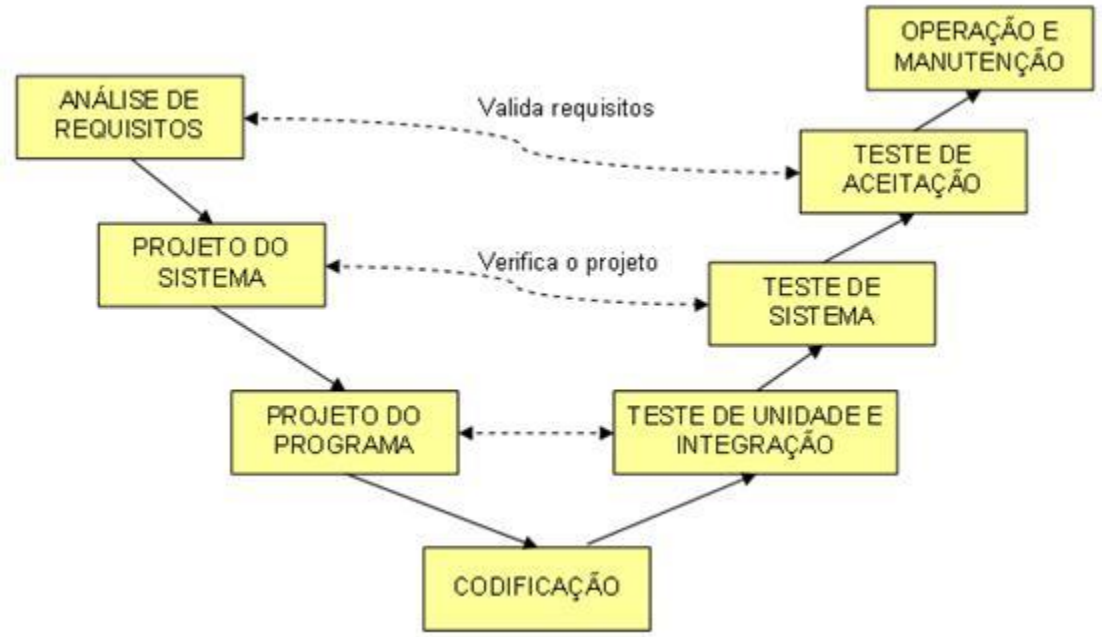
Introdução à Engenharia de Software
Atividade 21 de agosto de 2025

Pesquise sobre modelos de Ciclos de Vida

Modelo V

Desde 1992 o Ministério de Defesa da Alemanha, traz o modelo cascada em forma de "V". Do lado esquerdo do V ficam as análises de requisitos até o projeto, e a codificação fica no vértice e os testes, desenvolvimento, implantação e manutenção, ficam à direita como a figura 1.



A validação e verificação e posto em ênfase nesse modelo que é torna sua característica principal, que difere do modelo em Cascata: cada fase do lado esquerdo gera um plano de testes a ser executado pelo lado direito

O código entra em fase de teste posteriormente do mais baixo nível até o nível sistêmico para que haja a confirmação dos resultados, seguindo os respectivos planos de teste do "Modelo em V": o teste de unidade valida o projeto do programa, *o "teste de sistema"* valida o projeto do sistema e a satisfação do cliente e valida a análise de requisitos

