Estratégia Concursos

Aula 00

Caixa Econômica Federal - CEF (Técnico Bancário - TI) Engenharia de Software - 2021 (Pós-Edital)

Autor:
Diego Carvalho, Equipe Informática e TI, Fernando Pedrosa Lopes

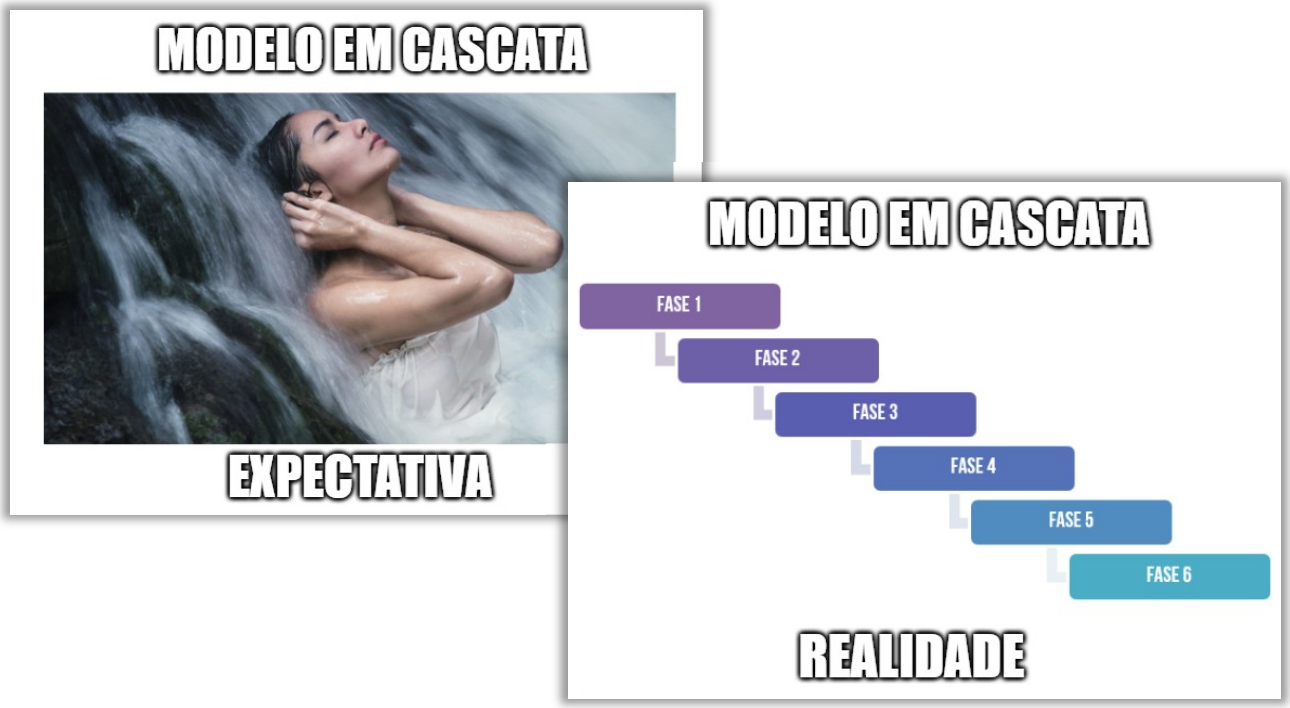
15 de Setembro de 2021

Sumário

APRESENTAÇÃO DA AULA

Pessoal, o tema da nossa aula é: **Metodologias de Desenvolvimento de Software**. A ideia aqui é entender os conceitos básicos do desenvolvimento de um software, passando pelas principais fases e etapas. Veremos também alguns processos de desenvolvimento bastante comuns e como eles se diferenciam dos demais. É um assunto tranquilo que nos ajudará a entender como softwares são desenvolvidos. Vamos lá...

PROFESSOR DIEGO CARVALHO - WWW.INSTAGRAM.COM/PROFESSORDIEGOCARVALHO



Galera, todos os tópicos da aula possuem Faixas de Incidência, que indicam se o assunto cai muito ou pouco em prova. *Diego, se cai pouco para que colocar em aula?* Cair pouco não significa que não cairá justamente na sua prova! A ideia aqui é: se você está com pouco tempo e precisa ver somente aquilo que cai mais, você pode filtrar pelas incidências média, alta e altíssima; se você tem tempo sobrando e quer ver tudo, vejam também as incidências baixas e baixíssimas. *Fechado?*

INCIDÊNCIA EM PROVA: BAIXÍSSIMA
INCIDÊNCIA EM PROVA: BAIXA
INCIDÊNCIA EM PROVA: MÉDIA
INCIDÊNCIA EM PROVA: ALTA
INCIDÊNCIA EM PROVA: ALTÍSSIMA

Além disso, essas faixas não são por banca – é baseado tanto na quantidade de vezes que caiu em prova independentemente da banca e também em minhas avaliações sobre cada assunto...

#ATENÇÃO

Avisos Importantes

O curso abrange todos os níveis de conhecimento...

Esse curso foi desenvolvido para ser acessível a **alunos com diversos níveis de conhecimento diferentes.** Temos alunos mais avançados que têm conhecimento prévio ou têm facilidade com o assunto. Por outro lado, temos alunos iniciantes, que nunca tiveram contato com a matéria ou até mesmo que têm trauma dessa disciplina. A ideia aqui é tentar atingir ambos os públicos - iniciantes e avançados - da melhor maneira possível..

Por que estou enfatizando isso?

O **material completo** é composto de muitas histórias, exemplos, metáforas, piadas, memes, questões, desafios, esquemas, diagramas, imagens, entre outros. Já o **material simplificado** possui exatamente o mesmo núcleo do material completo, mas ele é menor e bem mais objetivo. *Professor, eu devo estudar por qual material?* Se você quiser se aprofundar nos assuntos ou tem dificuldade com a matéria, necessitando de um material mais passo-a-passo, utilize o material completo. Se você não quer se aprofundar nos assuntos ou tem facilidade com a matéria, necessitando de um material mais direto ao ponto, utilize o material simplificado.

Por fim...

O curso contém diversas questões espalhadas em meio à teoria. Essas questões possuem um comentário mais simplificado porque **têm o único objetivo de apresentar ao aluno como bancas de concurso cobram o assunto previamente administrado**. A imensa maioria das questões para que o aluno avalie seus conhecimentos sobre a matéria estão dispostas ao final da aula na lista de exercícios e **possuem comentários bem mais completos, abrangentes e direcionados.**

ENGENHARIA DE SOFTWARE

1 – Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

Vamos começar falando um pouquinho sobre Engenharia de Software! *Em primeiro lugar, o que é um software?* Bem, em uma visão restritiva, muitas pessoas costumam associar o termo software aos programas de computador. **No entanto, software não é apenas o programa, mas também todos os dados de documentação e configuração associados, necessários para que o programa opere corretamente.**

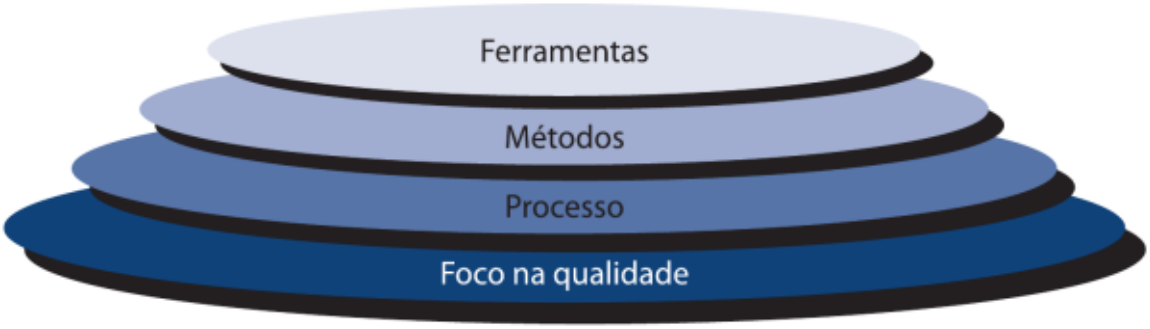
E a Engenharia de Software? **A IEEE define engenharia de software como a aplicação de uma abordagem sistemática, disciplinada e quantificável de desenvolvimento, operação e manutenção de software.** Já Friedrich Bauer conceitua como a criação e a utilização de sólidos princípios de engenharia a fim de obter software de maneira econômica, que seja confiável e que trabalhe em máquinas reais.

Em suma, trata-se de uma disciplina de engenharia que se ocupa de todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até a manutenção desse sistema, após sua entrada em produção **A meta principal da Engenharia de Software é desenvolver sistemas de software com boa relação custo-benefício.** Conceito bem simples, vamos prosseguir...

De acordo com Roger Pressman: "*A Engenharia de Software ocorre como consequência de um processo chamado Engenharia de Sistemas. Em vez de se concentrar somente no software, a engenharia de sistemas focaliza diversos elementos, analisando, projetando, e os organizando em um sistema que pode ser um produto, um serviço ou uma tecnologia para transformação da informação ou controle*".

Além disso, nosso renomadíssimo autor afirma que a engenharia de sistemas está preocupada com todos os aspectos do desenvolvimento de sistemas computacionais, **incluindo engenharia de hardware, engenharia de software e engenharia de processos.** Percebam, então, que a Engenharia de Sistemas está em um contexto maior – junto com várias outras engenharias. *Entenderam direitinho?*

A Engenharia de Software tem por objetivo a aplicação de teorias, modelos, formalismos, técnicas e ferramentas da ciência da computação e áreas afins para um desenvolvimento sistemático de software. Logo, associado ao desenvolvimento, é preciso também aplicar processos, métodos e ferramentas**, sendo que a pedra fundamental que sustenta a engenharia de software é o foco na qualidade conforme apresenta a imagem anterior.**



Tudo isso envolve planejamento de custos e prazos, montagem da equipe e garantia de qualidade do produto e do processo. Finalmente, a engenharia de software visa a produção da documentação formal do produto, do processo, dos critérios de qualidade e dos manuais de usuários finais. **Todos esses aspectos devem ser levados em consideração**.

Aliás, nosso outro renomadíssimo autor – Ian Sommerville – afirma em seu livro que: "*A engenharia de software não está relacionada apenas com os processos técnicos de desenvolvimento de software, mas também com atividades como o gerenciamento de projeto de software e o desenvolvimento de ferramentas, métodos e teorias que apoiem a produção de software*". Vamos ver como isso cai em prova...

A Engenharia de Software surgiu em meados da década de sessenta como uma tentativa de contornar a crise do software e dar um tratamento de engenharia ao desenvolvimento de software completo. Naquela época, o processo de desenvolvimento de software era completamente fora de controle, gerenciamento, monitoração e tinha grandes dificuldades em entregar o que era requisitado pelo cliente.

Já na década de oitenta, surgiu a Análise Estruturada e algumas Ferramentas CASE que permitiam automatizar algumas tarefas. Na década de noventa, surgiu a orientação a objetos, linguagens visuais, processo unificado, entre outros conceitos diversos. E na última década, surgiram as metodologias ágeis e outros paradigmas de desenvolvimento muito comuns hoje em dia no contexto de desenvolvimento de software.

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. *Professor, como a formalidade pode reduzir inconsistências?*

- **Cenário 1:** Estamos na fase de testes de software. O testador afirma que fez todos os testes, você - líder de projeto – acredita nele sem pestanejar, e passa o software ao cliente homologar se está tudo certo. **O cliente tenta utilizar o software e verifica que várias partes estão com erros grosseiros de funcionamento.** *Viram como a ausência de uma formalidade pode gerar inconsistências?*

- **Cenário 2:** Estamos na fase de testes de software. **O testador afirma que fez todos os testes e entrega um documento de testes com tudo que foi verificado no sistema.** Você - líder de projeto - lê o documento de testes e verifica que não foram feitos testes de carga e testes de segurança. Retorna para o testador e pede para ele refazer os testes. Feito isso, ele passa o software ao cliente, que fica feliz e satisfeito porque está tudo funcionando corretamente.

Vocês percebem que essas formalidades evitam aquele "telefone-sem-fio" que é relativamente comum? **Quanto mais eu seguir o processo, o passo-a-passo, tal qual foi devidamente definido por diversas pessoas ao longo do tempo a partir de suas experiências (de falhas e de sucesso) com vários projetos, maior a minha chance de obter êxito na construção do meu software.** *Bacana?* Vamos ver um resuminho...

A Engenharia de Software é uma disciplina dentro do contexto de Engenharia de Sistemas que se ocupa de todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até a manutenção desse sistema após a sua entrada em produção (apesar de o nome indicar o contrário, em produção significa que o software já está disponível no ambiente real de utilização do cliente), **passando por aspectos humanos, hardware, etc.**

Além disso, ela é composta de cinco princípios fundamentais: decomposição, generalização, flexibilização, formalidade e abstração. Por fim, é importante notar que ela se baseia em um conjunto de ferramentas, métodos e processos – sendo que a pedra fundamental que sustenta a engenharia de software é o foco na qualidade. Isso é apenas uma contextualização inicial – esse não é um assunto que cai com frequência em prova. *Fechou?* Vamos seguir então...

2 – Processos de Desenvolvimento

INCIDÊNCIA EM PROVA: BAIXA

Vamos começar falando sobre ciclo de vida! Esse termo surgiu lá no contexto da biologia. **Ele trata do ciclo de vida de um ser vivo, ou seja, trata do conjunto de transformações pelas quais passam os indivíduos de uma espécie durante toda sua vida**. *Vocês se lembram lá no ensino fundamental quando a gente aprende que um ser vivo nasce, cresce, reproduz, envelhece e morre?*

Vejam essa imagem: nós temos um bebê, que depois se torna uma criança, que depois um adolescente, que depois um jovem, que depois um adulto, que depois um velho, depois um ancião e depois morre! **Logo, o ciclo de vida de um ser vivo trata das fases pelas quais um ser vivo passa desde seu nascimento até a sua morte.** E esse conceito pode ser aplicado a diversos outros contextos.

Exemplos: ciclo de vida de uma organização, ciclo de vida de sistemas, ciclo de vida de produtos, ciclo de vida de projetos, ciclo de vida de planetas, entre vários outros. *E como esse conceito se dá em outros contextos?* Exatamente da mesma forma! Logo, o ciclo de vida de um produto trata da história completa de um produto através de suas fases (Ex: Introdução, Crescimento, Maturidade e Declínio).

Existe também o ciclo de vida de um projeto, que trata do conjunto de fases que compõem um projeto (Ex: Iniciação, Planejamento, Execução, Controle e Encerramento). *O que todos esses ciclos de vida têm em comum?* **Eles sempre tratam das fases pelas quais algo passa desde o seu início até o seu fim.** Então, é isso que vocês têm que decorar para qualquer contexto de ciclo de vida: fases pelas quais algo passa do seu início ao seu fim.

