Universidade Católica de Brasília (UCB)

Feature Driven Development (FDD): Metodologia Ágil Orientada a Funcionalidades

Autores: Amanda Alves Mendonça Gabriel Leite Queiroz Guilherme Lucas Pedro Henrique da Silva Romão Vinicius Almeida Lacerda
Disciplina: Metodologias Ágeis **Professor:** Weldes

Brasília, 04 de Novembro de 2025

Sumário

1. Introdução

O desenvolvimento de software, desde a sua concepção, tem sido uma busca contínua por processos que equilibrem previsibilidade, qualidade e velocidade. Durante décadas, o setor foi dominado por ciclos de vida prescritivos, notavelmente o modelo em Cascata (*Waterfall*). Conforme descrito por Pressman (2016), esses modelos, embora oferecessem uma estrutura lógica e sequencial, demonstravam-se fundamentalmente inadequados para um ambiente de negócios onde a mudança é a única constante. Sua rigidez intrínseca e a dificuldade em acomodar alterações de escopo resultavam frequentemente em atrasos, custos extrapolados e, o mais crítico, produtos que não atendiam às necessidades reais do cliente no momento da entrega.
Em resposta a essa "crise do software", emergiu um conjunto de novas abordagens no final da década de 1990, culminando na formalização do Manifesto Ágil em 2001 (BECK et al., 2001). Este manifesto representou uma mudança de paradigma, deslocando o foco de processos e ferramentas para indivíduos e interações; da documentação exaustiva para o software em funcionamento.

Dentro deste vasto universo ágil, que inclui metodologias populares como Scrum e *Extreme Programming* (XP), o *Feature Driven Development* (FDD), ou Desenvolvimento Orientado a Funcionalidades, ocupa um nicho singular. O FDD é frequentemente descrito como uma metodologia "híbrida" ou de "meio-termo". Ele combina a natureza adaptativa e iterativa do desenvolvimento ágil com o rigor da engenharia de software e da modelagem de domínio, práticas muitas vezes associadas a métodos mais tradicionais.

Sua importância reside precisamente em sua capacidade de escalar. Enquanto muitas abordagens ágeis encontram desafios em projetos de grande porte (com equipes de 50 ou mais desenvolvedores), o FDD foi concebido exatamente para esse cenário. Este trabalho, portanto, visa dissecar analiticamente o FDD, explorando não apenas seus processos e papéis, mas também seu posicionamento filosófico. O objetivo é demonstrar como o FDD oferece uma estrutura robusta para o desenvolvimento de sistemas complexos, centrada na entrega de valor tangível ao negócio (as funcionalidades), sem sacrificar a integridade arquitetural e a qualidade de longo prazo do software.

2. Desenvolvimento

2.1 Conceito e Princípios do Feature Driven Development (FDD)

O *Feature Driven Development* (FDD) é uma metodologia ágil de desenvolvimento de software que, como seu nome indica, organiza o ciclo de vida do projeto inteiramente em torno de "funcionalidades" (*features*). No FDD, a funcionalidade é a unidade fundamental de trabalho, planejamento e medição de progresso.
Uma funcionalidade é formalmente definida como "uma pequena parte de comportamento do sistema, valorizada pelo cliente, que pode ser implementada em até duas semanas" (PALMER; FELSING, 2002). É crucial a ênfase no "valor ao cliente". Isso distingue uma funcionalidade de uma "tarefa". Por exemplo, "Refatorar o módulo de conexão com o banco de dados" é uma tarefa técnica; "Gerar o relatório de vendas mensais" é uma funcionalidade. O FDD força a equipe a decompor o trabalho em entregáveis que são compreensíveis e relevantes para o negócio.
Para ser considerada uma funcionalidade válida no FDD, ela geralmente segue um padrão de nomenclatura específico que reforça seu propósito de negócio: <ação> <resultado> <objeto>.