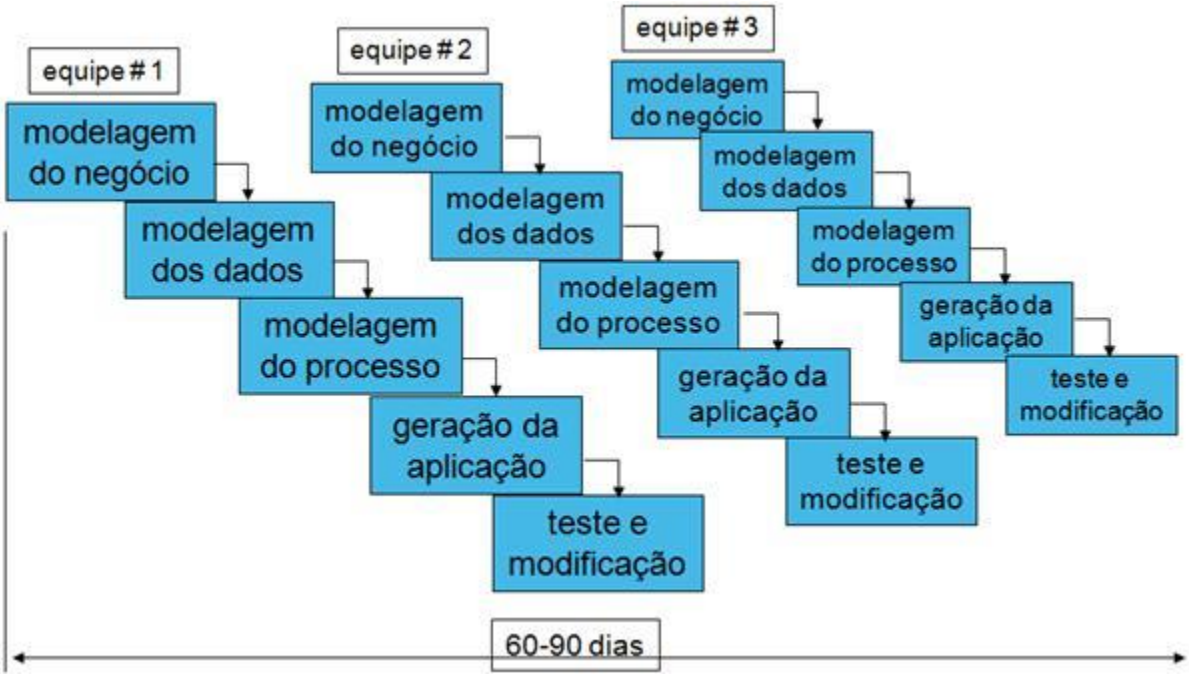
Devido o planejamento prévio do sistema, o cliente só recebe a primeira versão do software no final do ciclo, porém esse modelo apresenta menos riscos da mesma forma que o modelo em cascata

Modelo RAD (Rapid Application Development)

Com a evolução da prototipagem rápida James Martin em 1991 formalizou este modelo, destacando-se pelo desenvolvimento rápido da aplicação. O ciclo de vida e extremamente maior, de forma a encontrarem-se exemplos, na literatura de duração de 60 a 90 dias. É ideal para clientes buscando soluções pioneiras no mercado.

É um ciclo de vida incremental, interativo, onde e preferível que os requisitos tenham escopo restrito. Em comparação ao ciclo anterior o paralelismo das atividades e o grande foco, requerendo, assim, módulos bastantes independentes. Equipes diferentes trabalham ao mesmo tempo em incrementos para o software.

Além do paralelismo, a conquista do baixo tempo se dá graças à compreensão da fase de requerimentos e da fase de implantação. Workshop no lugar de entrevistas são usados para tornar o projeto mais ágil, e o desenvolvimento começa com um nível maior de abstração dos requisitos. Isso só e possível pois o usuário está mais envolvido e consegue ter uma visão do protótipo mais cedo, abaixo a figura 2 mostra como as equipes trabalhariam



A fábrica de software que resolvem adotar esse modelo deve ter uma estrutura previa diferencial de pessoal e ferramentas

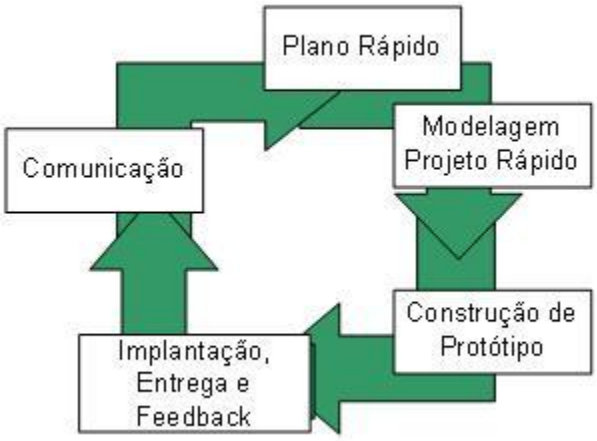
Prototipagem

Um exemplar construído com modelo de prototipagem vem do entendimento de requisitos capturados do cliente. Pode ser considerado um ciclo de vida ou pode ser usado como uma ferramenta para outros ciclos de vida.

Na engenharia de software a prototipagem pode ser assemelhar ao desenho de uma tela, um software contendo pequenas funções do sistema. É considerado operacional (quando estão já pode ser usado pelo usuário final em ambiente de produção) ou então não operacional (não aptos para o ambiente de produção). Podendo ser reaproveitados ou descartados evoluindo até a versão final.

