

Aula 2 - Paradigmas de desenvolvimento de software

Jayme Anchante

10 de junho de 2020

Paradigmas de desenvolvimento de software

Paradigmas e modelos

- ▶ Agile
- ▶ Cleanroom
- ▶ Incremental
- ▶ Prototyping
- ▶ Spiral
- ▶ V model
- ▶ Waterfall

Paradigmas

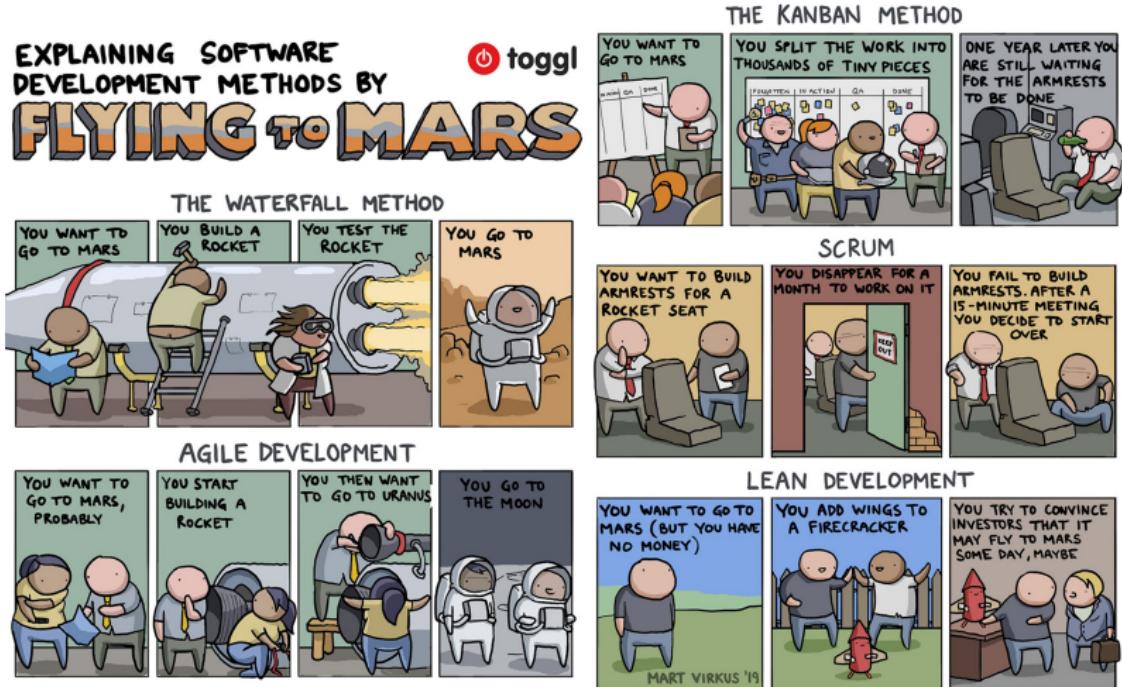


Figure 1: Voando a Marte

Waterfall

Conceito

Quebra as atividades do projeto em fases lineares e sequenciais em que cada fase depende dos entregáveis da fase anterior e corresponde a uma especialização das tarefas

Tende a ser menos iterativa e flexível já que o progresso flui em apenas uma direção (pra baixo, como uma cachoeira) das fases mencionados de desenvolvimento de software

Origem

Originou-se nos setores de manufaturas, indústria e construção civil em que o grande custo das estruturas físicas faz com que mudanças de design acabem sendo extremamente custosas

Artigo de 1970 de Winston Royce é dada como a primeira descrição formal do modelo

Em 1985, o Departamento de Defesa americano explicitava que os licitantes deveriam oferecer um software que passasse pelas seis fases de “análise de requisitos, design preliminar, design detalhado, codificação e teste unitário, integração e testes”

Modelo de Royce

1. Requisitos de sistema e software
2. Análise: modelos, esquemas, regras de negócio
3. Design: arquitetura de software
4. Codificação
5. Testes
6. Operações: instalação, migração, suporte e manutenção

Deve-se avançar para a próxima etapa apenas quando a fase precedente for revisada e validada

