Feature Driven Development & Empirical Modelling

0405426

Abstract

Feature Driven Development is a new software development methodology and its appeal lies in its natural applicability. This paper is an introduction of Feature Driven Development (FDD), with incorporated analysis of its likeness to Empirical Modelling. Out of this analysis, specific challenges for EM have been highlighted. Further, a teaching model, which describes the formation of features in FDD, has been produced using LSD principles. Suggestions for further work have also been detailed.

1 Introduction

Software development is a primary bond between industry and computation. If Empirical Modelling is to succeed as a foundation for computation, it must situate itself around software development such that it appeals to the industry as an alternative or a complementary framework.

Feature Driven Development is a fairly new methodology but its appeal lies in its natural applicability, as claimed by the founders of the methodology (Palmer, 2002). This is inline with the purpose of Empirical Modelling to make computation more appealing to the natural human disposition of computation through experimentation.

2 Feature Driven Development

Feature Driven Development (FDD) is a model-driven, iterative software development methodology. It begins by establishing an overall model. It then continues with a series of two-week "design by feature, build by feature" iterations. The features are small, client-valued functions and outputs. There are 5 Process Steps within FDD (Figure 1).

2.1 FDD Activities

In the first step, the client and the development team work together to develop an overall model of the client's domain. The client presents walkthroughs of the context and required system. While the development team produce models from this information. A process of adjusting the model shape results in a final agreed-upon model. It can be seen that this step is essentially interactive situational requirement gathering, with results that express knowledge about the required system, which is very much inline with Empirical modelling thinking.

The second step allows the team to formulise the information gathered in the first step through the production of a feature list. As mentioned above a feature is a small, client-valued function or output. It is small because a key concept in FDD is to generate results every two weeks. Therefore a given feature must take less than two weeks to implement. The team creates a feature list by identifying features,

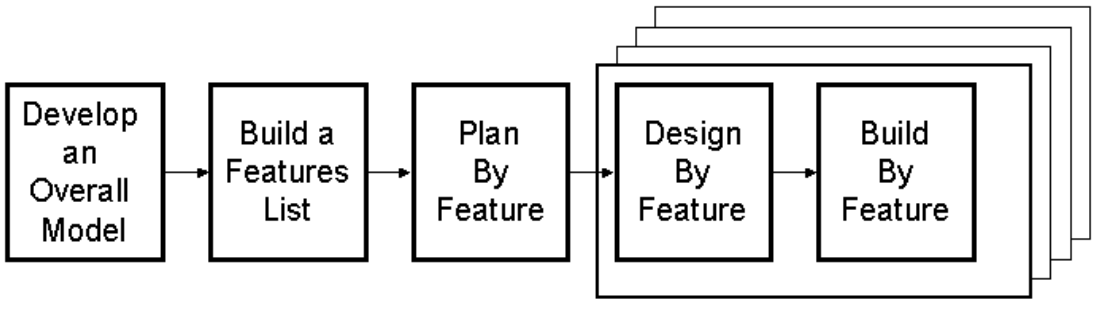

Figure 1: *The 5 Process Steps within FDD* (Palmer, 2002)

prioritising them in accordance with client satisfaction and weighing them to ensure they can be completed in less than two weeks (or else dividing them into smaller features). Features are also grouped together into feature sets to reflect dependencies and associations between features.

The third step involves using features to plan the development and organise the team in preparation for implementation. In planning for development, features from the features list are ordered to reflect the development sequence and transformed into milestones. In terms of team arrangements, a chief programmer for each feature set is selected to lead each two-week cycle involving that feature set. The chief programmer then forms a feature team by selecting a group of developers to work with, on assigned feature sets. Team management is an essential concept in software development and it determines the scalability of a methodology to large projects.

The last two steps are iterative and each iteration lasts 2 weeks. A chief programmer takes the next feature and identifies associated classes. The feature team work together on sequence diagrams and general design of features. Coding, testing, integrating and inspecting takes place in the final step of FDD. By this point, there is little room within FDD for experimentation and a more structured approach is adopted. A key benefit of this is that it ensures the project is completed within a reasonable time frame.

2.2 Key Principles in FDD

FDD is based on the idea that results reflect success and not the steps used to achieve those results. The founders of FDD have attempted to achieve this through making FDD a naturally applicable methodology. One of the main concepts used to accomplish this aim is to avoid process over-specification. In other words, a reduction in the amount of process description provided to team members. For this, a process description notation is has been provided in the form of a template named ETVX: Entry, Task, Verification, and eXit (Coad, 1999). The idea being that for each process step within FDD, the entry criteria is given, followed by a list of tasks and then methods to verify that the task has been completed and finally an exit criterion.

