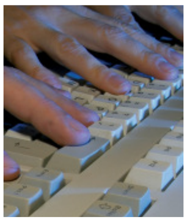
Jeff De Luca on Feature Driven Development



Interview

April 2007

it-agile consultant Stefan Roock[1] interviewed Jeff De Luca, who founded *Feature Driven Development* (*FDD*) 10 years ago. Jeff talks about the roots of FDD, the character of agile methods and the relationship of FDD to eXtreme Programming (XP) and Scrum.

Jeff argues about agile dogma: "*I am not religious about FDD as the one and only one true process, nor am I religious about process and method. What I am religious about is frequent, tangible working results, or reliably delivering working software in a timely manner.*"

When coming to the question what method is suitable for what type of project Jeff points out: "*I'm saying that the Agile methods are more suited to types of people and organisational cultures than types of project.*"

Jeff underlines the importance of a upfront high-level modeling activity: "*…there has to be some informational / analytical activity at the start to give us the knowledge to set a baseline that we can track and report against… FDD is the only agile method that gets this part right.*"

In contrast with XP, FDD has class ownership. Jeff has a strong opinion about class ownership: *"Collective ownership is code for 'no ownership'. It's not a structure I believe in."*

Interview Intro

Stefan: Hello Jeff
Jeff: Hello Stefan.
Stefan: How are you?
Jeff: I am fine thanks.

FDD Roots

Stefan: OK, let's start with the interview. What are the roots of FDD? When did FDD start?

Jeff: Well, that's a very interesting question actually. FDD started in name in 1997-1998 on a project I was running for a bank in Singapore. I had hired Peter Coad to lead the overall modelling for that project and that is how we both met.
Most of what I was doing I had already been doing for years and some of those influences date back to my time working in IBM programming laboratories in the USA. But Peter Coad had the notion of a very fine grained feature and that was an important concept. It was finer grained than the tracking I was using before that.
So I would say FDD's roots are things like the experiences I got in the IBM programming labs and then the notion of a fine grained feature from Coad in 1997. FDD was first written about and given a name in 1998.

Stefan: How did you recognize that FDD is worth an own name and method description?

Jeff: It was in the middle of that project, after I had a lunch meeting with John Gage the Chief Scientist at Sun Microsystems, that I went to back to Peter and agreed to write about the approach being used. Peter

[1] stefan.roock@akquinet.de

had asked me previously to write about the approach I used.
When I had lunch with John Gage, he talked about how he was also involved in Government think tanks and things like that. In fact, he was in the Asia region for the APEC meeting of governments (*asia pacific economic forum*). Anyway, he was talking about a particular think tank or some such thing he was involved in, and they were in a very large room, and the walls were on wheels and moveable, and each team did their work on these walls and then they did their presentation by moving their wall to the centre of the auditorium.

Stefan: That sounds familiar for people with an agile attitude.

Jeff: So then I started explaining to him how we did our modelling and requirements analysis with users by working in teams on walls, using coloured post-it notes on pieces of flipchart paper, each team bringing their piece of flipchart paper to the centre of the room to present their model, and so on. Plus I tracked and planned projects by using charts and reports on a wall (wall planning). Anyway, John got very excited and was challenging me. He said "who is your project anthropologist? who is writing this stuff down?" and so that is what got me to go back to Coad and agree to write about how I run software development projects.

Agile vs. Non-Agile

Stefan: FDD is known as a member of the community of agile methods. What are the main differences between FDD and other agile methods like XP or Scrum?

Jeff: There's a lot of differences at the detail level between the Agile methods, but I don't think it's the differences that are interesting. What's really interesting is what binds all the Agile methods together - and that is a common value system. This is captured by the four Agile value statements. i.e. we value working code more than... those statements. That is the common value system the various methods share. And it is a very good one. It takes great maturity to "get it".

Stefan: But we have a long way to go to until Agile became an accepted software development approach.

Jeff: A few years ago a common session at the major conferences, usually by some industry analyst, was about the differences between traditional or heavyweight methods and Agile methods. And these sessions would always have a slide title something like "Perceived problems with Agile" and it would say something like "Needs a customer to interact with" and "Needs good people". Um, yeah, no kidding I say! This is right. They are real problems with Agile because they are real problems with all software development. I mean, what the analyst is saying in effect is that if using a heavyweight method then I don't need to talk to a customer and I don't need good people? I think we know how successful such a project would be.
So this is nonsense. These are problems with all software development and what Agile does that is different to the traditional methods is that Agile makes such problems first class issues. Agile puts them front and centre. This is what the four value statements are doing.

