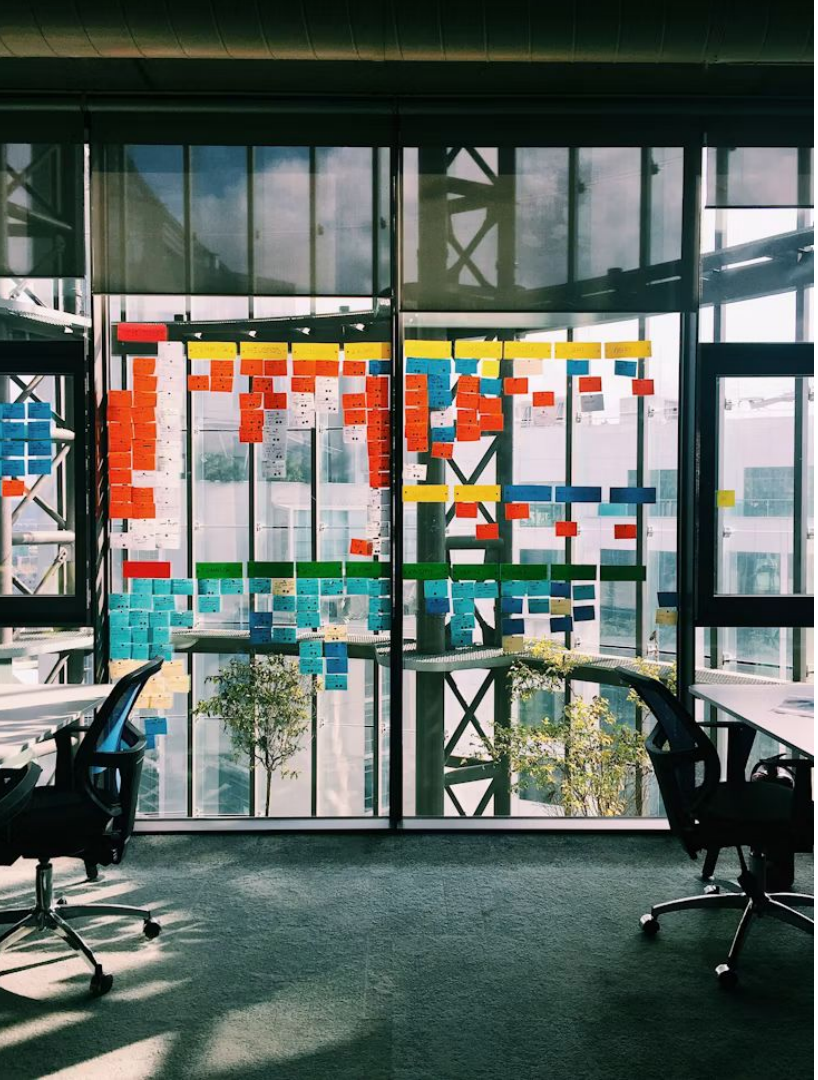


5954014 - Engenharia de Software

Aula 03 - Ciclo de Vida e Modelos de Processos

Prof. Dr. Denis Martins

DCM | FFCLRP | USP

Objetivos de Aprendizagem

01. Definição

Definir o que é um processo de software e compreender detalhadamente a sua importância para o desenvolvimento de sistemas.

02. Atividades

Reconhecer as quatro atividades fundamentais de engenharia que estão presentes em absolutamente qualquer projeto.

03. Cascata

Compreender a fundo o funcionamento do Modelo Cascata, analisando criticamente os seus sucessos e limitações de projeto.

04. Alternativas

Explorar e identificar as principais alternativas metodológicas ao tradicional Modelo Cascata no desenvolvimento moderno.

Créditos: **slides baseados no material de aula original da Profa. Alessandra Alaniz Macedo**

Engenharia de Software é uma abordagem sistemática, disciplinada e capaz de ser medida ao longo de um processo de construção de um software.

Conceito Fundamental: Processo

Origem do Termo

procedere

Verbo em latim que indica a ação de **avançar**, ir para frente (*pro* + *cedere*).

Conceito Prático

Conjunto de Ações

Série de atividades e tarefas coordenadas que objetivam **atingir uma meta** ou produzir um resultado esperado.

Processos em Engenharia de Software

Passos Ordenados

Conjunto de passos ordenados que guiam o processo de desenvolvimento.

Objetivo

Entregar um produto de software de maneira eficiente, previsível e que atinja as necessidades de negócio.

Atividades

Geralmente inclui análise de requisitos, projeto de software, programação, testes, entre outras tarefas.