Na Engenharia de Software, esse termo é geralmente aplicado a sistemas de software com o significado de mudanças que acontecem na vida de um produto de software. O ciclo de vida trata das fases identificadas entre o nascimento e a morte de um software. *Vocês viram?* É exatamente a mesma coisa – são as fases ou atividades pelas quais passa um software durante o decorrer da sua vida.

Uma definição interessante de ciclo de vida de software afirma que se trata das fases de um produto de software que vão desde quando ele é concebido inicialmente até quando ele não está mais disponível para uso. **Já Steve McConnell afirma que um modelo de ciclo de vida de software é uma representação que descreve todas as atividades que tratam da criação de um produto de software.**

Dessa forma, nós podemos concluir que um ciclo de vida de software se **refere às fases pelas quais um sistema de software atravessa desde sua concepção até sua retirada de produção**. Bem, entender esse conceito não é o problema, esse é um conceito muito simples. *Concordam comigo? E*

qual é o problema, professor? O problema é que existem dezenas de fases diferentes para os ciclos de vida de acordo com cada autor.

Como assim, professor? Infelizmente, não há um consenso entre os autores. **Cada um que lança um livro decide criar o ciclo de vida com as fases que ele acha mais corretas e isso acaba prejudicando nosso estudo.** Na UnB, eu tive um professor de Engenharia de Software chamado Fernando Albuquerque que já tinha escrito vários livros sobre esse assunto e ele tinha o seu próprio ciclo de vida com suas respectivas fases.

Agora imaginem a quantidade de autores de livros de Engenharia de Software lançados nas últimas três ou quatro décadas. Além disso, acontece de as vezes o próprio autor mudar seu entendimento sobre as fases. Se vocês olharem a quarta edição do livro do Roger Pressman, ele tem um conjunto de fases; se vocês olharem da sexta edição para frente, ele define um novo conjunto de fases. *O que aconteceu?* Ele mudou de ideia!

Então, eu já vi vários ciclos de vida diferentes em provas! *Qual a solução para esse problema, professor?* Galera, vocês sempre têm que pensar no custo/benefício. *Vale a pena decorar todos?* Na minha opinião, nem um pouco! *Por que?* Porque isso não é algo que cai muito em prova – principalmente nas provas recentes. **Guardem o espaço que vocês têm sobrando na cabeça para memorizar coisas que realmente caem.**

Sendo assim, percebam que há autores que descrevem as fases do ciclo de vida de software de maneira mais ampla e abstrata (com poucas e grandes fases) e autores que o fazem de maneira mais diminuta e detalhada (com muitas e pequenas fases). **Vamos começar a ver alguns ciclos de vida que existem por aí para que vocês visualizem isso com maior clareza!**

Vamos lá! Do lado esquerdo, temos um ciclo de vida de software só com quatro fases bem genéricas: Definição do Software, Desenvolvimento do Software, Operação do Software e Retirada

do Software. Eu só vi cair esse ciclo de vida uma vez, mas ele é útil para que vocês visualizem um ciclo de vida com poucas fases. **Observem que ele trata desde o início até a aposentadoria ou retirada do software.**

Do lado direito, eu coloquei um ciclo de vida um pouco mais detalhado. Vejam que ele destrincha mais as fases de desenvolvimento, separando em análise, projeto, implementação, testes, etc. *Por que eu coloquei esse ciclo de vida?* **Porque alguns autores afirmam que, em tese, todo ciclo de vida deveria contemplar todas essas atividades ou fases.** Relaxa, nós vamos ver isso com mais detalhes depois...

Outros exemplos: em verde, nós temos um ciclo de vida de software de acordo com nosso querido autor: Ian Sommerville. Ele contém quatro fases: Especificação, Desenvolvimento, Validação e Evolução. Sendo que, atenção aqui, o desenvolvimento pode ser dividido em Projeto e Implementação. *Tudo certo?* **Esse é um ciclo de vida de software que já cai com uma maior frequência (nada excepcional, só cai mais que os outros).**

Por fim, em laranja, nós temos um ciclo de vida de software de acordo com nosso outro querido autor: Roger Pressman. **Ele contém cinco fases: comunicação, planejamento, modelagem, construção e implantação**. Esse não cai tanto, mas é importante saber também. *Tudo bem até aqui?* Então vejam que não é tão complicado! Esses são os quatro ciclos de vida de software mais comuns em prova... e são raros!

Então, vamos recapitular: inicialmente, nós vimos o conceito de um Ciclo de Vida! Depois nós vimos o que é um Ciclo de Vida de Software. **E, agora, nós vamos ver um terceiro conceito, que é o conceito de Modelo de Ciclo de Vida de Software.** *Por que, professor?* Porque Ciclo de Vida de Software é diferente de Modelo de Ciclo de Vida de Software. *E o que é um modelo de ciclo de vida de software?*

Bem, um modelo de ciclo de vida de software é um modelo que apresenta não só as fases do ciclo de vida do software, mas também a forma como essas fases se relacionam. *Diego, não entendi isso muito bem!* Galera, um modelo contém as fases que nós acabamos de ver, mas ele também nos diz como essas fases se relacionam! *Vocês querem um exemplo?* Existe um modelo de ciclo de vida chamado Modelo em Cascata.

Esse modelo foi pioneiro – ele foi o primeiro modelo a aparecer na Engenharia de Software e nós vamos vê-lo em mais detalhes em outro momento. **O importante sobre o Modelo em Cascata é a forma como as fases se relacionam.** Nesse modelo, nós temos uma regra de ouro: uma fase somente se inicia após o término completo da fase anterior (exceto a primeira, evidentemente) – não é possível fazer fases paralelamente. *Viram como as fases se relaciona?*

Dessa forma, um Modelo de Ciclo de Vida de Software contém suas respectivas fases, **mas também contém como essas fases se relacionam.** Vejam os conceitos abaixo:

E o que seria um processo de software? **Então, um processo de software pode ser visto como o conjunto de atividades, métodos, práticas e transformações que guiam pessoas na produção de software.** Como o Modelo de Ciclo de Vida nos diz como as fases do ciclo de vida se relacionam, nós podemos – em termos de prova – considerar Modelo de Ciclo de Vida como sinônimo de Modelo de Processo! (Aliás, pessoal... infelizmente muitas questões ignoram isso!).

PROCESSOS DE SOFTWARE

- CONJUNTO DE ATIVIDADES, MÉTODOS, PRÁTICAS E TRANSFORMAÇÕES QUE GUIAM PESSOAS NA PRODUÇÃO DE SOFTWARE -

MODELO DE PROCESSO DE SOFTWARE

- MESMO CONCEITO DE MODELO DE CICLO DE VIDA - É UMA REPRESENTAÇÃO ABSTRATA DE UM PROCESSO DE SOFTWARE -

Um processo de software não pode ser definido de forma universal. **Para ser eficaz e conduzir à construção de produtos de boa qualidade, um processo deve ser adequado às especificidades do projeto em questão.** Deste modo, processos devem ser definidos caso a caso, considerando-se diversos aspectos específicos do projeto em questão, tais como:

#1. Características da aplicação (domínio do problema, tamanho do software, tipo e complexidade, entre outros);

#2. Tecnologia a ser adotada na sua construção (paradigma de desenvolvimento, linguagem de programação, mecanismo de persistência, entre outros);

#3. Organização onde o produto será desenvolvido e a equipe de desenvolvimento alocada (recursos humanos).

Há vários aspectos a serem considerados na definição de um processo de software. No centro da arquitetura de um processo de desenvolvimento estão as atividades-chave desse processo: análise e especificação de requisitos, projeto, implementação e testes, que são a base sobre a qual o processo de desenvolvimento deve ser construído. *Entendido?*

No entanto, **a definição de um processo envolve a escolha de um modelo de ciclo de vida (ou modelo de processo)**, o detalhamento (decomposição) de suas macro-atividades, a escolha de métodos, técnicas e roteiros (procedimentos) para a sua realização e a definição de recursos e artefatos necessários e produzidos – é bastante coisa!

Podemos dizer que se trata da representação abstrata de um esqueleto de processo, incluindo tipicamente algumas atividades principais, a ordem de precedência entre elas e, opcionalmente, artefatos requeridos e produzidos. **Em geral, um modelo de processo descreve uma filosofia de organização de atividades, estruturando-as em fases e definindo como essas fases estão relacionadas.**

A escolha de um modelo de ciclo de vida (para concursos, sinônimo de modelo de processo) é o ponto de partida para a definição de um processo de desenvolvimento de software. **Um modelo de**

ciclo de vida, geralmente, organiza as macro-atividades básicas do processo, estabelecendo precedência e dependência entre as mesmas. *Tudo certo?*

Conforme eu disse anteriormente, alguns autores afirmam que os modelos de ciclo de vida básicos, de maneira geral, contemplam pelo menos as fases de: **Planejamento; Análise e Especificação de Requisitos; Projeto; Implementação; Testes; Entrega e Implantação; Operação; e Manutenção.** Abaixo eu trago uma descrição genérica sobre cada uma dessas fases.

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.
ENTRE GA E IMPLA	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.

Existem outras fases em outros modelos, tais como: análise, responsável por modelar o problema (diferente do projeto, responsável por modelar a solução do problema); homologação, responsável pela aceitação pela parte interessada do produto; gerência de configuração, responsável pela estruturação sistemática dos produtos, artefatos, documentos, modelos, entre outros. *Bacana?*

Sommerville afirma que um processo de software é um conjunto de atividades e resultados associados que produz um produto de software. De acordo com ele, existem quatro atividades fundamentais de processo, que são comuns a todos os processos de software – são elas: **Especificação de Software; Desenvolvimento de Software (Projeto e Implementação); Validação de Software; e Evolução de Software.**

Também de acordo nosso querido autor, um modelo de processo de software é uma descrição simplificada desse processo de software que apresenta uma visão dele. Os modelos de processo incluem as atividades, que fazem parte do processo de software, e eventualmente os produtos de software e os papéis das pessoas envolvidas na engenharia de software. E não para por aí...

Ele ainda afirma que a maioria dos processos de software é baseada em três modelos gerais: modelo em cascata; desenvolvimento iterativo e engenharia de software baseada em componentes. Isso entra em contradição com o que dizem outros autores, isto é, **os principais modelos podem ser agrupados em três categorias: modelos sequenciais, modelos incrementais e modelos evolutivo.**

Por fim, existe mais um conceito importante nessa aula! É o conceito de Metodologia de Desenvolvimento de Software (também chamada de Processo de Desenvolvimento de Software). *O que é isso, professor?* **É basicamente uma caracterização prescritiva ou descritiva de como um produto de software deve ser desenvolvido**, isto é, ela define o quê, como e quando fazer algo para desenvolver um software. Calma, tudo ainda fará sentido...

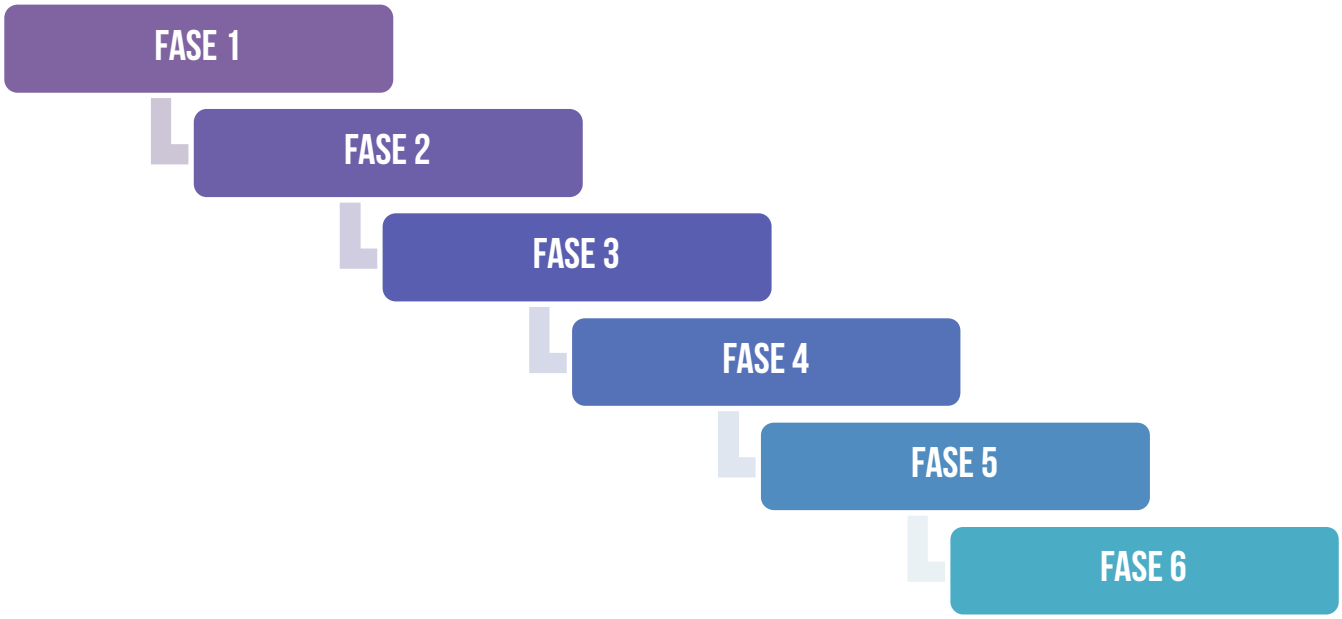
2.1 – Modelos Sequenciais

2.1.1 – Modelo em Cascata

INCIDÊNCIA EM PROVA: ALTÍSSIMA

Citado inicialmente em 1970 por W. Royce, é também denominado Modelo em Cascata, Clássico, Sequencial, Linear, Tradicional, Waterfall, Rígido, Top-Down ou Monolítico (todos esses nomes já caíram em prova!). Esse nome é devido ao encadeamento simples de uma fase com a outra. **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.**

Como assim, Diego? Por exemplo: a Fase 4 só pode ser iniciada após o término e aprovação da Fase 3; a Fase 5 só pode ser iniciada após o término e aprovação da Fase 4; e assim por diante!