Process description within FDD in general is based on similar thinking to Definitive Notation, in the sense that scripts are never completely closed and definition only stabilise as meanings are negotiated. This is reflected in the observation that many specific details in process description of the later steps in FDD have been left ambiguous and open. For example, integration testing is left at developer's discretion in that there is no mention of integration testing within the process description; although there is a separate discussion about it and it has been considered by the founders of the methodology (Palmer, 2002).

Further, FDD was founded in 1997 and as of yet, there have been two different definitions provided by the founders for the specific descriptions of process steps; one definition is the original and the other is a 'new and improved' version. This reflects the idea that new insight and experience can give rise to new definitions, or even removal of old definitions.

Another key principle within FDD is accommodation for shorter and shorter business cycles. The racing nature of technology requires shorter iterative business cycles. To explain further, consider a project with a 2-years lifecycle. This project may perhaps be unsuccessful if design is completed before any implementation is started because technology advancements will quickly outdate the output to this project. EM does not have concerns about outdating models because programming is done while modelling.

Scalability of FDD to large projects is a key strength. FDD provides just enough process to ensure this. By process, the indication is towards short design and implementation cycles,

results being produced every 2 weeks, reporting and tracking of progress and clear definitions of team roles.

The idea of features is a key attribute of FDD. A feature is essentially a sentence that has a particular composition. This, together with its small and easily comprehendible nature, makes a feature a powerful expressive tool for both the client and the development team. This concept sits well with Empirical Modelling, which also provides expressive power without the added the complication of formal proof.

A final principle that FDD shares with Empirical Modelling is that of interactive situational modelling of the user's construal. This is evident in the first stage of FDD, where a model of the problem context is created by using the referent that exists in the client's mind.

2.3 Empirical Modelling & Software Development

When considering software development, it is difficult to decide where Empirical modelling should be situated; as a complementary or an alternative or a substitute framework? (Beynon, 2006) If EM was to be an alternative to traditional software development, then there needs to be some process built into it as managing large teams requires such discipline. However this will be an unreasonable suggestion because it conflicts with a key fundamental EM principle which is the existence of truly experimental computation. With this in mind EM would perhaps be better situated as a complementary that offers its strengths at different points in a software development process.

To illustrate further, consider the shared reality that at the beginning of any project, the developers have limited knowledge of the client's domain and the client has vague knowledge about what they require. The bridging of this gap can only be achieved through providing expressive powers that both parties understand. As a solution to this, FDD provides informal feature lists and class diagrams during the developing of an overall model in the first step. However these tools of expression are unable to provide as much power as the Definitive Notation of EM.

FDD embraces an iterative approach which is unlike the Empirical Modelling approach where programming-as-modelling is used. In order to be a successful substitutive framework, the Empirical Modelling approach needs to define solutions for effective team management. Software development is a connecting link between industry and computation. Therefore it is necessary to work in accordance with industry standards and expectation. With this in mind, the EM approach needs to involve some scheduling disciplines which help meet time constraints set by the industry.

Another consideration is technological advancement. Should advancements in technology, in the commercial sense, impact results produced through Empirical Modelling disciplines? How can technological advancements be incorporated into an EM construction?

3 The Model

3.1 What to Model?

There are many possibilities for models within the above described Feature Driven Development context. It was difficult to decide what to model, primarily because so many concepts of FDD, especially in the earlier stages, overlap with Empirical Modelling principles.

The model could have resembled the lifecycle of a feature over the modelling and planning stages. This would be useful in illustrating the similarities between the nature of FDD and EM, while at the same time cater as a useful tool for FDD as a central repository for features. However, due to time constraints it was infeasible to construct such a model.

The lifecycle of a Feature consists of three main stages. First the forming of a feature which involves identifying a feature, labelling the fea-

ture using the standard template, ensuring the feature will take less than two weeks to implement and prioritising it before adding it to a Feature List. Second, the sequencing of features in preparation for development is done in accordance with client satisfaction and practical concerns. Third and last, features are grouped into feature sets and assigned to chief programmers. Having considered this, another possibility was to model simply one of the three stages in a feature lifecycle.

Further stimulus for the decision was provided while reading Meurig's analysis of software development and empirical modelling (Beynon, 2006). There is mention in the analysis of a dishwasher model (Wong, 2002), with particular attention drawn to UML depiction in the model.