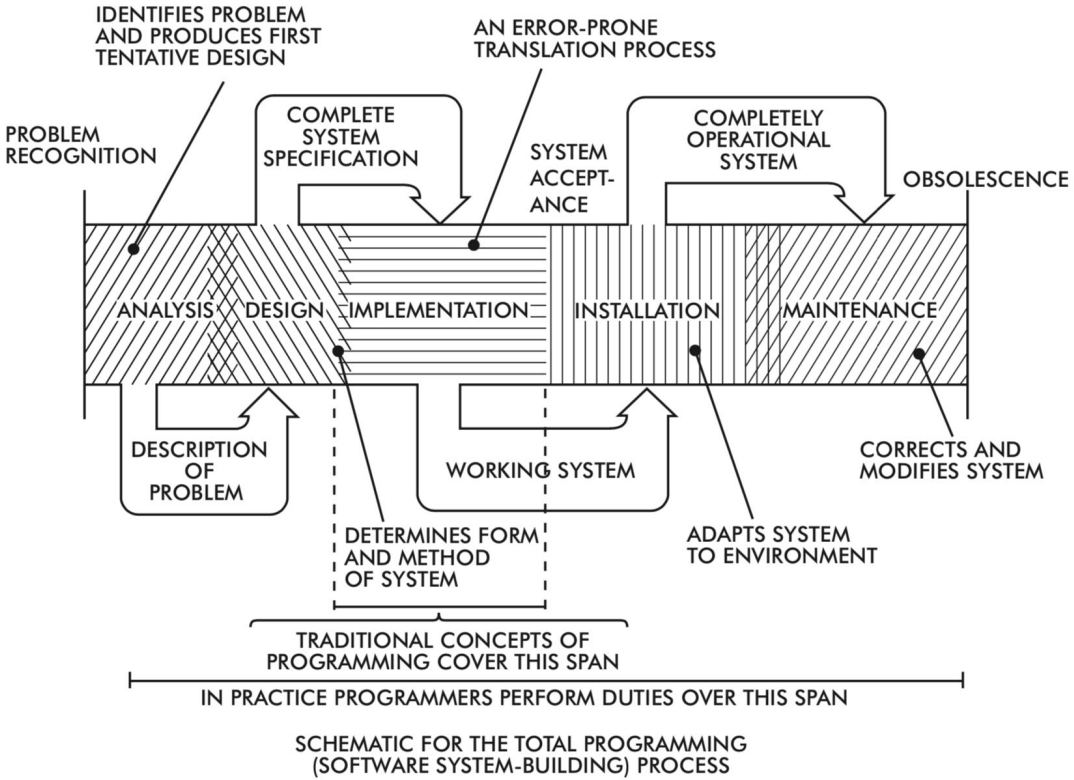
Define um conjunto de passos, tarefas, eventos e práticas.

Revisando

01 **Mapear o processo de construção de software é importante**

02 **Importância do projeto, da modularidade e do reuso**

03 **Planejamento detalhado e atividades sequenciais**



Fonte: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

Processo de Software

O que é um Processo?

Diferentes visões consagradas na literatura:

- **Sommerville**: Conjunto de atividades para especificação, projeto, implementação e teste de software.
- **Pressman**: Roteiro, ou conjunto de passos, previsível que ajuda a criar a tempo um software de alta qualidade.

Modelos de Processo

Como estruturamos e compreendemos o fluxo:

- **Representação Abstrata**: É uma representação simplificada ou abstrata de um processo de software.
- **Perspectiva Particular**: É a descrição de um processo a partir de uma perspectiva ou ponto de vista específico.

Processo de Software

O que prescreve?

- **Ordem e Frequência:** Prescreve a ordem e a frequência de cada fase ou atividade do projeto.
- **Critérios de Transição:** Especifica critérios claros para mudar de uma fase para outra.
- **Entregas Claras:** Define exatamente o que tem que ser entregue ao final de cada etapa.

O que NÃO significa?

- **Sobrecarga:** Não deve atuar como um processo burocrático engessado ou pesado.
- **Papelada Inútil:** Não gera documentação desnecessária que não agregue valor real.
- **Perda de Tempo:** Longe de ser desperdício, é uma otimização estratégica de equipe.

Resultados Positivos

- **Cronograma:** Essencial para guiar as estimativas e atender aos prazos com eficácia.
- **Mais Qualidade:** Garante a aplicação de padrões consistentes em todo o ciclo de vida.
- **Fácil de Manter:** Resulta em um código de arquitetura limpa e de fácil manutenção posterior.

Modelo de Processo de Software

Conceito Fundamental

Um modelo é uma representação **abstrata** de um processo de desenvolvimento de software.

