"Deliver frequent, tangible, working results"
Peter Coad

"Kanban is the science of not trying to do too much at once"
Stephen Palmer, 2012

"Speech is conveniently located midway between thought and action, where it often substitutes for both."
John Andrew Holmes

"We apply the analytic procedure in order to create a conceptual framework within which we can draw conclusions about the means by which a system solves its tasks. Indeed, we do this with the express purpose of establishing a solid foundation from which we can carry out a subsequent synthesis. This synthesis, in turn, acts to verify the conceptual model as an explanation. The process is an iterative one."
Tom Ritchey, Refactoring, 1991,

"Use models to find out how things work or to find solutions to puzzling dilemmas. Create models to communicate ideas and understand things you can?t see. Recognize models and the countless ways models are used for working, playing, teaching and explaining. Assess models for what they do and don't tell you about the real thing and how useful they are"
Boston Science Museum

"Human brain capacity is more or less fixed, but software complexity grows at least as fast as the square of the size of the program."
Gerald M. Weinberg, Quality Software Management: Volume 1 Systems Thinking, 1992

"The complexity of software is an essential property, not an accidental one. ... Many of the classical problems of developing software products derive from this essential complexity and its nonlinear increases with size."
Frederick P. Brooks, The Mythical Man-Month, 1995

"As the number of people increases, the ways they can interact tend to multiply faster than you can control them."
Gerald M. Weinberg, Quality Software Management: Volume 1 Systems Thinking, 1992

"chaos often results when a project's complexity is greater than its managers' ability to direct meaningful progress toward a goal."
Ken Schwaber, Agile Project Management with Scrum, 2004

"When people are factored in, nothing is simple. The complexity of individuals and individuals working in teams raises the noise level for all projects."
Ken Schwaber, Mike Beedle, Agile Software Development with Scrum, 2002

"It usually takes more than three weeks to prepare a good impromptu speech."
Mark Twain

"Simplicity is the final achievement. After one has played a vast quantity of notes and more notes, it is simplicity that emerges as the crowning reward of art."
Frederic Chopin

"If you can't explain it simply, you don't understand it well enough."
Albert Einstein

"Man will occasionally stumble over the truth, but most times he will pick himself up and carry on."
Winston Churchill

"The structure of a software system will reflect the communication structure of the team that built it."
R. E. Fairley

"I guess that as you go along you try to fill your tool-box, so that when you face these circumstances you have more options to choose from"
Jonny Wilkinson, Tom Fordyce's Blog, 2009

"We try to solve the problem by rushing through the design process so that enough time is left at the end of the project to uncover the errors that were made because we rushed through the design process"
Glenford Myers (via jJeff De Luca)

"Information Technology is 80% psychology and 20% technology "
Jeff De Luca, www.nebulon.com

"Perhaps the worst software technology of all time was the use of physical lines of code [for metrics]. Continued use of this approach, in the author?s opinion, should be considered professional malpractice."
Capers Jones, Applied Software Measurement

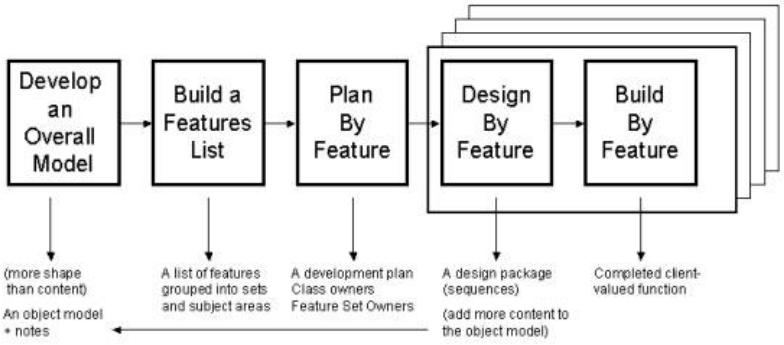
"Agile software development is about iteration not oscillation"
Jim Highsmith (paraphrased), Agile 2009

Feature-Driven Development (FDD)
Invented by Jeff De Luca, Feature-Driven development is an agile, model-centric, customer-valued-requirements-driven process for developing enterprise software.

Feature-Driven Development (FDD) is Jeff De Luca's pragmatic, agile, approach to developing enterprise software. It combines key advantages of other agile approaches such as Scrum and eXtreme Programming with model-centric techniques like Peter Coad's, 'modeling in color' and Eric Evan's Domain-Driven Design. This plus dynamic feature teams, peer reviews, and just enough design initially (JEDI), means that FDD scales easily to much larger teams and projects than generally recommended for other agile approaches. When done well, FDD produces meaningful, accurate and timely status reporting and progress tracking for all levels of project leadership and is the most satisfying way of working in a software development team that I have experienced to date.

The Five FDD Processes

The core of Feature-Driven Development is described in five 'processes', each written on one double-sided, letter-sized (A4/B4) piece of paper. The five processes are the rules of the game but like any software process, the rules are only one part of the equation; to play the game well and be successful, a team still needs ability, experience, discipline, and leadership. Feature-Driven Development is not a silver bullet.



