types of project that are too boring, or costs that are too high.

If this proposal requires technology that is so difficult it drives people away, I should select different technic protocols so that the technological ability of potential members can cover most of the demands. If the data structure or algorithm is too complicated to be implemented, or a certain module requires high-performance or top security, the suggestion of how they should be involved and integrated into this program needs to be changed. Since the foundation of *Free Software Movement* in 1983 by Richard Stallman and the widespread open source projects nowadays, various high quality libraries and frameworks are free to be used to replace the parts of the proposal that contain requirements that are too hard to be implemented.

Perhaps there are boring aspects related to the project that people may find unattractive. If that's the case, researching more of the popular trend of modern technology can contribute heavily towards how the project can be restructured. For example, in recent years, the communities of deep learning and artificial intelligence are growing rapidly. The world is shocked that Alpha Go designed by Google has beaten all known human go experts, AI assistants like Siri and Cortana became popular once it was published, massive deep neural networks granted machines stronger ability to learn abstract concepts. Hence, projects that can make a use of these technologies can really inspire and attract people with passion and talents to devote their time and energy into the projects.

Just like forming a team, choosing a team's project to contribute requires even more insight and wisdom of selection. People may enjoy the pleasure of conquering challenging tasks, but will hate the bitterness of failure. Hence, a proper difficulty of the projects shall be the first thing to be considered. More specifically, the most preferable projects should contain some concepts, languages or technologies that I don't know or do not feel comfortable and easy to use so that I can challenge myself during the development of this project and improve my skills. However, the project should not require too much time learning, or have to contribute a lot of time to make this project successful. Since it's a development project instead of research, a right amount of time splitting should be carefully balanced.

On the other hand, excellent leadership, cooperation and mutual understanding can also contribute a lot to the success of a project. Hence, when choosing a team to join in, I may search for the presence of a skillful leader. If not, I may consider how well those members can work with each other and prepare to take emergency leadership at any time.

After the team is successfully formed, the experience of software in the past time may be able to inspire team members to settle a proper, even better, development process to follow in order to reach the highest efficiency and quality.

In the projects of CS 427, extreme programming (XP) is followed by most team to support an agile, iterative and incremental development. Though fast and compact iteration, usable software can be quickly developed and improved. By following this methodology, we adopted several activities to meet the standard of it.

For analysis of requirements of the project, we applied writing of user stories by realizing functions and features into how the customers will use the system for and the features they want to use. Then we can divide those requirements into several specific simple tasks to work on. Since each user story is limited to a few sentences, we can guarantee that each task is tiny enough to be implemented and tested.

During the process of implementation, we adopted the method of pair programming. 2-3 programmers got paired up so that one of them can code and care about more details and the other(s) can navigate from a more general view of point. While working at the same part of the job for the same time, paired workers can mutually check if their partner did their part correctly. Hence, the efficiency and quality can be improved since fewer mistakes will be made during the cooperative process and less time and cost will be spent on fixing the flaws.

Since extreme programming promotes an agile development that does not require an overall design of the whole system, when continuously adding new modules and components to the system, so problems that affect the scalability of the system become obvious. When code smells that compromise our project appears, pairs of programmers will start to refactor the codes in order to eliminate redundancy and weak designs. By using design patterns, useless repetitions of code will exist in the code base and create interfaces that grant us ability to flexibly add more features without having to change too much repeated code in many different places.

It is also noteworthy that extreme programming is a test-driven method, which means that usually a full set of automated tests should be written even before we start working on the actual coding tasks. High coverage of test is the guarantee that our code completes the goals we set in user stories, and the change we made in refactoring would not introduce any new bugs into our existing code base that works fine.

All these methods contributed to our success in the project we did for CS 427, but after some self-reflections are made, we also find there are some details of this developing process we can improve in order to make a solid foundation for our new project for this course.

One of the most serious problem I experienced is that when people without matching ability of coding form a pair, pair programming will somehow regress to "solo programming". The one with more experience will take control of almost the entire coding, and leave the other one to be boring and passive during developing. As a result, in this course's project, some kinds of research should be done beforehand to match people with more similar technic background, so that both members of a pair can fully participate in the process of developing and more collaboratively work on the project.

To increase the speed of developing, coding of system and tests can also be done in parallel if the interfaces are settled. When we can a set of accurate signatures if functions and methods, their behaviors are entirely predictable and, as a result, testable. By simultaneously finishing the coding and testing, we may expect that less time will be taken, and as soon as the coding completed, debugging or refactoring driven by test can start immediately.

With the successful experience of using extreme programming, and the customized improvements based on the problems in the past, it is expected we can reach a good start of this challenging but rewarding experience.

List of Rules from Reading:

* Unclear writings can be caused by:
  + Abstract and turgid concepts
  + Disjointed sentenced
  + Technical language
* The first two principles of clear writing
  + The subjects of the sentences name the cast of characters
  + The verbs that go with those subjects name the crucial actions those characters are part of
* When you turn nouns into verbs, you have to use logical opera­ tors like because, although, and if to link the new sequences of clauses.
* Some characters do not appear in a sentence at all, so that when we revise, we have to supply them
* Pattern to fix useless nominalizations:
  + When the nominalization follows a verb, with little specific meaning, change the nominalization to a verb that can replace the empty verb
  + When the nominalization follows "there is" or "there are*"*, change the nominalization to a verb and find a subject
  + When the nominalization is the subject of an empty verb, change the nominalization to a verb and find a new subject
  + When you find consecutive nominalizations, turn the first one into a verb. Then either leave the second or turn it into a verb in a clause beginning with how or why
  + We have to revise more extensively when a nominalization in a subject is linked to a second nominalization in the predicate by a verb or phrase that logically connects them
* Useful Nominalizations:
  + The nominalization is a subject referring to a previous sentence
  + The nominalization names what would be the object of its verb
  + A succinct nominalization can replace an awkward "The fact that"
  + Some nominalizations refer to an often repeated concept
  + We often use a nominalization after there is/are to intro­duce a topic that we develop in subsequent sentences
  + And sometimes our topic seems so abstract that we think we can write about it only in nominalizations. Here are two passages about an abstract principle of law. In the first, the ab­stract nominalization recovery in equity acts virtually as a char­acter. It "requires," it "recovers," it "relaxes," just as a real character might
* Passives and Agents
  + In addition to avoiding abstract nominalizations, you can make your style more direct if you also avoid unnecessary pas­sive verbs