Sequenciais

Abordagens lineares onde o desenvolvimento flui sistematicamente através de fases bem definidas, dependendo da conclusão da etapa anterior.

Incrementais

Focado em entregar o software em partes funcionais progressivas, permitindo feedback constante e adaptação rápida a novas necessidades.

Prototipagem

Baseado na construção de versões de testes rápidas e preliminares para experimentar, validar requisitos e refinar os fluxos do sistema com usuários.

O que todo Processo de Software inclui?

Requisitos / Especificação

Definir detalhadamente o que o sistema deve fazer e quais as necessidades dos usuários.

Projeto & Implementação

Definir a arquitetura técnica, estruturar o banco de dados e escrever o código-fonte.

Validação & Testes

Verificar rigorosamente se o software atende ao que foi solicitado e corrigir possíveis falhas.

Evolução & Manutenção

Adaptar e expandir o software continuamente para atender a novas e futuras necessidades.

Atividade em Grupo

Instruções da Atividade

Ao lado estão tarefas de um projeto de software.
O grupo deve realizar as seguintes ações:

Classificar cada tarefa em uma das quatro atividades fundamentais:

- **Especificação,**
- **Projeto,**
- **Validação,** ou
- **Evolução**.

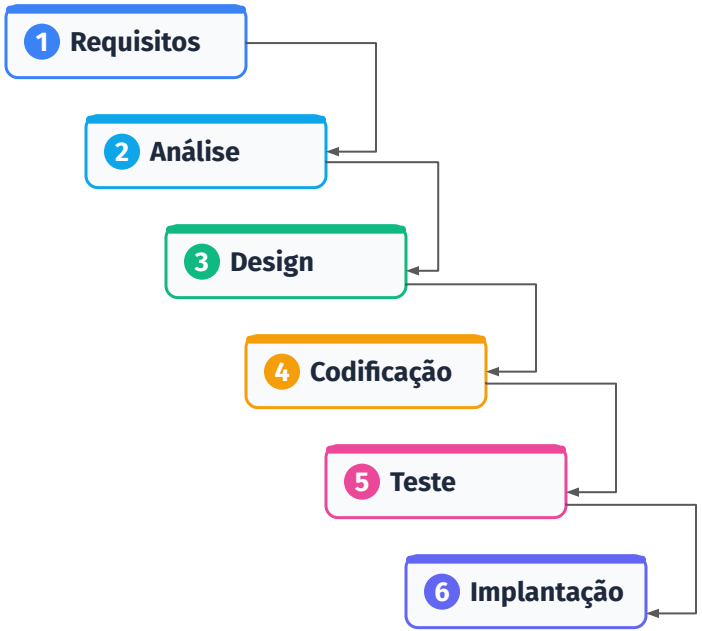
Identificar e justificar as **três tarefas** mais difíceis de classificar (as "zonas cinzentas").

Tarefas para Classificação

1. Escrever o manual do usuário.
2. Descobrir com o cliente qual a funcionalidade mais importante.
3. Consertar um bug reportado em produção.
4. Desenhar o diagrama o fluxo de dados do sistema.
5. Escrever um *script* de teste automatizado.
6. Negociar cronograma e prazos com o cliente.
7. Migrar dados de uma planilha legada para o novo banco de dados.
8. Realizar a revisão de código (*code review*) de um colega.
9. Escolher qual banco de dados relacional será utilizado.
10. Apresentar um protótipo navegável para validação do cliente.
11. Atualizar o sistema para nova versão do navegador.
12. Numerar e catalogar requisitos em documento.

Processo no Modelo Cascata (Waterfall) - Sequencial

Modelo tradicional nas engenharias: O resultado de uma fase é a entrada da próxima



01. Requisitos: Definição do escopo, levantamento de necessidades com os stakeholders e documentação do que o sistema deve fazer.

02. Análise: Modelagem do sistema, refinamento de regras de negócio, casos de uso e estudo detalhado de viabilidade técnica do projeto.

03. Design: Desenho da arquitetura geral do software, definição de esquemas de banco de dados e especificação de interfaces de usuário (UI/UX).

04. Codificação: Fase de desenvolvimento ativo onde programadores implementam as funcionalidades em código-fonte com base nos designs.

05. Teste: Validação da qualidade geral, execução de rotinas de verificação de bugs, testes de integração e homologação do software para uso.

06. Implantação: Lançamento oficial da aplicação estável no ambiente produtivo e entrega final de valor para o cliente e usuários finais.

As Particularidades de Software

Engenharia Tradicional

Lida com restrições e propriedades de materiais concretos.

- **Processos rígidos** e de difícil alteração pós-construção.
- **Produtos Físicos:** Carro, ponte, celular (hardware).
- Sujeito ao **desgaste** físico e depreciação material.