O nível de conhecimento dos requisitos não é tão exigente nesse modelo nos primeiros momentos. Isso é útil quando os requisitos não são amplamente conhecidos ou complexos ou mesmo confusos. Desta forma se o cliente não expressar o que deseja (muito frequente com o sistema não é legado) evitando perder tempo e recurso a prototipagem e a melhor forma de evitar mal entendimento entro cliente e desenvolvedor

Com esse tipo de modelo o cliente experimentara na prática parte do sistema na prática. Nesse primeiro contato o cliente expressa o que ficou mal-entendido, e aprofunda alguns conceitos e descobre o que realmente precisa. Com esses feedbacks, novos requisitos são colhidos e projeto toma maior profundidade. Outro protótipo e gerado e apresentado ao cliente que volta a expor ideias não colocadas na última apresentação, nesse modelo o cliente participa ativamente do projeto como mostra a figura 3.

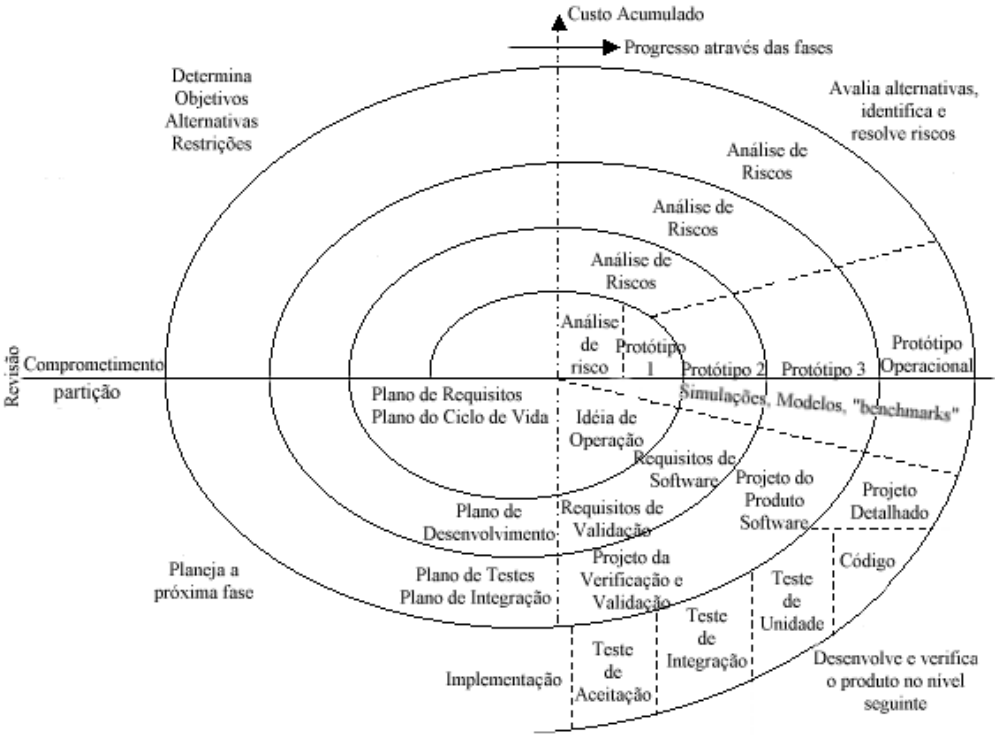


Espiral

O modelo de Boehm (1988) propõe um processo iterativo em que cada ciclo gera uma versão evoluída do sistema.

O modelo é voltado para projetos complexos e de grande porte, priorizando a análise de riscos a cada ciclo. Ele utiliza protótipos, mesmo que não funcionais, para garantir a participação ativa do cliente nas decisões.

Cada ciclo do modelo corresponde a uma etapa do desenvolvimento de software, como análise de viabilidade, definição de requisitos, construção e testes, sendo ainda dividido em quatro partes principais.



A imagem mostra o modelo espiral, que organiza o desenvolvimento de software em ciclos. Cada ciclo é dividido em quatro partes:

1. Definição de objetivos – identifica metas, funcionalidades, alternativas e restrições, gerando um plano detalhado.
2. Análise de riscos – avalia riscos e alternativas, podendo usar protótipos para apoio.
3. Desenvolvimento e validação – escolhe o método de construção (cascata, incremental etc.) e implementa o sistema.
4. Planejamento da próxima fase – revisa os resultados e decide os próximos passos.

