Metodología Ágil:
Feature Driven Development

Miguel Albertí Pons
Sofía Almeida Bruno
Pedro Manuel Flores Crespo
María Victoria Granados Pozo
Lidia Martín Chica

Índice

1. Introducción

Feature-Driven Development (FDD), es una metodología que ayuda a crear software mediante un ciclo de vida iterativo incremental que esta orientado a equipos más grandes, con más personas que aquellos donde normalmente se utilizan otras metodologias como SCRUM. FDD ve necesaria la figura del jefe de proyecto y una fase donde se define la arquitectura.

En este documento veramos qué procesos hay que seguir para aplicar FDD, sus ventajas y los roles con sus respectivas responsabilidades.

2. Procesos

La metodología FDD consta de cinco fases en su proceso. Son las siguientes:

2.1. Desarrollar el modelo global (*Develop overall model*)

En esta primera etapa los miembros del equipo de desarrollo trabajan juntos para construir un modelo del problema del dominio. Su objetivo es proponer el modelo para el área de dominio. El modelado FDD es una actividad interfuncional, iterativa y colaborativa. Los miembros del equipo (desarrollo, expertos en dominios y programadores principales) trabajan juntos para componer un modelo para el área de dominio y son guiados por un Arquitecto Jefe (*Chief Arquitect*). La idea es tener diferentes equipos proponiendo diferentes modelos y luego, después de ser revisados, seleccionar uno de los modelos propuestos o una combinación de modelos y que se convierta en el modelo para esa área de dominio. A medida que se desarrolle el modelo y el equipo aprenda, se agregarán detalles. Esto ayuda al equipo a tener una visión general clara de todo el proyecto, así como una comunicación sólida. Las tareas llevadas a cabo en esta etapa se resumen en la Tabla 1.

Tarea	Responsables
Formación del *Model Team*	*Project Manager*
Domain Walkthrough	*Modeling Team*
Estudio de los documentos	*Modeling Team*
Construcción de una *Feature List informal*	*Chief Arquitect*, *Chief* y *Programmers*
Desarrollo del *Modelo de Tareas*	*Modeling Team* dividido en pequeños grupos
Desarrollo del *Modelo Global*	*Chief Arquitect* y *Modeling Team*
Registro de alternaticas	*Chief Arquitect*, *Chief* y *Programmers*

Tabla 1: Tareas etapa del desarrollo del modelo global.

2.2. Construir lista de características (*Build Feature List*)

Esta etapa recibe como entrada el modelo de objetos y los requerimientos (*feature list informal*) obtenidos en la etapa anterior. Estos son agrupados según el área de dominio. Cada grupo de se denomina *Major List Sets*. Esta lista a su vez es dividida en otros subconjuntos según la funcionalidad. Posteriormente, cada característica o funcionalidad es priorizada y por último aquellas más complejas son divididas en otras más pequeñas. Esta etapa nos aporta como salida la *Feature List* que es revisada por los usuarios para su aprobación y validación. Las tareas llevadas a cabo en esta etapa se resumen en la Tabla 2.

Tarea	Responsables
Formación del *Feature-List Team*	*Project Manager* y *Development Manager*
Identificación de funcionalidades y formación del *Feature Set*	*Feature-List Team*
Priorización de las funcionalidades	*Feature-List Team*
División de las funcionalidades complejas	*Feature-List Team*

Tabla 2: Tareas etapa de construcción de la lista de características.

2.3. Planificado por características (*Plan by feature*)

Este paso gira entorno a la planificación, esto es, el orden en el que se van a implementar las características, la forma de hacerlo y quién las va implementar, asignando tareas a los desarrolladores. Durante la planificación consideramos diferentes aspectos como riesgos, complejidad, dependiencias, la carga de trabajo, etc, para evitar que surjan problemas complejos.

En esta etapa se incluye la creación de un plan de alto nivel, en el cual la *features list* se ordena en base a la prioridad y a la dependencia entre cada característica. A continuación, el gestor del proyecto, el desarrollador y el programador jefe establecen hitos y diseñan un cronograma de diseño y construcción.