Engenharia de Software

O software possui uma dinâmica inteiramente nova, intangível e maleável.

- **Evolução contínua** baseada em lógica e processos digitais.
- **Produto Abstrato:** Não possui limitações de peso, volume ou matéria física.
- Não se desgasta**,** mas **se deteriora.**

As Particularidades de Software

Engenharia Tradicional

Lida com restrições e propriedades de materiais concretos.

- **Processos rígidos** e de difícil alteração pós-construção.
- **Produtos Físicos:** Carro, ponte, celular (hardware).
- Sujeito ao **desgaste** físico e depreciação material.

Engenharia de Software

O software possui uma dinâmica inteiramente nova, intangível e maleável.

- **Evolução contínua** baseada em lógica e processos digitais.
- **Produto Abstrato:** Não possui limitações de peso, volume ou matéria física.
- Não se desgasta**,** mas **se deteriora.**

Modelo tradicional Waterfall não funcionou para Software!

Algumas Causas e Desafios

Desafio 1: Requisitos

Dificuldade de Definição

- **Clientes não sabem o que querem:** É um desafio comum entender as reais necessidades no início.
- **Funcionalidades "infinitas":** É extremamente difícil prever todas as variáveis.
- **O mundo muda:** O contexto do negócio se transforma rapidamente.

O Fator Tempo

Risco de Obsolescência

- **Ciclos muito longos:** Não funciona mais passar 1 ano levantando requisitos, 1 ano projetando e 1 ano implementando.
- **Resultado inevitável:** Quando o software finalmente ficar pronto, **ele já estará obsoleto!**

Algumas Causas e Desafios

Desafio 2: Documentação

Sintomas e Inutilidade

- **Extensas e pouco úteis:** Textos extremamente longos que raramente agregam valor real ao dia a dia do projeto.
- **Desconsideradas na prática:** Frequentemente ignoradas pelos desenvolvedores durante a fase de implementação real.

Falha de Paradigma

Plan-and-Document

- **Modelo Incompatível:** Tentar prever, planejar e documentar tudo antes de iniciar a construção simplesmente não funciona.
- **A Realidade do Software:** O código muda rapidamente, tornando documentações estáticas obsoletas quase que imediatamente.

Algumas Causas e Desafios

O Fator Espera

O Cliente Paciente

- **Exigência de paciência:** O modelo pressupõe que o cliente seja passivo e aguarde longos períodos.
- **Entrega tardia:** Software funcional só fica disponível em etapa criticamente avançada.
- **Risco altíssimo:** Descobrir desvios apenas no final gera retrabalho massivo e obsolescência.

O Fator Processo

Iterações e Desvios

- **Iterações onerosas:** Custos de documentação tornam as revisões caras e geram muito retrabalho.
- **Fases atropeladas:** É comum suspender etapas e prosseguir sem a devida validação prévia.
- **Desalinhamento final:** Softwares que não atendem ao cliente, exigindo evolução precoce na implantação.

Quando Utilizar o Modelo Waterfall?

CENÁRIO IDEAL

Requisitos Estáveis

Quando os requisitos são muito bem compreendidos e as mudanças são bastante limitadas durante o ciclo de desenvolvimento.

Importante: Poucos sistemas de negócio modernos possuem requisitos verdadeiramente estáveis.

VALOR DO MODELO

Abordagem Estruturada

Melhor do que uma abordagem **casual** de desenvolvimento de software.

Benefício Claro: O Modelo Cascata destaca-se por ser simples, fácil de usar, planejar e gerenciar na prática.

Alternativas ao Modelo Cascata?

Alternativas ao Modelo Cascata?

1. Feedback Cedo

O cliente só vê o software pronto no final do ciclo, gerando alto risco de desalinhamento.

Alternativa: Entregar versões funcionais frequentemente para validar requisitos e mitigar desvios de forma contínua.

Alternativas ao Modelo Cascata?

1. Feedback Cedo

O cliente só vê o software pronto no final do ciclo, gerando alto risco de desalinhamento.

Alternativa: Entregar versões funcionais frequentemente para validar requisitos e mitigar desvios de forma contínua.

2. Abraçar Mudanças

Mudanças de escopo são caras e desencorajadas devido ao congelamento de fases.

Alternativa: Projetar o processo assumindo que os requisitos vão evoluir conforme o negócio se transforma.

Alternativas ao Modelo Cascata?

1. Feedback Cedo

O cliente só vê o software pronto no final do ciclo, gerando alto risco de desalinhamento.

Alternativa: Entregar versões funcionais frequentemente para validar requisitos e mitigar desvios de forma contínua.