Argumentos favoráveis

- ▶ Tempo gasto nas primeiras etapas pode reduzir exponencialmente gastos nas fases subsequentes
- ▶ 20-40% do tempo nas duas primeiras fases, 30-40% nas duas fases seguintes e o restante para testes e implantação
- ▶ Ênfase na documentação (requisitos e design). Em outras metodologias, perda de membros da equipe assim como treinamento de novos integrantes pode ser bastante custoso
- ▶ Abordagem estruturada, linear e progressiva que é fácil de explicar e de entender. Os marcos são facilmente identificados durante o processo

Argumentos contrários

- ▶ Pouca margem para mudanças inesperadas ou revisões
- ▶ Clientes/usuários não sabem exatamente seus requisitos até que eles veem o software funcionando, gerando mudanças nos requisitos e todas as fases subsequentes
- ▶ Exclui/distancia o cliente/usuário final das fases intermediárias
- ▶ Deixa os testes para as últimas fases
- ▶ Movimento agile fala que o Waterfall é ineficiente (ou seria um argumento de mercado para reserva de mercado de trabalho, venda de cursos, certificações)

Modificações

- ▶ Waterfall incremental: os requisitos podem ser segmentados em fases incrementais do produto, cada um sendo desenvolvido independentemente
- ▶ Modelo sashimi de Peter DeGrace: waterfall com sobreposição de fases
- ▶ Waterfall com subprojetos

Modelo V

Conceito

É uma extensão do Waterfall

Em vez de se mover sempre “para baixo”, a fase de codificação representa um inflexão “para cima” em direção as fases de validação

Cada fase de verificação/desenvolvimento possui uma fase análoga de validação

Fases de verificação

- ▶ Análise de requisitos: necessidades do usuário
- ▶ Design do sistema: organização geral do sistema
- ▶ Design de arquitetura: lista de módulos, funcionalidades, relação de interfaces
- ▶ Design detalhado/dos módulos: design mais baixo nível com explicações para que o programador consiga começar imediatamente

Fases de validação

- ▶ Testes unitários: teste do design dos módulos
- ▶ Testes de integração: teste da arquitetura
- ▶ Testes de sistema: teste do design do sistema
- ▶ Teste de aceitação do usuário: teste de aceitação dos requisitos

Críticas

- ▶ Demasiadamente simples, levando a gestão um falso senso de segurança, um modelo de desenvolvimento de software que serve gerentes de projeto, advogados, contadores mas não para os desenvolvedores
- ▶ Rígido, estrutura linear e pouco espaço para responder a mudanças
- ▶ É apenas uma variação do Waterfall, sofrendo das mesmas críticas