Finalmente se realiza una verificación del plan, por parte del gestor del desarrollo y del jefe del producto, teniendo en cuenta la opinión de todos lo miembros del equipo. La salida es un plan de desarrollo incluyendo la fecha de finalización, para cada conjunto de características y para cada característica indicar el programador jefe asignado, y las fechas de inicio y fin. Por último para cada clase indicar el correspondiente programador de la clase.

2.4. Diseñando por características (*Design by feature*)

Esta etapa y la siguiente se realizan de forma iterativa, cada iteración debería durar entre unos pocos días y un máximo de dos semanas. Puede haber varios equipos de características trabajando simultáneamente en el diseño e implementación de su propio conjunto de características.

En esta etapa el *chief Programmer*, aprovechando el conocimiento obtenido en las primeras etapas del proceso, selecciona las siguientes características a realizar entre las que tenga en su lista de características asignadas. A continuación, organiza los equipos de características identificando los *class owners* (desarrolladores) que estarán involucrados en el desarrollo de las características seleccionadas y contacta con los expertos de dominio si es necesario. Parte del grupo puede trabajar en el diseño técnico mientras otros trabajan en la infraestructura, escribiendo clases, determinando los métodos y realizando los diagramas de secuencia correspondientes que el *chief Programmer* añadirá al modelo global. Asimismo, cada *class owner* actualizará la descripción de sus clases en base al diagrama de secuencia (añadiendo parámetros, tipos de retorno, mensajes enviados, etc).

Antes de pasar a la siguiente fase todo el equipo revisará y verificará el diseño.

Como salida de esta etapa se debe obtener: el diagrama de secuencia detallado, diagrama de clases actualizado, descripción de clases y métodos, notas del equipo significativas para el diseño.

2.5. (*Build by feature*)

3. Comparación con otros métodos

FDD es una mezcla entre eXtremeProgramming y Scrum, añadiendo técnicas de Domain Driven Design[1]. A continuación se muestran los puntos en los que se diferencia FDD con XP y Scrum.

Comunicación

Los métodos ágiles se centran bastante en la comunicación entre los mienbros del equipo y el resto de personas interesadas en el proyecto.

En XP y Scrum la documentación también es importante.

Centro de usuarios

El *software* debe diseñarse y desarrollarse por un centro de usuarios. Con XP por ejemplo, se necesita la participación de los clientes durante el desarrollo así como iteraciones cortas, al final de las cuales el cliente puede probar el *software*.

Duración del sprint

Lo normal es usar iteraciones cortas, por ejemplo entre 2 y 10 días. A diferencia de Scrum que dura entre 2 y 4 semanas y XP que puede durar 6 semanas.

Reuniones

Tanto la comunicación como las reuniones son importantes en los métodos ágiles. Con Srum y XP hay reuniones diarias donde participan todos los miembros del equipo y comentan en que punto del proyecto se encuentran.

En FDD es un poco diferente por que en general la información se comunica mediante la documentación.

Tamaño del proyecto

Dependiendo de cada proyecto hay que elegir que método ágil elegir. Para proyectos cortos y no muy complejos, habría que elegir XP, mientra que para proyectos más complejos y largos es mejor Scrum y FDD.

[1]Diseño guiado por el dominio (DDD), es un enfoque para el desarrollo de software con necesidades complejas mediante una conexión entre la implementación y los conceptos del modelo y núcleo del negocio. El DDD no es una tecnología ni una metodología, este provee una estructura de prácticas y terminologías para tomar decisiones de diseño.

4. Ventajas

FDD minimiza la complejidad del sistema por lo que es una excelente solución para proyectos grandes y complejos, dando al equipo la capacidad de dividir el problema en problemas más pequeños que se pueden resolver en menos tiempo.

La comprensión más profunda del sistema de software reduce el riesgo de encontrar problemas durante el desarrollo.