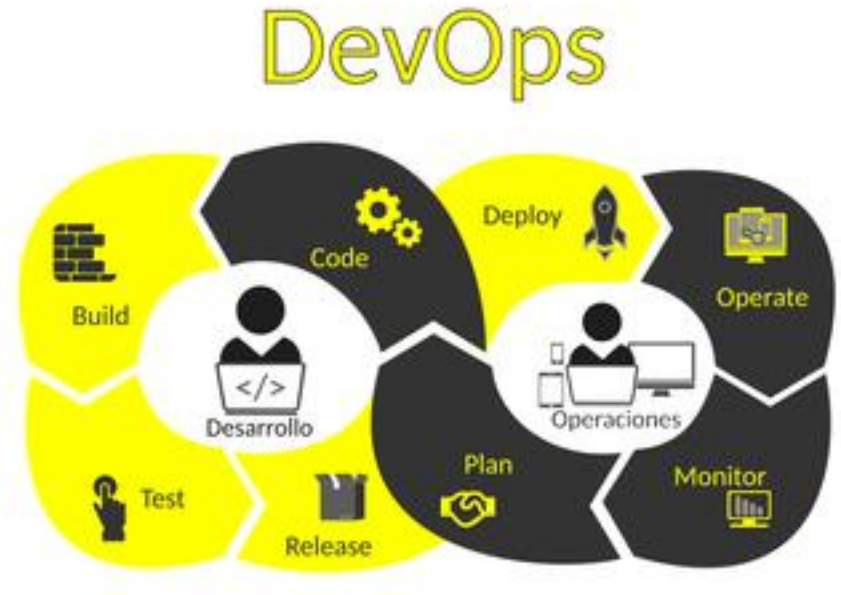
Assim, cada volta da espiral repete essas etapas, evoluindo o projeto.

DevOps

O modelo DevOps é baseado na integração contínua entre desenvolvimento e operações, buscando entregas rápidas, confiáveis e automatizadas. Ele é iterativo e contínuo, diferente de modelos lineares. As principais fases são:

1. Planejamento – definição de metas, backlog, requisitos e priorização.
2. Codificação – desenvolvimento do software, integração de ferramentas e versionamento.
3. Build/Integração Contínua – compilação, testes automáticos e integração do código.
4. Teste – testes unitários, de integração, de performance e segurança.
5. Release/Implantação Contínua – entrega automatizada e frequente em ambientes de produção ou pré-produção.
6. Operação/Monitoramento – acompanhamento de métricas, logs, performance e disponibilidade.
7. Feedback/Melhoria Contínua – coleta de dados de uso, incidentes e sugestões para ajustes e novas entregas.

Fluxo do DevOps: As fases formam um ciclo infinito (como um laço), representando integração e entrega contínua.



Apresente vantagens e desvantagens

1. Modelo em V

Vantagens:

- Forte ênfase em verificação e validação desde o início.
- Planejamento claro de testes para cada fase, garantindo qualidade.
- Boa para projetos bem definidos, com poucos riscos de mudanças.

Desvantagens:

- Pouca flexibilidade para mudanças de requisitos após o início.
- Cliente só vê o produto no final do ciclo.
- Pode ser caro e demorado para ajustes tardios.

2. Modelo RAD (Rapid Application Development)

Vantagens:

- Entregas rápidas (60 a 90 dias), ideal para soluções com prazos curtos.
- Permite feedback constante do cliente por meio de protótipos.
- Equipes trabalham em paralelo, acelerando o desenvolvimento.

Desvantagens:

- Exige equipes qualificadas e bem estruturadas.
- Melhor para projetos com escopo restrito e requisitos relativamente claros.
- Pode gerar problemas de integração se os módulos não forem bem coordenados.

3. Prototipagem

Vantagens:

- Bom para requisitos pouco claros ou complexos.
- Cliente participa ativamente, evitando mal-entendidos.
- Permite testes iniciais e coleta de feedback antes do sistema final.

Desvantagens:

- Pode gerar expectativas irreais sobre prazos e funcionalidades.
- Protótipos descartados podem significar esforço extra.
- Necessita boa comunicação entre cliente e equipe para não perder o foco.

4. Modelo Espiral

Vantagens:

- Focado na análise de riscos, ótimo para sistemas grandes e críticos.
- Combina características de outros modelos (incremental, prototipagem, cascata).
- Envolvimento constante do cliente em cada ciclo.

Desvantagens:

- Mais complexo e caro de gerenciar.
- Exige experiência da equipe para análise de riscos.
- Pode ser excessivo para projetos pequenos ou simples.

5. DevOps

Vantagens:

- Integração contínua e entregas rápidas e frequentes.
- Automatização reduz falhas humanas e acelera o ciclo de vida.
- Forte cultura de monitoramento e feedback, favorecendo melhorias contínuas.

Desvantagens:

- Requer maturidade cultural e técnica da equipe.
- Alto investimento inicial em ferramentas e infraestrutura.
- Nem todos os clientes ou ambientes permitem releases contínuos.