FDD Modelling

Stefan: One of the striking differences between FDD and Scrum/XP is the explicit modeling process in the beginning. Some may say that's big design upfront (BDUF)? Is it?

Jeff: No, not at all. BDUF has become a pejorative phrase and with some it seems that any activity before coding is labelled BDUF. This is not helpful. If you are a project manager one of your pain points is having to answer these two questions: "How far along are you?" and "How far is there left to go?" These questions are hard to answer. They are even harder to answer with accuracy, and even harder again to be accurate and meaningful to the client. The *Develop an Overall Model* process is not a heavy and detailed modeling activity. We build what is called a *shape model*. That is, we want to identify all the classes in the domain and their connections, but not try and identify every single attribute and method in every single class. This is done in a highly collaborative way together by the developers and the users (domain experts as they are called in FDD). You could think of this as perhaps high-level design, but it also is really requirements analysis and requirements discovery itself. While we produce a shape model in this activity, the real secret to it is the incredible knowledge transfer that takes place. Models are very expressive, they are visually explicit - there is very little wiggle room in a model, and capturing our understanding of the domain (via talking to the users) in this way gives us great information and knowledge.
So, really, this first process in FDD - *Develop an Overall Model* - is an informational / analytical activity to give us the knowledge to *Build a Feature List* and then plan the project with good accuracy and coverage. Once these startup-phase processes are complete, FDD's iterations or increments through the construction phase are more fine-grained than your typical iterative/incremental method. So, how long is this so-called up-front activity in FDD? We would model with the users for about 2 weeks for every 6 months of construction time in a project. Jim Highsmith summed this up well in one of his books: Thus, for a 6-month project, Jeff De Luca will be coding in week 3 and Ron Jeffries (XP) will be coding in week 1; hardly a significant difference.
Finally, and this is a very important point: When we moved from waterfall to iterative/incremental that was a big improvement. Iterative/incremental methods are some number of slices through most of the waterfall phases. But if we are too pure in our iterative/incremental approach - that is we are also slicing through requirements and analysis - then of course it is hard for us to answer the questions "How far along are you?" and “How far is there left to go?" because we haven't even looked across the rest of the requirements yet. So, there has to be some informational/analytical activity at the start to give us the knowledge to set a baseline that we can track and report against so that we can answer those questions. FDD is the only agile method that gets this part right. Not "all of the design up front" as the pejorative use of BDUF has come to mean - but "just enough design" and note that this first activity in FDD is as much about requirements and requirements analysis as it is about high-level design.

Stefan: You say that one major effect of the *Develop an Overall Model* process is understanding the domain and requirements. That seems very similar to the activity Eric Evans calls *knowledge crunching* in *Domain Driven Design*.

Jeff: If it is achieving the same outcome, then it is a good idea. When I teach people how to abstract and adapt FDD - this first FDD process is one of the places where you could substitute a different approach. Modeling the domain in the way FDD describes it is the best way I know how to do this. But if some other approach can satisfy the goal of an informational/analytical activity to give us enough knowledge to set a baseline with good coverage and good accuracy - then that is more than fine by me.

FDD Chief Architects and Project Managers

Stefan: As you described the *Develop an Overall Model* process is a common task of various project participants. One of the FDD roles is the *Chief Architect*. Has he a special role in this first process?

Jeff: Ah, the *Chief Architect* as a role is overplayed. I am long overdue to refine the FDD processes and this is one place that will change. The correct role name is *Chief Modeler* and that's what I'm using these days. The important roles and aspects are this: we bring together domain experts (users, business analysts, subject matter experts - these are all synonyms; a domain expert is simply someone that has expertise in part or all of the domain - the business) and developers under the leadership of facilitator and an experienced modeler. So *Chief Modeler* (or *Chief Architect* in the current process description) is simply referring to that experienced modeler that can lead the group as necessary, produce a strawman model to facilitate progress, etc. These are all just role names. Roles are like hats you can wear and you can wear multiple hats. As an example, if a person is really good they could play both the facilitator and *Chief Modeler* role - Peter Coad is an excellent example of such a person.

Stefan: Ah, that's an important statement since it differs from what many people may associate intuitively with the term *Chief Architect*.

Jeff: You are right. It's a problem name - anything with "architect" or "architecture" in it is a problem name in I.T. since those terms are so heavily overloaded.

Stefan: Is there a similar issue with the *Project Manager* role? Traditionally the *Project Manager* is responsible for defining tasks and assigning them to developers.