Vantagens

Mesmos do Waterfall

Ilustração

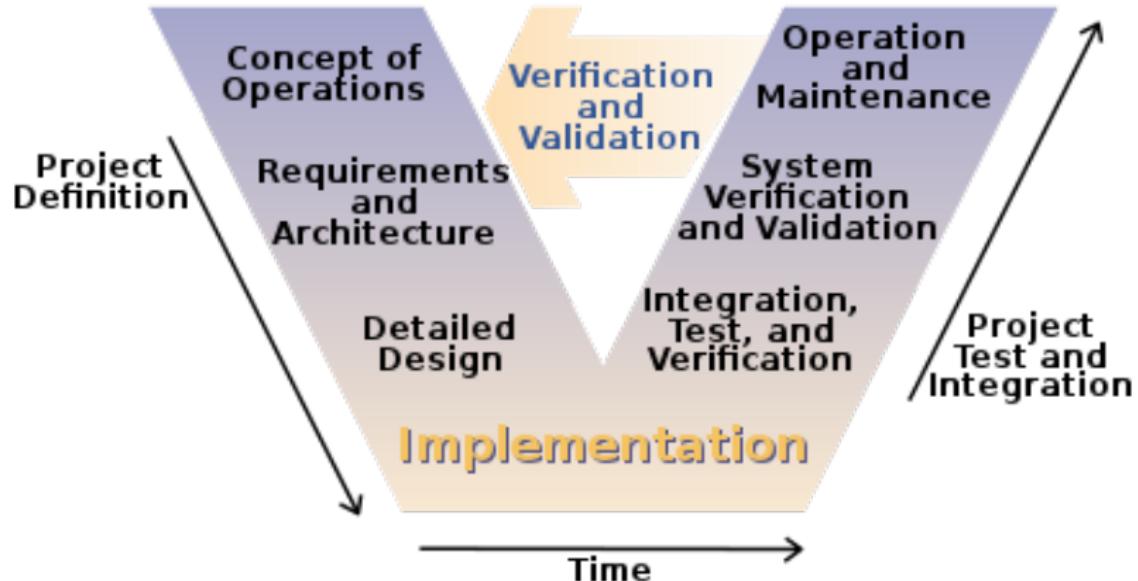


Figure 2: Modelo V

Incremental

Conceito

Produto é projetado, implementado e testado de forma incremental até que o produto é finalizado

Combina elementos da filosofia Waterfall e Prototípica

Produto é decomposto em componentes, cada um é projetado e construído separadamente (builds) e entregue ao cliente quando completado. Assim, é possível utilizar partes do produto e reduz tempo de espera, despesas de capital e o efeito (traumática) de introduzir um sistema completamente novo de uma vez só

Modelo

Uma série de lançamentos referidos como “incrementos” em que cada incremento dá mais funcionalidades, sendo que no primeiro deles, as funcionalidades centrais são entregues. Com feedback do cliente, é desenvolvido os próximos incrementos. O processo continua até que o produto é entregue

A filosofia incremental é adotada pelo agile

Características

1. Sistema é particionado em pequenos projetos de desenvolvimento
2. As partes são construídas para formar o sistema final
3. As prioridades mais altas são abordadas primeiro
4. Os requisitos são “congelados” assim que o incremento é entregue

Vantagens

- ▶ Testes podem ser realizados após cada incremento
- ▶ Clientes podem revisar às novas características e recomendar mudanças
- ▶ Entrega inicial é mais rápida e menos custosa

Desvantagens

- ▶ Custo final pode ser maior que o esperado
- ▶ A medida que funcionalidades são adicionadas, problemas de arquitetura podem surgir e se tornar mais evidentes

Ilustração

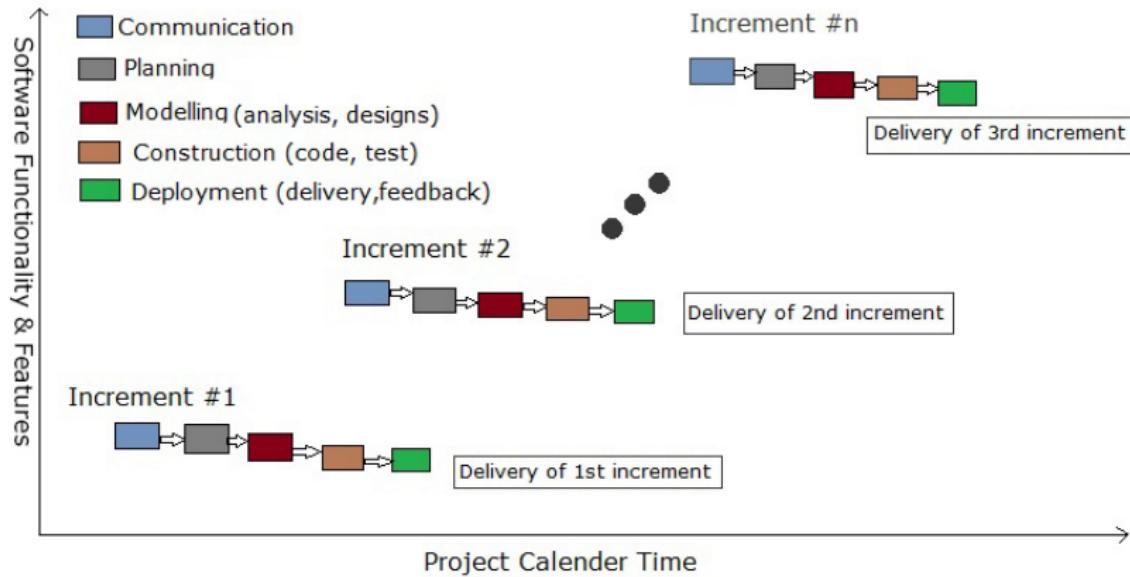


Figure 3: Incremental

Prototipal

Conceito

Atividade de criar protótipos de software (versões incompletas ou em desenvolvimento)

