Estratégia Concursos

Aula 00

Caixa Econômica Federal - CEF (Técnico Bancário - TI) Passo Estratégico de Conhecimentos Específicos - 2024 (Pós-Edital)

Autor:
Fernando Pedrosa Lopes

22 de Fevereiro de 2024

METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE

Sumário

CONTEÚDO

Engenharia de Software. Conceitos, camadas, processos. Modelos de Ciclo de vida. Metodologias de Desenvolvimento de Software. Modelo em cascata. Modelo iterativo e incremental. Modelo RAD. Prototipagem. Espiral. Componentes. Métodos formais. Orientação a aspectos.

ANÁLISE ESTATÍSTICA

Inicialmente, convém destacar o percentual de incidência do assunto, dentro da disciplina **Engenharia de Software** em concursos/cargos similares. Quanto maior o percentual de cobrança de um dado assunto, maior sua importância.

Obs.: *um mesmo assunto pode ser classificado em mais de um tópico devido à multidisciplinaridade de conteúdo*.

Assunto	Relevância na disciplina em concursos similares
UML	11.0 %
Processos de Software - Desenvolvimento Ágil	8.4 %
Engenharia de Requisitos	8.2 %
Teste de Software	6.7 %
Métricas de Software	5.1 %
Desenvolvimento de Software	4.6 %
Inteligencia Artificial	4.6 %
Processos de Software	4.6 %
Orientação a Objetos	4.5 %
Gestão de Projetos em Engenharia de Software	3.6 %
Qualidade de Software	3.4 %
Metodologia de desenvolvimento de software	2.5 %

Gerência de Configuração	1.9 %
Geoprocessamento em Engenharia de Software	1.6 %
Prototipação	1.2 %
Conceitos Básicos em Engenharia de Software	0.7 %
Análise Estruturada	0.6 %
Ferramentas CASE	0.4 %
Ferramentas de Desenvolvimento de Software	0.4 %
Software livre	0.3 %
Manutenção de Software	0.1 %
Análise Essencial	0.1 %
Web 2.0	0.1 %
Portal Web	0.1 %
Refatoração	0.1 %
Engenharia da Informação	0.1 %

GLOSSÁRIO DE TERMOS

Faremos uma lista de termos que são relevantes ao entendimento do assunto desta aula. Caso tenha alguma dúvida durante a leitura, esta seção pode lhe ajudar a esclarecer.

Engenharia de Software: É a aplicação de princípios de engenharia para projetar, desenvolver e manter software. Inclui técnicas para gerenciar todo o processo de desenvolvimento de software, garantindo que o produto final seja de alta qualidade, entregue a tempo e dentro do orçamento.

Ferramentas: Em engenharia de software, são programas ou aplicativos usados para auxiliar no desenvolvimento de software, como editores de código, compiladores, debuggers, ferramentas de controle de versão e ambientes de desenvolvimento integrados (IDEs).

Métodos: São as técnicas ou procedimentos usados para realizar um trabalho específico. Em engenharia de software, podem incluir práticas como programação orientada a objetos, programação funcional, análise e projeto de sistemas, entre outros.

Processos: Em engenharia de software, processos referem-se às atividades e tarefas realizadas durante o desenvolvimento de software, desde a concepção até a manutenção do produto final. Os processos podem incluir etapas como coleta de requisitos, design, codificação, teste e manutenção.

Ciclo de Vida de Software: É a sequência de etapas ou fases pelas quais um software passa, desde a concepção inicial até a descontinuação. Normalmente, inclui etapas como análise de requisitos, design, implementação, teste, implantação e manutenção.

Modelo de Ciclo de Vida: É uma representação abstrata do processo de desenvolvimento de software, que descreve as etapas a serem seguidas e a ordem em que devem ocorrer. Exemplos incluem o modelo em cascata, modelo iterativo e incremental, modelo em espiral, entre outros.

Modelo em Cascata: É um modelo de ciclo de vida de software que segue uma sequência linear de fases: análise e definição de requisitos, projeto de sistema e software, implementação e teste unitário, integração e teste de sistema, operação e manutenção.

Análise e Definição de Requisitos: É a fase do ciclo de vida de software onde os requisitos do sistema são coletados, analisados e documentados. Envolve a compreensão das necessidades dos usuários e a definição das funcionalidades que o software deve fornecer para atender a essas necessidades.

Projeto de Sistema e Software: É a fase do ciclo de vida de software onde o sistema é projetado para atender aos requisitos identificados. Envolve a definição da arquitetura do sistema, escolha de tecnologias, modelagem de dados e definição de interfaces.

Implementação e Teste Unitário: É a fase do ciclo de vida de software onde o software é codificado e os testes unitários são realizados. Envolve a escrita de código para implementar as funcionalidades definidas, e a realização de testes para verificar se cada unidade ou componente do software funciona corretamente.

Integração e Teste de Sistema: É a fase do ciclo de vida de software onde as unidades ou componentes individuais são combinados e testados como um sistema completo. Envolve a verificação de que todas as partes do sistema funcionam juntas corretamente e que o sistema como um todo atende aos requisitos.

Operação e Manutenção: É a fase do ciclo de vida de software onde o software é implantado e mantido. Envolve a instalação do software no ambiente do usuário, o treinamento dos usuários, a correção de erros e problemas e a atualização do software para adicionar novas funcionalidades ou para atender a novos requisitos.

Modelo Iterativo e Incremental: É um modelo de ciclo de vida de software onde o software é desenvolvido através de uma série de iterações ou incrementos. Cada iteração inclui todas as fases do ciclo de vida do software, e resulta em uma versão parcialmente completa, mas funcional, do software.

RAD (Rapid Application Development): É uma metodologia de desenvolvimento de software que enfatiza a rapidez e a flexibilidade. O RAD envolve o desenvolvimento iterativo, a prototipagem e a colaboração direta com os usuários finais para definir os requisitos e projetar o sistema.

Prototipagem: É uma técnica de desenvolvimento de software onde um protótipo ou versão preliminar do software é criado para demonstrar as funcionalidades e o design do sistema. O protótipo é então usado para coletar feedback dos usuários e refinar os requisitos e o design do sistema.

Protótipo: É uma versão preliminar ou exemplo de um produto que serve para demonstrar o conceito, o design ou as funcionalidades. Em desenvolvimento de software, um protótipo pode ser uma versão inicial do software que é usada para coletar feedback dos usuários e testar ideias de design.

Protótipo Descartável: É um tipo de protótipo que é criado para explorar ideias ou conceitos e coletar feedback dos usuários, mas que não é destinado a ser usado no produto final.

Protótipo Evolutivo: É um tipo de protótipo que é criado com a intenção de ser refinado e melhorado ao longo do tempo até se tornar o produto final.

Modelo em Espiral: É um modelo de ciclo de vida de software que combina elementos do modelo em cascata e do desenvolvimento iterativo, com uma ênfase na análise de riscos. O desenvolvimento progride em uma série de ciclos ou "espirais", cada um dos quais inclui fases de planejamento, análise de riscos, engenharia e avaliação do cliente.

Análise de Riscos: É o processo de identificar, avaliar e priorizar riscos, a fim de minimizar, monitorar e controlar a probabilidade e/ou o impacto de eventos indesejados. Em engenharia de software, os riscos podem incluir coisas como atrasos no cronograma, custos acima do orçamento, defeitos de software e mudanças nos requisitos.

Modelo Baseado em Componentes: É uma abordagem de desenvolvimento de software que enfatiza a reutilização de componentes de software preexistentes. Cada componente é uma unidade independente de funcionalidade que pode ser combinada com outros componentes para criar um sistema.

Componente de Software: É uma unidade independente de código que fornece uma funcionalidade específica e que pode ser combinada com outros componentes para criar um sistema. Exemplos incluem bibliotecas, módulos, classes e serviços web.

Método Formal: É uma técnica de engenharia de software que utiliza notações e técnicas matemáticas rigorosas para especificar, projetar e verificar sistemas. Os métodos formais podem ser usados para provar a correção de um sistema e para encontrar e corrigir erros.

Modelo Orientado a Aspectos: É uma abordagem de desenvolvimento de software que busca aumentar a modularidade permitindo a separação de preocupações transversais, que são aspectos do sistema que afetam outros aspectos do sistema e que são difíceis de modularizar usando técnicas tradicionais.

Preocupação Principal: São os requisitos e comportamentos centrais de um sistema que refletem os objetivos diretos da aplicação.

Preocupação Transversal: São os aspectos do sistema que afetam outros aspectos do sistema, tornando-os difíceis de modularizar usando técnicas de programação orientada a objetos ou procedural tradicionais. Exemplos comuns de preocupações transversais incluem logging, gerenciamento de transações, segurança e tratamento de erros.

ROTEIRO DE REVISÃO

A ideia desta seção é apresentar um roteiro para que você realize uma revisão completa do assunto e, ao mesmo tempo, destacar aspectos do conteúdo que merecem atenção.

Engenharia de Software

Engenharia de Software é um campo complexo e multidisciplinar, mas podemos definir de maneira geral a partir de duas perspectivas proeminentes - a do Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) e a de Roger Pressman, um conhecido autor e consultor da área.

Definição da IEEE:

A IEEE define a Engenharia de Software como: "*A aplicação de uma abordagem sistemática, disciplinada e quantificável ao desenvolvimento, operação e manutenção do software*." Isso inclui o estudo de abordagens para especificar, projetar, programar e testar um sistema de software.

Definição de Roger Pressman:

Roger S. Pressman, em seu livro "Engenharia de Software: Uma Abordagem Prática", define a Engenharia de Software como: "*Uma disciplina de engenharia que se envolve com todos os aspectos da produção de software, desde as primeiras etapas de especificação de requisitos do sistema até a manutenção do sistema após a entrega*."

Essas definições destacam a natureza sistemática e disciplinada da Engenharia de Software. Ela não é apenas a programação, mas envolve um processo estruturado e metódico para a criação de software, que abrange todo o ciclo de vida do software, desde a concepção até a manutenção. Além disso, a Engenharia de Software preocupa-se com práticas mensuráveis e quantificáveis para garantir a qualidade e a eficácia do software produzido.

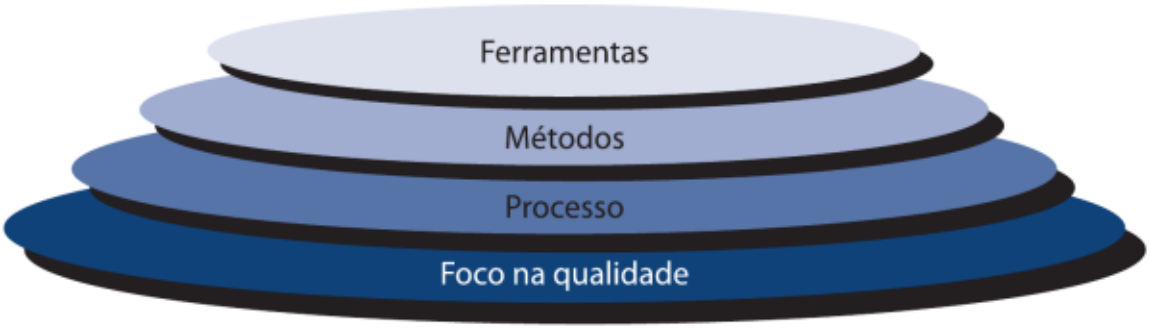
Princípios Fundamentais

A Engenharia de Software possui alguns princípios fundamentais, tais como: **Formalidade**, em que o software deve ser desenvolvido de acordo com passos definidos com precisão e seguidos de maneira efetiva; **Abstração**, em que existe uma preocupação com a identificação de um determinado fenômeno da realidade, sem se preocupar com detalhes, considerando apenas os aspectos mais relevantes.

Há a **Decomposição**, em que se divide o problema em partes, de maneira que cada uma possa ser resolvida de uma forma mais específica; **Generalização**, maneira usada para resolver um problema, de forma genérica, com o intuito de reaproveitar essa solução em outras situações; **Flexibilização** é o processo que permite que o software possa ser alterado, sem causar problemas para sua execução

Camadas

A Engenharia de Software é normalmente visualizada em três camadas distintas, mas inter-relacionadas: **ferramentas, métodos e processos**. Cada uma dessas camadas desempenha um papel essencial na produção eficaz de software de alta qualidade.



Processos:

Processos de Engenharia de Software se referem aos procedimentos e rotinas estruturadas que são seguidos para a produção de software. Eles fornecem uma estrutura para o desenvolvimento de software, incluindo atividades como especificação de requisitos, design, implementação, teste e manutenção. Processos também podem incluir elementos de gestão, como planejamento de projetos, gestão de recursos, e controle de qualidade.

Dentro desta camada, existem diversos modelos de ciclo de vida de software (como o modelo em cascata, espiral, ágil, entre outros), cada um com suas próprias etapas e sequência de atividades.

Métodos:

Métodos fornecem os detalhes técnicos de como o software é realmente desenvolvido dentro do contexto do processo escolhido. Isso inclui práticas para análise de requisitos, design de arquitetura de software, codificação, teste e depuração.

Métodos definem como e quando as tarefas de desenvolvimento específicas são realizadas, e quais técnicas ou práticas são empregadas para executá-las. Eles fornecem uma maneira de organizar o trabalho e criar produtos de software efetivamente.

Ferramentas:

Ferramentas de Engenharia de Software são os componentes tecnológicos usados para automatizar e auxiliar as atividades dos processos e métodos. Elas vão desde ambientes de desenvolvimento integrados (IDEs), sistemas de controle de versão, ferramentas de teste automatizado, até ferramentas de design de software, entre outras.

As ferramentas proporcionam o suporte tecnológico necessário para implementar efetivamente os métodos e processos, ajudando os desenvolvedores a realizar suas tarefas de forma mais eficiente e eficaz.

Cada uma dessas camadas trabalha em conjunto para facilitar o desenvolvimento de software eficiente e de alta **qualidade**. Os processos fornecem a estrutura geral, os métodos preenchem essa estrutura com práticas técnicas e as ferramentas auxiliam na execução dessas práticas.