1. Develop an Overall Model

FDD differs from Scrum and eXtreme Programming in demanding that a team invest just enough effort at the beginning of a project in exploring the structure of the problem by building an object model of the problem domain. Unlike the formal object modelling or other analysis and design activities in waterfall-style approaches, modelling in FDD is a cross-functional, collaborative, and time-boxed activity. The resulting model has just enough detail to form a good shared understanding, vocabulary (what Domain-Driven Design calls an ubiquitous language), and conceptual framework for the project. In addition, the modelling activity uncovers just enough knowledge to build a really good product backlog (called a Feature List in FDD) and initial, overall release-level project plan.

Rather than the derogatory *BDUF* (*Big Design Up Front*) term, I call this upfront modelling activity a *Jedi* approach (*Just Enough Design Initially*) to analysis and design. Knowing how much is 'just enough' is obviously the key and requires experience; just as in the StarWars movies, you do not become a Jedi Master overnight.

The model built in FDD is very much the same sort of model used in Domain-Driven Design, except that in FDD the modellers tend to apply Peter Coad's ' modeling in color' technique (although FDD does not mandate this).

Read more about object modelling...

Read more about modelling in colour...

Read more about FDD Process #1...

2. Build a Feature List

Using the knowledge gained during process 1, the team next constructs the FDD equivalent of an initial, overall product backlog. Instead of user stories or backlog items, FDD talks about features. In FDD, a feature is a small piece of client-valued function expressed in the form: <action> the <result> <by|for|of|to> a(n) <object>; for example 'calculate the total of a sale'.

Again the activity is a time-boxed, collaborative activity. Unlike Scrum or eXtreme Programming, the features list in FDD is not flat but typically a three level hierarchy comprising of subject or major functional *areas* broken into sets of features related by *activity*, each of which contains *features*. This three-level hierarchy is not dissimilar to the three levels used in the Scaled Agile Framework except that there the levels are called epics, features and stories.

Scrum says the Product Owner is responsible for turning up to the first sprint planning meeting with a prepared product backlog. It says nothing about how a Product Owner achieves that. FDD's modelling and feature list activities provide one proven, collaborative way of doing that.

Another way of thinking about the initial modelling activity, features list, and planning processes in FDD is to consider them as the application of a proven pattern for part of what typically happens within a sprint zero or initial release planning activity.

Read more about FDD Process #2...

3. Plan By Feature

The third activity in FDD, is sequencing the sets of features for activities into a high-level plan and assign them to chief programmers. Developers are also assigned to own particular classes identified in the overall object model. Again, this differs from eXtreme Programming's collective ownership of code, and when combined with FDD's dynamically-formed feature teams, it provides an alternative solution to the problems that collective ownership seek to resolve.

Read more about FDD Process #3...

4-5. Design By Feature / Build By Feature

FDD processes four and five guide the highly iterative, self-organising development engine room of the project. Again using the knowledge gained during and since the modelling activity in process #1, a lead developer (called a Chief Programmer in FDD terms) selects the small group of features that make most sense to develop over the next few days. The chief programmer identifies the domain classes likely to be involved, and the corresponding class owners form the feature team for this work. Depending on circumstances, others with specific skills such as testers, technical authors and user experience designers may join the team. This feature team works together with the help of a domain expert to analyse the details of each selected feature and design the solution for each. The team then work individually on the resulting coding and testing tasks, holding a code inspection when all is done. Once the chief programmer is satisfied that the work is complete, the completed features are promoted to the main, regular build, the feature team is disbanded, and the two processes are repeated for another small set of features.

It is quite easy to see from the process descriptions that the Chief Programmer role is critical in FDD. The Chief Programmer role fits most existing organisations concept of a lead developer, someone with both technical ability and experience, and just as importantly, able to lead a small cross-functional development team.

In addition, the Domain Expert role in FDD is hugely important, having many of the same responsibilities of the Product Owner role in Scrum. Interestingly, some of the more difficult tasks of a Scrum Product Owner are the responsibility of the Chief Programmer in FDD. This shift in responsibilities could possibly help Scrum teams struggling to find someone capable of fulfilling the hugely important Product Owner role.

The explicit identification of a team leader role also differs from eXtreme Programming. It views developers as a team of peers working in pairs. FDD recognises that in most enterprise software development organisations, skills, experience, and personalities often vary widely among project team members.

Read more about FDD Process #4...

Read more about FDD Process #5...

Read more about Feature Teams ...

Documents, Plans and Progress Reports

FDD mandates the creation of minimal amounts of design, planning, and progress documentation. Burn-down or burn-up charts similar to those at User Story or Product Backlog level in eXtreme Programming and Scrum are common. FDD also has a report that is the equivalent of a Scrum or Kanban style board but this

is organised slightly differently, capturing planned and actual dates for each milestone in a feature's development.

A higher-level, colour-coded chart known as a parking lot chart conveys a snapshot of the current status of a project to more senior management and stakeholders.