Mas que fases são essas, Diego? Bem, agora complica um pouco porque cada autor resolve criar suas próprias fases! Vejam só na tabelinha a seguir:

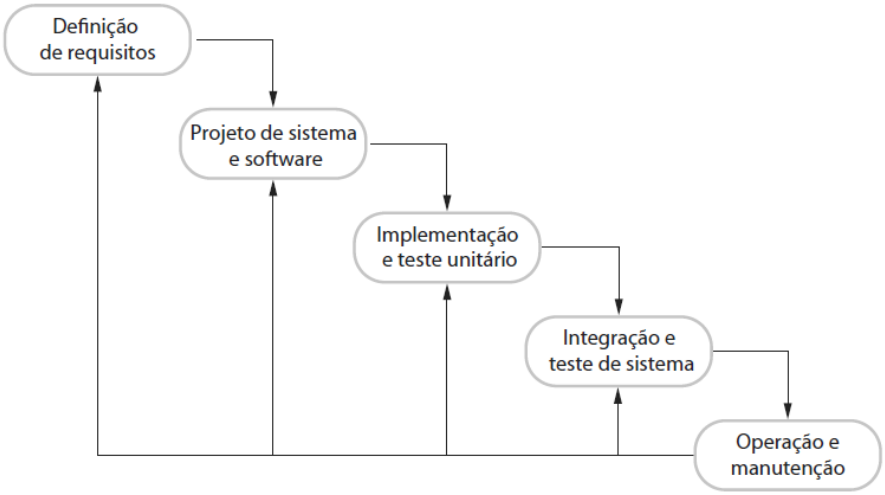
POR SOMMERVILLE	POR ROYCE	POR PRESSMAN (4ª ED)	POR PRESSMAN (6ª ED)
Análise e Definição de Requisitos	Requisitos de Sistema	Modelagem e Engenharia do Sistema/Informação	Comunicação
Projeto de Sistema e Software	Requisitos de Software	Análise de Requisitos de Software	Planejamento
Implementação e Teste de Unidade	Análise	Projeto	Modelagem
Integração e Teste de Sistema	Projeto	Geração de Código	Construção
Operação e Manutenção	Codificação	Teste e Manutenção	Implantação

	Teste		
	Operação		

Percebam que há grandes diferenças entre os autores! Inclusive, há divergências até entre autor e ele mesmo, dependendo da versão do livro (Exemplo: Pressman mudou as fases na última edição de seu livro). *Professor, você já viu isso cair em prova?* Sim, já vi! *E o que aconteceu?* Bem, polêmica, recursos, etc – não há o que fazer! Enfim... a minha classificação preferida é a do Royce por conter as fases que eu acho que realmente ocorrem no desenvolvimento de um software.

De toda forma, é interessante saber a visão dos dois autores mais consagrados. De acordo com Ian Sommerville, o modelo em cascata é um exemplo de um processo dirigido a planos. *Como assim, Diego?* **Ele quer dizer que, em princípio, você deve planejar e programar todas as atividades do processo antes de começar a trabalhar nelas.** Os principais estágios do modelo em cascata refletem diretamente as atividades fundamentais do desenvolvimento:

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.



Ainda de acordo com Ian Sommerville, em princípio, o resultado de cada estágio é a aprovação de um ou mais documentos (assinados). O estágio seguinte não deve ser iniciado até que a fase anterior seja concluída. Na prática, esses estágios se sobrepõem e alimentam uns aos outros de informações. Durante o projeto, os problemas com os requisitos são identificados; durante a codificação, problemas de projeto são encontrados e assim por diante.

Logo, na prática, o processo de software não é um modelo linear simples, mas envolve o feedback de uma fase para outra. **Dessa forma, os documentos produzidos em cada fase podem ser modificados para refletirem as alterações feitas em cada um deles.** Eventualmente, por causa dos custos de produção e aprovação de cada um dos documentos, as iterações podem ser dispendiosas e envolver um significativo retrabalho.

Após um pequeno número de iterações, é normal se congelarem partes do desenvolvimento, como a especificação, e dar-se continuidade aos estágios posteriores de desenvolvimento. A solução dos problemas fica para mais tarde, ignorada ou programada, quando possível. Esse congelamento prematuro pode significar que o sistema não fará o que o usuário quer e pode levar a sistemas mal estruturados, quando os problemas de projeto são contornados por artifícios de implementação.

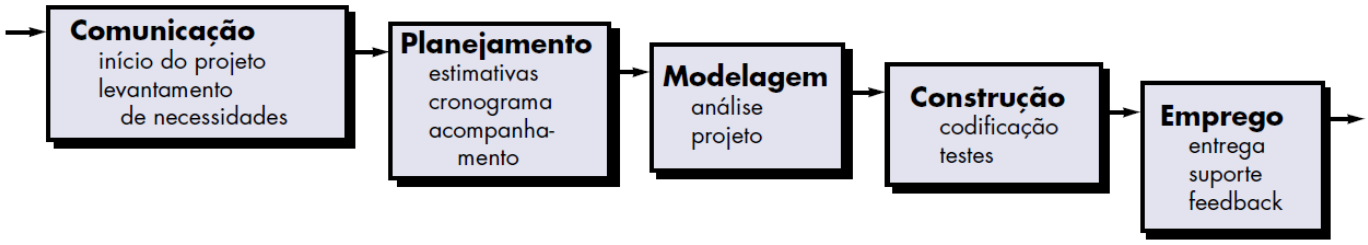
Durante o estágio final do ciclo de vida (operação e manutenção), o software é colocado em uso. Erros e omissões nos requisitos originais do software são descobertos. Os erros de programa e projeto aparecem e são identificadas novas necessidades funcionais. O sistema deve evoluir para permanecer útil. Fazer essas alterações (manutenção do software) pode implicar repetição de estágios anteriores do processo.

O modelo em cascata é consistente com outros modelos de processos de engenharia, e a documentação é produzida em cada fase do ciclo. Assim, o processo torna-se visível, e os gerentes podem monitorar o progresso de acordo com o plano de desenvolvimento. **Seu maior problema é a divisão inflexível do projeto em estágios distintos.** Os compromissos devem ser assumidos em um estágio inicial do processo, o que dificulta que atendam às mudanças de requisitos dos clientes.

Em princípio, o modelo em cascata deve ser usado apenas quando os requisitos são bem compreendidos e pouco provavelmente venham a ser radicalmente alterados durante o desenvolvimento do sistema. No entanto, ele reflete o tipo de processo usado em outros projetos de engenharia. Como é mais fácil usar um modelo de gerenciamento comum para todo o projeto, processos de software baseados no modelo em cascata ainda são comumente utilizados.

Já de acordo com Roger Pressman, há casos em que os requisitos de um problema são bem compreendidos – quando o trabalho flui da comunicação ao emprego de forma relativamente linear. Essa situação ocorre algumas vezes quando adaptações ou aperfeiçoamentos bem definidos precisam ser feitos em um sistema existente. Por exemplo: uma adaptação em software contábil exigida devido a mudanças nas normas governamentais.

Pode ocorrer também em um número limitado de novos esforços de desenvolvimento, mas apenas quando os requisitos estão bem definidos e são razoavelmente estáveis. O modelo em cascata sugere uma abordagem sequencial e sistemática para o desenvolvimento de software, começando com o levantamento de necessidades por parte do cliente, avançando pelas fases de planejamento, modelagem, construção, emprego e culminando no suporte contínuo do software concluído.



O modelo cascata tem sofrido diversas críticas a que fizeram com que até mesmo seus mais ardentes defensores questionassem sua eficácia. Entre os problemas encontrados, temos:

- Projetos reais raramente seguem o fluxo sequencial que o modelo propõe. Embora o modelo linear possa conter iterações, ele o faz indiretamente. Como consequência, mudanças podem provocar confusão à medida que a equipe de projeto prossegue.

- Frequentemente, é difícil para o cliente estabelecer explicitamente todas as necessidades. O modelo cascata requer isso e tem dificuldade para adequar a incerteza natural que existe no início de muitos projetos.

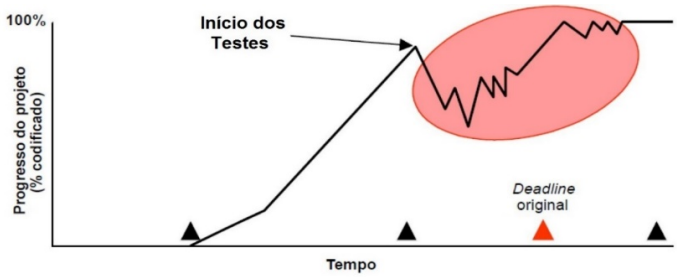
- O cliente deve ter paciência. Uma versão operacional do(s) programa(s) não estará disponível antes de estarmos próximos do final do projeto. Um erro grave, se não detectado até o programa operacional ser revisto, pode ser desastroso.

Em uma interessante análise de projetos reais, descobriu-se que a natureza linear do ciclo de vida clássico conduz a "estados de bloqueio", nos quais alguns membros da equipe do projeto têm de aguardar outros completarem tarefas dependentes. **De fato, o tempo gasto na espera pode**

exceder o tempo gasto em trabalho produtivo! Os estados de bloqueio tendem a prevalecer no início e no final de um processo sequencial linear.

Hoje em dia, o trabalho de software tem um ritmo acelerado e está sujeito a uma cadeia de mudanças intermináveis (em características, funções e conteúdo de informações). O modelo cascata é frequentemente inapropriado para tal trabalho. **Entretanto, ele pode servir como um modelo de processo útil em situações nas quais os requisitos são fixos e o trabalho deve ser realizado até sua finalização de forma linear.**

É importante destacar que **o desenvolvimento não continua até que o cliente esteja satisfeito com os resultados alcançados** e isso engessa o desenvolvimento.



O MODELO EM CASCATA ATRASA A REDUÇÃO DE RISCOS

O Modelo em Cascata tem um grande problema: ele atrasa a redução de riscos! *Como assim, Diego?* Bem, essa é uma desvantagem recorrente em provas! Como uma fase só se inicia após o término e aprovação da fase anterior, **em geral só é possível verificar se ocorreram erros nas fases finais, que é quando o sistema é efetivamente testado** – isso gera grandes riscos! Em outros modelos, os riscos são reduzidos desde as primeiras fases do processo de desenvolvimento.

Percebam que os riscos deveriam ser descobertos logo no início do processo de desenvolvimento, no entanto eles são descobertos somente após o início dos testes e implantação. Vocês podem notar no gráfico acima que, a partir da região vermelha, **o progresso do projeto sobe e desce diversas vezes**, porque provavelmente o sistema está sendo corrigido devido a requisitos modificados. *Descobriu um erro?* Desce! *Corrigiu o erro?* Sobe!

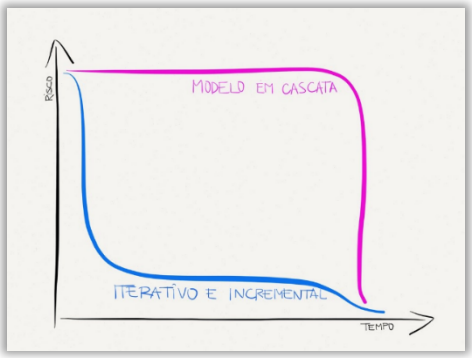
Vejam, também, que o projeto não terminou em seu deadline (prazo de conclusão) original. Como a redução de riscos atrasou, todo andamento do projeto também atrasou. Dessa forma, não se cumpriu nem o prazo do projeto e, provavelmente, nem o orçamento e talvez nem seu escopo – tendo em vista que, quanto mais ao fim do projeto um erro é identificado, mais caras se tornam as modificações.

Entenderam essa parte direitinho? **Um erro na fase de requisitos, por exemplo, que não foi corrigido e foi descoberto no final do processo de desenvolvimento, terá um custo de correção altíssimo**, visto que provavelmente terá que se refazer tudo novamente. *Vocês conhecem a Torre de Pisa?* Ela fica na Itália e é famosa por ser torta. Respondam: *seria mais fácil corrigir a torre logo no início da construção do primeiro andar ou no último?*

Ora, se alguém tivesse notado que a torre estava torta logo no início da construção do primeiro andar, seria possível corrigi-la sem ter tanto trabalho! Agora se somente ao final da construção alguém notasse que a torre estava bastante torta, seria muito mais trabalhoso e caro para corrigir. *Concordam comigo?* A mesma coisa acontece com um software! Se o cliente me pede uma coisa, mas eu entendo outra e somente mostro para ele quando estiver tudo pronto, é evidente que eu estou atrasando a redução de riscos. *Por que?* Porque se eu tivesse mostrado para ele cada passo do desenvolvimento desde o início, ele poderia identificar o erro de forma que eu pudesse corrigi-lo tempestivamente. **Em outras palavras, eu teria adiantado a redução de riscos – eu estaria reduzindo os riscos de fazer algo errado de maneira adianta!** *Sacaram?*

Portanto, não confundam essas duas coisas: **se o erro ocorreu no início e foi identificado no início, terá baixo custo de correção; se o erro ocorreu no início e foi identificado no final, terá alto custo de correção.** Dessa forma, o custo de correção de um erro está mais focado no momento em que um erro é identificado do que no momento em que ele de fato ocorreu. *Vocês entenderam legal? Então, vamos seguir...*

Outra maneira de visualizar o atraso é por meio de um gráfico Risco x Tempo, comparando o modelo em cascata com o Modelo Iterativo e Incremental (que veremos a seguir). **Observem que o Modelo Iterativo e Incremental já começa a reduzir os riscos desde o início do processo de desenvolvimento, em contraste com o Modelo em Cascata que acumula os riscos até a fase de testes, integração e implantação do sistema**. Vejam o gráfico ao lado e tentem entender essa interpretação que acabamos de ver!



Galera, a grande vantagem do Modelo em Cascata é que o desenvolvimento é dividido em fases distintas, padronizadas, entre outras. *Vantagem, professor?* Em relação ao que existia antes, sim! Antigamente, os softwares eram construídos quase que de maneira artesanal. **Ademais, é possível colocar pessoas com perfis específicos para cada fase**, isto é, quem tem facilidade de se comunicar vai ser analista de requisitos, programadores vão fazer a codificação, etc.

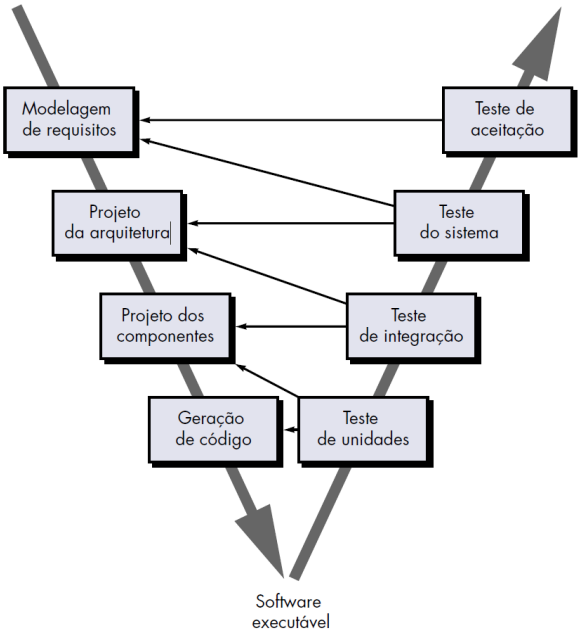
A grande desvantagem é que - em projetos complexos – demora-se muito para chegar até a fase de testes, sendo que o cliente não vê nada rodando até a implantação. **Então, pode acontecer de, nas fases finais, os requisitos da organização não serem mais os mesmos daqueles do início e o software não ter mais utilidade para organização**. Imaginem que vocês contratam um pedreiro para construir a casa de vocês.