Ciclo de Vida de Software

O ciclo de vida de software (ou sistemas de informação) é um **modelo que descreve as fases pelas quais um software passa desde sua concepção até a aposentadoria**. As fases típicas incluem, por exemplo, requisitos, design, implementação, testes, implantação e manutenção.

Durante cada fase, **há atividades específicas que devem ser realizadas** para garantir que o software seja desenvolvido de acordo com as especificações e atenda às necessidades do cliente.

Veja uma tabela mais completa que resume as fases típicas do ciclo de vida de software:

FASES	DESCRIÇÃO
PLANEJAMENTO	O objetivo do planejamento de projeto é fornecer uma estrutura que possibilite ao gerente fazer estimativas razoáveis de recursos, custos e prazos. Uma vez estabelecido o escopo de software, com um pequeno esboço dos requisitos, uma proposta de desenvolvimento deve ser elaborada, isto é, um plano de projeto deve ser elaborado configurando o processo a ser utilizado no desenvolvimento de software. À medida que o projeto progride, o planejamento deve ser detalhado e atualizado regularmente. Pelo menos ao final de cada uma das fases do desenvolvimento (análise e especificação de requisitos, projeto, implementação e testes), o planejamento como um todo deve ser revisto e o planejamento da etapa seguinte deve ser detalhado. O planejamento e o acompanhamento do progresso fazem parte do processo de gerência de projeto.
ANÁLISE E ESPECIFICAÇÃO DE REQUISITOS	Nesta fase, o processo de levantamento de requisitos é intensificado. O escopo deve ser refinado e os requisitos mais bem definidos. Para entender a natureza do software a ser construído, o engenheiro de software tem de compreender o domínio do problema, as restrições, as metas, as funcionalidades e o comportamento esperados – ele pode o fazer por meio de diversas técnicas de levantamento de requisitos (Ex: entrevistas). Uma vez capturados os requisitos do sistema a ser desenvolvido, estes devem ser modelados, avaliados e documentados. Uma parte vital desta fase é a construção de um modelo descrevendo o que o software tem de fazer (e não como fazê-lo).
PROJETO	Esta fase é responsável por incorporar requisitos tecnológicos aos requisitos essenciais do sistema, modelados na fase anterior e, portanto, requer que a plataforma de implementação seja conhecida. Basicamente, envolve duas grandes etapas: projeto da arquitetura do sistema e projeto detalhado. O objetivo da primeira etapa é definir a arquitetura geral do software, tendo por base o modelo construído na fase de análise de requisitos. Essa arquitetura deve descrever a estrutura de nível mais alto da aplicação e identificar seus principais componentes. O propósito do projeto detalhado é detalhar o projeto do software para cada componente identificado na etapa anterior. Os componentes de software devem ser sucessivamente refinados em níveis maiores de detalhamento (inclusive em relação à tecnologia adotada) até que possam ser codificados e testados.
IMPLEMENTAÇÃO	O projeto deve ser traduzido para uma forma passível de execução pela máquina. A fase de implementação realiza esta tarefa, isto é, cada unidade de software do projeto detalhado é implementada.
TESTES	Inclui diversos níveis de testes, a saber, teste de unidade, teste de integração e teste de sistema. Inicialmente, cada unidade de software implementada deve ser testada e os resultados documentados. A seguir, os diversos componentes devem ser integrados sucessivamente até se obter o sistema. Finalmente, o sistema como um todo deve ser testado.
ENTREGA E IMPLANTAÇÃO	Uma vez testado, o software deve ser colocado em produção. Para tal, contudo, é necessário treinar os usuários, configurar o ambiente de produção e, muitas vezes, converter bases de dados. O propósito desta fase é estabelecer que o software satisfaz os requisitos dos usuários. Isto é feito instalando o software e conduzindo testes de aceitação. Quando o

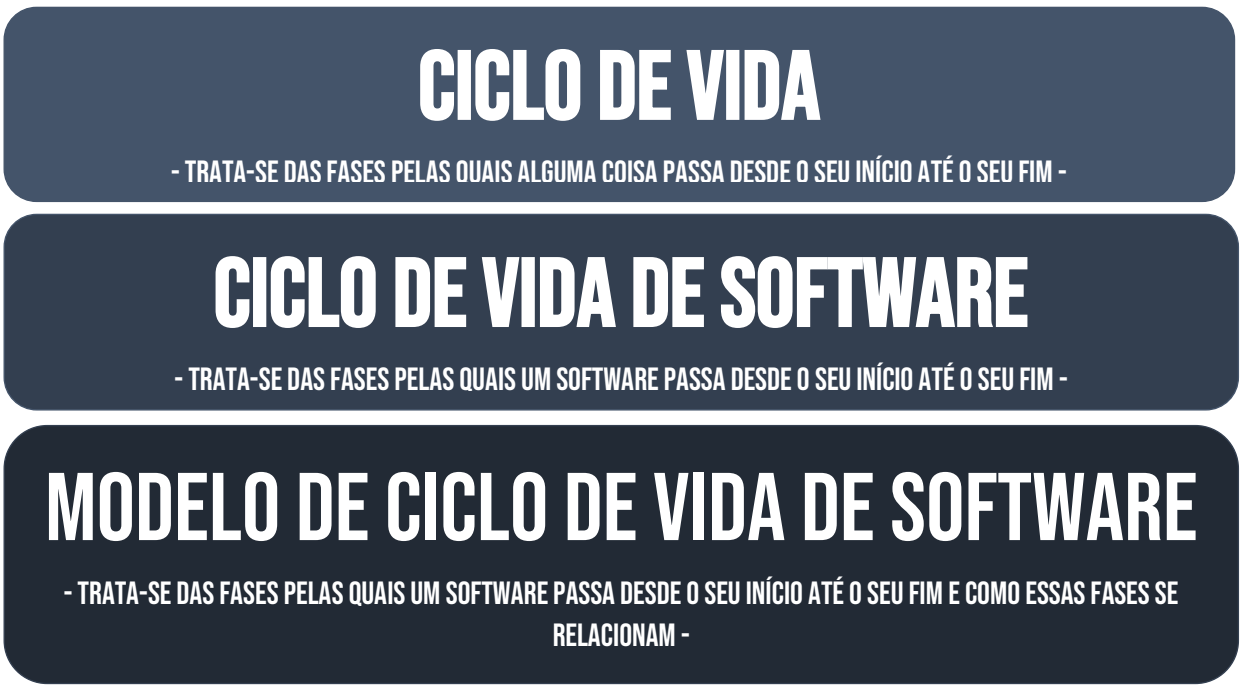
	software tiver demonstrado prover as capacidades requeridas, ele pode ser aceito e a operação iniciada.
OPERAÇÃO	Nesta fase, o software é utilizado pelos usuários no ambiente de produção, isto é, no ambiente real de uso do usuário.
MANUTENÇÃO	Indubitavelmente, o software sofrerá mudanças após ter sido entregue para o usuário. Alterações ocorrerão porque erros foram encontrados, porque o software precisa ser adaptado para acomodar mudanças em seu ambiente externo, ou porque o cliente necessita de funcionalidade adicional ou aumento de desempenho. Muitas vezes, dependendo do tipo e porte da manutenção necessária, essa fase pode requerer a definição de um novo processo, onde cada uma das fases precedentes é reaplicada no contexto de um software existente ao invés de um novo.

Não há um "padrão" para as fases do ciclo de vide de software, cada autor costuma ter a sua própria representação delas. As figuras a seguir resumem as fases de ciclo de vida citadas por Ian Sommerville e Roger Pressman, os dois principais autores da Engenharia de Software:

Modelo de Ciclo de Vida de Software

Modelos de ciclo de vida, também conhecidos como processos de desenvolvimento de software, são representações conceituais ou frameworks que descrevem as fases envolvidas no desenvolvimento de um software, desde a concepção inicial até a fase de manutenção. Eles fornecem uma estrutura sequencial para o processo de desenvolvimento de software e ajudam a estruturar, planejar e controlar o processo de transformar os requisitos do usuário em software operacional.

Perceba que um **modelo de ciclo de vida de software apresenta não só as fases do ciclo de vida do software, mas também a forma como essas fases se relacionam**.



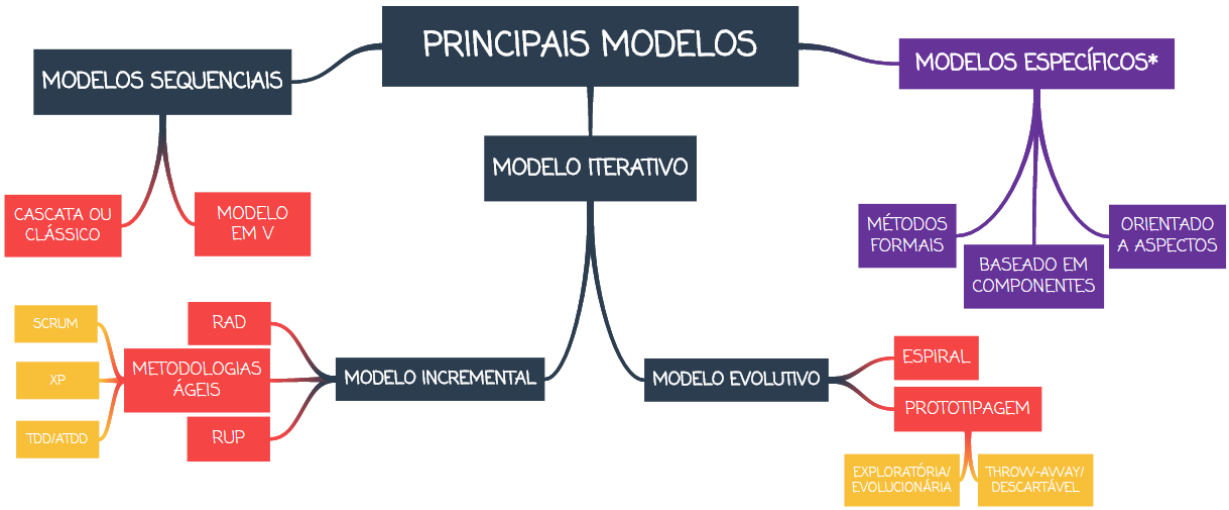
Alguns dos modelos mais comumente utilizados que estudaremos durante a nossa aula são, dentre outros:

- **Modelo em Cascata**: Este é um dos primeiros modelos de ciclo de vida de software. É uma abordagem sequencial que começa no nível de sistema e progressivamente detalha para a análise, design, codificação, teste e manutenção.
- **Modelo Iterativo e Incremental**: Foca na entrega e aprimoramento de pequenas parcelas de funcionalidade por meio de ciclos iterativos. Cada iteração passa por todas as fases do ciclo de vida.

- **Modelo Espiral**: Combina a natureza iterativa do desenvolvimento de software com a sistemática do modelo em cascata. Ele adiciona um elemento de gerenciamento de risco que não está presente no modelo em cascata.
- **Modelo RAD (Desenvolvimento Rápido de Aplicações)**: Prioriza o desenvolvimento rápido e a entrega contínua de protótipos de alta qualidade.

Cada modelo tem suas próprias vantagens e desvantagens e é adequado para diferentes tipos de projetos. A escolha do modelo de ciclo de vida depende de vários de fatores, como a natureza do projeto, os objetivos, os recursos disponíveis e a cultura da equipe ou organização.

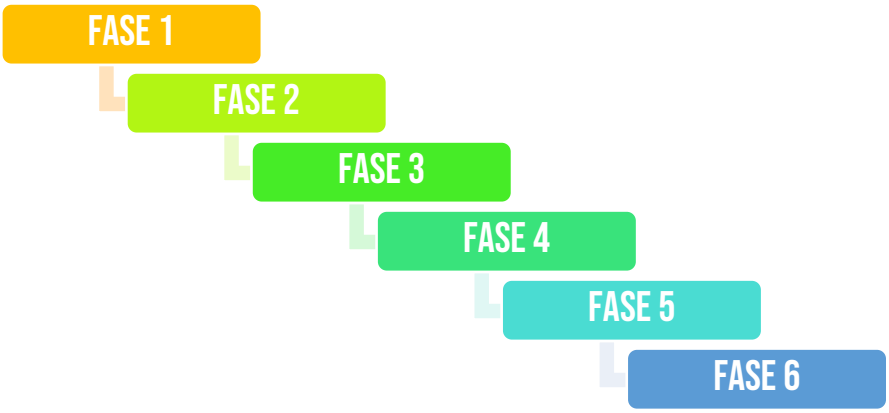
A imagem a seguir resume os principais modelos de ciclo de vida e processos de desenvolvimento utilizados na Engenharia de Software:



Modelo em Cascata

O modelo em cascata é uma metodologia **linear e sequencial** que prevê fases distintas de especificação e desenvolvimento de software. **Cada fase deve ser completada antes que a próxima fase possa começar, sem qualquer sobreposição entre elas**.

Cascata é um dos primeiros modelos de processo de desenvolvimento de software e ainda é amplamente utilizado em projetos onde os requisitos são claramente entendidos e são menos propensos a mudar ao longo do tempo. Ele recebe este nome por causa da maneira como as atividades de desenvolvimento de software fluem para baixo, como uma cascata, através das fases típicas de concepção, iniciação, análise, design, codificação, teste e manutenção.



O Modelo em Cascata foi o primeiro modelo de processo publicado, apresentado por Winston W. Royce em 1970. Embora Royce não tenha usado o termo "cascata" em seu artigo, ele apresentou o que agora conhecemos como Modelo em Cascata.

O artigo original de Royce na verdade apresentava o modelo em cascata como um exemplo de um processo que não funcionaria a menos que fossem incluídas várias salvaguardas, incluindo a revisão rigorosa do projeto e a implementação de protótipos. Porém, muitas organizações adotaram a visão mais simplista e linear do processo em cascata sem essas salvaguardas.