As the design and development of a feature proceed, a feature team collect any design diagrams, sketches and notes produced. These form a design package for the features, and formality varies from project to project according to the requirements for deliverable documentation.

Some of the ideas in this area of FDD have leaked into other agile approaches. The parking lot chart popped up in Mike Cohn's book, *Agile Estimating and Planning* and feature milestones have clearly been an influence in David Anderson's Kanban approach.

Read more about FDD Work and Design Packages...

Benefits fo FDD

One way to assess the potential of a process is to look at the sorts of common software development project problems that it helps solve. FDD was constructed with the strengths and weaknesses of human beings very much in mind. Software developers can be peculiar creatures at times and FDD addresses a number of the common behavioural problems that frequently afflict them. In addition, FDD has proven to be highly effective in rescuing complex projects that have stalled, or are in danger of failing completely.

Read more about FDD and developers...

Read more about rescuing projects using FDD...

FDD Background

Feature-Driven Development (FDD) evolved out of a necessity on a software development project in Singapore in 1997. Since then, FDD and FFD-inspired processes have been used on significant projects all over the world, in all sorts of business domains from telecommunications to banking to software tools. While extreme programming, scrum, lean, and kanban slog it out for the limelight in the conferences and on the web forums, FDD practitioners have applied FDD quietly and successfully on projects around the globe including Australia, New Zealand, the UK, Germany, and the USA.

Read more about the history of FDD...

Related articles elsewhere

- An Introduction to Feature-Driven Development: Part One
  Feature-Driven Development (FDD) is one of the agile processes not talked or written about very much. Often mentioned in passing in agile software development books and forums, few actually know much about it. However, if you need to apply agile to larger projects and teams, it is worthwhile taking the time to understand FDD a little more.
  Read the full article...

- An Introduction to Feature-Driven Development: Part Two
  Once there is an initial overall model (FDD Process #1), an initial overall features list (FDD Process #2), and an initial overall plan (FDD Process #3) in place, an FDD project is ready to start delivering the required software feature by feature.
  Read the full article...

- Best Practises used in Feature-Driven Development
  Like all good software development processes, Feature Driven Development is built around a core set of best practices blended into a cohesive whole.
  Read the full article...

- A Practical Guide to Feature Driven Development
  In the spring of 2001, Peter Coad asked Mac Felsing and myself to write a book about Feature-Driven Development for his new Coad Series published by Prentice Hall.
  Read the full article...

- Feature-Driven Development and Extreme Programming
  Feature Driven Development was introduced in Chapter 6 of the book, Java Modeling in Color with UML. Mac Felsing and I elaborated on the topic in our book,A Practical Guide to Feature-Driven Development. Software development process is an emotional issue so here's a few key quotes from the chapter to keep in mind when reading this...
  Read the full article...

- The Charge of the Light Brigade
  Tennysons poem about the infamous action of the Light Brigade at the battle of Balaclava
  Read the full article...

We think most process initiatives are silly. Well intentioned managers and teams get so wrapped up in executing process that they forget that they are being paid for results, not process execution
Peter Coad, Jeff De Luca *Java modelling in Color with UML Article*, 1999

A Practical Guide to FDD

In the spring of 2001, Peter Coad asked Mac Felsing and myself to write a book about Feature-Driven Development for his new Coad Series published by

Prentice Hall.
Read more...

FDD Roles and Responsibilities

FDD was designed originally for a team of mixed ability, varying experience, different racial backgrounds, and varying ages. The five processes of Feature Driven Development (FDD) explicitly mention six key project roles and implicitly suggest a number of supporting and additional roles.
Read more...

FDD Practices

Like all good software development processes, Feature Driven Development is built around a core set of 'best practices'. The chosen practices are not new but this particular blend is different. Each practice compliments and reinforces the others. The result is a whole greater than the sum of its parts.
Read more...

FDD Feature Teams

FDD's feature teams are small, cross-functional teams led by a technical lead (called a chief programmer in FDD), and enable a pragmatic alternative to the collective ownership paradigm of eXtreme Programming.
Read more...

FDD Design And Work Packages

We do not want to force developers to spend significant time creating unnecessarily large volumes of design documentation and project plans. FDD's design and work packages are a simple, effective alternative for document weary development teams.
Read more...

Just Enough Design Initially

We do not want to force developers to spend significant time creating unnecessarily large volumes of design documentation and project plans. FDD's design and work packages are a simple, effective alternative for document weary development teams.
Read more...

FDD to the Rescue

A look at how FDD can help put a project back on track by examining its influence in each activity within the core of a software development project.
Read more...

Developer Ailments

Developers. You got to love them ... or else they leave you ... with a half-coded system and no documentation!
Read more...

Related websites:

- Nebulon
- Feature-Driven DevelopmentCommunity
- The Scrum Alliance
- Scrum
- Adventures in Agility
- Software Engineering Institute

Follow me on Twitter...
Copyright 2014 Stephen R. Palmer. All rights reserved.