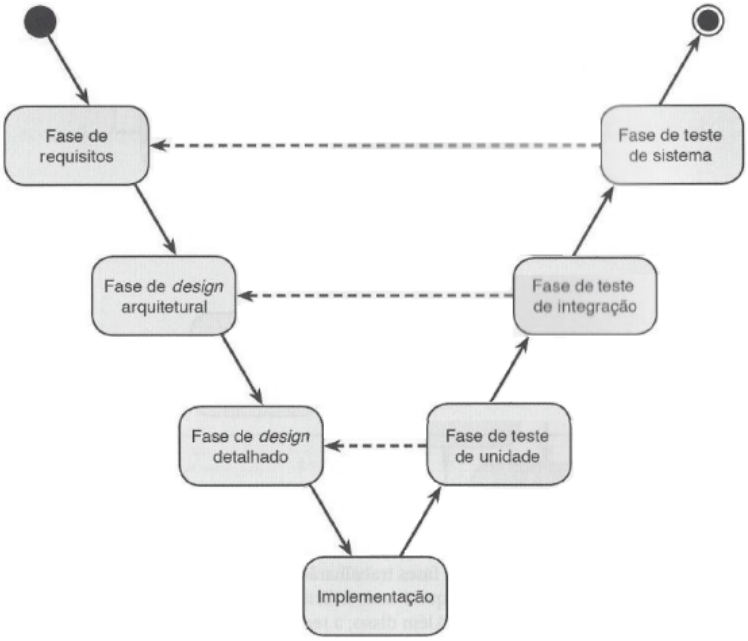
Só que ele não vai te mostrar nada de como a casa está ficando, ele só vai te mostrar quando já estiver tudo pronto daqui um ano. *É interessante?* Não! Primeiro, porque você vai morrer de

ansiedade. Segundo, porque isso atrasará a redução de riscos, aumentando a probabilidade de erros graves. Terceiro, porque demora tanto que durante esse ano você pode ter mudado de ideia – queria uma casa de um andar e mudou de ideia para uma casa de dois andares, por exemplo.

Então o Modelo em Cascata não deve ser usado em nenhuma hipótese? Calma lá, ele pode ser utilizado, sim – apesar de atualmente ser bem raro! No entanto, sua utilização deve ocorrer **preferencialmente quando os requisitos forem bem compreendidos e houver pouca probabilidade de mudanças radicais durante o desenvolvimento do sistema**. *Vocês entenderam?* Então vamos ver agora uma lista com as maiores vantagens e desvantagens.

VANTAGENS	DESVANTAGENS
É simples de entender e fácil de aplicar, facilitando o planejamento.	Divisão inflexível do projeto em estágios distintos.
Fixa pontos específicos para a entrega de artefatos.	Dificuldade em incorporar mudanças de requisitos.
Funciona bem para equipes tecnicamente fracas.	Clientes só visualizam resultados próximos ao final do projeto.
É fácil de gerenciar, devido a sua rigidez.	Atrasa a redução de riscos.
Realiza documentação extensa por cada fase ou estágio.	Apenas a fase final produz um artefato de software entregável.
Possibilita boa aderência a outros modelos de processo.	Cliente deve saber todos os requisitos no início do projeto.
Funciona bem com projetos pequenos e com requisitos bem conhecidos.	Modelo inicial (Royce) não permitia feedback entre as fases do projeto.
	Pressupõe que os requisitos ficarão estáveis ao longo do tempo.
	Não funciona bem com projetos complexos e OO, apesar de compatível.

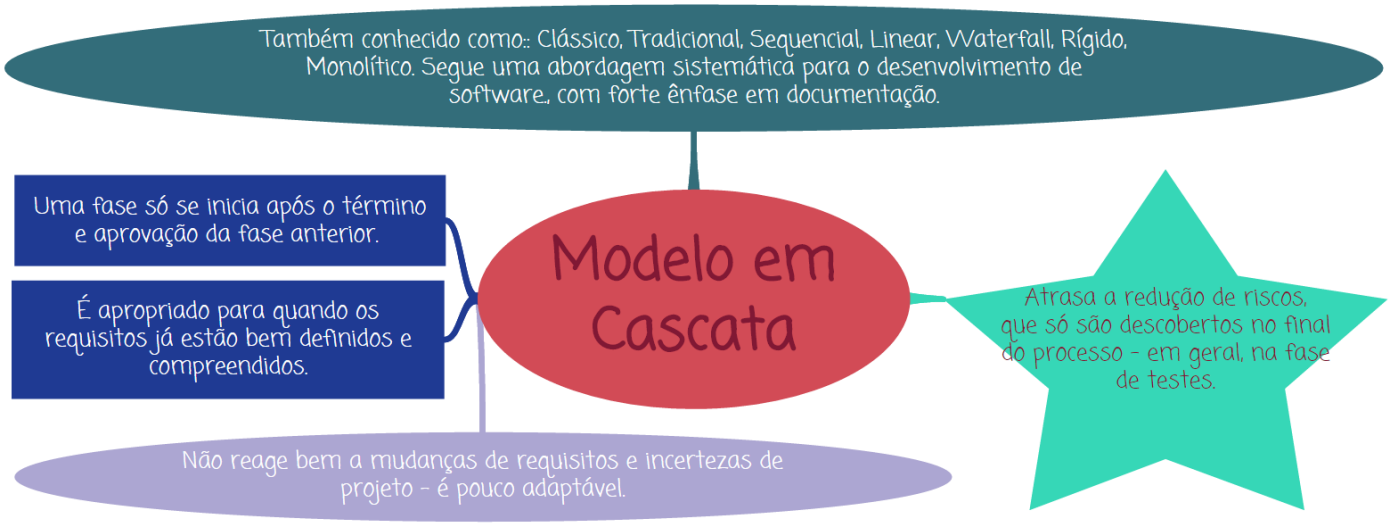




Para finalizar, podemos afirmar que o Modelo em Cascata é considerado um método tradicional e fortemente prescritivo, isto é, ele busca sempre dizer o que fazer, em geral, por meio de planos definidos no início do projeto. *Que planos, professor?* Escopo, custo, cronograma... tudo isso bem detalhado em uma extensa documentação. *Mudanças são bem-vindas?* Claro que não! Mudanças são bem-vindas no modelo do nosso próximo assunto...

Por fim, Roger Pressman trata de uma variação na representação do Modelo em Cascata chamado Modelo em V, que descreve a relação entre ações de garantia da qualidade e as ações associadas à comunicação, modelagem e atividades de construção iniciais. À medida que a equipe de software desce em direção ao lado esquerdo do V, os requisitos básicos do problema são refinados em representações cada vez mais detalhadas e técnicas do problema e de sua solução.

Uma vez que o código tenha sido gerado, a equipe se desloca para cima, no lado direito do V, realizando basicamente uma série de testes (ações de garantia da qualidade) que validem cada um dos modelos criados à medida que a equipe se desloca para baixo, no lado esquerdo do V. Lembrando que os testes são executados do lado direito do Modelo em V, mas são planejados do lado esquerdo do Modelo em V.



2.2 – Modelo Iterativo e Incremental

2.2.1 – Conceitos Básicos

INCIDÊNCIA EM PROVA: ALTÍSSIMA

Como foi dito anteriormente, o Modelo em Cascata acumulava riscos e vários projetos começaram a fracassar um atrás do outro ao utilizá-lo no mundo inteiro. *Diego, o que você quer dizer com fracassar?* O projeto não era entregue, ou era entregue de forma completamente errada, ou era entregue faltando partes, ou era entregue no dobro do prazo combinado, ou era entregue com o dobro do custo acordado, enfim... diversos motivos!

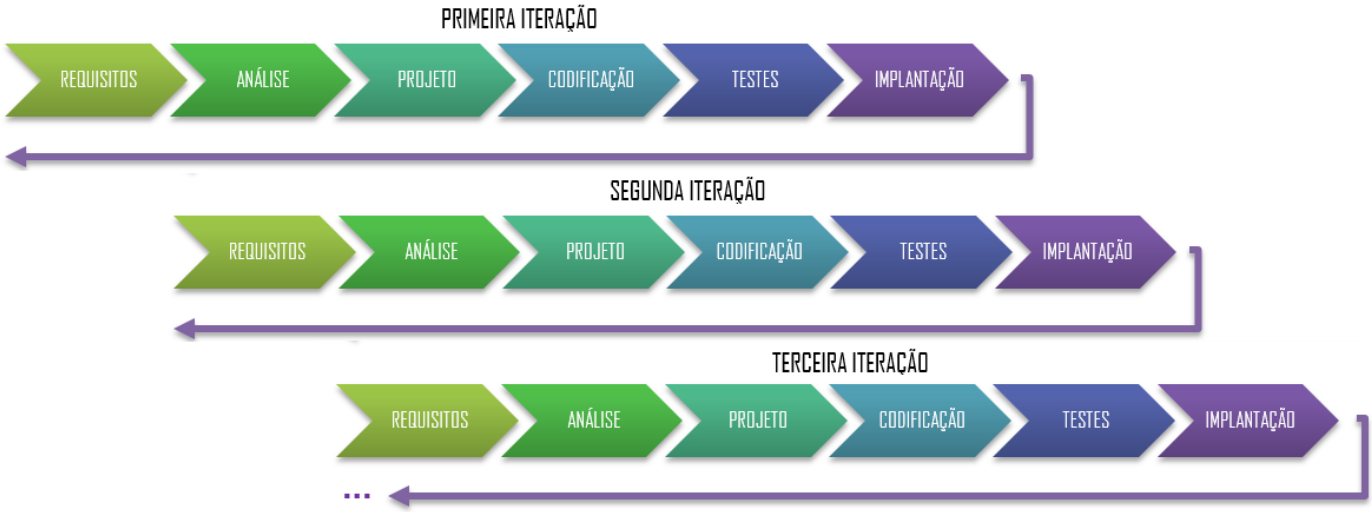
Foi quando surgiu o Modelo Iterativo e Incremental como uma tentativa de resolver esse problema de acúmulo de riscos. Vejamos as diferenças fundamentais:

- **No Modelo em Cascata**, caso haja cem requisitos, analisam-se os cem requisitos, projetam-se os cem requisitos, codificam-se os cem requisitos, testam-se os cem requisitos, e assim por diante sequencialmente;

- **No Modelo Incremental**, caso haja cem requisitos, dividem-se os cem requisitos em vinte miniprojetos de cinco requisitos cada e utiliza-se o modelo em cascata para cada miniprojeto;

- **No Modelo Iterativo**, caso haja cem requisitos, analisam-se, projetam-se, codificam-se, testam-se os cem requisitos, porém os requisitos são entregues incompletos e eu repito esse ciclo de refinamento até chegar ao produto final.

Sim, existe a abordagem incremental e a abordagem iterativa. **No entanto, na prática elas virão sempre de forma combinada no modelo iterativo e incremental para desenvolver os miniprojetos e entregar partes diferentes do projeto**. A imagem a seguir apresenta miniprojetos sendo feitos iterativamente em um modelo iterativo e incremental. Observem que cada iteração (passagem pelas fases de desenvolvimento) refinam cada vez mais a funcionalidade.



Dessa forma, os resultados são mais rápidos, há maior interação com o usuário e há um feedback mais intenso entre usuário e desenvolvedor – sendo possível reagir mais facilmente a mudanças. **Essa abordagem permite gerenciamento e mitigação de riscos**. *Professor, mas eu fiquei com uma dúvida: qual é a diferença entre a abordagem iterativa e abordagem incremental? Ou eles são exatamente a mesma coisa?*

Bem, galera... eu nunca vi até hoje em mais de dez anos no mundo de concursos nenhuma prova cobrar essa diferença entre modelo iterativo e modelo incremental! Como eu disse, quando se fala em modelo iterativo, já se presume que é incremental; e quando se fala em modelo

incremental, já se presume que é iterativo. Eles frequentemente andam lado a lado, mas há pequenas diferenças.

> Cuidado com a grafia das palavras iterativo e interativo! Eu já as vi utilizadas de forma invertida dezenas de vezes em provas e até em editais. Por vezes, bancas não acatam os recursos contra isso! De todo modo, saibam que: iterativo = reiterado ou repetitivo; interativo = participação ou ação mútua.

E se cair essa diferença em prova? Ora, caso caia em prova, a diferença é que, **no modelo incremental**, há várias equipes desenvolvendo uma parte do software a serem integradas ao final do desenvolvimento. Já **no modelo iterativo**, lança-se a versão 1.0, adicionam-se algumas funcionalidades; lança uma versão 2.0, adicionam-se mais algumas funcionalidades; e assim por diante. Vejamos isso mais claramente utilizando o clássico exemplo da Monalisa...

Modelo Incremental: observem que a imagem mostra um artista com uma ideia completa já sobre o quadro em sua cabeça, mas ele desenvolve cada parte do quadro separadamente até integrar as partes em uma imagem completa. É como se fosse um quebra-cabeças em que cada parte é entregue funcionando e depois integrada. **Ele produz *builds*, isto é, partes do software**.

Modelo Iterativo: observem que a imagem mostra um artista com um esboço do quadro, sendo que ele desenvolve várias versões até chegar ao resultado final que ele deseja. É como se fosse uma visão abstrata da imagem, que em seguida vai sendo melhorada até chegar a uma visão mais concreta. **Ele produz *releases*, isto é, versões constantemente melhoradas do software**.

Uma das vantagens do modelo iterativo e incremental é que o cliente pode receber e avaliar as entregas do produto mais cedo, já no início do desenvolvimento do software. Além disso, há maior tolerância a mudanças com consequência direta de redução do risco de falha do projeto, isto é, ele acomoda melhor mudanças – aumentando o reúso e a qualidade. *Lembram do exemplo do pedreiro e da casa?* Vale o mesmo aqui...

E como é a abordagem dos nossos autores consagrados? **Ian Sommerville afirma que o desenvolvimento incremental é baseado na ideia de desenvolver uma implementação inicial, expô-la aos comentários dos usuários e continuar por meio da criação de várias versões até que um sistema adequado seja desenvolvido.** Atividades de especificação, desenvolvimento e validação são intercaladas, e não separadas, com rápido feedback entre todas as atividades.

