Mike Solomon
CS 383
10/23/10

Feature- and Test-Driven Development for Gus

After reviewing each method, I think using a combination of Feature-Driven Development and Test-Driven Development is most likely to lead to a successful final product, as well as the most useful experience for our futures in the industry. Indeed, the recommendation from industry for our course was to study Test-Driven Development, and I find that a very good indication that we should be familiar with not only the idea, but the actual execution of it.

Test-Driven Development is a technique that is very close to the coding itself. It does not define requirements, design, implementation, verification and maintenance steps like the waterfall model. It does not truly have the explicit design phase that Extreme Programming has found the need for. It seems to me that it is a methodology for actually writing code, and perhaps some directly related tasks, but not a full-fledged development process. For example, its description in the texts linked from the lecture notes do not mention anything about the design process – how are developers to coordinate? How are they to know what tests to write (based on what code is needed)? Therefore, I propose that while Test-Driven Development should be the central and most strictly adhered-to method for the writing of code, the comprehensive software development process should be guided instead by the Feature-Driven Development ideology.

I do not think the strategies of Feature-Driven Development and Test-Driven Development are mutually exclusive – they actually deal with complementary aspects of software engineering. The process would be relatively straightforward. At the highest levels, Feature-Driven Development would rule, but at the lowest level – the coding process – Test-Driven Development would come into play. I have broken down how I envision the process to work during the different phases of development, many of which will happen during each iteration of our development cycle.

Overarching Model

The first step is one we would not repeat, but merely ensure hasn't changed at each iteration. This step is part of FDD. This is when we define the scope of the system, something we have already done a fair amount of. We then go into detail in each subcategory and model what each piece will need, then merge this back into the overarching model, adjusting it as necessary.

Build Feature List

This step is done once at the beginning, and checked after each iteration. It is a part of FDD. The model developed in the last step is used to break the domain of the problem into subject areas. In each subject area, a list of features is compiled. Features should be short and specific, e.g. ‘send an e-mail message.’ If they aren’t, we are to break the task down into smaller subtasks. This list should be sorted into subsections, and by priority within each subsection. This list will change as time goes on, due to the implementation of tasks and alterations of feature requirements.

Plan by Feature

This step is done at the beginning of each iteration. It is a part of FDD. Once a feature list has been created for each sub-domain, a plan for development can be established. A feature or set of features is assigned to the head of a sub-team that will be responsible for the development of a given feature. Usually this will be in the form of a class that a group leader is personally responsible for the contents of.

Design by Feature

This step is done during each iteration just after the planning of a given feature. It is a part of FDD. A design package is to be produced for each feature. This means considering design options and alternatives, updating the object model with new classes and class contents, and creating a calendar for the completion of sub-tasks assigned to specific team members or groups of members. The team must work with other teams whose classes are involved, possibly producing UML sequence diagrams to promote understanding of the relationships between classes. The overall design must be inspected by the teams to ensure functionality isn’t duplicated or unnecessarily complex.

Build by Feature

This step is done per feature, and is therefore done multiple times during each release iteration.   This step is a part of FDD, but TDD is used during the actual coding. Instead of just writing the code and inspecting it as FDD demands, the full rigor of TDD will be used. First a series of unit tests to test the feature must be implemented, and must fail. Then, the bare minimum of code is written to pass the battery of tests, and only to pass the tests. Once it does, the code must be refactored (which is to say, improved where needed). Then the entire set of tests is run again. So long as nothing fails, the code should be committed to the main repository (including, of course, the test code).

Repeat

Once all features assigned to a team are complete, the process begins again, especially the reassigning of features and the subsequent designing and building of the relevant features. Multiple iterations and releases are characteristically “agile.”

The complementary natures of FDD and TDD would make for an effective software engineering process. FDD would be used to supply design and task assignment to sub-teams within our class, and TDD would ensure rigorous and effectively micro-designed features within the whole. We would gain experience with two agile methods simultaneously and use them do effectively develop a useful product. Therefore, I submit that FDD and TDD combined would be most effective for our environment.