Jeff: *Project Manager* is different and the tasks you mentioned are handled differently in FDD. In terms of the FDD processes themselves, *Project Manager* barely rates a mention - only in 3 or 4 places - and usually as the role that "forms the team" to do the next set of tasks or activities. So, the work tasks in an FDD project - *Features* - are not defined by the *Project Manager*. The high-level planning is done collaboratively with the *Project Manager* and the developers. The assignment of features for design and build is done collaboratively by the *Project Manager* and developers. The actual detailed scheduling and planning of batches of features for design and build is done by the developers.

FDD Features

Stefan: Well. The *Develop an Overall Model* process precedes the *Build Feature List* process. One might argue that the order is wrong. Don't you need to know the concrete requirements / features to build the overall model?

Jeff: No - not as these things are defined by FDD. An FDD feature is a tiny, granular piece of client-valued function. They are very small and when building the features list you want the features as fine-grained as possible (as this gives you the most flexibility in the detailed sequencing of features during the construction phase and it also gives the most flexibility in allowing work to complete where one tiny part of work is still blocked). To be able to decompose the domain into a list of fine-grained features with good coverage and good accuracy, you must have an

informational/analytical activity before it to give you such knowledge. For example, the Singapore Lending project (an FDD case study) had over 1000 features just in the PD layer (business logic layer of the app). When you understand now what an FDD feature is - of course it is quite impossible to be able to build a features list as the first activity.
The other part of your question was about what do you need to know to develop the overall model. Well, what we are doing is modeling the domain; that subset of the domain bounded by the scope of this project or application. So the minimum entry criteria to develop the overall model is that we must at least have some idea of what that scope bounding box is. Without that there is nothing to facilitate against and you can end up modeling the whole world. In project terms - its hard to hit a target if you don't know what the target is. This is how the target is made clear in FDD. In I.T. we are particularly poor at this; we make too many assumptions, we don't clearly define the target.

Class Ownership

Stefan: When it comes to feature construction there is a obvious difference concerning class ownership. XP and others propose collective ownership. FDD has class ownership. What is the reason? How strict is the class ownership handled in typical FDD projects?

Jeff: The concept of module or component ownership has long been practised and long been known to be a best practice. That's how all professional development at scale is done. Everyone can't know everything about everything - that is brittle and it can't scale. Brooks (*The Mythical Man Month*) told us so nearly 30 years ago; many others have too. For some reason, this concept has gone away in some methods and we need to put it back.
If you think of a sequence diagram for a model: Some number of classes collaborate to perform a function - such as calculating the weight of a shipment - in the product sales domain. There are 4 or 5 classes that are involved in such a feature. If you don't practice safe computing; if you don't do class ownership then when you assign that *calcweight* feature to a individual developer, that developer has to touch 4 or 5 classes in the system. He has to have knowledge of 4 or 5 classes in the system. Now, think of a very high-use class in some system - such as the *LoanApplication* class in a Lending system. Here's a class with hundreds of methods and attributes - each written by a different developer at a different date and time.... just saying this makes the hairs on the back of my neck stand up!
And what's especially ironic about this is that one of the fundamental principles of OO is encapsulation; how a class does what it does is private and internal to that class and those implementation details can vary wildly as long as the class presents a consistent interface to the rest of the world (the other classes). Well, humans naturally encapsulate. If you practice class ownership you get much better consistency of implementation and interface.

Now common sense of course applies here as it should everywhere. Class ownership is not a life sentence. If you get the Customer class you're not stuck on Customer forever. These owners can and do change throughout a project. At the end of the day, all we are saying is that a developer owns some number of classes. You could model that as Developer-----*Class. Now, if it's a small project, maybe only 4 people or so, we'll "chunk it out." e.g. “Fred - you do the customer stuff, Paul - you do the account stuff, Alex - you do the admin stuff, Phil - you do the UI.” We'll only think about it at that level.
If it is a larger project with a larger or more complex model then we'll put more thought into it. We'll look at the classes in the model and we'll think about class complexities and our developer skills. e.g. there are

two kinds of class complexity - *pervasive complexity* (which is a class that is touched by a lot of features - but it doesn't do much itself except to delegate away) and *algorithmic complexity*. So in a larger model we'll put more thought into it. We don't want to assign many pervasively complex classes and algorithmically complex classes to the same developer because we'll end up blocking on that developer during construction. In both cases - the small and the large - what you end up with is Developer----*Class. Collective ownership is code for "no ownership". It's not a structure I believe in.

Testing and Code Inspection

Stefan: Testing is emphasized by many agile methods. FDD has code inspections. What is the role of testing in FDD?