2. Abraçar Mudanças

Mudanças de escopo são caras e desencorajadas devido ao congelamento de fases.

Alternativa: Projetar o processo assumindo que os requisitos vão evoluir conforme o negócio se transforma.

3. Redução de Risco

Integração e testes tardios acumulam riscos catastróficos para o final do projeto.

Alternativa: Desenvolver de forma iterativa, permitindo testes de integração contínuos e correção rápida de falhas.

Modelo Incremental

Entregar o produto em fatias completas, com funcionalidades ponta a ponta.

1. Escopo Fatiado

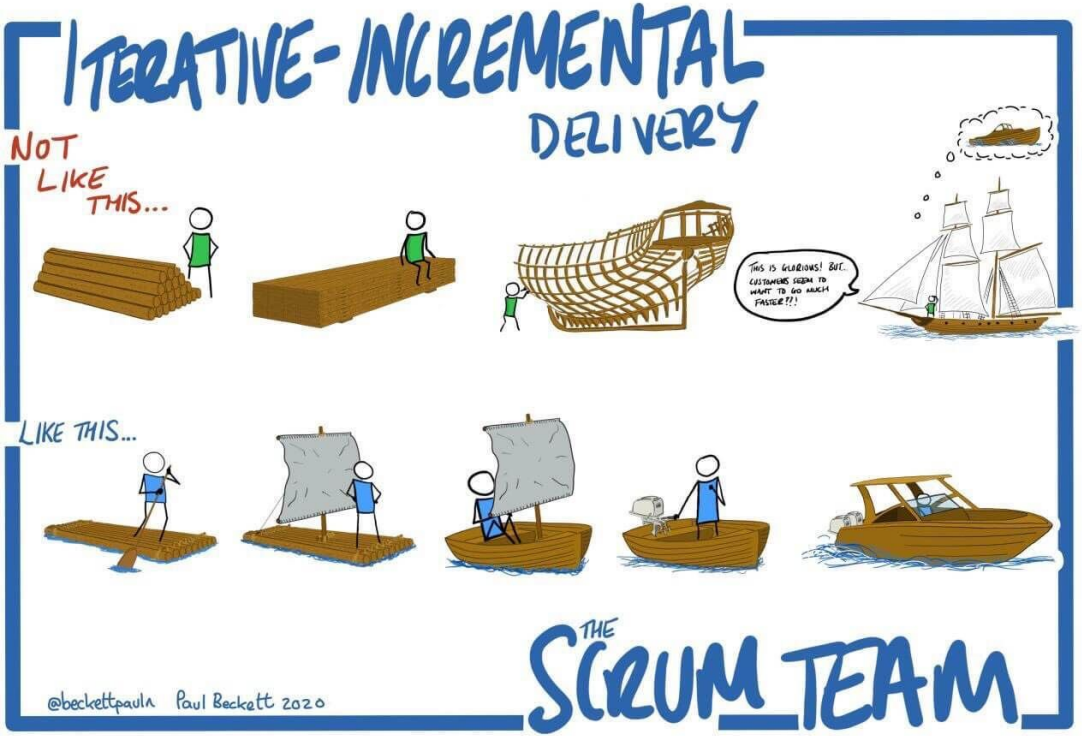
Cada incremento fornece parte da funcionalidade solicitada, entregando valor utilizável ao cliente desde o início.

2. Priorização

Os requisitos do sistema são sempre priorizados de acordo com o valor de negócio.

3. Evolução Controlada

Uma vez iniciado o desenvolvimento de um incremento, seus requisitos específicos são congelados.