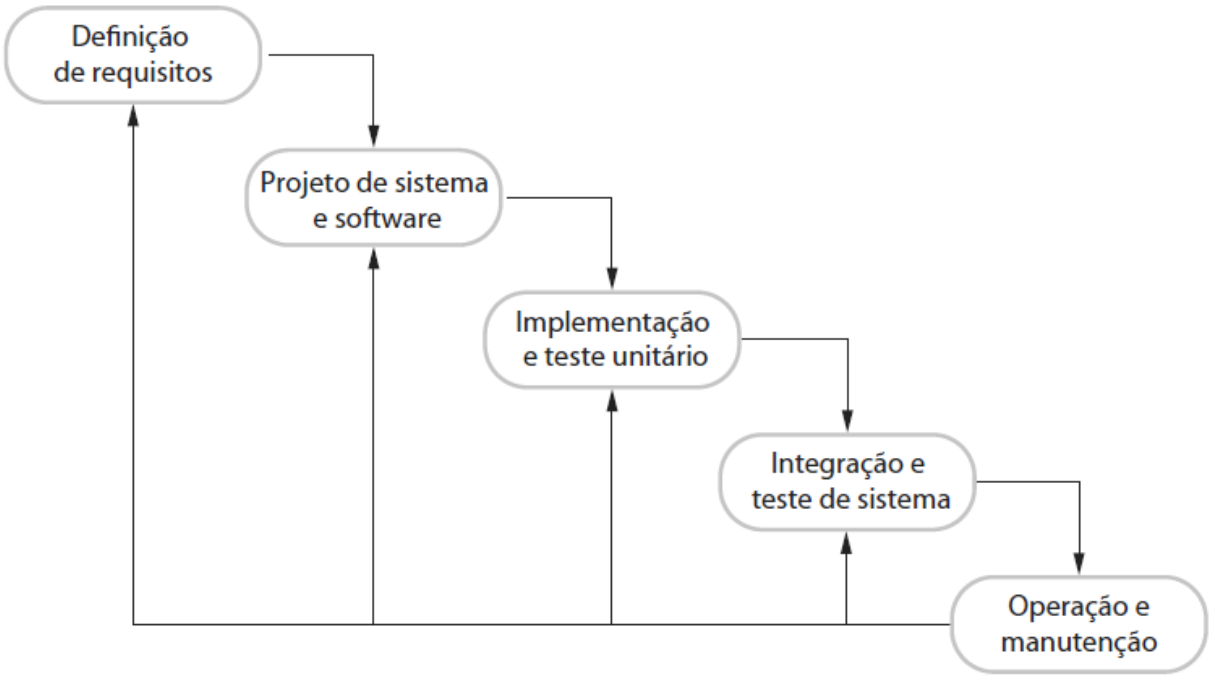
Ao longo dos anos, o Modelo em Cascata tem sido amplamente criticado por sua falta de flexibilidade em relação às mudanças nos requisitos, mas ainda é popular em muitos contextos em que os requisitos são bem compreendidos e não mudam muito ao longo do tempo, como no desenvolvimento de sistemas críticos em setores como aeroespacial e defesa.

É importante notar que não há um "padrão" para as fases do modelo em cascata. Cada autor costuma representá-las de maneiras diferentes. Veja as principais representações:

POR SOMMERVILLE	POR ROYCE	POR PRESSMAN (6ª ED)
Análise e Definição de Requisitos	Requisitos de Sistema	Comunicação
Projeto de Sistema e Software	Requisitos de Software	Planejamento
Implementação e Teste de Unidade	Análise	Modelagem
Integração e Teste de Sistema	Projeto	Construção
Operação e Manutenção	Codificação	Implantação
	Teste	
	Operação	

Por exemplo, veja as descrições das fases de acordo com o Sommerville (autor muito cobrado em prova):

FASES (IAN SOMMERVILLE)	DESCRIÇÃO
Análise e Definição de requisitos	Os serviços, restrições e metas do sistema são estabelecidos por meio de consulta aos usuários. Em seguida, são definidos em detalhes e funcionam como uma especificação do sistema.
Projeto de sistema e software	O processo de projeto de sistemas aloca os requisitos tanto para sistemas de hardware como para sistemas de software, por meio da definição de uma arquitetura geral do sistema. O projeto de software envolve identificação e descrição das abstrações fundamentais do sistema de software e seus relacionamentos.
Implementação e teste unitário	Durante esse estágio, o projeto do software é desenvolvido como um conjunto de programas ou unidades de programa. O teste unitário envolve a verificação de que cada unidade atenda a sua especificação.
Integração e teste de sistema	As unidades individuais do programa ou programas são integradas e testadas como um sistema completo para assegurar que os requisitos do software tenham sido atendidos. Após o teste, o sistema de software é entregue ao cliente.
Operação e manutenção	Normalmente (embora não necessariamente), essa é a fase mais longa do ciclo de vida. O sistema é instalado e colocado em uso. A manutenção envolve a correção de erros que não foram descobertos em estágios iniciais do ciclo de vida, com melhora da implementação das unidades do sistema e ampliação de seus serviços em resposta às descobertas de novos requisitos.



Vantagens e Desvantagens

Como qualquer método, o Modelo em Cascata tem suas vantagens e desvantagens, e é mais adequado para certos tipos de projetos do que outros.

Vantagens:

- **Facilidade de Entendimento e Uso**: Como um modelo linear, o modelo em cascata é bastante simples de entender e usar. Cada fase tem um resultado e um processo de revisão definidos, tornando o progresso fácil de acompanhar e entender.
- **Estrutura Rígida**: O modelo é útil em situações em que os requisitos são bem compreendidos e não mudam muito. Isso permite que o projeto prossiga de uma maneira muito sistemática e disciplinada.
- **Documentação**: Como cada fase é projetada para ser concluída antes do início da próxima, a documentação é produzida a cada fase, facilitando a compreensão do produto em todos os níveis.
- **Especialização de Funções**: O modelo permite a especialização de funções, já que cada fase do projeto requer habilidades específicas. Isso possibilita que os membros da equipe se concentrem nas áreas em que são especialistas.

Desvantagens:

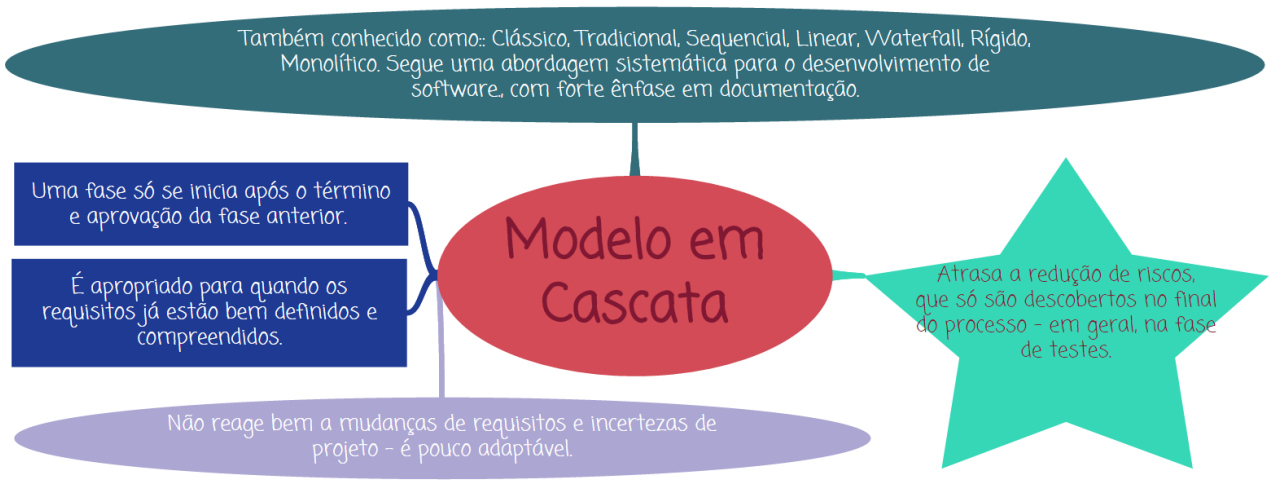
- **Inflexibilidade**: A maior desvantagem do modelo em cascata é sua rigidez. Uma vez que uma fase é concluída, é difícil ou impossível voltar atrás e fazer alterações. Isso pode ser problemático se os requisitos mudarem durante o curso do projeto.
- **Descoberta Tardia de Problemas**: Como o teste é uma fase tardia do processo, os problemas só são descobertos após a fase de implementação. Isso pode levar a atrasos significativos no projeto e, em alguns casos, à necessidade de reiniciar todo o projeto.

O MODELO EM CASCATA ATRASA A REDUÇÃO DE RISCOS

- **Alto Risco e Incerteza**: O modelo pode ser arriscado para projetos grandes e complexos onde os requisitos não são bem compreendidos desde o início. Além disso, ele pode ser inadequado para projetos de longo prazo, onde a tecnologia ou as demandas do usuário podem mudar.
- **Pouco Cliente Envolvimento**: No modelo em cascata, o cliente geralmente só é envolvido no início (durante a coleta de requisitos) e no final (durante a implantação). Isso significa que pode haver surpresas desagradáveis no final do projeto, quando é difícil e caro fazer alterações.

Ou seja, enquanto o modelo em cascata pode funcionar bem para projetos pequenos e de requisitos bem definidos, ele tem limitações significativas que podem torná-lo inadequado para projetos maiores, mais complexos ou mais incertos.

A figura a seguir ilustra e resume o modelo em cascata:



Modelo Iterativo e Incremental

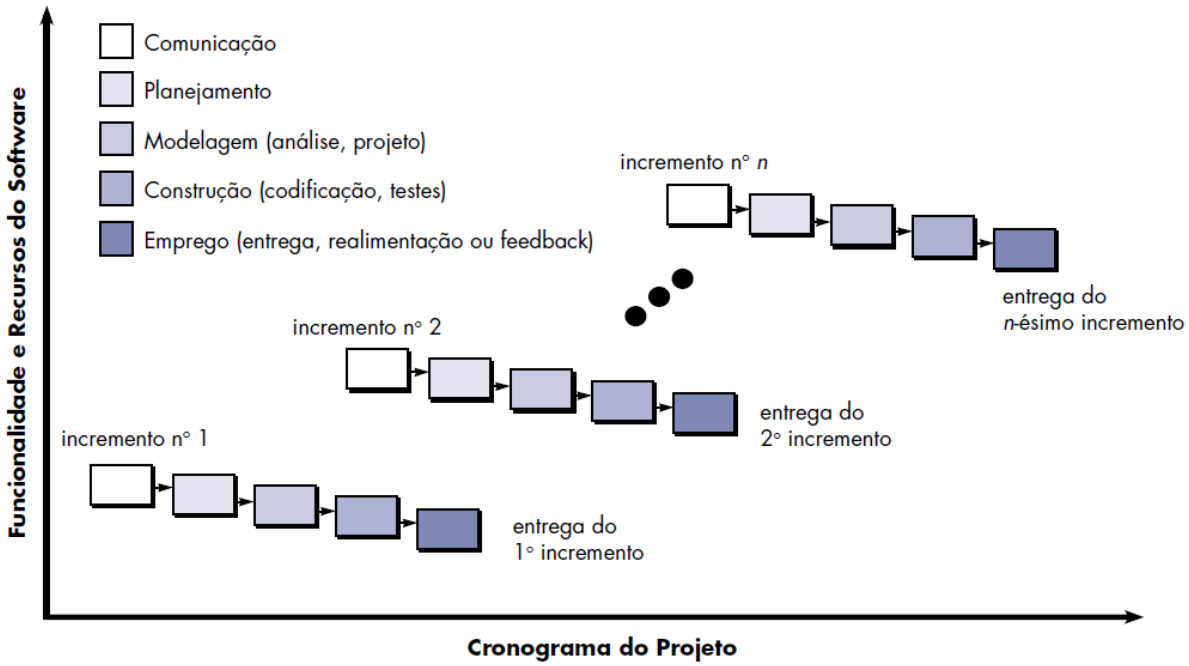
O Modelo Iterativo e Incremental é uma abordagem de desenvolvimento de software que enfatiza a entrega e aprimoramento repetidos de **pequenos incrementos de funcionalidade**. Em vez de tentar entregar todo o sistema de uma só vez, como acontece no modelo em cascata, o desenvolvimento iterativo e incremental procura construir e entregar pequenas porções úteis de funcionalidade em cada iteração.

De maneira geral, funciona da seguinte forma:

- **Planejamento**: Antes de iniciar o desenvolvimento, a equipe planeja o escopo geral do software, identifica os requisitos gerais e esboça uma estratégia de alto nível para o desenvolvimento.
- **Divisão em Incrementos**: O projeto é então dividido em pequenos incrementos, cada um dos quais adiciona alguma funcionalidade ao software. Cada incremento é suficientemente pequeno para ser compreendido e desenvolvido em uma única iteração.
- **Desenvolvimento Iterativo**: Cada iteração pode passar por todas as fases do ciclo de vida de desenvolvimento de software - análise de requisitos, design, implementação, teste, etc. Essa "passada" escolhe as atividades que são necessárias. Em cada iteração, o software é aprimorado através do acréscimo de novas funcionalidades.
- **Avaliação e Feedback**: Após cada iteração, o software é avaliado. Os stakeholders podem fornecer feedback que pode ser usado para informar o desenvolvimento das próximas iterações. Isso permite que a equipe ajuste o software em resposta às mudanças nos requisitos ou no entendimento do problema.

- **Repetição**: O processo continua, com cada iteração construindo e melhorando o software, até que o produto final esteja completo.

Veja uma representação visual do modelo iterativo e incremental:



A principal vantagem desta abordagem é que ela permite que os desenvolvedores **respondam a mudanças nos requisitos** do software à medida que o entendimento do problema se desenvolve. Além disso, ao entregar incrementos úteis do software regularmente, os stakeholders podem começar a obter valor do projeto mais cedo.

