Project Methodology

elo9@aber.ac.uk

February 10, 2017

Contents

1 Feature Driven Development

Feature Driven Development (FDD) is a plan driven methodology for development in software engineering. FDD has well defined roles, such as Project Manager and Chief Architect, and develops a system by designing and building features. Features are normal single operations or functionalities that are desired by the Domain Expert (Customer). Several features are created during every iteration which is a period of time (normally 1 - 3 weeks) agreed upon at the begining of the project for when the next iteration of the software will be complete. FDD follows a five step plan with the final two steps being carried out repeatedly for each feature in the feature list and multiple times per iteration [1].

1. **Develop an Overall Model**: At the start of a project, an overall model of the system is created with the Domain Expert present using Domain specific language. This model is designed to form the basis of how the system architecture is likely to be implemented. The model should cover all entities related to the system and will consist of various UML diagrams. FDD also uses colour UML, where different classes are coloured to represent real-world classifications of objects such as, a role in the system or an interval of time.
   Roles: Domain Experts • Chief Programmers • Chief Architect

2. **Build a Feature List**: A feature is an isolated function that is required for the system to perform as the Domain Expert expects. A Feature should be written in Domain specific language and model a business activity. At this stage, all features are written into a list but have no weighting (prioritisation). Features are designed to take no longer than 2 weeks to develop. Features are written in the form:
   '<action> the <result> <by|for|of|to> a(n) <object>'
   An example of this is:
   'Display the contents of a shopping basket'.
   Roles: Domain Experts • Chief Programmers • Chief Architect

3. **Plan by Feature**: An overall development plan regarding teams to be assigned to each feature, feature prioritisation, scheduling of developer teams time and features to be completed in each iteration is produced. At this stage, the development teams are created, and the features to be developed are distributed amongst the chief programmers in charge of those teams. Furthermore, dependencies between features are established to ensure the correct sequence of development for the project.
   Roles: Project Manager • Chief Programmers • Chief Architect

4. **Design by Feature**: A single feature from the 'inbox' of features is selected by the chief programmer and then a design is produced for the implementation of that feature. The design for the feature will include specific UML diagrams as well as the creation of interfaces and function definitions. The overall design for the system will most likely be updated at this point if required.
   Roles: Chief Programmers • Development team

5. **Build by Feature**: Once the feature has been successfully designed, the classes and methods associated with that feature can be implemented. This is followed by a code inspection and unit testing. Providing this is successful, the change is promoted to the build and becomes part of the clean code base ready to be shipped in the next iteration.
   Roles: Chief Programmers • Development team

FDD is normally considered for larger projects as it provides a framework for distributed development by dividing developers into smaller teams that tackle features one at a time. Furthermore, the up front planning stage is generally indicative of projects that are more stable as, whilst the it can be adapted throughout the process, the overall model is normally added to and the core architecture remains reasonably static.

As per the outline project specification, the steps to complete this project seem well defined and I believe the project would be best suited to having some form of up front design. Furthermore, the continuous integration (CI) style of FDD offers a good way to produce a functional prototype at various stages of the project which is a significant aid for the mid project review and potentially showcasing a prototype at this years science week. However, this is a university project and therefore I will be the only member of the development team.

2 Single Person FDD

In order to successfully use FDD for this project, I will have to make several adaptations to the normal processes of the methodology. The most notable is the developer team is only myself and therefore, I will be required to assume many different roles throughout the project. In the initial stages (1 and 2) that require a Domain Expert (Customer) my supervisor will fulfil this role and I shall be both the Chief Programmer and Chief Architect.

Steps such as establishing developer teams and scheduling developer teams work throughout the project have been made easier and this will reduce the steps required in stage 3. The main tasks to consider at this point will be the prioritisation of features and the scheduling of features per iteration.

Stages 4 and 5 will remain similar to the original specification. I will take the role of both the developer team and Chief Programmer and perform both the design and building of features, as well as the updated to the overall model. I will be using GitHub for version control and will create an issue that will correspond to each feature in the feature list. To allow for a code review/walk

through, I plan to leave completed features as open pull requests for a day before reviewing the code. This should ensure good quality code and this process will be followed for all pull requests unless they are urgent, or highly dependant on further development.

3 Additional processes

To ensure good quality code, I plan to incorporate several additional processes alongside using FDD.

3.1 Test Driven Development(TDD)

TDD will allow my project to ensure that the code is written with the unit, and system, functionality in mind. By writing the tests for the code first, I can be sure that the correct functionality is implemented and I have tests in place to ensure that if a depending part of the code base changes, a test will fail and alert me to the problem. As I am using GitHub for this project, I hope to pair potentially pair this with CI tool for running tests such as Jenkins [2]. Jenkins will be able to run my tests automatically on every pull request I create to ensure that the code base is running correctly before being merged with the master branch.

3.2 Coding standards

By introducing coding standards, code readability and consistency will be improved. Furthermore, many accepted standards exist for languages already and Jenkins has tools that allow you to check both style and syntax for errors as well as coding standards. An example of this for python is Pylint [3].

3.3 Refactoring

Refactoring will allow be a process that could potentially take place as frequently as after, or during, every feature. To ensure the quality and readability of the code, it is good practice to change or improve the way the code is written without changing the functionality. Refactoring, will often lead to a better more understandable code base and when paired with TDD, will be easy to do as the test can ensure that the functionality of the refactored code still adheres to the criteria of tests. Refactoring will take place as part of the TDD process using Red-Green-Refactor.

4 Bibliography

References

[1] S. W. Ambler, “"feature driven development (fdd) and agile modeling",” 2014, accessed Feburary 2017. [Online]. Available: http://agilemodeling.com/essays/fdd.htm

[2] Jenkins, “Jenkins - build great things at any scale,” 2017, accessed Feburary 2017. [Online]. Available: https://jenkins.io/

[3] LogiLab, “Pylint - star your python code,” 2017, accessed Feburary 2017. [Online]. Available: https://www.pylint.org/