Fonte da Imagem: https://productive.io/blog/scrum-framework/

Modelo Incremental

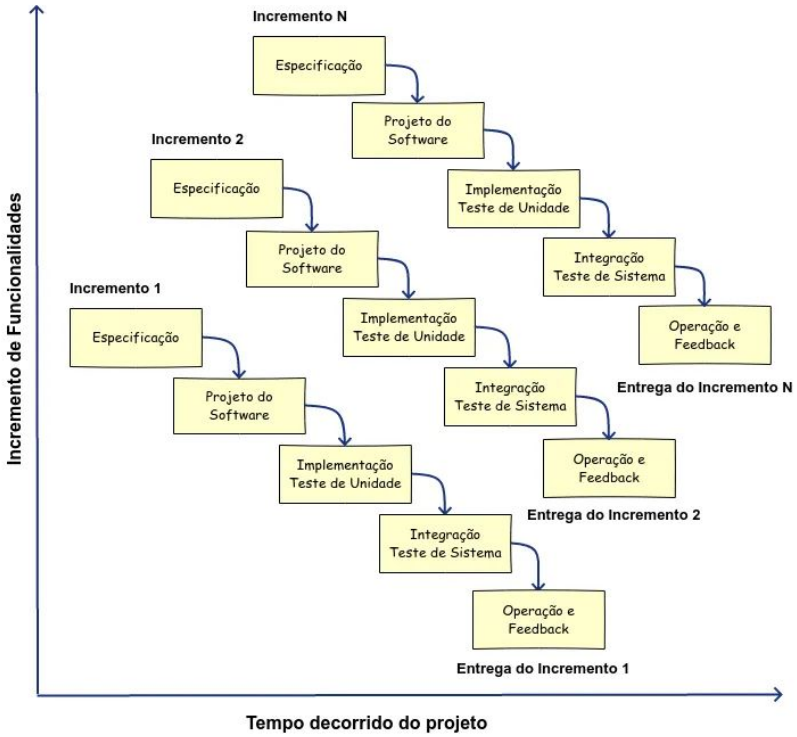
Entregar o produto em fatias completas, com funcionalidades ponta a ponta.

Cada incremento é desenvolvido de forma linear, como no Modelo em Cascata, e em seguida exposto aos comentários dos clientes.

Em caso de alterações na implementação, um novo incremento é desenvolvido o produto resultante é novamente apresentado.

Atividades de Especificação, Projeto, Implementação e Validação são intercaladas, acontecendo em cada nova versão.

Vantagens: Riscos menores de falha geral do projeto e facilidade de receber *feedback*.



Fonte da Imagem: https://medium.com/contexto-delimitado/o-modelo-incremental-b41fc06cac04

Quando Utilizar o Modelo Incremental?

Restrição de Recursos e Prazos
Quando não há mão de obra disponível para uma implementação completa dentro do prazo de entrega estabelecido.

Decomposição em Partes Úteis
Quando o sistema permite ser entregue em funcionalidades completas de ponta a ponta (ex: consultar antes de reservar).

Necessidade de Valor Precoce
Indicado para projetos que precisam gerar resultados e feedback rapidamente, evitando a obsolescência.

Aprendizado do Domínio
Quando a equipe ou o cliente precisam "ver para crer", usando o sistema para descobrir novos requisitos.

Modelo Evolucionário

Necessidade de processo projetado para produtos que sempre evoluem

- Permite elaborar uma versão reduzida em cenários de prazos de entrega curtos
- Ideal para quando os requisitos básicos de um produto ou sistema são bem entendidos

Prototipação

Foca em um "projeto rápido" voltado para os aspectos visíveis ao usuário.

Aprimoramento de Requisitos:

Funciona como uma ferramenta essencial para refinar e validar requisitos através de feedback imediato do usuário.

Modelo Espiral

Associa a natureza iterativa de desenvolvimento ao controle sistemático do modelo em Cascata.

Análise de Riscos Explícita:

Possui como característica chave a análise rigorosa de riscos em cada volta da espiral.

Gera versões progressivamente mais completas conforme o projeto se expande.

Modelo de Prototipação

Protótipo: É um sistema/modelo sem todos os elementos funcionais, mas apenas com as funcionalidades gráficas e algumas básicas. É utilizado para aprovação de quem solicita o software.

Interessante quando:

Definição de Requisitos

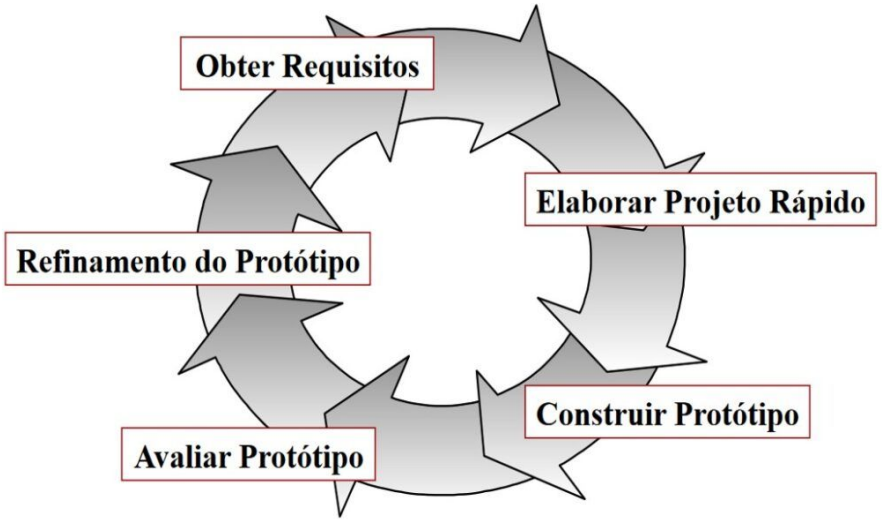
É necessário entender os requisitos do usuário e, então obter uma melhor definição dos requisitos do sistema.

