Agile Methodologies Proposal for GUS

Joran Beasley

Contents

Introduction

For the design of gus I would recommend a combination of several Agile Methodologies. I will start with a quick discussion of what each of the 3 Agile models is and some pro's and con's for each of them. I will conclude with my views of how we could incorporate aspects of them into GUS.

Various Agile Methodologies

Feature Driven Development (FDD)

Feature Driven Development is a type of spiral development cycle, where in each iteration of the cycle you focus on specific a specific feature of the system you are engineering. The work flow for feature driven development is to Develop an overall model, then from that model you build a list of features that have been identified. Once the feature list has been built, a small group of features that should be feasible to complete in a two week time window is meted out to various teams. after each feature is coded and passed unit tests and quality inspections all the parts are put together to form a cohesive whole. each stage is marked by 6 milestones ; Domain Walkthrough(exploration),Design,Design Inspection, Code, Code Inspection, Promote into build. Feature Driven Development is well suited to medium to large scale systems with many feature sets, and well designed requirements documents. It would not be well suited to small scale projects, or projects with uncertain requirements. There are several pre-existing technologies for progress tracking in a FDD framework, one such example is fddtools (http://fddtools.sourceforge.net/)

Test Driven Development (TDD)

Test driven development focuses on very short development cycles, in which the team is given a test that the system initially fails, their job is to pass the test before the end of the cycle. The process is usually to add a new test each iteration of the spiral, then run all tests on the unmodified system to ensure only the new test fails and none of the old tests fail, you then write production code in order to pass the test, once you pass the test re-run all tests to ensure you did not break any of the other tests. lastly you must clean up and refactor the new code , then repeat the cycle until the system is complete. TDD is well suited where the developers may need to take occasional small steps, and complicated systems that are easily testable or can be broken into easily testable parts. It is poorly suited for projects that are not easily testable for

correctness (ie. User Interface), or where the use cases are not clearly defined (as it requires good use cases or you may miss core functionality). There are several free unit test tools available online such as cppUnit (http://sourceforge.net/apps/mediawiki/cppunit/index.php)

Scrum

Scrum involves a lengthy scrum planning session at the beginning of each scrum, followed by quick update meetings everyday for the duration of the scrum(typically one week to one month), and a debriefing meeting at the end of each scrum, where each scrum could be viewed a cycle of the spiral. There are 3 main roles in scrums the ScrumMaster , the ProductOwner , and the Team. The ScrumMaster maintains the scrum process (much like a project manager). The ProductOwner represents the stakeholders in the project. the Team is a cross-functional small group that does the actual analysis, design, implementation, testing. All role fall into two distinct categories the Pigs, and Chickens. Pigs are committed to the project while in the scrum(the ones with their bacon on the line), and the Chickens are the ones who have a stake in the project but are not part of the development process(all they lose is a few eggs) . This is a system well suited to medium to large scale projects especially projects in which the requirements are fluid and changing with each iteration of the spiral, as one of the key concepts in scrum is that requirements change, and it is a methodology that adapts very well to requirement changes. Like most spiral design methods it is not well suited to small systems (like projects for 100 level CS courses.) There are some free tools for managing scrums , such as scrumworks (http://danube.com/scrumworks), although most of the good ones cost money, but then again scrum management may just have to be mostly manual using whiteboards, etc.)

Rejected Methodologies

I considered and rejected the following methodologies. Pair Programming adds significantly to the overall development process(at least in my opinion, there is some supporting evidence available at wikipedia.) and adds very little in terms of code correctness over the other methodologies(especially TDD). Extreme Programming requires pair programming which I have already decided to avoid, it also lacks the requirement of having a fleshed out design in advance and can lead to scope slowly increasing until it reaches an unmanageable level. Cowboy coding was thrown out , because it just seems to hectic, and that it advocates death-march code-a-thons.

Relating it to the Developement of GUS for CS383

My Recommendation is that we use a combination of all 3 of the methodologies I have mentioned. Primarily base the development cycle around FDD, where each cycle of the main spiral is made up of the different subsystems of GUS. but each cycle of that is made up of a few cycles of test driven development. I also believe we should implement the lengthy pre meeting(one full class) and regular quick meetings of a scrum, but not quite daily (10-15 minutes at the beginning of each class plus 1 extra quick meeting each week). FDD will work well as the primary development model largely because it has some well established (and freely available) tools for managing and tracking FDD projects.