Desenvolvimento incremental é uma parte fundamental de abordagens ágeis – é melhor que a abordagem em cascata para a maioria dos sistemas de negócios, e-commerce e sistemas pessoais. **Desenvolvimento incremental reflete a maneira como resolvemos os problemas.** Raramente elaboramos uma completa solução do problema com antecedência; geralmente movemo-nos passo a passo em direção a uma solução, recuando quando percebemos que cometemos um erro.

Ao desenvolver um software de forma incremental, é mais barato e mais fácil fazer mudanças no software durante seu desenvolvimento. **Cada incremento/versão do sistema incorpora alguma funcionalidade necessária para o cliente.** Em geral, incrementos iniciais incluem a funcionalidade mais importante ou mais urgente. Isso significa que o cliente pode avaliar o sistema em um estágio relativamente inicial do desenvolvimento para ver se ele oferece o que foi requisitado.

Em caso negativo, só o incremento que estiver em desenvolvimento no momento precisará ser alterado e, possivelmente, nova funcionalidade deverá ser definida para incrementos posteriores. O desenvolvimento incremental tem três vantagens importantes quando comparado ao modelo em cascata: (1) o custo de acomodar as mudanças nos requisitos do cliente é reduzido. A quantidade de análise e documentação a ser refeita é muito menor do que o necessário no modelo em cascata;

(2) É mais fácil obter feedback dos clientes sobre o desenvolvimento que foi feito. Os clientes podem fazer comentários sobre as demonstrações do software e ver o quanto foi implementado. Os clientes têm dificuldade em avaliar a evolução por meio de documentos de projeto de software. (3) É possível obter entrega e implementação rápida de um software útil ao cliente, mesmo se toda a funcionalidade não for incluída – o que é melhor do que seria com um processo em cascata.

O desenvolvimento incremental, atualmente, é a abordagem mais comum para o desenvolvimento de sistemas. Ele pode ser tanto dirigido a planos, ágil, ou, o mais comum, uma mescla dessas abordagens. Em uma abordagem dirigida a planos, os incrementos do sistema são identificados previamente; se uma abordagem ágil for adotada, os incrementos iniciais são identificados, mas o desenvolvimento de incrementos posteriores depende do progresso e das prioridades dos clientes.

Bem, sabemos que a abordagem evolucionária é mais eficaz que abordagem em cascata. *Por que?* Porque a especificação pode ser desenvolvida de forma incremental e, não, integralmente

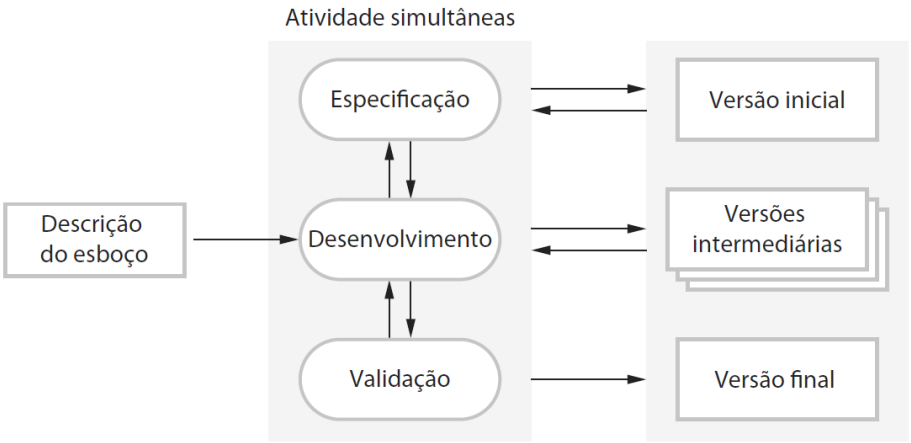
como naquele modelo. À medida que os usuários compreendem melhor seu problema, isso pode ser refletido no sistema de software. No entanto, essa abordagem possui dois problemas (que, inclusive, já caíram no Senado Federal):

a) O processo não é visível: se os sistemas são desenvolvidos rapidamente, não é viável economicamente produzir documentos para cada versão do sistema.

b) Os sistemas são frequentemente mal estruturados: mudanças contínuas tendem a corromper a estrutura do software e tornar mudanças difíceis e caras.

Os problemas do desenvolvimento incremental são particularmente críticos para os sistemas de vida-longa, grandes e complexos, nos quais várias equipes desenvolvem diferentes partes do sistema. Sistemas de grande porte necessitam de um framework/arquitetura estável, e as responsabilidades das equipes de trabalho do sistema precisam ser claramente definidas. Isso deve ser planejado com antecedência e não desenvolvido de forma incremental.

Você pode desenvolver um sistema de forma incremental e expô-lo aos comentários dos clientes, sem realmente entregá-lo e implantá-lo no ambiente do cliente. Entrega e implantação incremental significa que o software é usado em processos operacionais reais. Isso nem sempre é possível, pois experimentações com o novo software podem interromper os processos normais de negócios.



Um esboço simples do processo de desenvolvimento incremental é apresentado na imagem anterior. **As atividades de especificação, desenvolvimento e validação são intercaladas, em vez de separadas, com feedback rápido que permeia as atividades.** Desenvolve-se rapidamente uma implementação inicial do software (protótipo) a partir de especificações bastante abstratas e são feitas modificações de acordo com sua avaliação.

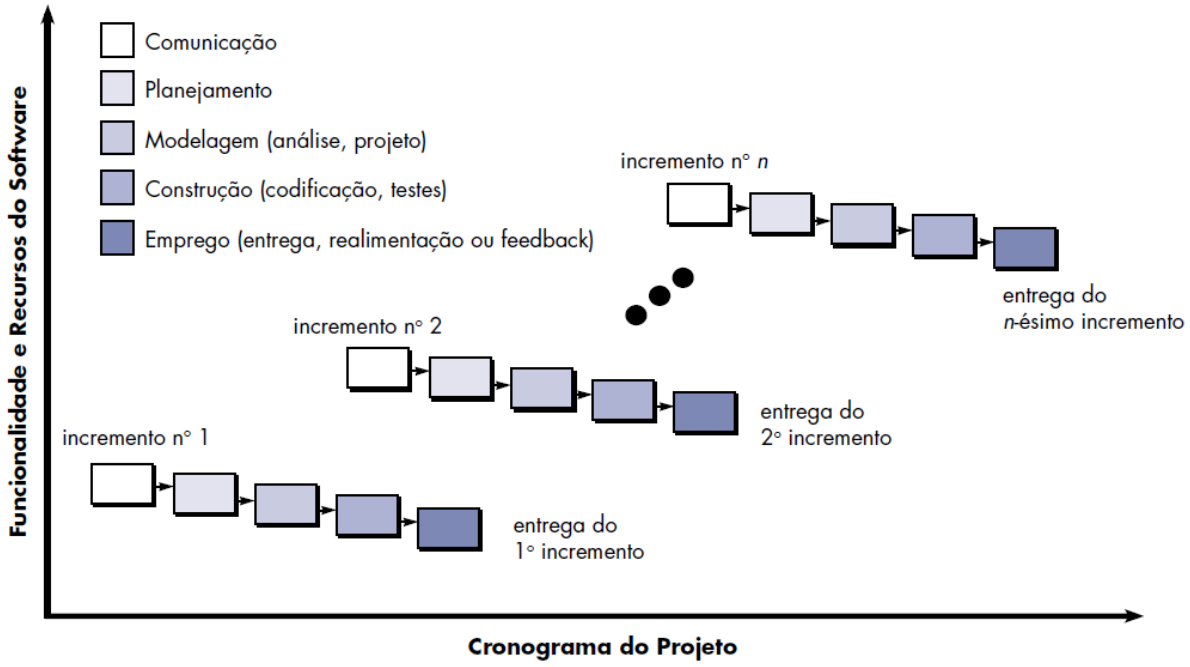
Cada versão do programa herda as melhores características das versões anteriores. **Cada versão é refinada com base no feedback recebido dos clientes para produzir um sistema que satisfaça suas necessidades**. Neste ponto, o sistema pode ser entregue ou pode ser reimplementado

utilizando uma abordagem mais estruturada para aumentar a robustez e a facilidade de manutenções!

As atividades de especificação, desenvolvimento e validação são concorrentes e apresentam um forte feedback entre si, como mostra a imagem apresentada anteriormente. **Já na visão de Roger Pressman, em várias situações, os requisitos iniciais do software são razoavelmente bem definidos, entretanto, devido ao escopo geral do trabalho de desenvolvimento, o uso de um processo puramente linear não é utilizado.**

Pode ser necessário o rápido fornecimento de um determinado conjunto funcional aos usuários, para somente após esse fornecimento, refinar e expandir sua funcionalidade em versões de software posteriores. Em tais casos, pode-se optar por um modelo de processo projetado para desenvolver o software de forma incremental. **O modelo incremental combina elementos dos fluxos de processos lineares e paralelos.**

Na imagem a seguir, o modelo incremental aplica sequências lineares, de forma escalonada, à medida que o tempo vai avançando. Cada sequência linear gera "incrementais" (entregáveis / aprovados / liberados) do software de maneira similar aos incrementais gerados por um fluxo de processos evolucionários (que veremos adiante). Vamos pensar, por exemplo, em um software de processamento de texto desenvolvido com o emprego do paradigma incremental.



Ele poderia liberar funções básicas de gerenciamento de arquivos, edição e produção de documentos no primeiro incremento; recursos mais sofisticados de edição e produção de documentos no segundo; revisão ortográfica e gramatical no terceiro; e, finalmente, recursos avançados de formatação (layout) de página no quarto incremento. Deve-se notar que o fluxo de processos para qualquer incremento pode incorporar o paradigma da prototipação.

Quando se utiliza um modelo incremental, frequentemente, o primeiro incremento é um produto essencial. Isto é, as os requisitos básicos são atendidos, porém, muitos recursos complementares (alguns conhecidos, outros não) ainda não são entregues. Esse produto essencial é utilizado pelo cliente (ou passa por uma avaliação detalhada). Como resultado do uso e/ou avaliação, é desenvolvido um planejamento para o incremento seguinte.

O planejamento já considera a modificação do produto essencial para melhor se adequar às necessidades do cliente e à entrega de recursos e funcionalidades adicionais. **Esse processo é repetido após a liberação de cada incremento, até que seja produzido o produto completo.** O modelo de processo incremental tem seu foco voltado para a entrega de um produto operacional com cada incremento.

Os primeiros incrementos são versões seccionadas do produto final, mas eles realmente possuem capacidade para atender ao usuário e também oferecem uma plataforma para avaliação do usuário. O desenvolvimento incremental é particularmente útil nos casos em que não há pessoal disponível para uma completa implementação na época de vencimento do prazo estabelecido para o projeto. **Os primeiros incrementos podem ser implementados com número mais reduzido de pessoal.**

Se o produto essencial for bem acolhido, então um pessoal adicional (se necessário) poderá ser acrescentado para implementar o incremento seguinte. Além disso, os incrementos podem ser planejados para administrar riscos técnicos. Por exemplo: um sistema importante pode exigir a disponibilidade de novo hardware que ainda está em desenvolvimento e cuja data de entrega é incerta.

Poderia ser possível planejar incrementos iniciais de modo a evitar o uso desse hardware, possibilitando a liberação de funcionalidade parcial aos usuários finais, sem um atraso excessivo.

2.2.2 – Rapid Application Development (RAD)

INCIDÊNCIA EM PROVA: MÉDIA

O RAD (Rapid Application Development) é um modelo iterativo e incremental, que **enfatiza o ciclo de desenvolvimento curto (60 a 90 dias)**. Esse desenvolvimento ocorre tão rápido, porque é utilizada o reúso de componentes a exaustão. Como muitos componentes já estão testados, pode-se reduzir o tempo total de desenvolvimento. As fases são descritas na tabela seguinte e exibidas na imagem da próxima página:

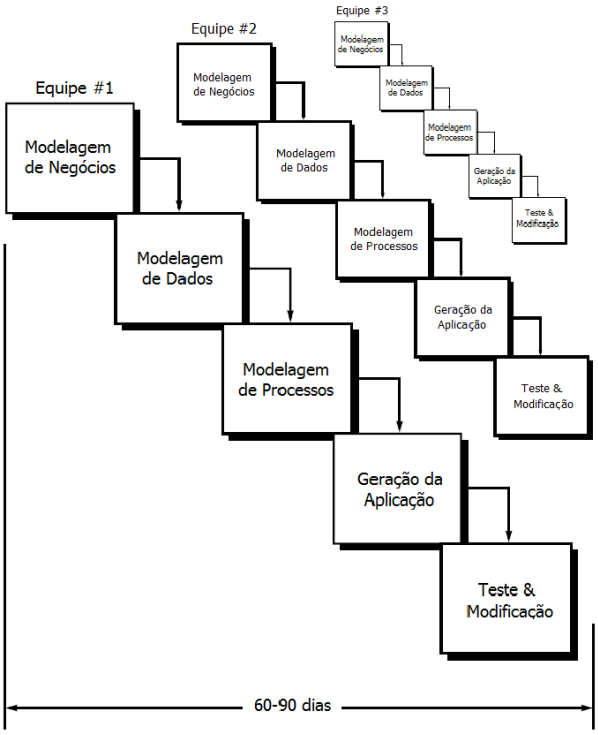
FASES	DESCRIÇÃO
MODELAGEM DE NEGÓCIO	O fluxo de informações entre as funções de negócio é modelado de modo a responder que informação direciona o processo de negócio; que informação é gerada; quem gera essa informação; para onde vai a informação gerada; e, por fim, quem processa a informação.
MODELAGEM 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.

Detalhe interessante: as etapas de Modelagem de Negócio, Dados e Processos são frequentemente condensadas na etapa de Modelagem; e Geração da Aplicação, Teste e Modificação são frequentemente condensados na etapa de Construção. **Lembrando que, como pode ser observado pela imagem apresentada anteriormente, essas etapas podem ser distribuídas por diversas equipes**.

Neste modelo, há uma interação direta e intensa com o usuário e uso frequente de programação de banco de dados e ferramentas de apoio ao desenvolvimento, como geradores de telas e relatórios. Mais abaixo, pode-se ver as vantagens e desvantagens do modelo de desenvolvimento rápido de aplicações. **Galera, ele não pode ser utilizado em qualquer situação. Recomenda-se utilizá-lo em situações específicas.**

Por exemplo: quando a aplicação não necessita de softwares auxiliares (standalone); quando é possível fazer uso de classes pré-existentes; quando a performance não é o mais importante;

quando o risco técnico é reduzido; quando a distribuição do produto no mercado é pequena; quando o escopo do projeto é restrito; quando o sistema pode ser dividido em vários módulos; quando o risco de mudança tecnológica é baixo.