Objetivos Indefinidos

Quando o cliente definiu objetivos gerais para o software, mas ainda não os detalhou, ou quando os requisitos estão confusos.

Insegurança Técnica

Quando o engenheiro de software pode se sentir inseguro quanto à eficiência dos algoritmos, adaptação a mudanças ou hardware específico.



Fonte da Imagem: https://www.nucleodoconhecimento.com.br/engenharia-da-computacao/testes-de-software

Modelo Espiral (Boehm, 1986)

Desenvolvimento Guiado por Riscos

O esforço de desenvolvimento é proporcional ao risco identificado. O foco principal é minimizar incertezas antes de avançar.

Evolução Contínua e Iterativa

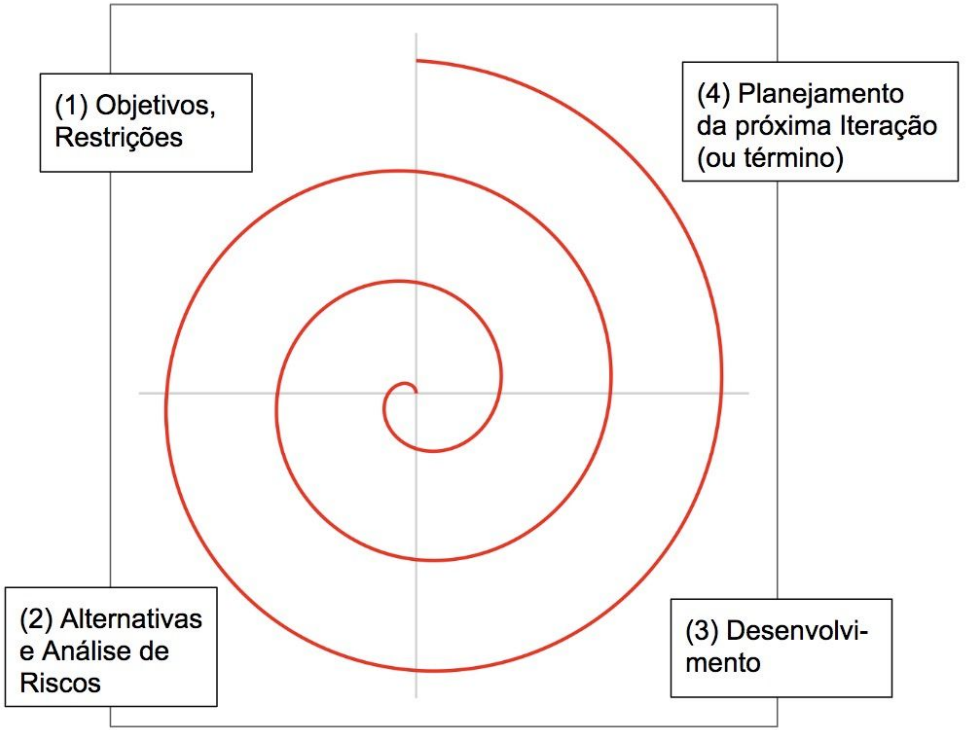
Cada "volta" na espiral resulta em refinamento. Diferente do Cascata, os requisitos evoluem e se consolidam a cada iteração.

Abordagem Híbrida Eficaz

Combina com maestria a natureza sistemática e controlada do modelo em Cascata com a flexibilidade da prototipação.

Processo de Alto Rigor

É considerado um processo "pesado" devido à necessidade de documentação extensiva para garantir a segurança em cada etapa.



Fonte da Imagem: https://engsoftmoderna.info/cap2.html

Outro Modelo: Rational Unified Process (RUP)

Origem e Modelo

Desenvolvido na década de 1990, o RUP surgiu para unificar as melhores práticas de engenharia de software.

