## Introduction

For the project, we’ll be using a selection of agile technologies including feature-driven development (FDD), test-driven development (TDD), and scrum.

## Feature-Driven Development

The key idea of feature-driven development is to break down the work required to implement a system into small units of work called ***features***. Each feature is implemented in its entirety before moving to the next piece or work. A feature is often a smaller unit of work than a use case. This avoids the need to estimate partial completion of work; a unit of work is either completed, or not. Feature-driven development is often used for large teams (hundreds of programmers or more) and is often associated with substantial up-front architectural work. We’ll take a more agile approach with architecture developed ***just-in-time***, and refactored as needed. For further reading on FDD, see the following:

<http://www.agilemodeling.com/essays/fdd.htm>

<http://agile.dzone.com/articles/introduction-feature-driven>

<http://en.wikipedia.org/wiki/Feature-driven_development>

<http://csis.pace.edu/~marchese/CS616/Agile/FDD/fdd.pdf>

## Test Driven Development

The key idea in test-driven development is that, rather than writing a class and then thinking about we can test it, we write the test early. Writing the test helps clarify just what the class should do, so it’s a good idea to do this before writing the class, or as we are writing it. It’s normal to have failing tests on code we are actively developing – in fact we expect it. Our focus is on refining our class until it passes the tests. Our tests will be organised into an automated framework that runs all tests so it will be easy to verify that our new class does not break any existing code. You can read more about test-driven development here:

<http://en.wikipedia.org/wiki/Test-driven_development>

<http://www.agiledata.org/essays/tdd.html>

<http://java.dzone.com/articles/agile-decompiled-test-driven>

<http://msdn.microsoft.com/en-us/library/aa730844(v=vs.80).aspx>

<http://radar.oreilly.com/2014/03/the-case-for-test-driven-development.html>

## Scrum agile development

Scrum is an iterative development approach in which development is carried out in sprints. In each sprint (iteration), a small set of features or use cases is taken through the complete development cycle, including testing and peer review through to a ***ready for release*** state. Each sprint is strictly ***time-boxed***; we’ll use one week, but two weeks is common in industry. There’s no such thing as 95% complete – each feature is complete, or not, at the end of the sprint.

The development starts with iteration zero: The task here is to generate a complete list of features (or user stories or use cases) and add them to a ***backlog***. The backlog is a living document and new features can be added at any time throughout the project. Each entry on the backlog is a unit or work that can be potentially implemented in full during a sprint cycle. Aim for about five hours work as a guideline.

Each entry on the backlog should have an estimated time to implement in hours and be given a priority. You may also want to note any dependencies. Priorities should be structured into a major classification (must do, should do, could do, won’t do, unclassified) and a minor classification (basically sequence within the major classification). Each team will have to negotiate a process for agreeing on priorities.

Each sprint after iteration zero starts with the scrum master sorting the backlog into priority order and selecting the features that will be implemented in the sprint (normally those with the highest priority). These are then allocated to individual members of the team. Individual team members then work on their assigned tasks to implement and test the feature, undertake peer review, and get a ***sign-off*** from the team. Each team will have to work out the process for sign-off. At the end of the sprint, signed-off features are added to the completed tasks. It is up to the scrum master of the next cycle whether unfinished features are re-allocated to the same team member, allocated to another, or returned to the backlog.

The size of the backlog is measured (in estimated hours) at the end of each sprint to produce a **burn-down** chart of progress. We’ll use the **iceberg** variant (http://alistair.cockburn.us/Earned-value+and+burn+charts).

For further reading:

<http://en.wikipedia.org/wiki/Scrum_(software_development)>

<http://scrummethodology.com/>

<https://www.scrumalliance.org/why-scrum>

<https://www.scrum.org/Resources/What-is-Scrum>

## Sample Backlog

Here’s what a sample backlog might look like:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Feature | Depends on | Priority | Hours |
| **Backlog** |  |  |  |  |
| 11 |  |  |  |  |
| 12 |  |  |  |  |
| 13 |  |  |  |  |
| 14 |  |  |  |  |
| 15 |  |  |  |  |
| 16 |  |  |  |  |
| 17 |  |  |  |  |
| 18 |  |  |  |  |
| 19 |  |  |  |  |
| 20 |  |  |  |  |
| 21 |  |  |  |  |
| 22 |  |  |  |  |
| 23 |  |  |  |  |
| 24 |  |  |  |  |
| 25 |  |  |  |  |
| 26 |  |  |  |  |
|  |  |  |  |  |

The first column is just an ID. I’d suggest simply allocating sequential numbers. The next column is a short name for the feature. The third column is just a list of the IDs of other features that have to be completed before the feature can be implemented. The fourth column is the priority code (discussed in the next paragraph) and the fifth column is the estimated time required to implement the feature. Once allocated we won’t change this; we’ll track the project by measuring estimated hours actually achieved.

I’d suggest a simple priority code:

|  |  |
| --- | --- |
| 4nn | **Must do** features (nn = priority sequence where high = do first) |
| 3nn | **Should do** features (nn = priority sequence where high = do first) |
| 2nn | **Could do** features (nn = priority sequence where high = do first) |
| 1nn | **Will not** do features (nn = priority sequence where high = do first) |
| 0 | Not yet classified |

A simple ***sanity check*** should verify that no task depends on a lower priority task. In each sprint, the scrum master will then sort the backlog in descending priority code and allocate from the top of the list having regard to dependencies and the workload and capabilities of team members. Dependencies can be crossed-off as the referred to features are implemented.