Por exemplo:

- **Bom:** "Calcular o total de uma venda"
- **Bom:** "Validar a senha de um usuário"
- **Ruim:** "Formulário de Pedido" (não descreve uma ação)
- **Ruim:** "Corrigir bug #501" (não é uma funcionalidade de negócio)

O FDD baseia-se em um conjunto de "melhores práticas" da engenharia de software, como modelagem de domínio, propriedade de código e inspeções formais, integrando-as em um processo ágil de cinco etapas. Ele não é tão leve quanto o Scrum, pois exige uma fase inicial de modelagem (embora não seja um *Big Design Up Front* - BDUF, mas sim um *Just Enough Design Up Front* - JEDUF), nem tão prescritivo quanto processos como o RUP (*Rational Unified Process*).

2.2 Contexto Histórico e Criadores: A Gênese Pragmática do FDD

A origem do FDD é fundamental para entender sua filosofia. Ele não nasceu em um laboratório de pesquisa acadêmica, mas sim na "trincheira" de um projeto de alta complexidade. Entre 1997 e 1999, **Jeff De Luca**, um experiente gerente de projetos e arquiteto de software, estava liderando o desenvolvimento de um sistema de *core banking* para um grande banco em Singapura. O projeto era massivo, envolvendo 50 desenvolvedores (que mais tarde cresceram para 250) e um domínio de negócio intrincado.

Os métodos tradicionais falharam em fornecer a tração necessária, e as metodologias ágeis emergentes da época não pareciam robustas o suficiente para a escala do desafio. De Luca, então, buscou a colaboração de **Peter Coad**, uma autoridade mundial em modelagem orientada a objetos (OO), conhecido por seu trabalho em *UML* e *Object Modeling in Color*.

Juntos, eles sintetizaram um processo pragmático que fundia as melhores práticas de ambas as suas experiências. De Coad, veio a ênfase na modelagem de domínio OO como a espinha dorsal do sistema. De De Luca, veio a estrutura de processo iterativo, o gerenciamento focado em funcionalidades e a necessidade de relatórios de progresso claros e tangíveis. O FDD foi, portanto, forjado pela necessidade prática de gerenciar a complexidade e escalar o desenvolvimento ágil, sendo posteriormente formalizado no livro *Java Modeling in Color with UML* (COAD; DE LUCA; LEFEBVRE, 1999).

2.3 A Arquitetura do Processo FDD: Os Cinco Passos Detalhados

O FDD é definido por cinco processos sequenciais. Os três primeiros estabelecem a fundação do projeto (a fase de "setup"), enquanto os dois últimos formam o ciclo iterativo de construção (a fase de "manufatura").

2.3.1 Desenvolvimento de um Modelo Geral (*Develop an Overall Model*)

Este é o primeiro e um dos mais distintivos processos do FDD. Envolve uma colaboração intensa entre os Especialistas de Domínio (clientes, usuários) e os desenvolvedores-chave (liderados pelo Arquiteto-Chefe). O objetivo não é criar uma arquitetura de software detalhada, mas sim um **modelo de domínio** abrangente.

Através de sessões chamadas *Domain Walkthroughs*, a equipe explora o negócio e identifica os principais objetos de domínio, seus atributos e seus relacionamentos. O resultado é um

conjunto de diagramas de classes UML (frequentemente usando a técnica de "modelagem em cores" de Coad) que serve como um "mapa" conceitual do sistema. Este modelo garante que toda a equipe compartilhe um vocabulário e um entendimento comum do problema, servindo como a fundação arquitetural sobre a qual as funcionalidades serão construídas.

2.3.2 Construção da Lista de Funcionalidades (*Build a Features List*)

Com o modelo de domínio estabelecido, a equipe decompõe funcionalmente o sistema. Este processo é hierárquico:

1. **Áreas de Negócio:** Grandes agrupamentos lógicos (ex: "Gestão de Vendas", "Controle de Estoque").
2. **Atividades de Negócio:** Passos de processo dentro de uma área (ex: "Processar um Pedido de Cliente" dentro de "Gestão de Vendas").
3. **Funcionalidades:** Ações detalhadas que compõem uma atividade (ex: "Adicionar item ao carrinho", "Calcular frete", "Aplicar cupom de desconto").

Essa lista de funcionalidades, formatada com o padrão <ação> <resultado> <objeto>, torna-se o *backlog* principal do projeto. Ela é exaustiva e serve como o escopo mestre para o planejamento.