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)

QUESTÕES COMENTADAS – DIVERSAS BANCAS

ENGENHARIA DE SOFTWARE

1. **(FCC / TST – 2012)** A Engenharia de Software:

a) é uma área da computação que visa abordar de modo sistemático as questões técnicas e não técnicas no projeto, implantação, operação e manutenção no desenvolvimento de um software.

b) consiste em uma disciplina da computação que aborda assuntos relacionados a técnicas para a otimização de algoritmos e elaboração de ambientes de desenvolvimento.

c) trata-se de um ramo da TI que discute os aspectos técnicos e empíricos nos processos de desenvolvimento de sistemas, tal como a definição de artefatos para a modelagem ágil.

d) envolve um conjunto de itens que abordam os aspectos de análise de mercado, concepção e projeto de software, sendo independente da engenharia de um sistema.

e) agrupa as melhores práticas para a concepção, projeto, operação e manutenção de artefatos que suportam a execução de programas de computador, tais como as técnicas de armazenamento e as estruturas em memória principal.

Comentários:

De acordo com Pressman: "*A Engenharia de Software ocorre como consequência de um processo chamado Engenharia de Sistemas. Em vez de se concentrar somente no software, a engenharia de sistemas focaliza diversos elementos, analisando, projetando, e os organizando em um sistema que pode ser um produto, um serviço ou uma tecnologia para transformação da informação ou controle*". Logo, vamos aos julgamentos:

(a) Perfeito, observem as palavras-chave: modo sistemático; questões técnicas e não técnicas; projeto, implantação, operação e manutenção de desenvolvimento de software.

(b) *Técnicas para otimização de algoritmos e elaboração de ambientes de desenvolvimento?* Não, isso não é Engenharia de Software.

(c) Pessoal, discordo do gabarito! Certa vez, um aluno me disse que talvez fosse porque aspectos empíricos são mais voltados para metodologias ágeis. Sim, é verdade! No entanto, a engenharia de software trata também de metodologias ágeis. Se alguém encontrar o erro, avise :-]

(d) A Análise de Mercado serve mais como uma técnica para Análise de Interfaces, mas pode ser vista como um dos aspectos que envolvem a Engenharia de Software. Pressman afirma que: "*A Análise de Mercado pode ser inestimável na definição de segmentos de mercado e no entendimento de*

como cada segmento poderia usar o software de modos sutilmente diferentes". De todo modo, a questão está errada porque a Engenharia de Software depende da Engenharia de Sistema (como é mostrado acima).

(e) *Suportam a execução?* Não, suportam o desenvolvimento de programas de computador.

Gabarito: Letra A

---

2. **(FCC / TRT6 – 2012)** Considere: *é uma disciplina que se ocupa de todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até a manutenção desse sistema, depois que ele entrou em operação. Seu principal objetivo é fornecer uma estrutura metodológica para a construção de software com alta qualidade.* A definição refere-se:

a) ao ciclo de vida do software.
b) à programação orientada a objetos.
c) à análise de sistemas.
d) à engenharia de requisitos.
e) à engenharia de software.

Comentários:

Engenharia de Software é uma disciplina de engenharia que se ocupa de todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até a manutenção desse sistema, após sua entrada em produção. A meta principal da Engenharia de Software é desenvolver sistemas de software com boa relação custo-benefício. Essa é a pura definição de Engenharia de Software!

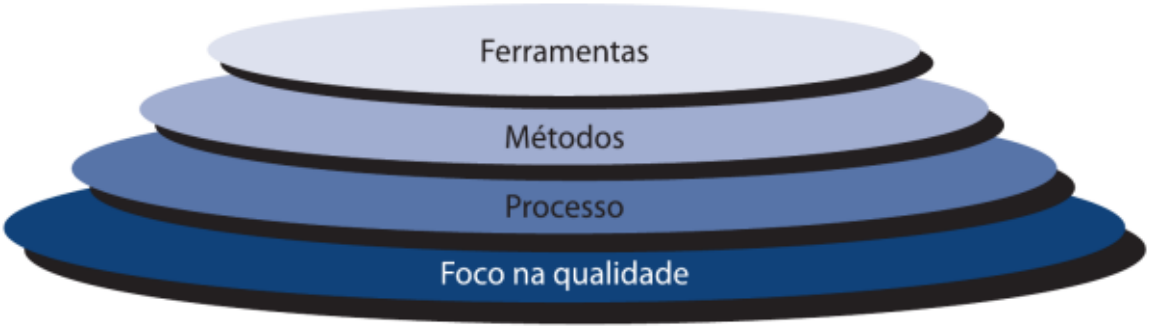
Gabarito: Letra E

---

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: Letra B

---

4. (FCC / TRE-AM – 2010) A Engenharia de Software:

a) não tem como método a abordagem estruturada para o desenvolvimento de software, pois baseia-se exclusivamente nos modelos de software, notações, regras e técnicas de desenvolvimento.

b) se confunde com a Ciência da Computação quando ambas tratam do desenvolvimento de teorias, fundamentações e práticas de desenvolvimento de software.

c) tendo como foco apenas o tratamento dos aspectos de construção de software, subsidia a Engenharia de Sistemas no tratamento dos sistemas baseados em computadores, incluindo hardware e software.

d) tem como foco principal estabelecer uma abordagem sistemática de desenvolvimento, através de ferramentas e técnicas apropriadas, dependendo do problema a ser abordado, considerando restrições e recursos disponíveis.

e) segue princípios, tais como, o da Abstração, que identifica os aspectos importantes sem ignorar os detalhes e o da Composição, que agrupa as atividades em um único processo para distribuição aos especialistas.

Comentários:

(a) Errado. Pelo contrário, ela se baseia em uma abordagem estruturada e sistemática! A IEEE define engenharia de software como a aplicação de uma abordagem sistemática, disciplinada e quantificável de desenvolvimento, operação e manutenção de software. Já Friedrich Bauer conceitua como a criação e a utilização de sólidos princípios de engenharia a fim de obter software de maneira econômica, que seja confiável e que trabalhe em máquinas reais.

(b) Errado. Na verdade, Engenharia de Software é uma disciplina da Ciência da Computação. A Engenharia de Software tem por objetivos a aplicação de teoria, modelos, formalismos, técnicas e ferramentas da ciência da computação e áreas afins para a desenvolvimento sistemático de software. Associado ao desenvolvimento, é preciso também aplicar processos, métodos e ferramentas sendo que a pedra fundamental que sustenta a engenharia de software é a qualidade.

(c) *Apenas o tratamento dos aspectos de construção de software? Só construção?* Não! (d) Correto, é exatamente isso! (e) *Composição?* Não, Decomposição! Divide-se o problema em partes para que cada uma possa ser resolvida de uma forma mais específica. Além disso, a abstração ignora detalhes!

Gabarito: Letra D

---

5. (FCC / INFRAERO – 2011) Em relação à Engenharia de Software, é INCORRETO afirmar:

a) O design de software, ao descrever os diversos aspectos que estarão presentes no sistema quando construído, permite que se faça a avaliação prévia para garantir que ele alcance os objetivos propostos pelos interessados.

b) A representação de um design de software mais simples para representar apenas as suas características essenciais busca atender ao princípio da abstração.

c) Iniciar a entrevista para obtenção dos requisitos de software com perguntas mais genéricas e finalizar com perguntas mais específicas sobre o sistema é o que caracteriza a técnica de entrevista estruturada em funil.

d) No contexto de levantamento de requisitos, funcionalidade é um dos aspectos que deve ser levado em conta na abordagem dos requisitos funcionais.

e) A representação é a linguagem do design, cujo único propósito é descrever um sistema de software que seja possível construir.

Comentários:

Descrever um sistema de software que seja possível construir não é o único, mas um dos objetivos da representação. Ela auxilia a comunicação entre as partes interessadas e serve também como documentação.

Gabarito: Letra E

---

6. **(CESPE / TCE-TO – 2009 – Item A)** Quanto maior e mais complexo o projeto de software, mais simples deve ser o modelo de processo a ser adotado.

Comentários:

Galera, não existe essa relação! Em geral, quanto mais complexo o projeto mais complexo o modelo. No entanto, isso também não é uma regra. Hoje em dia, existem projetos megacomplexos feitos utilizando metodologias ágeis, por exemplo.

Gabarito: Errado

---

7. **(CESPE / TCE-TO – 2009 – Item B)** O modelo de ciclo de vida do software serve para delimitar o alvo do software. Nessa visão, não são consideradas as atividades necessárias e o relacionamento entre elas.

Comentários:

A escolha de um modelo de ciclo de vida (para concursos, sinônimo de modelo de processo) é o ponto de partida para a definição de um processo de desenvolvimento de software. Um modelo de ciclo de vida, geralmente, organiza as macro-atividades básicas do processo, estabelecendo precedência e dependência entre as mesmas. Logo, o alvo do software serve para delimitar o modelo de ciclo de vida a ser escolhido. Primeiro, define-se o alvo do software para, só então, escolher o modelo de ciclo de vida mais adequado. Ademais, são consideradas as atividades necessárias e o relacionamento entre elas.

Gabarito: Errado

---

8. **(CESPE / MEC – 2011)** O processo de desenvolvimento de software é uma caracterização descritiva ou prescritiva de como um produto de software deve ser desenvolvido.

Comentários:

Metodologia de Desenvolvimento de Software É uma caracterização prescritiva ou descritiva de como um produto de software deve ser desenvolvido, isto é, define o quê, como e quando fazer algo – um exemplo clássico é o RUP! Segundo Pressman, um modelo prescritivo consiste em um conjunto específico de atividades de arcabouço, como – por exemplo – a NBR ISSO/IEC 12207, que estabelece uma estrutura comum para os processos de ciclo de vida de software. Já um modelo descritivo apresenta o processo em detalhes, é como se fosse um guia para o desenvolvimento de software. Ambas as formas podem ser utilizadas.

Gabarito: Correto

---

9. **(CESPE / TRT10 – 2013)** As atividades fundamentais relacionadas ao processo de construção de um software incluem a especificação, o desenvolvimento, a validação e a evolução do software.

Comentários:

Sommerville afirma que um processo de software é um conjunto de atividades e resultados associados que produz um produto de software. De acordo com ele, existem quatro atividades fundamentais de processo, que são comuns a todos os processos de software – são elas: Especificação de Software; Desenvolvimento de Software (Projeto e Implementação); Validação de Software; e Evolução de Software. Logo, a questão está corretíssima!

Gabarito: Correto

---

10. **(CESPE / TRE-BA – 2010)** Um modelo de processo de software consiste em uma representação complexa de um processo de software, apresentada a partir de uma perspectiva genérica.

Comentários:

Um modelo de processo é uma representação abstrata de um esqueleto de processo, incluindo tipicamente algumas atividades principais, a ordem de precedência entre elas e, opcionalmente, artefatos requeridos e produzidos. Em geral, um modelo de processo descreve uma filosofia de organização de atividades, estruturando-as em fases e definindo como essas fases estão relacionadas. Logo, um modelo de processo de software ou modelo de ciclo de vida trata da representação abstrata/simplificada de um processo de software, apresentada a partir de uma perspectiva genérica.

Gabarito: Errado

MODELO EM CASCATA

11. (CESPE / EBSERH – 2018) O modelo de ciclo de vida em cascata tem como características o estabelecimento, no início do projeto, de requisitos de maneira completa, correta e clara, e a possibilidade de disponibilização de várias versões operacionais do software antes da conclusão do projeto.

Comentários:

O modelo de ciclo de vida em cascata realmente tem como característica o estabelecimento, no início do projeto, de requisitos de maneira completa, correta e clara. No entanto, não há versões operacionais do software antes da conclusão do projeto, apenas ao fim – essa é uma característica do modelo iterativo e incremental.

Gabarito: Errado

---

12. (FAURGS / TJ-RS – 2018) Considere as seguintes afirmações sobre o modelo cascata de desenvolvimento de software.

I - É um exemplo de processo dirigido a planos; em princípio, deve-se planejar todas as atividades do processo antes de se começar a trabalhar nelas.

II - É consistente com outros modelos de processos de engenharia e a documentação é produzida em cada fase do ciclo. Dessa forma, o processo torna-se visível e os gerentes podem monitorar o progresso de acordo com o plano de desenvolvimento.

III- Sua maior vantagem é a divisão inflexível do projeto em estágios distintos, de forma que os compromissos devem ser assumidos em um estágio inicial do processo, o que facilita que atendam às mudanças de requisitos dos clientes.

Quais estão corretas?

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

Comentários:

(I) Correto, trata-se realmente de uma metodologia dirigida a planos; (II) Correto, o modelo em cascata é – sim – consistente com outros modelos de processos de engenharia e temos uma documentação ao fim de cada fase. Isso torna o processo visível e o monitoramento do progresso

mais simples; (III) Errado, essa é sua maior desvantagem, visto que dificulta que atendam às mudanças de requisitos dos clientes.

Gabarito: Letra B

---

13. (FAURGS / BANRISUL – 2018) Há vários modelos de processo de software, sendo que cada um define um fluxo de processo que invoca cada atividade do desenvolvimento de forma diversa. O modelo _____, algumas vezes chamado ciclo de vida clássico, é um exemplo de processo dirigido a planos, pois deve-se planejar todas as atividades (estágios) do processo antes de começar a trabalhar nelas. Em princípio, o estágio seguinte não deve ser iniciado até que o estágio anterior seja concluído, mas na prática este processo não é um modelo linear simples, envolvendo o feedback de um estágio a outro. Assim os documentos e artefatos produzidos em cada estágio podem ser modificados para refletirem as alterações em cada um deles. Este modelo é consistente com outros modelos de processo de engenharia, e a documentação é produzida em cada estágio do ciclo. Desta forma, o processo torna-se visível e os gerentes podem monitorar o progresso de acordo com o plano de desenvolvimento. Seu maior problema é a divisão inflexível do projeto em estágios distintos e, por isso, deve ser usado apenas quando os requisitos são bem compreendidos e pouco provavelmente venham a ser radicalmente alterados durante o desenvolvimento.

Assinale a alternativa que preenche corretamente a lacuna do texto acima:

a) cascata (waterfall)
b) espiral
c) orientado a desenvolvimento incremental
d) baseado em componentes
e) prototipação

Comentários:

Modelo de ciclo de vida clássico? Modelo em Cascata (Waterfall).