Protótipo simula apenas alguns aspectos do produto final

Visão geral

Próposito do protótipo é permitir que os usuários avaliem a proposta de software ao experimentá-lo

Contrário a visão monolítica de desenvolver o programa inteiro antes de mostrar ao usuário final

Etapas

1. Requisitos básicos
2. Desenvolver o primeiro protótipo
3. Revisão do usuário final
4. Melhora do protótipo: voltar a fase 3 até que o usuário esteja satisfeito

Dimensões

- ▶ Horizontal: dá todas as funcionalidades de todo sistema, mas em um nível baixo de interação (acesso ao banco de dados diretamente, chamadas de API)
- ▶ Vertical: elaboração completa de uma única função ou subsistema, ideal para obter feedback dessa função

Tipos

- ▶ Dispensável: protótipo rápido que não fará parte do produto final, serve apenas para demonstração
- ▶ Evolucionário: criar um protótipo robusto e ir melhorando a partir dele
- ▶ Incremental: vários protótipo são construídos separadamente, ao final são combinados em um único design
- ▶ Extremo: usado em aplicações web. Primeiro é feito o HTML, em seguida as telas são programadas para serem funcionais, em terceiro lugar os serviços são implementados. Segunda fase é chave, já que a interface não se importa com o serviço em si, mas apenas com o contrato

Vantagens

- ▶ Reduz tempo e custo: melhorando a qualidade dos requisitos, implementando mudanças nos primeiros protótipos
- ▶ Engajamento do usuário: usuário irá interagir com o protótipo, dando feedback e considerações

Desvantagens

- ▶ Análise insuficiente: protótipo limitado podem cegar/distrair desenvolvedores do projeto como um todo, negligenciando soluções melhores a longo prazo, soluções que são difíceis de dar manutenção e não escalam
- ▶ Confusão do usuário com protótipo versus produto final
- ▶ Tempo excessivo gasto com um protótipo
- ▶ Custo inicial de implementar um protótipo

Casos de uso

Protótipos podem ser utilizados quase em todos os projetos

Os que mais se beneficiam são aqueles que possuem muitas interações com o usuário, como sistemas online

Sistemas com pouca interação com usuário - processamentos intensivos ou cálculos - se beneficiam pouco

Ferramentas

- ▶ Geradores de telas
- ▶ Ferramentas de design
- ▶ Fábricas de software (drag and drop)

Ágil

Conceito

Reunião de várias abordagens em que os requisitos e as soluções evoluem de forma colaborativa pela autoorganização e times com expertises mistas e o cliente

Defende o planejamento adaptativo, desenvolvimento evolucionário, entregas sem demora, melhora contínua e flexibilidade de resposta frente a mudanças

Histórico

Métodos iterativos e incrementais surgiram nos anos 1950, e gerenciamento de projetos evolucionário e desenvolvimento de software adaptativo nos anos 1970

Durante os anos 1990, uma série de metodologias compactas surgiram em reação aos paradigmas padrão e robustos (Waterfall) por serem muito rígidas e com microgerenciamento, como o desenvolvimento de aplicações rápida (RAD) de 1991, Scrum de 1995, programação extrema em 1996 e outras. Hoje, coletivamente conhecidas como metodologias do movimento Ágil

Em 2001, um grupo de 17 desenvolvedores de software se reuniram em Utah para publicar o Manifesto para o Desenvolvimento de Software Ágil. Em 2005, um grupo de gerentes de projeto fizeram uma Declaração de Interdependência entre o gerenciamento de projetos e o ágil. Adendos foram feitos como Manifesto do Artesão de Software em 2009 e o compêndio Guia para os Princípios Ágeis em 2011

Manifesto para o Desenvolvimento de Software Ágil

Tradução em português

Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar:

Indivíduos e interações mais que processos e ferramentas
Software em funcionamento mais que documentação abrangente
Colaboração com o cliente mais que negociação de contratos
Responder a mudanças mais que seguir um plano

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.