2.3.3 Planejamento por Funcionalidade (*Plan By Feature*)

Nesta fase, a equipe de gerenciamento (Gerente de Projeto, Arquiteto-Chefe) e os Programadores-Chefes colaboram para criar o plano de desenvolvimento. As funcionalidades são sequenciadas com base em múltiplos critérios: valor de negócio (prioridade do cliente), risco técnico, dependências entre funcionalidades e complexidade.

Funcionalidades relacionadas são agrupadas em "conjuntos de funcionalidades" (*feature sets*), que por sua vez são atribuídos a Programadores-Chefes específicos. O resultado é um plano de alto nível que define a ordem de construção e quem é responsável por cada parte.

2.3.4 Projeto por Funcionalidade (*Design By Feature*)

Este é o primeiro passo do ciclo iterativo (a "fase de manufatura"). Para cada funcionalidade (ou pequeno conjunto delas) selecionada para implementação, o Programador-Chefe responsável monta uma **Equipe de Funcionalidade** (*Feature Team*), composta por ele mesmo e os Proprietários de Classe relevantes.

Esta equipe realiza o design técnico "just-in-time" daquela funcionalidade. Eles produzem diagramas de sequência detalhados, definem as APIs das classes envolvidas e atualizam o modelo de domínio, se necessário. Este design é então submetido a uma **Inspeção de Design** (uma revisão formal por pares), que atua como um portão de qualidade crucial antes que qualquer código seja escrito.

2.3.5 Construção por Funcionalidade (*Build By Feature*)

Após a aprovação do design, a Equipe de Funcionalidade implementa o código. O FDD utiliza um conceito fundamental chamado **Propriedade de Classe** (*Class Ownership*). Cada classe no código-fonte é "propriedade" de um desenvolvedor específico (o Proprietário da

Classe). Embora outros possam visualizar o código, apenas o proprietário pode realizar alterações. Se a implementação de uma funcionalidade exigir a alteração de classes de diferentes proprietários, os proprietários são trazidos para a Equipe de Funcionalidade.

Este processo inclui codificação, testes de unidade e integração. Uma vez concluída, a funcionalidade passa por uma **Inspeção de Código** rigorosa (para garantir a adesão ao design e aos padrões) antes de ser promovida para o *build* principal.

2.4 Papéis e Responsabilidades: Uma Estrutura Hierárquica Ágil

O FDD se destaca por definir papéis claros, o que facilita sua escalabilidade. Não é uma estrutura "plana" como a de algumas equipes ágeis.

- **Gerente de Projeto (*Project Manager*):** Tem a visão administrativa, gerenciando escopo, cronograma, orçamento e sendo a interface principal com os *stakeholders* executivos.
- **Arquiteto-Chefe (*Chief Architect*):** Responsável pela concepção e integridade do modelo de domínio (Passo 1) e pela arquitetura geral do sistema. Define os padrões técnicos.
- **Gerente de Desenvolvimento (*Development Manager*):** Lida com a alocação de recursos, resolução de conflitos entre equipes e o ambiente de desenvolvimento.
- **Programador-Chefe (*Chief Programmer*):** Este é o papel mais crucial e distinto do FDD. É um desenvolvedor sênior, altamente qualificado, que atua como um "líder servidor" técnico. Ele *não* é um gerente de pessoas. Ele lidera as Equipes de Funcionalidade, é responsável pelo design detalhado (Passo 4), orienta os desenvolvedores mais juniores, revisa o código e garante a qualidade técnica das funcionalidades sob sua responsabilidade.
- **Proprietário da Classe (*Class Owner*):** O desenvolvedor "dono" de um conjunto de classes. Ele é responsável pela implementação, teste de unidade e manutenção dessas classes. A Propriedade de Classe fomenta a especialização e a responsabilidade.
- **Especialista de Domínio (*Domain Expert*):** O cliente, usuário ou analista de negócios. Ele é a fonte de verdade para as regras de negócio e é essencial no Passo 1 (Modelo) e na validação das funcionalidades construídas.