Gabarito: Letra A

---

14. (COSEPE / UFPI – 2018) O modelo cascata é um dos paradigmas mais antigos da engenharia de software. Dentre os problemas às vezes encontrados quando se aplica o modelo cascata, tem-se:

a) A etapa de comunicação ser responsável pelo levantamento das necessidades.
b) A existência de uma variação na representação do modelo, denominada de modelo V.
c) O modelo ser equivocadamente aplicado a problemas com requisitos bem definidos e razoavelmente estáveis.

d) O uso do fluxo sequencial proposto pelo modelo, visto que projetos reais raramente seguem tal fluxo.
e) A existência de somente cinco etapas no modelo, da comunicação ao emprego.

Comentários:

(a) Errado. Isso não é um problema do modelo em cascata; (b) Errado, Isso não é um problema do modelo em cascata; (c) Errado. Essa é uma característica do modelo e, não, um problema; (d) Correto. De acordo com Pressman, projetos reais raramente seguem o fluxo sequencial que o modelo propõe. Embora o modelo linear possa conter iterações, ele o faz indiretamente. Como consequência, mudanças podem provocar confusão à medida que a equipe de projeto prossegue; (e) Errado. Isso não é um problema do modelo em cascata.

Gabarito: Letra D

---

15. **(Inaz do Pará / CORE-SP – 2019)** "O Modelo em Cascata (do inglês: Waterfall Model) é um modelo de desenvolvimento de software sequencial no qual o processo é visto como um fluir constante para frente (como uma cascata)"

Disponível em: https://pt.wikipedia.org/wiki/Modelo_em_cascata. Acesso em: 13.12.2018

No que tange ao processo de desenvolvimento de software em cascata, qual a afirmativa correta?

a) O modelo em cascata ou clássico também pode ser conhecido como "Bottom-UP".
b) Este modelo está defasado e não é mais utilizado, tendo sido descontinuado desde a década de 90.
c) As fases do modelo em cascata seguem a seguinte ordem: (1) Requerimento, (2) Verificação, (3) Projeto, (4) Implementação e (5) Manutenção.
d) As fases do modelo são como uma cascata, mantendo o fluxo do trabalho de cima para baixo, não podendo voltar às fases iniciais, somente pular etapas para frente.
e) A saída produzida em cada fase será utilizada como entrada da fase seguinte, tornando o modelo em cascata um modelo simples de entender e controlar.

Comentários:

(a) Errado, ele é conhecido como um modelo top-down – assim como uma cascata; (b) Errado, ele está defasado, mas continua sendo utilizado em diversos locais; (c) Errado, as fases dependem do autor, mas nenhum apresenta as fases da questão; (d) Errado, não é permitido pular etapas para frente; (e) Correto, a saída de uma fase é realmente utilizada como entrada para a fase seguinte, tornando-o um modelo simples de entender e controlar.

Gabarito: Letra E

---

16. **(Gestão Concursos / EMATER – 2018)** O processo de um software é um conjunto de atividades que conduz ao desenvolvimento do produto software e o modelo de processo é uma descrição simplificada do processo. Qual é a característica que define o modelo cascata?

a) Atividades intercaladas.
b) Atividades sequenciais.
c) Rápida entrega do software.
d) Existência de componentes reusáveis.

Comentários:

(a) Errado, atividades são sequenciais; (b) Correto, atividades são sequenciais; (c) Errado, não há garantia de rápida entrega de software; (d) Errado, em geral não se utiliza componentes reusáveis.

Gabarito: Letra B

---

17. **(FADESP / IF-PA – 2018)** O modelo de desenvolvimento de software em cascata, também conhecido como ciclo de vida clássico, sugere uma abordagem sistemática e sequencial para o desenvolvimento de softwares que começa com a especificação dos requisitos e termina na manutenção do software acabado. Nos últimos anos, este modelo de ciclo de desenvolvimento vem sofrendo várias críticas quanto a sua eficácia. Assim, é correto afirmar que um dos possíveis problemas do ciclo de vida clássico é:

a) a exigência do modelo para que o cliente estabeleça todos os requisitos explicitamente.
b) a construção problemática dos componentes, caso o sistema não possa ser adequadamente modularizado.
c) a responsabilidade do levantamento das necessidades pela etapa de comunicação.
d) a aplicação do modelo de forma incorreta a problemas com requisitos bem definidos e razoavelmente estáveis.
e) a existência de somente cinco etapas no modelo, da comunicação à imantação.

Comentários:

(a) Correto, um dos principais problemas é que o cliente deve estabelecer todos os reqisitos explicitamente no início do projeto; (b) Errado, isso não tem nenhuma relação com o modelo em cascata; (c) Errado, esse não é um problema do modelo em cascata – é apenas uma característica; (d) Errado, esse não é um problema específico do modelo em cascata; (e) Errado, esse não é um problema do modelo em cascata – é apenas uma característica.

Gabarito: Letra A

---

18. **(CESPE / IPHAN – 2018)** No modelo em cascata, com exceção do sequenciamento dos estágios de requisitos e de análise, os demais são executados em paralelo, iniciando-se antes do término do estágio seguinte.

Comentários:

Na verdade, todos os estágios são sequenciais – não há exceção.

Gabarito: Errado

---

19. **(QUADRIX / CRM-PR – 2018)** É no estágio final do modelo em cascata, ou ciclo de vida de software, operação e manutenção, que o software é colocado em uso.

Comentários:

Pefeito! No modelo em cascata, o software só entra em uso no final do ciclo de vida.

Gabarito: Correto

---

20. **(QUADRIX / CRM-PR – 2018)** O modelo em cascata é composto por três estágios, que são independentes entre si: análise e definição de requisitos; implementação e teste unitário; e operação e manutenção.

Comentários:

Segundo Sommerville, as etapas do modelo em cascata são: **análise e definição de requisitos**; projeto de sistema e software; **implementação e teste de unidade**; integração e teste de sistema; e **operação e manutenção**. Logo, faltam dois estágios!

Gabarito: Errado

MODELO ITERATIVO E INCREMENTAL

21. (INSTITUTO AOCP / Prefeitura de Novo Hamburgo - RS – 2020) Existem diversos modelos de desenvolvimento de software na literatura. Sabendo disso é correto afirmar que o modelo que se baseia na ideia de desenvolver uma versão inicial do produto, apresentá-la para os comentários dos clientes e continuar o desenvolvimento, por meio da criação de diversas versões, até que um produto final adequado seja alcançado, é o:

a) modelo orientado a objetos.
b) modelo orientado ao reúso.
c) modelo incremental.
d) modelo cascata.
e) modelo híbrido.

Comentários:

A ideia de desenvolver uma versão inicial do produto, apresentá-la para os comentários dos clientes e continuar o desenvolvimento, por meio da criação de diversas versões, até que um produto final adequado seja alcançado, é a definição clássica do Modelo Incremental.

Gabarito: Letra C

---

22. (IBFC / Emdec – 2019) Sobre alguns modelos do ciclo de vida de desenvolvimento de software, assinale a alternativa correta.

a) Incremental
b) Curva
c) Estrela
d) Circular

Comentários:

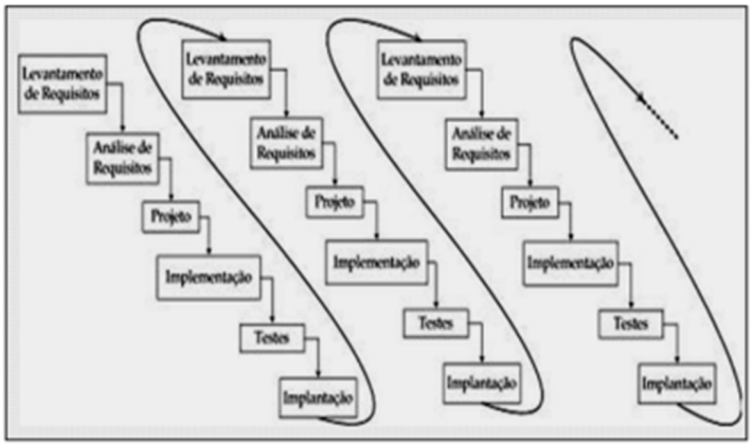
Circular, Estrela e Curva não são modelos de ciclo de desenvolvimento de software – incremental, sim.

Gabarito: Letra A

---

23. (SELECON / Prefeitura de Boa Vista - RR – 2019) No que tange à Engenharia de Software, um dos ciclos de vida para o desenvolvimento de sistemas preconiza que os requisitos do cliente são obtidos e, de acordo com a funcionalidade, são agrupados em módulos. Após este agrupamento, a equipe, junto ao cliente, define a prioridade em que cada módulo será desenvolvido, escolha baseada na importância daquela funcionalidade ao negócio do cliente. Cada módulo passa por todas as fases de projeto, conforme se observa na figura a seguir, e será

entregue ao cliente um software operacional. Assim, o cliente receberá parte do produto final em menos tempo.



Esse ciclo de vida é conhecido como:

a) cascata
b) espiral
c) incremental
d) evolutivo

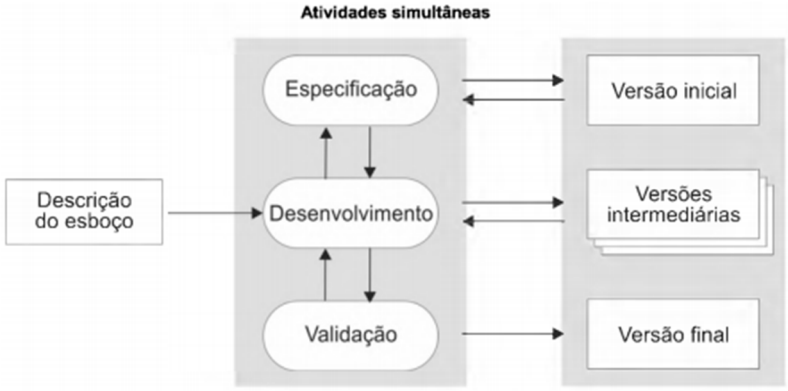
Comentários:

Esse ciclo que se repete iterativamente incrementando funcionalidades é conhecido como iterativo e incremental (ou simplesmente incremental).

Gabarito: Letra C

---

24. (FCC / TRF - 3ª REGIÃO – 2019) Considere a figura abaixo.



O modelo de processo de software representado é:

a) orientado a reuso.

b) desenvolvimento incremental.
c) em cascata.
d) processo empírico.
e) processo unificado.

Comentários:

A imagem representa o modelo de processo de software representado pelo desenvolvimento incremental (vimos a mesma imagem na aula).

Gabarito: Letra B

---

25. (CESPE / SERPRO – 2021) No modelo iterativo, as iterações na fase de construção concentram-se nas atividades de requisitos, gerenciamento, design e testes.

Comentários:

Um projeto que usa o desenvolvimento iterativo tem um ciclo de vida que consiste em várias iterações. Uma iteração incorpora um conjunto quase sequencial de tarefas em modelagem de negócio, requisitos, análise e design, implementação, teste e implementação, em várias proporções, dependendo do local em que ela está localizada no ciclo de desenvolvimento. As iterações nas fases de iniciação e de elaboração concentram-se nas atividades de gerenciamento, requisitos e design. As iterações na fase de construção concentram-se no design, na implementação e no teste. E as iterações na fase de transição concentram-se no teste e na implementação. As iterações devem ser gerenciadas em um estilo com caixa de hora, ou seja, o planejamento de uma iteração deve ser considerado fixo e o escopo do conteúdo da iteração gerenciado ativamente para atender a esse planejamento.

Gabarito: Errado

---

26. (CESPE / TST – 2008) O modelo RAD (Rapid Application Development) consiste em uma forma de prototipação para esclarecer dúvidas da especificação do software.

Comentários:

RAD é um processo de desenvolvimento de software iterativo e incremental que enfatiza um ciclo de desenvolvimento curto. Ele não é uma forma de prototipação, apesar de poder utilizá-la.

Gabarito: Errado

---

27. (CESPE / BASA – 2004) O modelo embasado em prototipagem é um modelo de processo incremental que enfatiza um ciclo de desenvolvimento extremamente curto. A primeira fase do processo é a modelagem de negócio e a última é a fase de teste e entrega.

Comentários:

Ciclo de desenvolvimento curto? Isso é RAD e, não, Prototipagem! O RAD é um modelo iterativo e incremental, que **enfatiza o ciclo de desenvolvimento curto (60 a 90 dias)**. Esse desenvolvimento ocorre tão rápido, porque é utilizada o reúso de componentes a exaustão. Como muitos componentes já estão testados, pode-se reduzir o tempo total de desenvolvimento.

Gabarito: Errado

---

28. (CESPE / MPE-AM – 2005) O modelo RAD (rapid application development) é específico para projetos de software que empregam linguagens de programação de terceira geração.

Comentários:

Não existe qualquer limitação quanto a isso! Em geral, ele mais utilizado com linguagens de programação de quarta geração, mas não se limita a elas. Lembrando que, grosso modo, as linguagens de 1ª Geração são as Linguagens de Máquina (Ex: Binário); 2ª Geração são Linguagens de Montagem (Ex: Assembler); as Linguagens de 3ª Geração são as Linguagens de Alto Nível (Ex: Java, C++, etc); e as Linguagens de 4ª Geração são as Linguagens de Altíssimo Nível com objetivos específicos (Ex: SQL para Bancos de Dados, APEX para RAD, MatLab para cálculo numérico).

Gabarito: Letra E

---

29. (CESPE / AL-ES – 2011) O ciclo de vida RAD (Rapid Application Development), por privilegiar a rapidez do desenvolvimento, não possui etapa de modelagem.

Comentários:

Como não? Existe Modelagem de Negócio, Dados e Processo!

Gabarito: Errado

---

30. (CESPE / IGEPREV – 2005) O modelo Rapid Application Development (RAD) é apropriado para projetos que envolvem grandes riscos técnicos.

Comentários:

Pelo contrário, é apropriado para projetos que envolvem pequenos riscos técnicos.

Gabarito: Errado

LISTA DE QUESTÕES – DIVERSAS BANCAS

ENGENHARIA DE SOFTWARE

1. **(FCC / TST – 2012)** A Engenharia de Software:

a) é uma área da computação que visa abordar de modo sistemático as questões técnicas e não técnicas no projeto, implantação, operação e manutenção no desenvolvimento de um software.

b) consiste em uma disciplina da computação que aborda assuntos relacionados a técnicas para a otimização de algoritmos e elaboração de ambientes de desenvolvimento.

c) trata-se de um ramo da TI que discute os aspectos técnicos e empíricos nos processos de desenvolvimento de sistemas, tal como a definição de artefatos para a modelagem ágil.

d) envolve um conjunto de itens que abordam os aspectos de análise de mercado, concepção e projeto de software, sendo independente da engenharia de um sistema.

e) agrupa as melhores práticas para a concepção, projeto, operação e manutenção de artefatos que suportam a execução de programas de computador, tais como as técnicas de armazenamento e as estruturas em memória principal.