During research for FDD, there was mention of a UML reporting component that particularly captures Feature Progress during FDD (Coad, 1999). The component, as shown in Figure 2, is a rectangle with a set of inner rectangles that are representative of different observables related to a particular feature set. There is use of colour to identify state-changing events. Agency is also depicted in how one of these rectangles sits in a 'Major Feature Set' with many others (as shown in Figure 3). This was another possibility for the construction of a model that captured the progress of features and reflected observables, dependencies and agencies of FDD in one place. Further, experimental interactions could be created to allow an agent to change feature set progress.

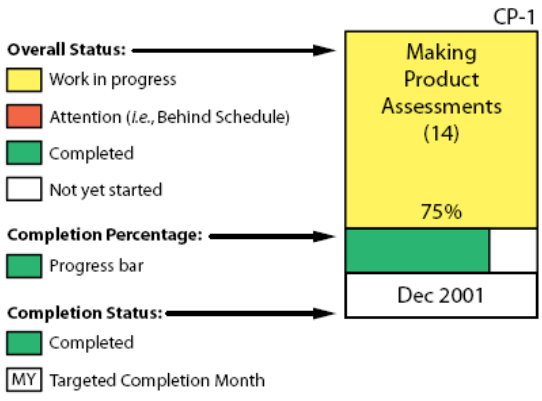

Figure 2: *UML Reporting Component that depicts Feature Set Progress* (Coad, 1999)

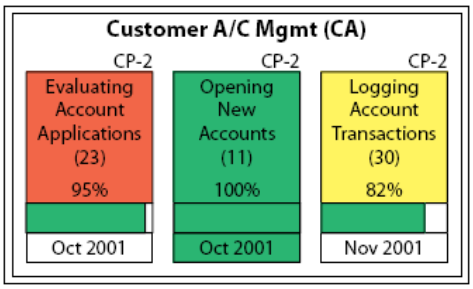

Figure 3: *UML Reporting Component that depicts a Major Feature Set containing 3 Feature Sets* (Coad, 1999)

However, additional motivation came from the dishwasher model itself. Having seen the model, it seemed like an obvious basis for the depiction of one of the 3 stages in a feature lifecycle (mentioned above). In particular, it was suitable for the first stage of forming a feature because that is the most elaborate stage and requires the most explanation.

In deciding between, a reporting model and a feature-forming model, it was decided that the feature-forming model was more appropriate at this stage because FDD is a new methodology and knowledge about how it works is more significant knowledge than the reporting of progress in FDD.

3.2 Model Description

The model is a visualisation of and an executable model of the process of forming a feature and adding it to a feature list. The model is called the Feature-list model. There are 3 windows (Figure 4); the first window (top left) allows a user to start and pause and re-start the simulation and it also provides the user with descriptions of process steps and progress while the simulation is running. The second window (bottom left) is a state-changing state chart. The third window (right) is the feature space that provides a visual abstract representation of the process of forming a feature.

The model has been created with 3 main agents in mind:

- First, it acts as a simulator for the activities of a feature-list team during the 'Build a Feature List' step in FDD.

- Second, it provides space for a modeller to express an abstract interpretation of the activities of the feature-list team. For instance, a feature is represented by a circle.

- Third, it acts as a teaching tool for anyone wishing to learn about the activities that take place during the 'Build a Feature List' step in FDD (the term student is used for this agent in the model).

Other agents include a programmer and a user interface designer, who implements and conceptualises the student's desired interaction. There is also a Process Manager who expresses the process that the Feature-list team follows as a state chart.

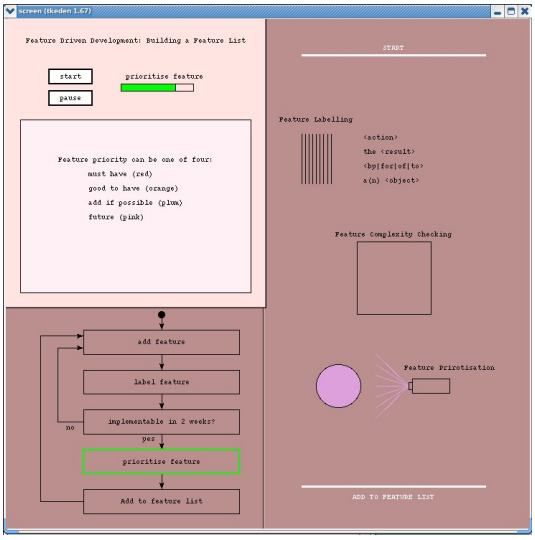

Figure 4: *A screen shot of the Feature-List Model.*

3.2 Empirical Modelling Principles in the Model

The Feature-list model works in a Multi-agent modelling environment. Through using this approach, the potential use of LSD as an alternative approach to software development was explored and it also captured the relationship between the agents. This made it easier to formalise how different agents observe the system and therefore results in producing a more full system that does justice to all involved agents as appose to the normal 'do what is necessary for the system to work' approach.