É importante notar que o desenvolvimento iterativo e incremental requer uma gestão de projetos cuidadosa para garantir que o projeto permaneça no caminho certo, e também pode exigir mais recursos de testes, pois cada incremento de software precisa ser testado individualmente e como parte do todo.

A tabela a seguir faz uma comparação entre o desenvolvimento iterativo e incremental versus o modelo em cascata:

Fatores	Modelo em Cascata	Modelo Iterativo e Incremental
Sequência de Desenvolvimento	Linear e sequencial. As fases não se repetem ou se sobrepõem.	Iterativa. Cada incremento passa por todas as fases do ciclo de vida.

Flexibilidade para Mudanças	Baixa. Mudanças nos requisitos podem ser caras e difíceis de implementar.	Alta. Facilita a incorporação de mudanças e ajustes ao longo do processo de desenvolvimento.
Identificação de Problemas	Problemas geralmente só são identificados na fase de teste, o que pode levar a grandes modificações.	Problemas são identificados mais cedo, já que cada iteração passa por um ciclo de teste.
Risco	Alto. Uma vez que a fase é concluída, voltar atrás é difícil e caro.	Baixo a médio. Cada iteração é relativamente pequena, então os problemas podem ser corrigidos antes que se tornem grandes.
Entrega de Valor	O valor é entregue apenas no final do projeto, quando o produto final é entregue.	O valor é entregue incrementalmente. O cliente pode ver e usar partes do sistema a cada iteração.
Envolvimento do Cliente	O cliente é envolvido principalmente no início (coleta de requisitos) e no final (validação).	O cliente está envolvido ao longo de todo o processo, fornecendo feedback a cada iteração.
Gestão do Projeto	Relativamente simples, pois as fases são sequenciais e não se sobrepõem.	Pode ser mais complexa, pois requer planejamento e execução contínuos para cada iteração.

Modelo RAD

O modelo RAD, ou Desenvolvimento Rápido de Aplicações, é uma **metodologia de desenvolvimento de software que enfatiza a rapidez e a eficiência**. Ele foi desenvolvido em resposta à necessidade de desenvolver software de alta qualidade em um tempo de

desenvolvimento muito menor do que o tradicionalmente permitido pelos modelos clássicos de ciclo de vida.

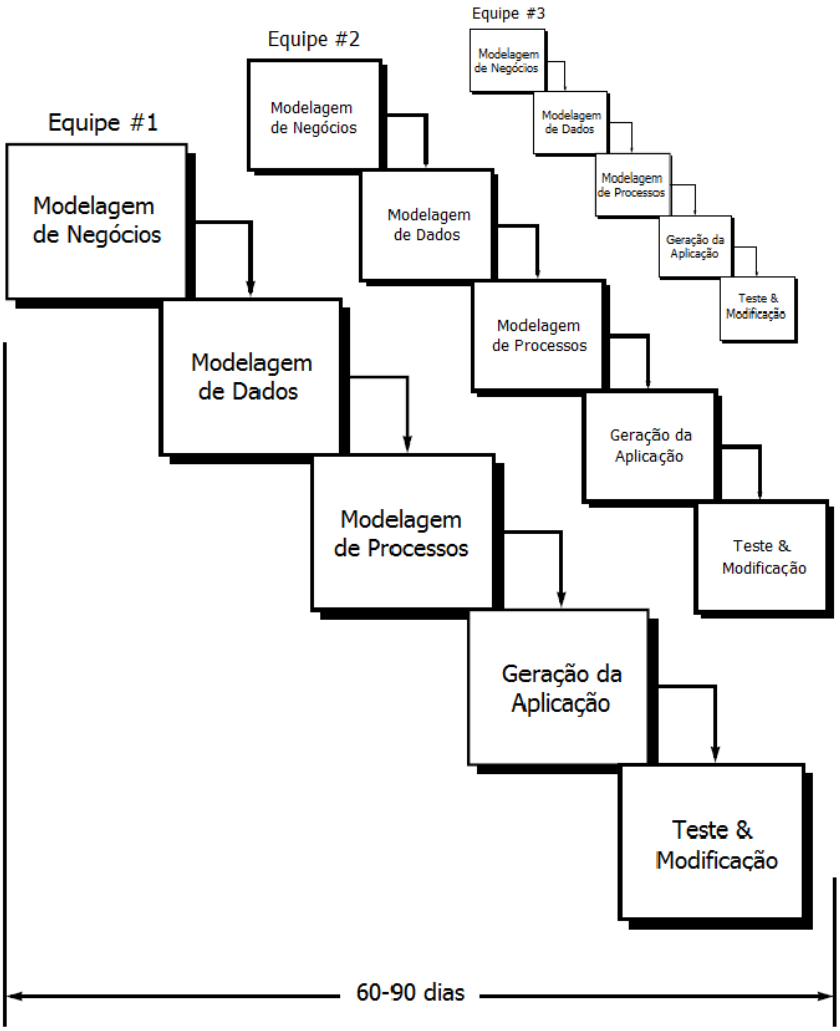
O modelo é baseado nos seguintes princípios fundamentais:

- **Envolvimento do Usuário**: O envolvimento ativo e contínuo do usuário é crucial em todas as fases do desenvolvimento. Isso ajuda a garantir que o produto final atenda às necessidades e expectativas do usuário.
- **Prototipagem**: O RAD utiliza a prototipagem extensiva e a interação direta com os usuários para obter um entendimento claro dos requisitos do sistema, em vez de tentar definir todos os requisitos detalhadamente no início do projeto.
- **Desenvolvimento Iterativo**: Em vez de um desenvolvimento sequencial de um grande sistema, o RAD enfatiza o desenvolvimento iterativo de componentes menores e independentes.
- **Reutilização de Componentes**: RAD promove a reutilização de componentes de software sempre que possível, para acelerar o processo de desenvolvimento.

Veja quais são as fases do modelo RAD:

FASES	DESCRIÇÃO
Modelagem de negócio	O fluxo de informações entre as funções de negócio é modelado de modo a responder que informação direciona o processo de negócio; que informação é gerada; quem gera essa informação; para onde vai a informação gerada; e, por fim, quem processa a informação.
Modelagem de dados	O fluxo de informação definido na fase de modelagem de negócio é refinado em um conjunto de objetos de dados que são necessários para suportar o negócio. Os atributos de cada objeto são identificados e os relacionamentos entre esses objetos são definidos.
Modelagem de processo	Os objetos de dados definidos na modelagem de dados são transformados para conseguir o fluxo necessário para implementar uma função do negócio. Descrições do processamento são criadas para adicionar, modificar, descartar ou recuperar um objeto de dados.
Geração da aplicação	Considera o uso de técnicas de quarta geração, trabalha com a reutilização de componentes de programa existentes quando possível, ou cria componentes reusáveis. São usadas ferramentas automatizadas para facilitar a construção do software.
Teste e modificação	Como o processo enfatiza o reúso, muitos componentes já estão testados e isso reduz o tempo total de teste. No entanto, os novos componentes devem ser testados e todas as interfaces devem ser exaustivamente exercitadas para colocar o resultado em produção.

O modelo RAD oferece vantagens significativas, incluindo a entrega mais rápida (utiliza ciclos de 60 a 90 dias geralmente) de software de alta qualidade e maior satisfação do usuário devido ao seu envolvimento ativo. No entanto, também requer equipes altamente qualificadas e dedicadas e não é adequado para todos os tipos de projetos, especialmente aqueles que são grandes, complexos ou mal definidos.



A tabela a seguir resume as vantagens e desvantagens do modelo RAD:

VANTAGENS	DESVANTAGENS
Permite o desenvolvimento rápido e/ou a prototipagem de aplicações.	Exige recursos humanos caros e experientes.
Criação e reutilização de componentes.	O envolvimento com o usuário tem que ser ativo.
Desenvolvimento é conduzido em um nível mais alto de abstração.	Comprometimento da equipe do projeto.

Grande redução de codificação manual com *wizards*.	Custo alto do conjunto de ferramentas e hardware para rodar a aplicação;
Cada função pode ser direcionada para a uma equipe separada.	Mais difícil de acompanhar o projeto.
Maior flexibilidade (desenvolvedores podem reprojetar à vontade).	Perda de precisão científica (pela falta de métodos formais).
Provável custo reduzido (tempo é dinheiro e também devido ao reuso).	Pode levar ao retorno das práticas caóticas no desenvolvimento.
Tempo de desenvolvimento curto.	Pode construir funções desnecessárias.
Protótipos permitem uma visualização mais cedo.	Requisitos podem não se encaixar (conflitos entre desenvolvedores e clientes).
Envolvimento maior do usuário.	Padronização (aparência diferente entre os módulos e componentes)

Prototipagem

O Modelo de Prototipagem é uma abordagem de desenvolvimento de software que se concentra na criação de um protótipo funcional inicial que pode ser refinado com base no feedback dos usuários. Este modelo é útil para projetos onde os requisitos são difíceis de definir claramente no início, ou onde uma nova tecnologia ou abordagem está sendo usada.

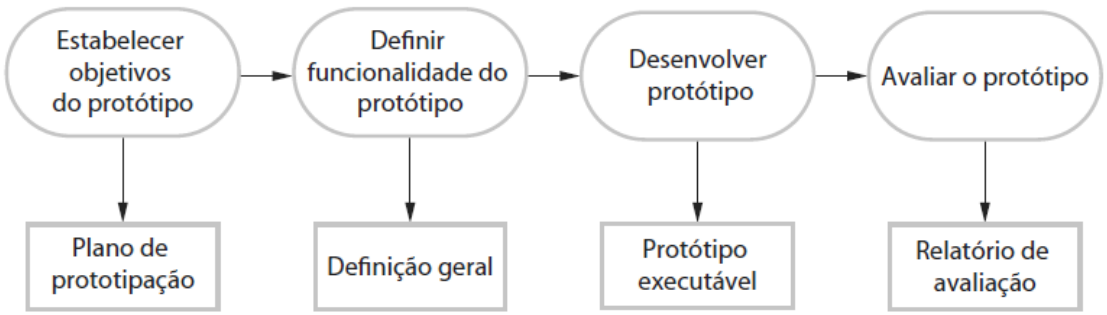
Um protótipo é uma **versão inicial de um sistema de software utilizado para demonstrar conceitos, experimentar opções de projeto e, geralmente, conhecer mais sobre o problema e suas possíveis soluções**. O desenvolvimento rápido e iterativo do protótipo é essencial para que os custos sejam controlados e os stakeholders do sistema possam experimentá-lo no início do processo de software.

Em geral o processo de prototipagem funciona de acordo com as seguintes etapas:

- **Coleta de Requisitos Iniciais**: A equipe de desenvolvimento e os stakeholders se reúnem para discutir requisitos gerais. Em vez de tentar definir todos os detalhes, o foco está em entender os conceitos gerais do que o software deve fazer.
- **Desenvolvimento de Protótipo Inicial**: A equipe então cria um protótipo inicial. Este protótipo pode não ter toda a funcionalidade do sistema final e pode usar atalhos ou soluções temporárias para demonstrar rapidamente como o sistema final pode funcionar.
- **Revisão do Protótipo**: Os stakeholders então revisam o protótipo e fornecem feedback. Eles podem experimentar o protótipo e sugerir alterações ou melhorias.

- **Refinamento do Protótipo**: Com base no feedback dos stakeholders, a equipe de desenvolvimento refina o protótipo, fazendo as alterações sugeridas e melhorando a funcionalidade e qualidade do sistema.
- **Iteração**: Os passos de revisão e refinamento são repetidos quantas vezes forem necessárias até que os stakeholders estejam satisfeitos com o produto.
- **Desenvolvimento do Sistema Final**: Uma vez que o protótipo esteja satisfatório, a equipe de desenvolvimento começa a construir o sistema final. Eles podem usar o protótipo como uma espécie de "*blueprint*" (modelo) para orientar o desenvolvimento.

A figura abaixo resume essas etapas de forma visual:



A principal vantagem do Modelo de Prototipagem é que ele **permite que os usuários e desenvolvedores explorarem ideias antes de se comprometerem com elas**. Isso pode resultar em um sistema final que está mais alinhado com as necessidades dos usuários. No entanto, também pode ser um processo mais demorado e custoso, especialmente se muitas iterações forem necessárias. Além disso, **existe o risco de que os stakeholders se concentrem muito nos detalhes do protótipo e percam de vista os objetivos gerais do sistema**.

A tabela a seguir resume vantagens e desvantagens da Prototipagem:

Vantagens	Desvantagens
Comunicação Melhorada: A prototipagem pode ajudar a melhorar a comunicação e a compreensão entre a equipe de desenvolvimento e os stakeholders, permitindo que eles vejam e interajam com uma versão funcional do sistema.	**Expectativas Infladas**: Os stakeholders podem ter expectativas irreais sobre o produto final baseadas no protótipo, especialmente se o protótipo for mais uma demonstração de conceito do que uma representação precisa do produto final.

Identificação Precoce de Problemas: Ao criar um protótipo, os problemas de design e usabilidade podem ser identificados e corrigidos antes que o desenvolvimento completo do sistema comece.	**Custo e Tempo**: A prototipagem pode ser demorada e cara, especialmente se muitas iterações forem necessárias. Além disso, o tempo gasto na prototipagem pode prolongar o cronograma geral do projeto.
Redução de Riscos: A prototipagem pode reduzir os riscos associados ao desenvolvimento de um novo sistema, permitindo que a equipe de desenvolvimento e os stakeholders explorem ideias e conceitos antes de se comprometerem com eles.	**Descarte de Trabalho**: No caso de protótipos descartáveis, pode haver uma sensação de trabalho desperdiçado quando o protótipo é eventualmente descartado.
Feedback do Usuário: Os usuários podem fornecer feedback direto sobre o protótipo, o que pode resultar em um produto final que está mais alinhado com as necessidades e expectativas dos usuários.	**Foco no Protótipo**: Os stakeholders podem se concentrar demais nos detalhes do protótipo e perder de vista os objetivos gerais do sistema.