2.5 Análise Crítica: Vantagens, Limitações e Casos de Uso

Vantagens:

1. **Escalabilidade Comprovada:** A estrutura hierárquica (Programadores-Chefes liderando Equipes de Funcionalidade) permite que o FDD escale para grandes equipes (50+) de forma mais eficaz do que o Scrum puro, que pode sofrer com a sobrecarga de comunicação.
2. **Visibilidade e Rastreamento de Progresso:** O FDD oferece relatórios de progresso extremamente claros e granulares, baseados em funcionalidades concluídas. Ferramentas visuais como os *Parking Lot Diagrams* (Diagramas de Estacionamento) permitem que os *stakeholders* vejam, em tempo real, o status de cada funcionalidade do projeto.
3. **Qualidade Arquitetural Embutida:** A ênfase no modelo de domínio (Passo 1) e nas inspeções formais de design e código (Passos 4 e 5) garante que a "saúde" técnica do projeto não seja sacrificada pela velocidade, mitigando o débito técnico.
4. **Processo Centrado no Cliente:** Ao decompor tudo em funcionalidades de negócio, o

FDD garante que cada linha de código escrita esteja diretamente ligada a uma entrega de valor perceptível pelo cliente.

Limitações:

1. **Dependência Crítica de Pessoal Sênior:** O sucesso do FDD depende maciçamente da existência de "Programadores-Chefes" competentes. Se a organização não possui esses talentos (que são uma mistura rara de excelência técnica e habilidades de mentoria), o processo pode falhar.
2. **Burocracia para Projetos Pequenos:** Para equipes pequenas (ex: 3-5 desenvolvedores), a estrutura de papéis do FDD (Gerente de Projeto, Arquiteto, Programador-Chefe, etc.) é excessiva e contraproducente.
3. **Resistência à Modelagem Inicial:** A fase de "Desenvolver um Modelo Geral" pode ser mal interpretada como um *Big Design Up Front* (BDUF) por agilistas puristas, gerando resistência cultural. Embora seja um modelo de domínio (o "quê") e não de design (o "como"), essa fase inicial requer um investimento que nem todos os projetos estão dispostos a fazer.

Casos de Uso: O FDD é ideal para **projetos *enterprise* de média a grande escala**, com **domínios de negócio complexos** (ex: sistemas de *core banking*, plataformas de cálculo de apólices de seguro, sistemas de ERP, logística complexa) onde um entendimento profundo das regras de negócio é essencial. É também altamente eficaz em projetos de **modernização de sistemas legados**, onde o Passo 1 (Modelo) é crucial para dissecar e entender a lógica existente antes de substituí-la.

2.6 Análise Comparativa com Outras Metodologias Ágeis (Scrum, XP, Kanban)

O FDD não é um concorrente direto, mas uma alternativa com focos distintos.

Característica	Feature Driven Development (FDD)	Scrum	Extreme Programming (XP)	Kanban
Unidade de Trabalho	Funcionalidade (Feature) (Valor de negócio)	Item do Backlog da Sprint (PBI) (Definido pelo PO)	User Story / Tarefa Técnica	Item de Trabalho (Card)
Ritmo (Iteração)	Curta e orientada à feature (1-2 semanas)	Fixa (Sprint, *time-boxed*, 2-4 semanas)	Fixa (Iteração, 1-2 semanas)	Fluxo Contínuo (sem iteração prescrita)
Foco em Design	Alto: Modelagem de domínio inicial (Passo 1) e design por feature (Passo 4).	Baixo (Prescrito): O framework não dita práticas de design; deixa para a equipe.	Alto: Design Simples, Refatoração, TDD, Design Emergente.	Nulo (Prescrito): Foca na otimização do fluxo, não em práticas de engenharia.
Papéis-Chave	Programador-Chefe, Proprietário de Classe, Arquiteto-Chefe.	Product Owner, Scrum Master, Equipe de Desenvolvimento.	Coach, Cliente On-site, Programador (em pares).	Não prescritivo (sugere papéis como *Service Delivery Manager*).