La mayoría de los procesos requieren un flujo de comunicación constante y la realización frecuente de los informes de progreso en casi cualquier nivel del desarrollo del proyecto, permiten al equipo realizar un seguimiento del progreso y los resultados. Además, FDD permite que los miembros del equipo se comuniquen más fácilmente al tiempo que fomenta la creatividad e innovación del equipo.

El hecho de que en FDD se realicen compilaciones regulares garantiza que se puedan identificar fácilmente errores en el código y permite mantener el proyecto actualizado regularmente, identificar cualquier error y que se pueda mostrar al cliente en cualquier momento.

En FDD se maximiza la calidad ya que el concepto de calidad no solo incluye la prueba del código, sino que también incluye estándares de codificación, auditorías de medición y métricas en el código.

5. Roles y responsabilidades

Los proyectos están formados por gente, procesos y tecnología. En esta sección describiremos cómo se organizan los participantes en un proyecto.
La metodología FDD clasifica sus roles en tres categorías: clave, de apoyo y adicionales. Un miembro de un equipo puede tomar diferentes roles y un rol se puede compartir entre varias personas.

A continuación se exponen los seis roles clave junto con sus tareas y responsabilidades:

- *Project Manager*. Es el líder administrativo y financiero del proyecto. Una de sus tareas es proteger al equipo de distracciones externas y permitir que el equipo trabaje en las condiciones apropiadas. Es responsable de señalar el progreso, manejar los presupuestos, buscar la plantilla, gestionar el equipo, espacio y recursos. Es el que tiene la última palabra sobre el alcance, planificación y personal del proyecto.

- *Chief Architect*. El jefe de diseño es responsable del diseño global del sistema y de llevar a cabo las sesiones de diseño del sistema con el equipo. Asimismo, toma todas las decisiones finales en los asuntos de diseño. Este puesto requiere de habilidades técnicas y de modelado además de capacidad de asesoramiento. Si es necesario este rol se puede dividir entre arquitecto del dominio (*Domain Architect*) y arquitecto técnico (*Tecnical Arquitect*).

- *Development Manager*. Es el responsable de liderar las actividades de desarrollo diarias y resolver los problemas que ocurran entre los miembros del equipo. Es

el encargado de resolver los conflictos por los recursos cuando cuando los *Chief Programmers* no consigan ponerse de acuerdo. Este rol se puede combinar con el de *chief architect* o *project manager*.

- *Chief Programmer*. Es un programador con cierta experiencia ya que ha pasado por todo el proceso de desarrollo varias veces. Se encarga del análisis de requisitos a un alto nivel, de actividades relacionadas con el diseño y de la dirección de equipos pequeños (3 a 6 programadores) para el análisis, diseño y desarrollo de nuevas funcionalidades. También escoge las funcionalidades a desarrollar de la lista, identifica las clases y el *Class Owner* que se necesita en el equipo para la iteración. Con la ayuda de otros *Chief Programmers* resuelve problemas técnicos y de recursos, y sigue el progreso del equipo.

- *Class Owner*. Trabaja bajo la dirección del *Chief Programmer* en tareas de diseño, programación, pruebas y documentación. Es el responsable de las clases que se asignaron y participa en la decisión de qué clase será incluida en la lista de características de la próxima iteración.

- *Domain Expert*. Son usuarios, patrocinadores, analistas o una mezcla de ellos. Son los que tienen el conocimiento base acerca del software que se está desarrollando y por ello su participación es crítica para el éxito del sistema.

Entre los roles de apoyo de FDD se incluyen:

- *Domain Manager*. Lidera a los expertos en el dominio y resuelve sus diferencias de opinión respecto a los requisitos del sistema.

- *Release Manager*. Controla el progreso del proceso asegurándose de que los *chief programmers* informen sobre su progreso semanalmente, revisando sus informes y teniendo reuniones cortas con ellos. Entonces informará directamente al *project manager*.

- *Language Lawyer/Guru*. Es un miembro del equipo responsable de tener un amplio conocimiento sobre un lenguaje o tecnología concreta. Esta papel tiene más importancia cuando el equipo se enfrenta a una nueva tecnología.