Protótipo Descartável versus Protótipo Evolutivo

Existem dois tipos principais de protótipos em desenvolvimento de software: **descartáveis e evolutivos**. Ambos têm suas vantagens e desvantagens, e a escolha entre eles depende das necessidades específicas do projeto.

Protótipos Descartáveis:

Este tipo de protótipo é usado para explorar ideias, conceitos ou projetos que são incertos ou pouco compreendidos. Um protótipo descartável é desenvolvido rapidamente para permitir que os usuários interajam com ele e forneçam feedback. Ele pode não ser funcionalmente completo ou tecnicamente sólido, e pode usar soluções temporárias ou "atalhos" para ilustrar certos aspectos do sistema. Uma vez que seu propósito seja atendido (por exemplo, coletar feedback dos usuários, validar um conceito, etc.), o protótipo é descartado. O desenvolvimento do sistema real começa do zero, mas é informado pelo que foi aprendido com o protótipo descartável.

É importante notar que a Prototipagem Descartável também é considerada uma técnica de elicitação (levantamento) de requisitos.

Vantagens dos protótipos descartáveis incluem a rapidez no desenvolvimento e a flexibilidade para experimentar ideias. A desvantagem é que eles não contribuem diretamente para o produto final, o que pode levar a um esforço "desperdiçado".

Protótipos Evolutivos:

Ao contrário dos protótipos descartáveis, os protótipos evolutivos são destinados a se tornarem o produto final. Eles começam como uma versão simples do sistema, que é então refinada e expandida ao longo do tempo, até que se torne o sistema completo. Cada iteração do protótipo é funcional e pode ser entregue aos usuários para feedback e uso.

A principal vantagem dos protótipos evolutivos é que todo o trabalho contribui diretamente para o produto final, sem esforço "desperdiçado". No entanto, eles exigem um maior cuidado no desenvolvimento, pois cada iteração deve ser robusta e de alta qualidade o suficiente para ser potencialmente usada em um ambiente de produção. Isso pode tornar o processo de prototipagem evolutiva mais lento e mais caro do que a prototipagem descartável.

A tabela a seguir resume as características de ambas abordagens:

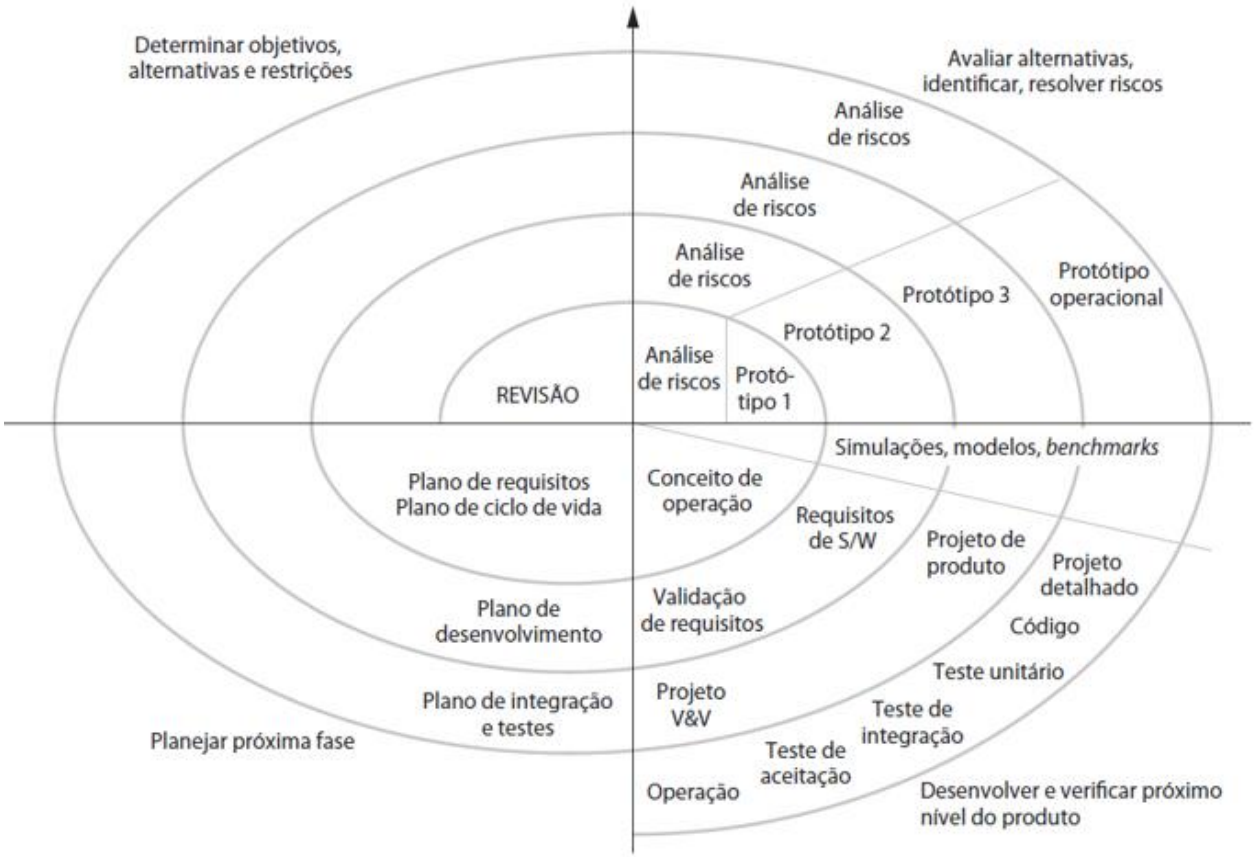
DESENVOLVIMENTO EXPLORATÓRIO OU EVOLUCIONÁRIO	O objetivo do processo de desenvolvimento exploratório é trabalhar com o cliente para explorar os requisitos e entregar um sistema final. O desenvolvimento começa com as partes do sistema compreendidas. O sistema evolui por meio da adição de novas características propostas pelo cliente.
PROTOTIPAÇÃO THROWAWAY OU DESTARTÁVEL	O objetivo do processo de prototipação throwaway é compreender os requisitos do cliente e, a partir disso, desenvolver melhor definição de requisitos para o sistema. O protótipo se concentra na experimentação dos requisitos mal compreendidos do cliente, mas é posteriormente descartado.

Modelo em Espiral

O Modelo em Espiral é uma metodologia de desenvolvimento de software que combina elementos do Modelo em Cascata (por sua abordagem sistemática e sequencial) e do Modelo de Prototipagem (pelo uso de protótipos para refinar requisitos e design). Este modelo foi introduzido por Barry Boehm em 1986 e é especialmente útil para projetos grandes, complexos e de alto risco.

O Modelo é visualizado como uma espiral com muitos loops ou iterações, cada um representando uma fase do processo de desenvolvimento. Cada loop da espiral é dividido em quatro quadrantes ou atividades.

A principal característica do Modelo em Espiral é a **análise de riscos**, que avalia a viabilidade de continuidade do projeto com base nos riscos e alternativas disponíveis.

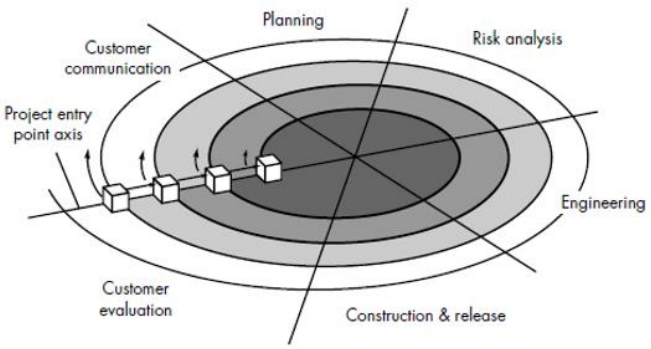


As atividades (setores) do Modelo em Espiral são:

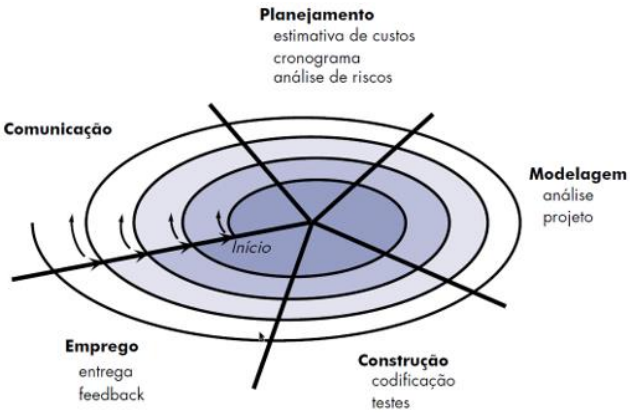
1. **Determinação de Objetivos**: Define-se os objetivos específicos para essa fase do projeto, como a identificação de requisitos ou a criação de um protótipo.
2. **Análise de Riscos**: Identificam-se potenciais problemas ou riscos que podem afetar a conclusão bem-sucedida dessa fase. Esses riscos são então analisados e estratégias são desenvolvidas para lidar com eles.
3. **Desenvolvimento e Testes**: Dependendo do objetivo da iteração, essa fase pode envolver a coleta e validação de requisitos, o design do sistema, a codificação, a realização de testes ou a integração de componentes.

4. **Planejamento das próximas iterações**: O projeto é revisto e o plano para a próxima iteração é desenvolvido. Isso pode envolver a decisão de continuar para a próxima iteração, a implementação do sistema completo ou o término do projeto se ele não for viável.

Com o passar do tempo, outros autores propuseram diferentes representações para o Modelo em Espiral. Veja as representações do Pressman:



PRESSMAN - 5ª EDIÇÃO



PRESSMAN – 7ª EDIÇÃO

De acordo com Pressman, cada espiral é dividida em cinco setores:

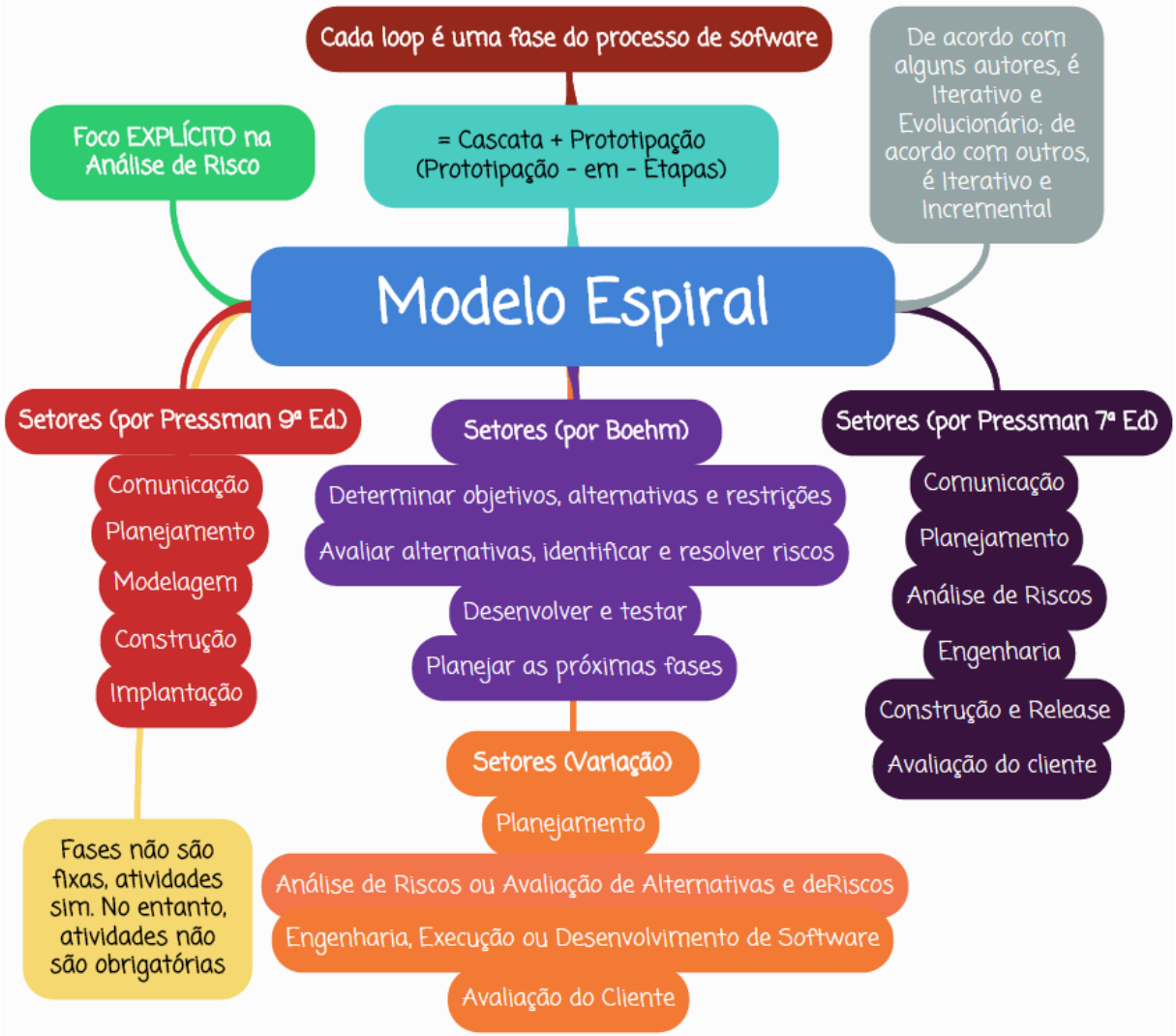
Setores (por Pressman)	DESCRIÇÃO
Comunicação	É a comunicação em si
Planejamento	Estimativa de custos, cronograma e **análise de riscos**
Modelagem	Análise e design
Construção	Codificação e teste
implantação	Entrega e feedback

A tabela a seguir resume as maiores vantagens e desvantagens do Espiral:

VANTAGENS	DESVANTAGENS
Suporta mecanismos de redução de riscos.	Exige analistas de risco bastante experientes.
Obtêm-se versões do sistema a cada iteração.	Exige uma equipe de desenvolvimento extremamente qualificada.
Entregando produtos cada vez mais refinados e de melhor qualidade.	Exige um gerenciamento de processo mais complexo.
Reflete as práticas reais de engenharia atual.	Não é recomendado resolver problemas mais simples e pequenos.
Apresenta uma abordagem sistemática.	