Característica	Feature Driven Development (FDD)	Scrum	Extreme Programming (XP)	Kanban
Filosofia Central	Processo robusto para escalar a entrega de valor com qualidade de engenharia.	Framework de gestão para lidar com a complexidade e maximizar o valor.	Conjunto de práticas de engenharia para alta qualidade técnica e feedback.	Método para visualizar, gerenciar e otimizar o fluxo de trabalho.

Filosoficamente, o Scrum é um **framework de gestão** (o "o quê" e "quando" da entrega, mas não o "como" técnico). O XP é um conjunto de **práticas de engenharia** (o "como" técnico). O FDD é um **processo de desenvolvimento** mais completo, que prescreve tanto a gestão (os 5 passos) quanto práticas de engenharia (modelagem, inspeções).

2.7 Aplicações Práticas e Ferramentas de Suporte

Em um cenário prático, considere a modernização de um sistema de gestão de logística de uma grande varejista. O sistema legado é um monolito em COBOL, e a empresa deseja migrar para uma arquitetura de microsserviços.
Aplicando o FDD:

1. **Passo 1 (Modelo):** Seria crucial. Uma equipe de arquitetos e especialistas de domínio passaria semanas mapeando as regras de negócio do COBOL em um modelo de domínio OO, identificando os *bounded contexts* (contextos delimitados) que se tornariam os novos microsserviços.
2. **Passo 2 (Lista):** O sistema seria decomposto em funcionalidades como "Roteirizar uma entrega", "Calcular o custo do frete", "Confirmar recebimento pelo cliente".
3. **Passo 3-5 (Iteração):** Múltiplas Equipes de Funcionalidade trabalhariam em paralelo. Uma equipe (liderada por um CP) poderia estar construindo o novo microsserviço de "Roteirização", enquanto outra (liderada por outro CP) estaria construindo o de "Cálculo de Frete".

O progresso seria monitorado não por "horas gastas", mas por "funcionalidades do legado migradas e validadas". Ferramentas como *Parking Lot Diagrams* (muitas vezes implementados em JIRA, Trello ou ferramentas especializadas como o FDD Project Manager) forneceriam à alta gestão uma visão clara do avanço da modernização.

3. Conclusão

O *Feature Driven Development* (FDD) apresenta-se como uma metodologia ágil singular, que ousa prescrever um nível de rigor de engenharia e estrutura de papéis frequentemente ausente em abordagens mais leves. Ele não é uma panaceia; sua aplicação em projetos pequenos é inadequada. Contudo, sua relevância histórica e prática reside em sua resposta ao desafio da **escalabilidade ágil**.

Ao centrar o universo do projeto na "funcionalidade", o FDD cria uma linguagem comum entre o negócio e a tecnologia. Ao instituir processos formais de modelagem e inspeção, ele protege a integridade arquitetural do produto a longo prazo. E, ao definir papéis claros como o do Programador-Chefe, ele estabelece um modelo de mentoria e liderança técnica que permite a

grandes equipes operarem de forma coesa.
Dessa forma, o FDD contribui para o sucesso de equipes ágeis ao fornecer um modelo que equilibra, de forma pragmática, a flexibilidade e a velocidade (demandas do Manifesto Ágil) com a disciplina e a robustez (demandas da Engenharia de Software clássica).

4. Referências Bibliográficas

AGILE ALLIANCE. (n.d.). *Feature Driven Development (FDD)*. Agile Alliance. Retirado de https://www.agilealliance.org/glossary/fdd

BECK, K., et al. (2001). *Manifesto for Agile Software Development*. Agile Manifesto. Retirado de https://agilemanifesto.org/iso/ptbr/manifesto.html

COAD, P., DE LUCA, J., & LEFEBVRE, E. (1999). *Java modeling in color with UML: Enterprise components and process*. Prentice Hall.

PALMER, S. R., & FELSING, J. M. (2002). *A practical guide to feature-driven development*. Prentice Hall.

PRESSMAN, R. S. (2016). *Engenharia de Software: Uma Abordagem Profissional* (8ª ed.). McGraw-Hill.

SOMMERVILLE, I. (2019). *Engenharia de Software* (10ª ed.). Pearson.