The modelling and simulation of this artefact was constructed with a view to imitate a physical experiment and observation of a 'student' viewing the process of forming a feature. This was successfully done, although it was difficult to capture the discussions of a feature-list team while the feature is being identified, named and prioritised.

One of the main focuses of the Feature-List Model is to be a teaching tool. This makes it open to exploration and experimentation. The variables defined within the script can be manipulated to depict more useful states or complete nonsense states. For example, if the user felt the simulation was too fast, it is possible to increase the `endtime` variable to make the simulation slower. As another example, the x and y coordinates of the feature can be changed to make it appear anywhere on the feature space or even off the feature space.

Different views of the problem domain have been provided to increase comprehensibility of the referent being modelled. Colour is used to create dependency between the current activity and another construal of that activity in a different window. For instance, the progress bar and state chart show the same information about the current activity but in different ways. They are indivisibly linked in change.

The user is able to start and pause the process while it's running at any point and explore what is happening at that stage at their own pace. This makes the model interactive as well as representative of knowledge.

The power of definitive notation is also explored in this model. A simple and small script is able to store state information about the entire process of forming a feature. If an alternative object-orientated approach was used, then much more effort would have been required. If an alternative, UML approach was used then it would not have been as expressive and comprehendible as this model.

3.3 Issues

It was difficult to capture the process of dividing a feature when the feature is too complicated to implement within 2 weeks. The main issue was that two or more features needed to exist on the feature space at once but it was not possible to establish how many features are likely to be in the space because a complicated feature could potentially be divided in to 100s of smaller features. It was difficult to capture a dynamic number of observables i.e. each new observable has to be given its own definition effort from the modeller.

4 Future Expansions

Some ideas for future expansion have already been explored above; in particular the reference to a central feature repository model and a progress reporting model.

The main purpose of a teaching model of this sort is to assist a user, who is new to FDD, in sense-making of features and internalising of how features are formed. The Feature-List model only explores one aspect of FDD. For this concept of teaching and simulating to be successful, further work needs to be done in terms of defining models for all steps of FDD.

A study of Features with a view to reveal the value of applying Empirical Modelling principles to Features Driven Development would be useful in establishing a more concrete relationship between EM and software development as a whole. Such a study is likely to successful because key Empirical Modelling principles already exist within FDD. Features are clearly identifiable observables that are established through experience and experimentation conducted with both client and development team present. There is always a referent present in the eyes of the client. Feature sets allow features to be indivisibly linked in change. Agents are clear to define as roles are very specific within FDD. Defining notations exist in the form of feature templates and process description notation.

5 Conclusion

Feature Driven Development and Empirical Modelling are based on many similar principles. The main one being a shared desire to be naturally applicable. This goal is accomplished in both approaches through the use of definitive notation, be it different forms of definitive notation. There also exist differences between the approaches which can be used to improve both 'methodologies'. Overall, both approaches are relatively new to software development and their contribution will only be evident with the passing of time.

Acknowledgements

Meurig Beynon for enthusiastically teaching the principles of EM and writing such wonderful lecture notes about EM.

Allan Wong for creating the dishwasher model.

References

Beynon W.M., 2006. *Software Development*. University of Warwick. [Online]. Available at: http://www2.warwick.ac.uk/fac/sci/dcs/research/em/applications/softwaredevelopment/ [accessed 05 May 2008].

Beynon, W. M., 2008. *The LSD notation for domain analysis and description*. University of Warwick. [Online] Available at: http://www2.warwick.ac.uk/fac/sci/dcs/research/em/teaching/cs405/concsys/concsyslecture4 [accessed 05 May 2008].

Coad, P. Lefebvre, E. & De Luca, J., 1999. *Java Modelling color with UML*. Upper Saddle River: Prentice Hall.

Palmer, S. R. & Felsing, J. M., 2002. *A Practical Guide to Feature Driven Development*. Upper Saddle River: Prentice Hall.

Wong A., 2002. *Dishwasher.* University of Warwick. [Online]. Available at: http://empublic.dcs.warwick.ac.uk/projects/dishwasherWong2002/ [accessed 05 May 2008].