Apresenta estimativas realistas.	

A figura a seguir resume o modelo de forma visual:



Modelo baseado em Componentes

O Modelo Baseado em Componentes é uma abordagem de desenvolvimento de software que se concentra no **reuso de componentes de software existentes para construir novos sistemas**. Esses componentes são peças de software "pré-feitas" e modulares que podem ser usadas para realizar tarefas comuns e podem ser combinadas de várias maneiras para criar diferentes aplicações de software.

Segundo o Pressman, Componente é um **bloco de construção modular**, isto é, uma parte do sistema modular, executável, implantável, independente, padronizada e reutilizável que encapsula a implementação e expõe um conjunto de interfaces do sistema

O uso de componentes economiza tempo e esforço, já que reduz a quantidade de código que precisa ser escrita do zero. Isso também melhora a qualidade do software, já que os componentes são previamente testados e depurados.

Veja as principais fases do modelo baseado em componentes:

Fase do modelo baseado em componentes	DESCRIÇÃO
Especificação de Requisitos	Tem o objetivo de traduzir as informações coletadas durante a atividade de análise em um documento que define um conjunto de requisitos de software. Devem ser incluídos dois tipos de requisitos nesse documento: os Requisitos de Usuário e Requisitos de Sistema.
Análise de Componentes	Nesta fase, é feita uma busca pelos componentes para implementar essa especificação. Geralmente, não existe uma correspondência exata entre o componente encontrado e o procurado. Muitas vezes, os componentes que podem ser usados fornecem apenas parte da funcionalidade necessária.
Modificação de Requisitos	Os requisitos são analisados usando as informações sobre os componentes encontrados, que são modificados para refletir os componentes disponíveis. Quando as modificações são impossíveis, a atividade de análise de componentes pode ser novamente realizada para procurar alternativas.
Projeto de Sistema com Reúso	O framework do sistema é projetado ou um framework existente é reutilizado. Os projetistas levam em consideração os componentes reusados. Pode ser necessário projetar algum software novo caso os componentes reusáveis não estejam disponíveis para aquisição para o sistema.

Desenvolvimento e Integração	Software que não pode ser adquirido externamente é desenvolvido e os componentes e os sistemas COTS são integrados para criar os novos sistemas. A integração de sistema, neste modelo, pode ser parte do processo de desenvolvimento, em vez de ser uma atividade separada.
Validação de Sistema	Processo de verificação de se um sistema atende às necessidades e expectativas do cliente. O que são Sistemas COTS? Esse é o acrônimo de Commercial Off-The-Shelf, que é um conjunto de soluções pré-fabricadas e disponíveis no mercado, podendo ser compradas ou licenciadas, i.e., uma grande biblioteca de componentes prontos.

O desenvolvimento baseado em componentes tem muitas **vantagens**, incluindo a economia de tempo e esforço, a reutilização de software comprovado e a capacidade de responder rapidamente às mudanças. No entanto, também tem **desvantagens**, como a dependência de fornecedores de componentes terceirizados, a possibilidade de conflitos entre componentes e a dificuldade de integrar componentes que não foram projetados para trabalhar juntos.

Métodos Formais

Métodos Formais é uma abordagem de desenvolvimento de software que **utiliza linguagens matemáticas rigorosas para especificar, desenvolver e verificar sistemas**. Ao contrário de outras metodologias que podem depender mais de experiências práticas, os Métodos Formais buscam tornar o processo de desenvolvimento de software tão preciso e previsível quanto possível.

A metodologia inclui várias técnicas e ferramentas, mas todas elas compartilham algumas características comuns, tais como:

- **Especificação Formal**: Envolve a descrição precisa dos requisitos do sistema e do comportamento do sistema em uma linguagem matemática. Isso fornece uma base sólida para o desenvolvimento subsequente e ajuda a evitar mal-entendidos ou ambiguidades.
- **Verificação Formal**: A verificação formal envolve a prova de que o software atende à sua especificação formal. Abrange a prova de propriedades do software ou a verificação de que o software se comporta conforme esperado em todas as possíveis condições de entrada.
- **Desenvolvimento Formal**: O desenvolvimento formal envolve a transformação da especificação formal em um projeto ou código de software, novamente utilizando linguagem matemática. Inclui a aplicação de regras rigorosas de transformação ou a utilização de ferramentas automatizadas.

Na especificação formal de sistemas, são utilizados conceitos de lógica, conjuntos e funções, que são campos fundamentais da matemática. Estes oferecem um meio de descrever e analisar sistemas de forma precisa e rigorosa.

Por exemplo, considere um sistema simples de gerenciamento de banco de dados. Poderíamos usar conjuntos para representar as tabelas no banco de dados, com cada tabela sendo um conjunto de registros. As funções poderiam ser usadas para representar as operações do banco de dados, como adicionar um registro a uma tabela ou recuperar um registro de uma tabela. E a lógica poderia ser usada para fazer afirmações sobre o banco de dados, como "todos os registros em uma tabela têm um identificador único" ou "a operação de adicionar um registro sempre aumenta o tamanho da tabela em um".

A principal vantagem dos Métodos Formais é que eles podem fornecer um **alto grau de confiança de que o software está correto** e potencialmente livre de bugs. Eles são particularmente úteis para sistemas críticos de segurança ou missão, onde os erros podem ter consequências graves.

No entanto, Métodos Formais também têm desvantagens. Eles **são complexos e difíceis de usar, e exigem um alto grau de conhecimento e experiência**. Além disso, eles são demorados e caros de aplicar, especialmente em grandes sistemas. Por último, embora os Métodos Formais possam provar a correção em relação à especificação, eles não podem garantir que a especificação em si seja correta ou completa.

Modelo Orientado a Aspectos

Desenvolvimento Orientado a Aspectos (Aspect-Oriented Programming, AOP) é uma abordagem de desenvolvimento de software que busca **aumentar a modularidade ao permitir a separação de preocupações ou interesses "principais" (core concerns) de preocupações "transversais" ou "ortogonais" (cross-cutting concerns)**.

Interesses principais capturam as funcionalidades de negócio dos módulos e estão diretamente relacionadas ao domínio da aplicação, ou seja, são os serviços oferecidos pelo software.

Já as preocupações transversais são aspectos do sistema que afetam outros aspectos do sistema, tornando-os difíceis de modularizar usando técnicas de programação orientada a objetos ou procedural tradicionais. Exemplos comuns de preocupações transversais incluem logging, gerenciamento de transações, segurança e tratamento de erros.

Em um sistema tradicional, o código para lidar com essas preocupações transversais pode ser espalhado por todo o sistema, resultando em código repetitivo e um acoplamento alto entre módulos. Isso pode dificultar a manutenção e a evolução do sistema.

O Desenvolvimento Orientado a Aspectos aborda esse problema permitindo que os desenvolvedores definam "aspectos", que encapsulam comportamentos que cortam vários módulos ou classes de objetos. Esses aspectos podem então ser "costurados" no sistema em pontos de junção especificados (join points), como chamadas de método ou manipulações de exceções.

Veja as diferenças entre preocupações principais e ortogonais:

	Preocupações Principais	Preocupações Transversais
Definição	São os requisitos funcionais e comportamentos centrais do sistema que refletem os objetivos diretos da aplicação.	São os requisitos e comportamentos que não estão ligados ao comportamento central da aplicação, mas que são necessários para o funcionamento correto e eficiente do sistema. Esses aspectos podem afetar múltiplos módulos ou componentes.
Exemplos	Em um sistema bancário, as preocupações principais podem incluir transações como depósito, saque e transferência de dinheiro.	Em um sistema bancário, as preocupações transversais podem incluir logging, segurança, autenticação, autorização e tratamento de erros.
Modularização	São normalmente encapsuladas em módulos ou objetos específicos, de acordo com os princípios da programação orientada a objetos.	Podem ser difíceis de modularizar usando abordagens tradicionais, pois podem atravessar vários módulos ou objetos. São normalmente encapsuladas em "aspectos" na programação orientada a aspectos.
Impacto no sistema	Modificar uma preocupação principal pode afetar diretamente a funcionalidade central do sistema.	Modificar uma preocupação transversal pode ter efeitos indiretos em várias partes do sistema, tornando o impacto de tais mudanças menos previsível.

Por exemplo, um aspecto de logging pode ser definido para registrar informações sobre chamadas de método. Esse aspecto pode então ser costurado em todas as chamadas de método no sistema, sem a necessidade de modificar o código de cada método individualmente.

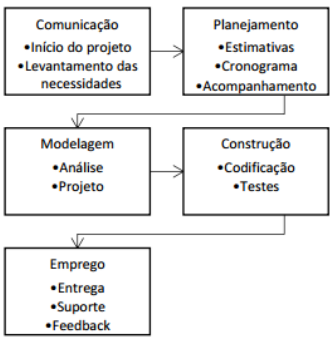
Os benefícios do Desenvolvimento Orientado a Aspectos incluem **maior modularidade, menor acoplamento entre módulos e maior facilidade de manutenção**. Porém, também pode ser mais complexo de entender e usar do que abordagens de programação tradicionais, e a introdução de aspectos pode levar a interações não intencionais ou difíceis de prever entre diferentes partes do sistema.

QUESTÕES ESTRATÉGICAS

Nesta seção, apresentamos e comentamos uma amostra de questões objetivas selecionadas estrategicamente: são questões com nível de dificuldade semelhante ao que você deve esperar para a sua prova e que, em conjunto, abordam os principais pontos do assunto.

A ideia, aqui, não é que você fixe o conteúdo por meio de uma bateria extensa de questões, mas que você faça uma boa revisão global do assunto a partir de, relativamente, poucas questões

1. **(FGV / COMPESA – 2016)** Observe a figura a seguir, que representa um modelo de processo de software.



Este modelo, algumas vezes chamado ciclo de vida clássico, sugere uma abordagem sequencial e sistemática para o desenvolvimento de software nos casos em que os requisitos de um problema são bem compreendidos e quando o trabalho flui da comunicação ao emprego de forma relativamente linear. O modelo apresentado é denominado:

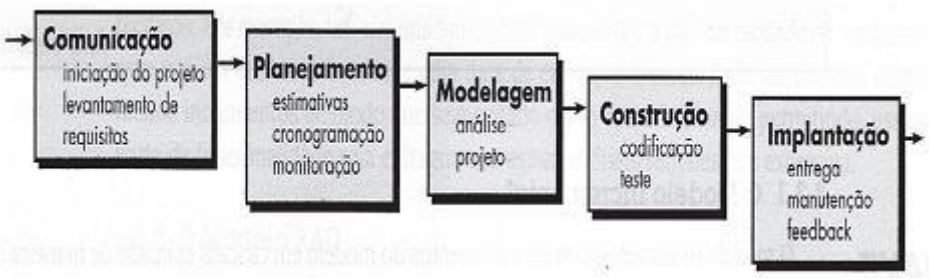
a) incremental.
b) cascata.
c) evolucionário.
d) unificado
e) especializado.

Comentários:

Palavras-chave: ciclo de vida clássico; abordagem sequencial e sistemática; útil quando requisitos são bem compreendidos. Trata-se do famoso Modelo em Cascata (Waterfall).

Gabarito: B

2. **(FGV / PGE-RO – 2015)** A figura abaixo ilustra um modelo de processo, que prescreve um conjunto de elementos de processo como atividades de arcabouço, ações de engenharia de software, tarefas, produtos de trabalho, mecanismos de garantia de qualidade e de controle de modificações para cada projeto.



Esse modelo é conhecido como Modelo:

a) por funções.
b) em cascata.
c) incremental.
d) em pacotes.
e) por módulos.

Comentários:

POR SOMMERVILLE	POR ROYCE	POR PRESSMAN (6ª ED)
Análise e Definição de Requisitos	Requisitos de Sistema	Comunicação
Projeto de Sistema e Software	Requisitos de Software	Planejamento
Implementação e Teste de Unidade	Análise	Modelagem
Integração e Teste de Sistema	Projeto	Construção
Operação e Manutenção	Codificação	Implantação
	Teste	

	Operação	

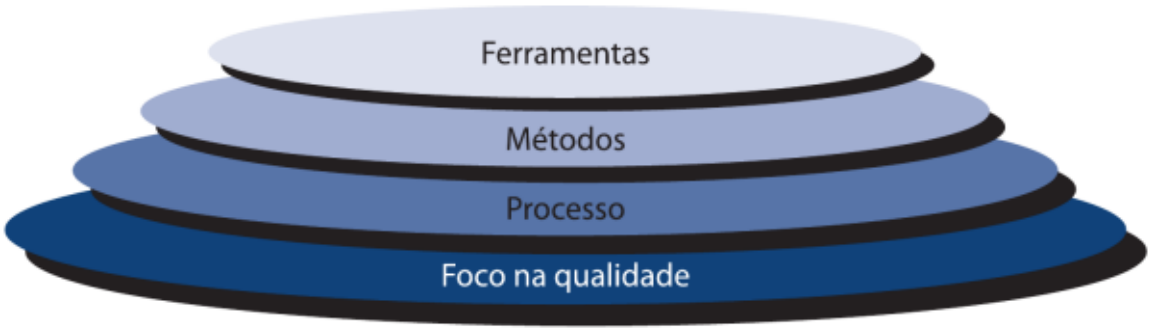
Trata-se das fases descritas pelo Pressman para o Modelo em Cascata.

Gabarito: B

3. **(FGV / BADESC – 2010)** De acordo com Pressman, a engenharia de software é baseada em camadas, com foco na qualidade. Essas camadas são:

a) métodos, processo e teste.
b) ferramentas, métodos e processo.
c) métodos, construção, teste e implantação.
d) planejamento, modelagem, construção, validação e implantação.
e) comunicação, planejamento, modelagem, construção e implantação.