- *Build Engineer*. Es el responsable de configurar, mantener y llevar a cabo el proceso de producción incluyendo el mantenimiento de las versiones y publicación de la documentación.

- *Toolsmith*. Construye herramientas específicas para el desarrollo, conversión de datos y realización de pruebas. Puede trabajar en la preparación y mantenimiento tanto de bases de datos o páginas web destinadas al proyecto.

- *System Administrator*. Configura, administra y repara los servidores, equipos de trabajo, desarrollo y pruebas utilizados.

De forma adicional se tienen los siguientes roles:

- *Tester*. Se encargan de verificar de forma independiente de que el sistema cumple los requisitos del usuario y que realiza esas funciones de la manera adecuada. Puede ser una persona independiente o no del equipo del proyecto.
- *Deployer*. Convierte la información existente al formato requerido por el sistema y participa en el despliegue físico de los nuevos productos.
- *Technical Writer*. Escribe la documentación para los usuarios.

6. Prácticas

Las prácticas más usadas en FDD son:

1. *Domain Object Modeling*: Consiste en construir diagramas de clases que representan los objetos más significativos dentro de un problema y las relaciones entre ellos. Es una forma de descomposición de objetos. El diseño e implementación de cada objeto o la clase identificada en el modelo es un problema menor para resolver. Una técnica utilizada dentro de esta práctica es *UML in color*. Básicamente, consiste en que las clases se clasifican en diferentes categorías y cada categoría lleva asociada un color. Por ejemplo, el amarillo se usa para clases que representan algún rol (persona yu organización) o el azul para aquellas que guardan la información sobre el objeto que representan.
2. *Developing by Feature*: las funciones que son demasiado complicadas se descomponen en otras más pequeñas hasta tener subproblemas que representen una sola característica.
3. *Class (Code) Ownership*: cada persona o rol es el responsable de sus clases o código implementado.
4. *Feature Teams*: cada característica detectada puede necesitar más de una clase y en consecuencia varios *class owners* por lo que el dueño de cada característica es el encargado de coordinar el trabajo de estos programadores.
5. *Inspections*: aplicar las técnicas de detección de fallos más conocidas y aprovechar las oportunidades que proporcionan para promover buenas prácticas de desarrollo.
6. *Regular Build Schedule* : regularmente se debe tomar todo el código desarrollado y construir el sistema completo para asegurarnos que siempre una versión funcional.
7. *Configuration Management*: sirve para identificar las últimas versiones de los archivos completados y proporciona un seguimiento de todos los artefactos usados.
8. *Progress Reporting*: a lo largo de todo el proyecto realizar informes sencillos que contengan el progreso realizado para informar a todos los roles dentro y fuera del proyecto.

7. Bibliografía

Referencias

[1] Fdd; its processes and comparison to other agile methodologies. https://apiumhub.com/tech-blog-barcelona/feature-driven-development/. Último acceso: 15/11/2019.

[2] Feature driven development. https://activecollab.com/blog/project-management/feature-driven-development/. Último acceso: 15/11/2019.

[3] Feature driven development (fdd) and agile modeling. http://agilemodeling.com/essays/fdd.htm. Último acceso: 15/11/2019.

[4] La metodología ágil fdd. una metodología ágil para equipos/empresas/proyectos grandes. https://www.javiergarzas.com/2012/09/metodologia-gil-fdd-1.html. Último acceso: 15/11/2019.

[5] What is fdd in agile. https://leankit.com/learn/agile/fdd-agile/. Último acceso: 15/11/2019.

[6] Luis Calabria. *Metodología FDD*. Universidad ORT Uruguay, 2003.

[7] Sadhna Goyal. Major seminar on feature driven development. In *Technical University Munich*, 2007.

[8] John Hunt. *Agile Software Construction*, chapter 9, pages 161–192. London: Springer London, 2006.

[9] Jussi Ronkainen & Juhani Warsta Pekka Abrahamsson, Outi Salo. *Agile software development methods. Review and analysis*, chapter 3.4, 4, pages 47–55, 86–92. JULKAISIJA – UTGIVARE – PUBLISHER, 2002.