Jeff: Well you have to start by asking these other methods what the goal of testing is in their method - and the answer is "to remove defects." Well that goal gets even more emphasis in FDD - but not solely through testing because testing is one of the least effective ways to remove defects. This is not speculative - it is a very well experimented and measured thing in I.T. Go read Capers Jones for example. Inspections are the most effective form of defect removal (they are also a great team building activity as they actively propagate team culture as well as syntactic and semantic standards). So, what does FDD do about testing? Well what it does is utilise proven best practices to ship the fewest number of defects to test in the first place. We have the collaborative modeling with the users where we are pushing back hard on requirements and requirements analysis since models are so visually explicit. Then we do formal design and code inspections - but on tiny features - which makes the implementation of these known best practices so much easier. All of this is so we have the fewest number of defects possible before we get to unit testing which is a mandatory step in the FDD *Build by Feature* process.
And once you get there we do just as much for testing as any other currently fashionable testing approach. We do unit testing, we do testing automation, coverage, etc. FDD does far more for the goal of testing in other methods - defect removal.

Management

Stefan: What do managers and users like about FDD?

Jeff: What managers and users love about FDD is it's tracking and reporting. It's transparency and communicability. Describing the work to be done in terms a client-valued features and reporting against a decomposition of the business itself (the domain) resonates really strongly with most line of business development. The features list and the *parking lot chart* (one of the reporting visualisations in FDD) have been very widely received and accepted. Many other methods are using those things now too. Project managers want to be able to answer those two questions (“how far along are you? how far is there left to go?”) and communicate in a meaningful way to the client. This is what draws most people to FDD.

FDD and other agile methods

Stefan: It is common practice to mix different agile approaches in projects, like having the Scrum Meetings and Pair Programming. Are practices from Scrum and XP typically present in FDD projects also?

Jeff: Yes they are. Many of the 12 practices in XP are present of course - they are not new. A daily standup is something I've used myself for decades. In fact, daily standups were used during the early part of that lending project in Singapore (1997). The burn down chart in Scrum is similar to the Trend chart in FDD (one plots down, the other plots up). At this level there are many similarities and many differences. FDD doesn't use timebox management like a Scrum, FDD doesn't pair program. But the goals of both of those things are achieved in FDD but in different ways.

Project Types

Stefan: You said collective ownership doesn't scale well. How large are typical FDD projects? What are the largest FDD projects you did and you know of?

Jeff: Well that does depend on how "large" is defined. The Lending project in Singapore was one of the largest Java projects at the time - that one had about 50 people. The largest developer team I've ever managed in a project is 250 developers. At the small scale, I've done many 2 person FDD projects. Most projects these days should be smaller durations (most businesses aren't interested in projects that take longer than a year to develop) and so this will also scale team sizes down. I'd say most projects of significance are between 6 and 50 developers. There are of course significant projects with more than 50 developers, but it's not the majority case. I'm also talking mainly to commercial and corporate line of business development.
And I did say collective ownership doesn't scale well - and that is true, but I also believe it just doesn't work as well as other approaches. It's not just about scale.

Stefan: Would you say, that FDD is more suitable for certain project types than for others?

Jeff: No I don't think the Agile methods classify that way. By project type I mean. When it comes to this sort of thing I tell people to get Jim Highsmith's Agile Software Development Ecosystems book. That's the only book that talks about all the Agile methods, in some detail, in the one book. You read that book, and one or two of the methods are going to resonate with you and your team. And whichever methods those are, they are the ones you should explore.
In other words I'm saying that the Agile methods are more suited to types of people and organisational cultures than types of project.

The FDD Future

Stefan: Are their any special enhancements of FDD you are working on currently?

Jeff: Not enhancements to FDD as such, just describing and explaining more of it and also the relationship of such approaches to other parts of I.T. such as program management, governance, etc.

The End

Stefan: Jeff, thank you very much for the interview. I look forward to meet you in Hamburg.

References

- http://www.nebulon.com/articles/fdd/download/fddprocessesA4.pdf
- http://www.featuredrivendevelopment.com
- FDD training in Germany:: http://fdd.it-agile.de
- Web site of Jeff De Luca: http://www.nebulon.com

FDD trainings in Germany

It-agile offers FDD trainings in Germany.

In 2007 the father of FDD Jeff De Luca leads two FDD trainings and certifications in Germany:

- ***8th and 9th of May 2007 in Hamburg***
- ***21st and 22nd of November 2007 in Karlsruhe***

More information and registration at:
http://fdd.it-agile.de
or via e-mail:
info@it-agile.de