Comentários:



Bastava lembrar da imagem para responder à questão!

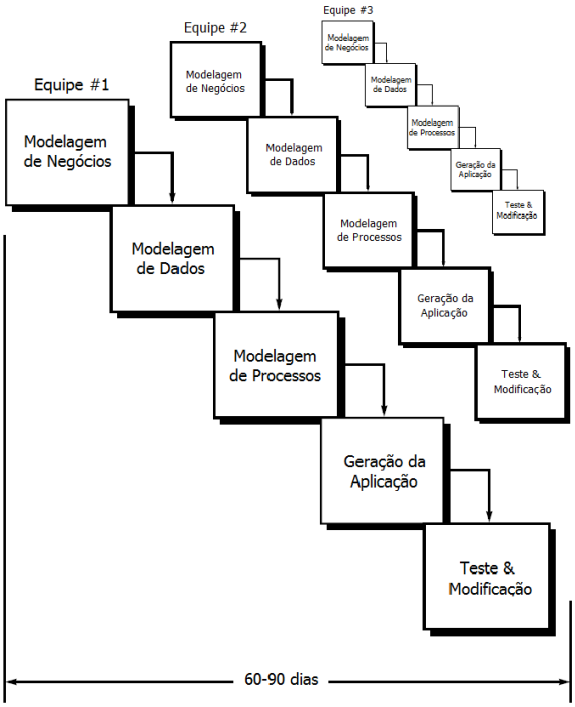
Gabarito: B

4. **(FGV / Fiocruz – 2010)** Rapid Application Development (RAD) é um modelo de processo de software incremental que enfatiza um ciclo de desenvolvimento curto, com o uso de uma abordagem de construção baseada em componentes. Nesse modelo, três das principais fases são abrangidas pelas modelagens:

a) do negócio, dos recursos financeiros e das funções gerenciais.
b) do gerenciamento, dos recursos de TI e dos processos.

c) do planejamento, dos dados e das funções gerenciais.
d) do planejamento, dos recursos de TI e dos projetos
e) do negócio, dos dados e dos processos.

Comentários:



Nesse modelo, três das principais fases são abrangidas pelas modelagens: Modelagem de Negócios, Modelagem de Dados e Modelagem de Processos.

Gabarito: E

5. **(FGV / CODESP-SP – 2010)** A UML é uma linguagem visual para modelar sistemas orientados a objetos, sendo independente tanto de linguagens de programação quanto de processos de desenvolvimento. Nesse contexto, analise a figura abaixo, que representa um modelo de ciclo de vida para desenvolvimento de sistemas. Essa abordagem divide o desenvolvimento de software em ciclos, em que, em cada ciclo, podem ser identificadas as fases de análise, projeto, implementação e testes. Cada um dos ciclos considera um subconjunto de requisitos, e estes são desenvolvidos uma vez que sejam alocados a um ciclo de desenvolvimento. Esse modelo de ciclo de vida é denominado:

a) clássico;

b) em cascata;
c) prototipação;
d) estruturado por fases;
e) incremental e iterativo.

Comentários:

Uma informação bizarra: a questão menciona uma figura, mas a prova não trouxe figura alguma e a questão não foi anulada. De toda forma, a abordagem em ciclos em que, em cada ciclo, podem ser identificadas as fases de análise, projeto, implementação e testes, sendo que cada um dos ciclos considera um subconjunto de requisitos, e estes são desenvolvidos uma vez que sejam alocados a um ciclo de desenvolvimento é o modelo iterativo e incremental.

Gabarito: E

6. **(FGV / Senado Federal – 2008)** Considere as seguintes assertivas sobre modelos de processos de software:

I. No modelo em cascata, a fase seguinte não deve iniciar antes que a fase precedente tenha sido concluída.
II. No modelo evolucionário, a mudança constante tende a corromper a estrutura do software
III. A explícita consideração dos riscos no modelo em espiral distingue esse modelo dos modelos em cascata e evolucionário.

As assertivas corretas são:

a) somente I.
b) somente I e II.
c) somente I e III.
d) somente II e III.
e) I, II e III.

Comentários:

(I) Correto, uma fase só se inicia após o término e aprovação da fase anterior; (II) Correto, muitas mudanças tendem a corromper a estrutura do software e isso as tornam difíceis e caras; (III) Correto, a ideia do modelo espiral é representar um processo de software orientado a riscos, o que o diferencia dos demais modelos.

Gabarito: E

7. **(VUNESP / TJM-SP – 2021)** O modelo de desenvolvimento de software RAD (Rapid Application Development) conta com uma fase de Modelagem, que compreende a modelagem de:

a) Negócio, Dados e Processo.
b) Teste, Integração e Negócio.
c) Protótipo, Entrega e Dados.
d) Comunicação, Integração e Teste.
e) Entrega, Comunicação e Protótipo

Comentários:

RAD conta com uma fase de modelagem que compreende modelagem de negócio, modelagem de dados e modelagem de processo.

Gabarito: A

8. **(VUNESP / PRODEST-ES – 2014)** No modelo de ciclo de vida de software conhecido como RAD (Rapid Application Development) há duas atividades, cujas tarefas podem ser distribuídas por diversas equipes. Essas atividades são:

a) comunicação e modelagem
b) comunicação e planejamento.
c) integração e construção.
d) modelagem e construção.
e) planejamento e integração.

Comentários:

Modelagem de Negócio, Dados e Processos são frequentemente condensados na etapa de Modelagem e Geração da Aplicação, Teste e Modificação são frequentemente condensados na etapa de Construção. Logo, trata-se de Modelagem e Construção.

Gabarito: D

9. **(VUNESP / SPTrans – 2012)** Uma das abordagens do processo de desenvolvimento da engenharia de software prevê a divisão em etapas, em que o fim de uma é a entrada para a próxima. Esse processo é conhecido como modelo:

a) Transformação.
b) Incremental.
c) Evolutivo.
d) Espiral.
e) Cascata.

Comentários:

No Modelo em Cascata, uma fase só se inicia após o término e aprovação da fase anterior, isto é, há uma sequência de desenvolvimento do projeto. Por exemplo, a Fase 4 só é iniciada após o término e aprovação da Fase 3. A Fase 5 só é iniciada após o término e aprovação da Fase 4. Logo, conforme vimos em aula, trata-se do Modelo em Cascata.

Gabarito: E

10. **(FGV / IBGE – 2016)** A empresa SONOVATOS desenvolve sistemas há pouco tempo no mercado e, como padrão, sempre utilizou o modelo Cascata de ciclo de vida. Alguns clientes ficaram insatisfeitos com os produtos desenvolvidos pela empresa por não estarem de acordo com suas necessidades. Atualmente a SONOVATOS está desenvolvendo sistemas muito maiores, com duração de vários anos, e com requisitos ainda instáveis. O próprio processo de desenvolvimento da empresa também está em reformulação. Assim, a adoção de um novo modelo de ciclo de vida está sendo avaliada pelos gerentes da empresa. A intenção da SONOVATOS é, principalmente, gerenciar riscos e poder reavaliar constantemente o processo de desenvolvimento ao longo do projeto, o que permitiria correções nesse processo ou até mudança do tipo de processo. O modelo mais adequado para os sistemas atuais de longa duração da SONOVATOS é:

a) Rapid Application Development (RAD);
b) Espiral;
c) Extremme Programming;
d) Prototipação;
e) Modelo V

Comentários:

A questão menciona algumas palavras-chave: requisitos instáveis, gerenciamento dos riscos, reavaliação constante, correções/mudanças do processo. Dito isso, vamos analisar: (a) Errado, esse modelo não trata de gerenciamento de riscos e é mais voltado para projetos com entregas rápidas; (b) Correto, esse modelo traz explicitamente uma preocupação com o gerenciamento de riscos, além de permitir uma reavaliação/correção/adaptação/mudança constantemente; (c) Errado, esse modelo não trata explicitamente do gerenciamento de riscos – apesar de concordar que caberia recurso aqui; (d) Errado, esse modelo também não trata explicitamente de riscos; (e) Errado, esse modelo é uma variação do modelo em cascata.

Como a questão afirma que a intenção principal da empresa é gerenciar riscos, o modelo que explicitamente coloca um foco no gerenciamento de riscos é o modelo em espiral.

Gabarito: B

11. **(FGV / TCM-SP – 2015)** Software, assim como todos os sistemas complexos, evolui ao longo do tempo. Modelos de processos evolucionários reconhecem a natureza iterativa e incremental da maioria dos projetos de engenharia de software e são projetados para adequar mudanças. Os modelos a serem utilizados em um processo evolucionário são:

a) cascata e modelo V;
b) prototipação e modelo espiral;
c) concorrente e métodos formais;
d) incremental e baseado em componentes;
e) processo unificado e orientado a aspectos.

Comentários:

Os modelos a serem utilizados em um processo evolucionário são: prototipação e espiral.

Gabarito: B

12. **(FGV / Fiocruz – 2010)** Como modelo evolucionário do processo de software, uma característica da prototipagem é:

a) independer do estabelecimento e da definição de requisitos.
b) configurar um processo interativo e rápido de desenvolvimento.

c) iniciar o processo de desenvolvimento pela implantação e pelos testes.
d) gerar uma primeira versão do sistema completa e isenta de erros.
e) descartar a participação do cliente no processo de desenvolvimento e de implantação.

Comentários:

(a) Errado, claro que depende da definição de requisitos; (b) Correto, é um processo interativo, iterativo e rápido de desenvolvimento; (c) Errado. Você vai implantar e testar o que? Você primeiro tem que levantar, especificar e desenvolver; (d) Errado, se é uma primeira versão, não é completa. Além disso, não há sistemas sem erros; (e) Errado, você deve incentivar a participação do cliente.

Gabarito: B

13. **(FGV / BADESC – 2010)** O Modelo Espiral, segundo Pressman (1995), incorpora as melhores características do Ciclo de Vida Clássico e da Prototipação e acrescenta o seguinte elemento:

a) análise dos riscos.
b) análise de projetos.
c) avaliação de usuários.
d) refinamento de requisitos.
e) refinamento de protótipos.

Comentários:

O Modelo Espiral foi o primeiro modelo a tratar explicitamente de análise de riscos.

Gabarito: A

14. **(VUNESP / TJM-SP – 2021)** Algumas atividades que fazem parte do modelo espiral de desenvolvimento de software são:

Construção – Implantação – Comunicação –
Planejamento – Modelagem

A ordem correta com que tais atividades são executadas, considerando o modelo espiral, é:

a) Comunicação, Planejamento, Modelagem, Construção e Implantação.
b) Construção, Implantação, Comunicação, Modelagem e Planejamento.
c) Modelagem, Planejamento, Construção, Implantação e Comunicação.
d) Planejamento, Construção, Implantação, Comunicação e Modelagem.
e) Planejamento, Modelagem, Comunicação, Construção e Implantação.

Comentários:

De acordo com Pressman, cada espiral é dividida em cinco setores:

Setores (por Pressman)	DESCRIÇÃO
Comunicação	É a comunicação em si
Planejamento	Estimativa de custos, cronograma e **análise de riscos**
Modelagem	Análise e design
Construção	Codificação e teste
implantação	Entrega e feedback

Gabarito: A

15. **(VUNESP / CETESB – 2009)** Considere um sistema cujos requisitos de interface são definidos apenas quando o cliente realiza um test-drive na aplicação e aprova essa interface. Assinale a alternativa que apresenta o modelo mais adequado para o desenvolvimento da interface desse sistema.

a) Ágil.
b) Cascata.
c) Iterativo incremental.
d) Prototipação.
e) Rapid Application Development.

Comentários:

Prototipação é uma maneira do usuário "degustar" o sistema antes de ser realizado o desenvolvimento real da aplicação.

Gabarito: D

QUESTIONÁRIO DE REVISÃO E APERFEIÇOAMENTO

A ideia do questionário é elevar o nível da sua compreensão no assunto e, ao mesmo tempo, proporcionar uma outra forma de revisão de pontos importantes do conteúdo, a partir de perguntas que exigem respostas subjetivas.

São questões um pouco mais desafiadoras, porque a redação de seu enunciado não ajuda na sua resolução, como ocorre nas clássicas questões objetivas.

O objetivo é que você realize uma auto explicação mental de alguns pontos do conteúdo, para consolidar melhor o que aprendeu ;)

Além disso, as questões objetivas, em regra, abordam pontos isolados de um dado assunto. Assim, ao resolver várias questões objetivas, o candidato acaba memorizando pontos isolados do conteúdo, mas muitas vezes acaba não entendendo como esses pontos se conectam.

Assim, no questionário, buscaremos trazer também situações que ajudem você a conectar melhor os diversos pontos do conteúdo, na medida do possível.

É importante frisar que não estamos adentrando em um nível de profundidade maior que o exigido na sua prova, mas apenas permitindo que você compreenda melhor o assunto de modo a facilitar a resolução de questões objetivas típicas de concursos, ok?

Nosso compromisso é proporcionar a você uma revisão de alto nível!

Vamos ao nosso questionário:

Perguntas

1. O que é Engenharia de Software?
2. O que são ferramentas, métodos e processos na Engenharia de Software?
3. Qual é o objetivo de um ciclo de vida de software?
4. Descreva o modelo em cascata.
5. Quais são as vantagens e desvantagens do modelo em cascata?
6. O que é o modelo iterativo e incremental?

7. Como o modelo iterativo e incremental se compara ao modelo em cascata?
8. O que é o modelo RAD?
9. O que é o modelo de Prototipagem?
10. Quais são os tipos de protótipos?
11. O que é o modelo em espiral?
12. O que é o modelo baseado em componentes?
13. O que são métodos formais em desenvolvimento de software?
14. O que é a modelagem orientada a aspectos?
15. O que é uma preocupação principal?
16. O que é uma preocupação transversal?
17. O que é um protótipo descartável?
18. O que é um protótipo evolutivo?
19. O que é um componente de software no modelo baseado em componentes?
20. O que são ferramentas na engenharia de software?