2. **(FCC / TRT6 – 2012)** Considere: *é uma disciplina que se ocupa de todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até a manutenção desse sistema, depois que ele entrou em operação. Seu principal objetivo é fornecer uma estrutura metodológica para a construção de software com alta qualidade.* A definição refere-se:

a) ao ciclo de vida do software.
b) à programação orientada a objetos.
c) à análise de sistemas.
d) à engenharia de requisitos.
e) à engenharia de software.

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. **(FCC / TRE-AM – 2010)** A Engenharia de Software:

a) não tem como método a abordagem estruturada para o desenvolvimento de software, pois baseia-se exclusivamente nos modelos de software, notações, regras e técnicas de desenvolvimento.

b) se confunde com a Ciência da Computação quando ambas tratam do desenvolvimento de teorias, fundamentações e práticas de desenvolvimento de software.

c) tendo como foco apenas o tratamento dos aspectos de construção de software, subsidia a Engenharia de Sistemas no tratamento dos sistemas baseados em computadores, incluindo hardware e software.

d) tem como foco principal estabelecer uma abordagem sistemática de desenvolvimento, através de ferramentas e técnicas apropriadas, dependendo do problema a ser abordado, considerando restrições e recursos disponíveis.

e) segue princípios, tais como, o da Abstração, que identifica os aspectos importantes sem ignorar os detalhes e o da Composição, que agrupa as atividades em um único processo para distribuição aos especialistas.

5. **(FCC / INFRAERO – 2011)** Em relação à Engenharia de Software, é INCORRETO afirmar:

a) O design de software, ao descrever os diversos aspectos que estarão presentes no sistema quando construído, permite que se faça a avaliação prévia para garantir que ele alcance os objetivos propostos pelos interessados.

b) A representação de um design de software mais simples para representar apenas as suas características essenciais busca atender ao princípio da abstração.

c) Iniciar a entrevista para obtenção dos requisitos de software com perguntas mais genéricas e finalizar com perguntas mais específicas sobre o sistema é o que caracteriza a técnica de entrevista estruturada em funil.

d) No contexto de levantamento de requisitos, funcionalidade é um dos aspectos que deve ser levado em conta na abordagem dos requisitos funcionais.

e) A representação é a linguagem do design, cujo único propósito é descrever um sistema de software que seja possível construir.

6. **(CESPE / TCE-TO – 2009 – Item A)** Quanto maior e mais complexo o projeto de software, mais simples deve ser o modelo de processo a ser adotado.

7. **(CESPE / TCE-TO – 2009 – Item B)** O modelo de ciclo de vida do software serve para delimitar o alvo do software. Nessa visão, não são consideradas as atividades necessárias e o relacionamento entre elas.

8. **(CESPE / MEC – 2011)** O processo de desenvolvimento de software é uma caracterização descritiva ou prescritiva de como um produto de software deve ser desenvolvido.

9. **(CESPE / TRT10 – 2013)** As atividades fundamentais relacionadas ao processo de construção de um software incluem a especificação, o desenvolvimento, a validação e a evolução do software.

10. **(CESPE / TRE-BA – 2010)** Um modelo de processo de software consiste em uma representação complexa de um processo de software, apresentada a partir de uma perspectiva genérica.

MODELO EM CASCATA

11. **(CESPE / EBSERH – 2018)** O modelo de ciclo de vida em cascata tem como características o estabelecimento, no início do projeto, de requisitos de maneira completa, correta e clara, e a possibilidade de disponibilização de várias versões operacionais do software antes da conclusão do projeto.

12. **(FAURGS / TJ-RS – 2018)** Considere as seguintes afirmações sobre o modelo cascata de desenvolvimento de software.

I - É um exemplo de processo dirigido a planos; em princípio, deve-se planejar todas as atividades do processo antes de se começar a trabalhar nelas.

II - É consistente com outros modelos de processos de engenharia e a documentação é produzida em cada fase do ciclo. Dessa forma, o processo torna-se visível e os gerentes podem monitorar o progresso de acordo com o plano de desenvolvimento.

III- Sua maior vantagem é a divisão inflexível do projeto em estágios distintos, de forma que os compromissos devem ser assumidos em um estágio inicial do processo, o que facilita que atendam às mudanças de requisitos dos clientes.

Quais estão corretas?

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

13. **(FAURGS / BANRISUL – 2018)** Há vários modelos de processo de software, sendo que cada um define um fluxo de processo que invoca cada atividade do desenvolvimento de forma diversa. O modelo _____, algumas vezes chamado ciclo de vida clássico, é um exemplo de processo dirigido a planos, pois deve-se planejar todas as atividades (estágios) do processo antes de começar a trabalhar nelas. Em princípio, o estágio seguinte não deve ser iniciado até que o estágio anterior seja concluído, mas na prática este processo não é um modelo linear simples, envolvendo o feedback de um estágio a outro. Assim os documentos e artefatos produzidos em cada estágio podem ser modificados para refletirem as alterações em cada um deles. Este modelo é consistente com outros modelos de processo de engenharia, e a documentação é produzida em cada estágio do ciclo. Desta forma, o processo torna-se visível e os gerentes podem monitorar o progresso de acordo com o plano de desenvolvimento. Seu maior problema é a divisão inflexível do projeto em estágios distintos e, por isso, deve ser usado apenas quando os requisitos são bem compreendidos e pouco provavelmente venham a ser radicalmente alterados durante o desenvolvimento.

Assinale a alternativa que preenche corretamente a lacuna do texto acima:

a) cascata (waterfall)
b) espiral
c) orientado a desenvolvimento incremental
d) baseado em componentes
e) prototipação

14. **(COSEPE / UFPI – 2018)** O modelo cascata é um dos paradigmas mais antigos da engenharia de software. Dentre os problemas às vezes encontrados quando se aplica o modelo cascata, tem-se:

a) A etapa de comunicação ser responsável pelo levantamento das necessidades.
b) A existência de uma variação na representação do modelo, denominada de modelo V.
c) O modelo ser equivocadamente aplicado a problemas com requisitos bem definidos e razoavelmente estáveis.
d) O uso do fluxo sequencial proposto pelo modelo, visto que projetos reais raramente seguem tal fluxo.
e) A existência de somente cinco etapas no modelo, da comunicação ao emprego.

15. **(Inaz do Pará / CORE-SP – 2019)** "O Modelo em Cascata (do inglês: Waterfall Model) é um modelo de desenvolvimento de software sequencial no qual o processo é visto como um fluir constante para frente (como uma cascata)"

Disponível em: https://pt.wikipedia.org/wiki/Modelo_em_cascata. Acesso em: 13.12.2018

No que tange ao processo de desenvolvimento de software em cascata, qual a afirmativa correta?

a) O modelo em cascata ou clássico também pode ser conhecido como "Bottom-UP".
b) Este modelo está defasado e não é mais utilizado, tendo sido descontinuado desde a década de 90.
c) As fases do modelo em cascata seguem a seguinte ordem: (1) Requerimento, (2) Verificação, (3) Projeto, (4) Implementação e (5) Manutenção.
d) As fases do modelo são como uma cascata, mantendo o fluxo do trabalho de cima para baixo, não podendo voltar às fases iniciais, somente pular etapas para frente.
e) A saída produzida em cada fase será utilizada como entrada da fase seguinte, tornando o modelo em cascata um modelo simples de entender e controlar.

16. **(Gestão Concursos / EMATER – 2018)** O processo de um software é um conjunto de atividades que conduz ao desenvolvimento do produto software e o modelo de processo é uma descrição simplificada do processo. Qual é a característica que define o modelo cascata?

a) Atividades intercaladas.

b) Atividades sequenciais.
c) Rápida entrega do software.
d) Existência de componentes reusáveis.

17. **(FADESP / IF-PA – 2018)** O modelo de desenvolvimento de software em cascata, também conhecido como ciclo de vida clássico, sugere uma abordagem sistemática e sequencial para o desenvolvimento de softwares que começa com a especificação dos requisitos e termina na manutenção do software acabado. Nos últimos anos, este modelo de ciclo de desenvolvimento vem sofrendo várias críticas quanto a sua eficácia. Assim, é correto afirmar que um dos possíveis problemas do ciclo de vida clássico é:

a) a exigência do modelo para que o cliente estabeleça todos os requisitos explicitamente.
b) a construção problemática dos componentes, caso o sistema não possa ser adequadamente modularizado.
c) a responsabilidade do levantamento das necessidades pela etapa de comunicação.
d) a aplicação do modelo de forma incorreta a problemas com requisitos bem definidos e razoavelmente estáveis.
e) a existência de somente cinco etapas no modelo, da comunicação à imantação.

18. **(CESPE / IPHAN – 2018)** No modelo em cascata, com exceção do sequenciamento dos estágios de requisitos e de análise, os demais são executados em paralelo, iniciando-se antes do término do estágio seguinte.

19. **(QUADRIX / CRM-PR – 2018)** É no estágio final do modelo em cascata, ou ciclo de vida de software, operação e manutenção, que o software é colocado em uso.

20. **(QUADRIX / CRM-PR – 2018)** O modelo em cascata é composto por três estágios, que são independentes entre si: análise e definição de requisitos; implementação e teste unitário; e operação e manutenção.

MODELO ITERATIVO E INCREMENTAL

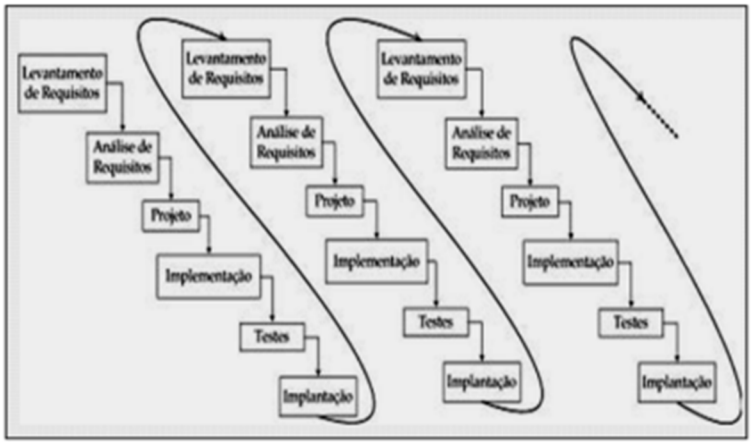
21. **(INSTITUTO AOCP / Prefeitura de Novo Hamburgo - RS – 2020)** Existem diversos modelos de desenvolvimento de software na literatura. Sabendo disso é correto afirmar que o modelo que se baseia na ideia de desenvolver uma versão inicial do produto, apresentá-la para os comentários dos clientes e continuar o desenvolvimento, por meio da criação de diversas versões, até que um produto final adequado seja alcançado, é o:

a) modelo orientado a objetos.
b) modelo orientado ao reúso.
c) modelo incremental.
d) modelo cascata.
e) modelo híbrido.

22. **(IBFC / Emdec – 2019)** Sobre alguns modelos do ciclo de vida de desenvolvimento de software, assinale a alternativa correta.

a) Incremental
b) Curva
c) Estrela
d) Circular

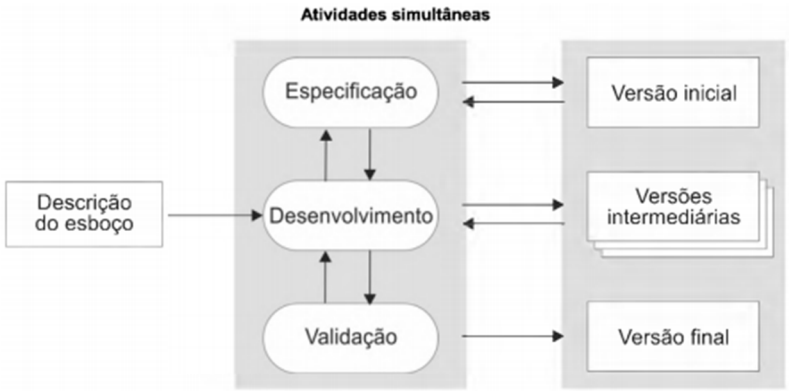
23. **(SELECON / Prefeitura de Boa Vista - RR – 2019)** No que tange à Engenharia de Software, um dos ciclos de vida para o desenvolvimento de sistemas preconiza que os requisitos do cliente são obtidos e, de acordo com a funcionalidade, são agrupados em módulos. Após este agrupamento, a equipe, junto ao cliente, define a prioridade em que cada módulo será desenvolvido, escolha baseada na importância daquela funcionalidade ao negócio do cliente. Cada módulo passa por todas as fases de projeto, conforme se observa na figura a seguir, e será entregue ao cliente um software operacional. Assim, o cliente receberá parte do produto final em menos tempo.



Esse ciclo de vida é conhecido como:

a) cascata
b) espiral
c) incremental
d) evolutivo

24. (FCC / TRF - 3ª REGIÃO – 2019) Considere a figura abaixo.



O modelo de processo de software representado é:

a) orientado a reuso.
b) desenvolvimento incremental.
c) em cascata.
d) processo empírico.
e) processo unificado.

25. (CESPE / SERPRO – 2021) No modelo iterativo, as iterações na fase de construção concentram-se nas atividades de requisitos, gerenciamento, design e testes.

26. **(CESPE / TST – 2008)** O modelo RAD (Rapid Application Development) consiste em uma forma de prototipação para esclarecer dúvidas da especificação do software.

27. **(CESPE / BASA – 2004)** O modelo embasado em prototipagem é um modelo de processo incremental que enfatiza um ciclo de desenvolvimento extremamente curto. A primeira fase do processo é a modelagem de negócio e a última é a fase de teste e entrega.

28. **(CESPE / MPE-AM – 2005)** O modelo RAD (rapid application development) é específico para projetos de software que empregam linguagens de programação de terceira geração.

29. **(CESPE / AL-ES – 2011)** O ciclo de vida RAD (Rapid Application Development), por privilegiar a rapidez do desenvolvimento, não possui etapa de modelagem.

30. **(CESPE / IGEPREV – 2005)** O modelo Rapid Application Development (RAD) é apropriado para projetos que envolvem grandes riscos técnicos.

GABARITO – DIVERSAS BANCAS

1. LETRA A
2. LETRA E
3. LETRA B
4. LETRA D
5. LETRA E
6. ERRADO
7. ERRADO
8. CORRETO
9. CORRETO
10. ERRADO
11. ERRADO
12. LETRA B
13. LETRA A
14. LETRA D
15. LETRA E
16. LETRA B
17. LETRA A
18. ERRADO
19. CORRETO
20. ERRADO
21. LETRA C
22. LETRA A
23. LETRA C
24. LETRA B
25. ERRADO
26. ERRADO
27. ERRADO
28. LETRA E
29. ERRADO
30. ERRADO

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.