12 Princípios do Software Ágil

We follow **these principles**:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

12 Princípios do Software Ágil - continuação

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity—the art of maximizing the amount of work not done—is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Visão geral

- ▶ Iteração, incremento
- ▶ Comunicação eficiente e face a face
- ▶ Tempo curto entre feedbacks
- ▶ Foco na qualidade

Iteração

- ▶ Quebra do produto em pequenos incrementos
- ▶ Iterações (ou sprints no jargão) são períodos curtos entre uma e quatro semanas em que todo time com funções mistas trabalha em todas as etapas (planejamento, design, codificação, testes e validação) e ao final uma entrega de valor para os stakeholders

Comunicação

- ▶ Funcionários da mesma equipe devem estar juntos para estabelecer uma identidade e melhorar a comunicação
- ▶ Independente do método, o time deve incluir um representante do cliente (*product owner* no caso do Scrum) e que age em nome dos stakeholders e está sempre disponível para o time de desenvolvimento para questões e esclarecimentos
- ▶ Ao final da iteração, os stakeholders e o cliente revisa o progresso e reavalia as prioridades
- ▶ É comum existir um display físico da evolução do projeto para que todos possam ver e estar cientes do desenvolvimento

Tempo curto entre feedbacks

Uma característica comum são as reuniões diárias entre o time, normalmente em pé (daily no Scrum)

Em resumo, cada membro fala o que fez no dia anterior em relação ao objetivo da equipe, o que pretende fazer hoje, e quaisquer bloqueios ou impedimentos que se preveja

Qualidade

Ferramentas e técnicas específicas são utilizadas como: integração contínua, testes unitários automatizados, programação em par, desenvolvimento orientado a testes, padrões de design, desenvolvimento orientado a comportamento, desenvolvimento orientado a domínio, refatoração de código e outras técnicas para entregar valor ao final de cada iteração

Casos

Ágil se destina a sistemas e produtos complexos, com características dinâmicas, não determinísticas e não lineares

Estimativas, planos e previsões são difíceis no início

Ágil vs Waterfall

Waterfall o teste é separado e posterior a codificação; em Ágil, ambas acontecem ao mesmo tempo

A abordagem iterativa apoia em pensamento de produto em vez de projeto. Isso provê flexibilidade no processo enquanto que no projeto os requisitos são definidos e bloqueados do início ao fim.

Desenvolvimento iterativo permite respostas frente a mudanças de negócio e de mercado

Pela questão da iteração, tem fortes laços com o conceito de *Lean Startup*

Críticas

- ▶ Steven Rakitin escreveu uma carta a IEEE criticando o ágil como sendo uma tentativa de enfraquecer a disciplina de engenharia de software e criticando fortemente o descaso com a documentação
- ▶ Falta de visão do produto final, falhas de design exigindo refatorações massivas no sistema
- ▶ Aumento indefinido do número de casos de uso (histórias do usuário)
- ▶ Falta de um sponsor
- ▶ Treinamento insuficiente
- ▶ Figura o Dono do Produto não estar preenchida
- ▶ Preparação e planejamento excessivo

Críticas - continuação

- ▶ Dailies demoradas e divergentes
- ▶ Scum master sendo um programador
- ▶ Falta de testes automatizados
- ▶ Permitir que os débitos técnicos se acumulem
- ▶ Tentar resolver muitos problemas em uma única iteração
- ▶ Tempo, recursos e qualidade são fixos, mas o escopo é variável
- ▶ Burnout dos desenvolvedores
- ▶ Rótulo de mania de gerentes dando novos jargões às boas práticas já existentes

Metodologias

Existem inúmeras metodologias e práticas, veremos as principais no próximo encontro

Personalização da estrutura

A situação, equipe, contexto, habilidades etc devem ser consideradas para se adaptar a metodologia ágil ao que for mais apropriado

Desafios

Ágil tem sido bem visto para certos ambientes, como equipes pequenas de experts trabalhando em projetos inovadores, mas existem muitos desafios em grandes empresas tradicionais com sistemas/infraestruturas legados, muita documentação e negócios bem desenvolvidos e entendidos

Existem diversas adaptações das metodologias para essas realidades