Perguntas e Respostas

1. O que é Engenharia de Software?
   Resposta: A Engenharia de Software é a disciplina que envolve todos os aspectos de produção de software, desde a concepção, passando pelo desenvolvimento de software até a manutenção do sistema após a entrega.
2. O que são ferramentas, métodos e processos na Engenharia de Software?
   Resposta: As ferramentas são os softwares ou aplicações usados para auxiliar o desenvolvimento de software. Os métodos referem-se às práticas e técnicas usadas para realizar um trabalho específico. Os processos são o conjunto de atividades relacionadas e coerentes que envolvem métodos, ferramentas e procedimentos para desenvolver ou manter software.
3. Qual é o objetivo de um ciclo de vida de software?
   Resposta: O objetivo de um ciclo de vida de software é fornecer um modelo que descreva as fases do ciclo de vida de um software, desde a concepção até a descontinuação.
4. Descreva o modelo em cascata.
   Resposta: O modelo em cascata é um processo sequencial que começa com a definição de requisitos e avança através de várias fases, como design, implementação, verificação e manutenção.
5. Quais são as vantagens e desvantagens do modelo em cascata?
   Resposta: As vantagens incluem simplicidade e estrutura clara. As desvantagens incluem a

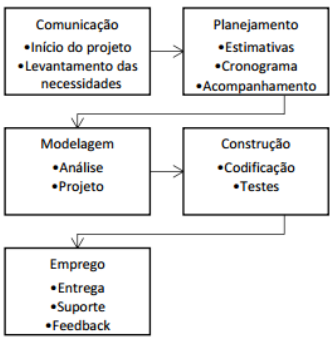
dificuldade de mudança após uma fase ser concluída e a possibilidade de o cliente só ver o software após a conclusão do projeto.

6. O que é o modelo iterativo e incremental?
   Resposta: É um modelo de desenvolvimento de software que enfatiza a repetição de ciclos de desenvolvimento (iterações) e a entrega incremental de funcionalidades ao longo do tempo.
7. Como o modelo iterativo e incremental se compara ao modelo em cascata?
   Resposta: O modelo iterativo e incremental permite mudanças e ajustes ao longo do processo de desenvolvimento, diferentemente do modelo em cascata. Além disso, ele permite obter feedback do usuário mais cedo no processo.
8. O que é o modelo RAD?
   Resposta: RAD, ou Rapid Application Development, é uma metodologia que enfatiza a rápida criação de protótipos e a iteração com o usuário para obter feedback e refinamento.
9. O que é o modelo de Prototipagem?
   Resposta: É uma metodologia de desenvolvimento de software que envolve a criação de protótipos rápidos para demonstrar funcionalidades e obter feedback do usuário antes do desenvolvimento do produto final.
10. Quais são os tipos de protótipos?
    Resposta: Os dois tipos principais são os descartáveis e os evolutivos. Os descartáveis são usados para explorar ideias e receber feedback, mas não são usados no produto final. Os evolutivos são continuamente refinados até se tornarem o produto final.
11. O que é o modelo em espiral?
    Resposta: O modelo em espiral é uma metodologia de desenvolvimento de software que combina elementos de abordagens de cascata e iterativa, com ênfase em avaliação contínua de riscos.
12. O que é o modelo baseado em componentes?
    Resposta: É uma abordagem para o desenvolvimento de software que envolve a criação de componentes individuais de software que podem ser reutilizados em diferentes sistemas.
13. O que são métodos formais em desenvolvimento de software?
    Resposta: Os métodos formais são técnicas que utilizam notações e técnicas matemáticas rigorosas para especificar, projetar e verificar sistemas.
14. O que é a modelagem orientada a aspectos?
    Resposta: A modelagem orientada a aspectos é uma abordagem de desenvolvimento de software que procura aumentar a modularidade, permitindo a separação de preocupações transversais, que são aspectos do sistema que afetam outros aspectos do sistema.

15. O que é uma preocupação principal?
    Resposta: Uma preocupação principal é um requisito ou comportamento central de um sistema que reflete os objetivos diretos da aplicação.
16. O que é uma preocupação transversal?
    Resposta: Uma preocupação transversal é um aspecto do sistema que afeta outros aspectos do sistema e é difícil de modularizar usando técnicas tradicionais.
17. O que é um protótipo descartável?
    Resposta: É um tipo de protótipo que é criado para explorar ideias ou conceitos e coletar feedback dos usuários, mas que não é destinado a ser usado no produto final.
18. O que é um protótipo evolutivo?
    Resposta: É um tipo de protótipo que é criado com a intenção de ser refinado e melhorado ao longo do tempo até se tornar o produto final.
19. O que é um componente de software no modelo baseado em componentes?
    Resposta: É uma unidade independente de código que fornece uma funcionalidade específica e que pode ser combinada com outros componentes para criar um sistema.
20. O que são ferramentas na engenharia de software?
    Resposta: Ferramentas são softwares ou aplicações usadas para auxiliar as atividades de desenvolvimento de software, como programação, gerenciamento de projetos, controle de versões, entre outros.

LISTA DE QUESTÕES ESTRATÉGICAS

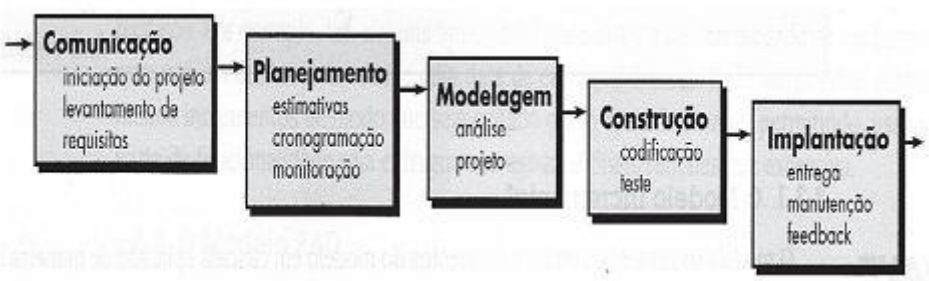
1. **(FGV / COMPESA – 2016)** Observe a figura a seguir, que representa um modelo de processo de software.



Este modelo, algumas vezes chamado ciclo de vida clássico, sugere uma abordagem sequencial e sistemática para o desenvolvimento de software nos casos em que os requisitos de um problema são bem compreendidos e quando o trabalho flui da comunicação ao emprego de forma relativamente linear. O modelo apresentado é denominado:

a) incremental.
b) cascata.
c) evolucionário.
d) unificado
e) especializado.

2. **(FGV / PGE-RO – 2015)** A figura abaixo ilustra um modelo de processo, que prescreve um conjunto de elementos de processo como atividades de arcabouço, ações de engenharia de software, tarefas, produtos de trabalho, mecanismos de garantia de qualidade e de controle de modificações para cada projeto.



Esse modelo é conhecido como Modelo:

a) por funções.
b) em cascata.
c) incremental.
d) em pacotes.
e) por módulos.

3. **(FGV / BADESC – 2010)** De acordo com Pressman, a engenharia de software é baseada em camadas, com foco na qualidade. Essas camadas são:

a) métodos, processo e teste.
b) ferramentas, métodos e processo.
c) métodos, construção, teste e implantação.
d) planejamento, modelagem, construção, validação e implantação.
e) comunicação, planejamento, modelagem, construção e implantação.

4. **(FGV / Fiocruz – 2010)** Rapid Application Development (RAD) é um modelo de processo de software incremental que enfatiza um ciclo de desenvolvimento curto, com o uso de uma abordagem de construção baseada em componentes. Nesse modelo, três das principais fases são abrangidas pelas modelagens:

   a) do negócio, dos recursos financeiros e das funções gerenciais.
   b) do gerenciamento, dos recursos de TI e dos processos.
   c) do planejamento, dos dados e das funções gerenciais.
   d) do planejamento, dos recursos de TI e dos projetos
   e) do negócio, dos dados e dos processos.

5. **(FGV / CODESP-SP – 2010)** A UML é uma linguagem visual para modelar sistemas orientados a objetos, sendo independente tanto de linguagens de programação quanto de processos de desenvolvimento. Nesse contexto, analise a figura abaixo, que representa um modelo de ciclo de vida para desenvolvimento de sistemas. Essa abordagem divide o desenvolvimento de software em ciclos, em que, em cada ciclo, podem ser identificadas as fases de análise, projeto, implementação e testes. Cada um dos ciclos considera um subconjunto de requisitos, e estes são desenvolvidos uma vez que sejam alocados a um ciclo de desenvolvimento. Esse modelo de ciclo de vida é denominado:

   a) clássico;
   b) em cascata;
   c) prototipação;
   d) estruturado por fases;
   e) incremental e iterativo.

6. **(FGV / Senado Federal – 2008)** Considere as seguintes assertivas sobre modelos de processos de software:

   I. No modelo em cascata, a fase seguinte não deve iniciar antes que a fase precedente tenha sido concluída.
   II. No modelo evolucionário, a mudança constante tende a corromper a estrutura do software
   III. A explícita consideração dos riscos no modelo em espiral distingue esse modelo dos modelos em cascata e evolucionário.

   As assertivas corretas são:

   a) somente I.
   b) somente I e II.

c) somente I e III.
d) somente II e III.
e) I, II e III.

7. **(VUNESP / TJM-SP – 2021)** O modelo de desenvolvimento de software RAD (Rapid Application Development) conta com uma fase de Modelagem, que compreende a modelagem de:

a) Negócio, Dados e Processo.
b) Teste, Integração e Negócio.
c) Protótipo, Entrega e Dados.
d) Comunicação, Integração e Teste.
e) Entrega, Comunicação e Protótipo

8. **(VUNESP / PRODEST-ES – 2014)** No modelo de ciclo de vida de software conhecido como RAD (Rapid Application Development) há duas atividades, cujas tarefas podem ser distribuídas por diversas equipes. Essas atividades são:

a) comunicação e modelagem
b) comunicação e planejamento.
c) integração e construção.
d) modelagem e construção.
e) planejamento e integração.

9. **(VUNESP / SPTrans – 2012)** Uma das abordagens do processo de desenvolvimento da engenharia de software prevê a divisão em etapas, em que o fim de uma é a entrada para a próxima. Esse processo é conhecido como modelo:

a) Transformação.
b) Incremental.
c) Evolutivo.
d) Espiral.
e) Cascata.

10. **(FGV / IBGE – 2016)** A empresa SONOVATOS desenvolve sistemas há pouco tempo no mercado e, como padrão, sempre utilizou o modelo Cascata de ciclo de vida. Alguns clientes ficaram insatisfeitos com os produtos desenvolvidos pela empresa por não estarem de acordo com suas necessidades. Atualmente a SONOVATOS está desenvolvendo sistemas muito maiores, com duração de vários anos, e com requisitos ainda instáveis. O próprio processo de desenvolvimento da empresa também está em reformulação. Assim, a adoção de um novo modelo de ciclo de vida está sendo avaliada

pelos gerentes da empresa. A intenção da SONOVATOS é, principalmente, gerenciar riscos e poder reavaliar constantemente o processo de desenvolvimento ao longo do projeto, o que permitiria correções nesse processo ou até mudança do tipo de processo. O modelo mais adequado para os sistemas atuais de longa duração da SONOVATOS é:

a) Rapid Application Development (RAD);
b) Espiral;
c) Extremme Programming;
d) Prototipação;
e) Modelo V

11. **(FGV / TCM-SP – 2015)** Software, assim como todos os sistemas complexos, evolui ao longo do tempo. Modelos de processos evolucionários reconhecem a natureza iterativa e incremental da maioria dos projetos de engenharia de software e são projetados para adequar mudanças. Os modelos a serem utilizados em um processo evolucionário são:

a) cascata e modelo V;
b) prototipação e modelo espiral;
c) concorrente e métodos formais;
d) incremental e baseado em componentes;
e) processo unificado e orientado a aspectos.

12. **(FGV / Fiocruz – 2010)** Como modelo evolucionário do processo de software, uma característica da prototipagem é:

a) independer do estabelecimento e da definição de requisitos.
b) configurar um processo interativo e rápido de desenvolvimento.
c) iniciar o processo de desenvolvimento pela implantação e pelos testes.
d) gerar uma primeira versão do sistema completa e isenta de erros.
e) descartar a participação do cliente no processo de desenvolvimento e de implantação.

13. **(FGV / BADESC – 2010)** O Modelo Espiral, segundo Pressman (1995), incorpora as melhores características do Ciclo de Vida Clássico e da Prototipação e acrescenta o seguinte elemento:

a) análise dos riscos.
b) análise de projetos.
c) avaliação de usuários.
d) refinamento de requisitos.

e) refinamento de protótipos.

14. **(VUNESP / TJM-SP – 2021)** Algumas atividades que fazem parte do modelo espiral de desenvolvimento de software são:

Construção – Implantação – Comunicação –
Planejamento – Modelagem

A ordem correta com que tais atividades são executadas, considerando o modelo espiral, é:

a) Comunicação, Planejamento, Modelagem, Construção e Implantação.
b) Construção, Implantação, Comunicação, Modelagem e Planejamento.
c) Modelagem, Planejamento, Construção, Implantação e Comunicação.
d) Planejamento, Construção, Implantação, Comunicação e Modelagem.
e) Planejamento, Modelagem, Comunicação, Construção e Implantação.

15. **(VUNESP / CETESB – 2009)** Considere um sistema cujos requisitos de interface são definidos apenas quando o cliente realiza um test-drive na aplicação e aprova essa interface. Assinale a alternativa que apresenta o modelo mais adequado para o desenvolvimento da interface desse sistema.

a) Ágil.
b) Cascata.
c) Iterativo incremental.
d) Prototipação.
e) Rapid Application Development.

Gabaritos

1. B
2. B
3. B
4. E
5. E
6. E
7. A
8. D
9. E

10. B
11. B
12. B
13. A
14. A
15. D

ESSA LEI TODO MUNDO CONHECE:
PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.

Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.