|  |
| --- |
| Cegep Heritage College |

Development Methodologies

|  |  |
| --- | --- |
| To: | Richard Chan |
| From: | Philip Dumaresq |
| Date: | August 31, 2017 |
| Re: | Methodologies |

This report will cover which development methodology to be used for the development project system, discussing the pros and cons of agile development, waterfall development, and throw away prototyping and recommend one for the development of the project. It will also cover the differences and importance of both user stories and use case diagrams to justify whether ach should be included in the system. Then I will discuss the differences between functional and non-functional requirements so that they can be properly outlined for this system, followed up by which UML diagrams are key to have in the final documentation and reports.

Depending on the type of system being designed and what the user requirements are, different system development methodologies should be used. The three different methodologies that will be focused on are waterfall development, throwaway prototyping, and agile. Waterfall development is a very structured method of approaching software development. You do your system analysis, you write your documentation, you develop, you test, you deploy. You do everything step by step and don’t move on until you’re sure you’re done. Because of all this structure, it can be a time consuming methodology. Waterfall development is good if you have very rigid customer requirements and a long time to complete the project, however once you have fewer requirements from the product owner, waterfall begins to fall apart. This is where throw away prototyping comes into play. Throw away prototyping a method where you’ll develop a prototype, present it to the system owner, get feedback, the restart and make the next version better. This is great regardless of the time frame of project, and you can begin with little to no user requirements. You can build something loosely based off a conversation, throw something together, and then scrap it and change it based on their wants and needs. This makes it ideal uncertain owners. However, with an end goal in sight, but a desire to push the system out as soon as possible with changing requirements, agile is fantastic. Agile is done in two-week sprints, where each iteration you add more and more functionality. Agile requires very little from the product owner to begin throwing something together. Because of this cyclical process, within two weeks you can have a first version up and running, then just keep changing based on the user’s needs. However, with much larger or complex projects, agile can be too quickly overwhelmed with a lack of documentation and the code can quickly become messy. Because we’re going to be developing our system relatively quickly, and they won’t be huge systems, I’d recommend moving towards agile development. Agile will allow us to adapt to changing needs, give us a greater deal of flexibility in our timing amongst the team, and will promote lots of communication for our first real system development.

Focusing on agile development, two important types of documents are user stories and use case diagrams. The two types of documents are similar, however there are significant differences in how each one is used. User stories are useful to get a snapshot of everything that needs to be included in the system. They’re one-sentence explanations of a single required functionality of the system for one actor. An example of a use case would be something like “As a customer, I would like to be able to sign up for this service, so that I can access these software tools. Then they allow you to ensure that everything requested, works. Use case diagrams are a slightly lower level version of user stories. They take all the data out of all user stories and build that into a diagram. So, you have each use case, which is what the user wants to be able to do, then you have each actor in the system, where each actor is a different person across each user story. For each actor that shows up, you want to say what their use case was, and what other systems (or actors) it acts against, and include all those actions and interactions in a diagram. For the development project, I would include both documents. Because the system is going to need to be maintained, and we will be going lower level than use case diagrams, I feel like they’re a very important step to keep in to show the steps that we used to get from user stories to our activity or sequence diagrams. It will also allow people in the future to better understand the system.

When developing a system, some things that need to be identified include the functional and non-functional requirement. Functional requirements are things that describe what the system must be able to do. For example, one functional requirement for many websites is when you’ve signed up for an account, the system must send you an email to confirm your email address. That’s a functional requirement because it describes something the system does. A non-functional requirement however, focuses on constraints that must be put on the functions of a system. Non-functional requirements are often constraints on things like the required robustness of the system, how secure it is, how scalable it is, or how testable it is. Following the functional example, a non-functional requirement for that functionality would be that a confirmation email must be sent within 2 hours of signing up for the website.

UML models are also a very important part of system development. They allow you to capture a snapshot of both the static and dynamic aspects of a system all in one diagram, which can be incredibly useful in thinking about how to design the actual algorithms, figuring out how sub-systems interact, and allowing people to look back at them to be able to properly maintain a system. The UML diagrams I would absolutely construct would be use case diagrams, because they can be treated as a list of everything the system must be able to do. I would also include sequence diagrams, because they’re kind of like a high-level image of the pseudo-code of an algorithm and how each class in a system ties together. They show how many different parts work together while giving developers and idea of how the code should work. I would also include class diagrams, because they’re useful for the developers to get the specifications on how to use a class, and useful for people developing the database to know what to include in the modelling. Including activity diagrams, package diagrams and deployment diagrams is also very useful, however I think the other 3 are the most important ones to include.

In conclusion, this report has covered 3 different system development methodologies, waterfall, throwaway prototyping, and agile as the methodology of choice for the system. I’ve covered the differences between, the importance and inclusion of user stories and use case diagrams, and discussed the differences between functional and non-functional requirements that must be defined for a system. For the development project system, I’ve also described which UML diagrams I think are key in the documentation for the system.