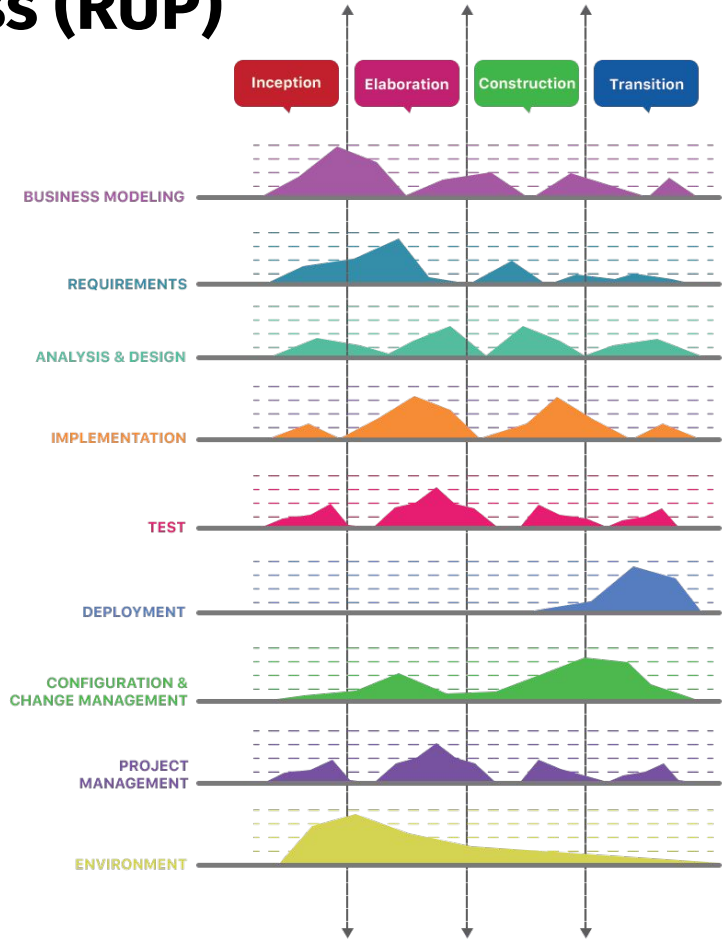
Combina de forma estruturada a abordagem sistemática do modelo em **Cascata (Waterfall)** com a flexibilidade e controle de riscos do modelo **Espiral**.

UML como Padrão

Utiliza a Unified Modeling Language (UML) como padrão essencial para a criação de diagramas e toda a documentação do sistema.

Foco em Arquitetura

Direciona esforços com foco central na definição robusta da arquitetura de software e no mapeamento de casos de uso.



Fonte da Imagem: https://www.weblineindia.com/blog/top-15-software-development-methodologies/

Atividade

Para cada cenário ao lado, escolha o modelo mais adequado (dentre os vistos) respondendo às 5 perguntas fundamentais:

1. Os requisitos vão mudar?
Se sim, congelá-los cedo custará caro.

2. Quanto custa um erro em produção?
Riscos de vida, financeiros ou judiciais pedem mais verificação.

3. O cliente está disponível continuamente?
Sem o cliente presente, o time acaba "inventando" requisitos.

4. O sistema pode ser entregue em "fatias"?
Alguns sistemas não funcionam pela metade (ex: marca-passo).

5. Existe alguma exigência externa?
Contratos de escopo fechado, auditorias ou certificações.

Cenário A: Firmware de Catraca de Acesso

Desenvolvimento do software básico para nova catraca de laboratório. O código é gravado fisicamente no hardware na fabricação; qualquer atualização exige troca física do chip.

Cenário B: Reserva de Espaços Acadêmicos

Novo sistema de reservas. A secretaria (cliente) fica ao lado, os requisitos iniciais são vagos e o sistema pode começar a ser útil permitindo apenas a consulta prévia de salas.

Cenário C: Sistema de Matrícula Crítico

Deve estar obrigatoriamente estável em produção em data fixa e inadiável (1º dia de aulas). Não pode falhar no lançamento, pois não há alternativa manual para o volume.

Entrega: Um parágrafo de justificativa por cenário, utilizando as respostas às cinco perguntas para sustentar a decisão.

Conclusão

Atividades Universais

Especificação, Projeto/Implementação, Validação e Evolução ocorrem em todo projeto, variando apenas a ordem e a frequência.

Decisão de Projeto

A escolha entre modelos deve ser baseada em fatos como incerteza de requisitos e custo do erro.

Processo como Roteiro

É o conjunto de passos que provê estabilidade, organização e coordenação para o esforço da equipe.

Foco no Valor

O objetivo final de qualquer processo é a entrega sustentável de software funcional que resolva problemas reais.

Próxima Aula

Manifesto Ágil: Valoriza indivíduos, software funcional e resposta a mudanças.

Referências da Aula

Valente, M. T. Engenharia de Software Moderna, 2020.

Fox, A.; Patterson, D. Engineering Software as a Service, 2ª ed., 2021.

Sommerville, I. Engenharia de Software, 10ª ed.

Brooks, F. O Mítico Homem-Mês. Alta Books, 2018.

Dúvidas e Discussão

Veja também: Domine o ciclo de vida do software e transforme sua carreira dev (YouTube)