Corey Gumbs Aug 28, 2015

Software Development Life Cycle Comparison

For these assignments I have chosen 2 SDLC models to discuss that I have recently been involved in with out even knowing it. My partner and I are working on an app for our personal company while in the beginning of a start up phase for another company. We don’t have a clear approach to either project at this point, or up until this assignment, I didn’t believe we did. Now I see that we have some semblance of SDLC models on both projects. I find that thought the two types I will discuss are more closely to how we are approaching these projects; I do see some other methodologies also in our planning.

I always assumed that creating a website, programs, etc. always just started with someone coding. I guess I can blame the movie industry for my view on “Cowboy Coding” or Code and Fix. I am a “hands on” person and usually start doing projects without any careful planning and make adjustments as I move along. This has also been how I learn the fastest as well; sink or swim as the saying goes. As a software development process, I can see how Code and Fix would seem like the most viable solution. You just get started and the “writing” of code helps to give the impression of progress.

Writing the code immediately is tangible, it delivers an instantaneous result. I find that it is symbiotic with chaos model development, as you are writing you are handling the biggest bugs first. From a software development point of view the only benefit I see of it is that you have something tangible in front of you once you write and interpret/compile your code immediately. From the perspective of a lone programmer or extremely small team, It may be good to get a real bare bones, simple prototype for demonstration purposes, but its not a good option for long term product development.

I don’t see any other benefit. Because there is a rush to get a product together, or out this type of development produces a low quality, mediocre product at best. Should the product take off, because there was no clear design, plan, or process, bringing team members on or into the project seems it can be a time consuming, and error-ridden process. Because there are no clear roles, directives, goals, etc; “Cowboy Coding” also leaves a lot of room for things such as egos, personalities, etc to overtake a project as team members are added. This can cause more issues than results.

Lastly, this particular type of development can lead to a lot of errors in documentation, revisions, etc as since most people are making changes as they go along, they may forget or not have time to take notes on every little nuance or issue which can lead to bugs, delays, and conflicts in direction of the development process. This is how I have approached my learning, and have been slowly moving away from it as my skills become better and projects harder. I have also been blessed that my partner is a software engineer and has been teaching me how the process works over the past year.

Recently he mentioned that he was going to start teaching me the Agile/Scrum process, as this is the process we will most likely begin to use on our projects. However, I do not know enough to comment on it. However, one methodology that caught my interest is the Iterative and Incremental process, but mainly with an emphasis on the Incremental process. This process is much better than the Code and Fix method. It uses a more structured approach by utilizing the Waterfalls methodology to break the process into smaller segments. Because a particular segment or requirements has to be completed or met, there is control on the outcomes as well as progress on the project. These requirements ensure goals/or results are met and ensure a product not only addresses issues such as code base, security, usability, but also a quality product is produced meeting at least an acceptable minimal and viable product.

The incremental process also allows bringing on necessary team members when you need them rather than have them on board from the beginning, which can exhaust resources, convolute communication and directives, etc. Allowing requirements/goals to be met ensures that all issues have been addressed. However it can also have its drawbacks. One draw back is budget, if there isn’t a viable solution, or a particular process fails to meet requirements, this type of delay can extend the cost of the project. Another draw back of this process is it can also draw back on the creative or innovative aspect of the process. Time requirements may not allow for the deviation in the “normal” process of coding, or using certain languages or design. It also can breed an environment of all products having the same feel and design of other products on the market. This can be dangerous for both the development company and/or their clients.

As for the iteration component, it helps in ensuring that all issues are continuously reviewed and improved each time around. It should in theory ensure standards are followed or improved on, code base meets any established requirements, and that the product in that particular phase is functioning as planned. The issue with iteration is that goals/requirements must be met before the project can be pushed forward, it can cause delays and increase costs. If it isn’t in the plans from the beginning an issue like a bug that could be fixed with a patch or revised code after product launch can cause a product to hang and increase its cost because it continues to fail the iteration test. On some projects this may be acceptable in their quality assurance process but for others it may not always work in the best interest of the development and deployment of a product. However, for me iteration on my work helps me find issues, clean up code, and improve the quality of my work.