Estratégia Concursos

Aula 00

SEMSA Manaus (Especialista em Saúde - Analista de Sistemas) Engenharia de Software - 2021 IBFC

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

09 de Abril de 2021

Sumário

Engenharia de Software 3

1 – Conceitos Básicos 3

2 – Processos de Desenvolvimento 6

2.1 – Modelos Sequenciais 13

2.2 – Modelo Iterativo e Incremental 18

Exercícios Comentados 23

Lista de Exercícios 36

Gabarito 42

APRESENTAÇÃO

PROFESSOR DIEGO CARVALHO - WWW.INSTAGRAM.COM/PROFESSORDIEGOCARVALHO

ENGENHARIA DE SOFTWARE

1 – Conceitos Básicos

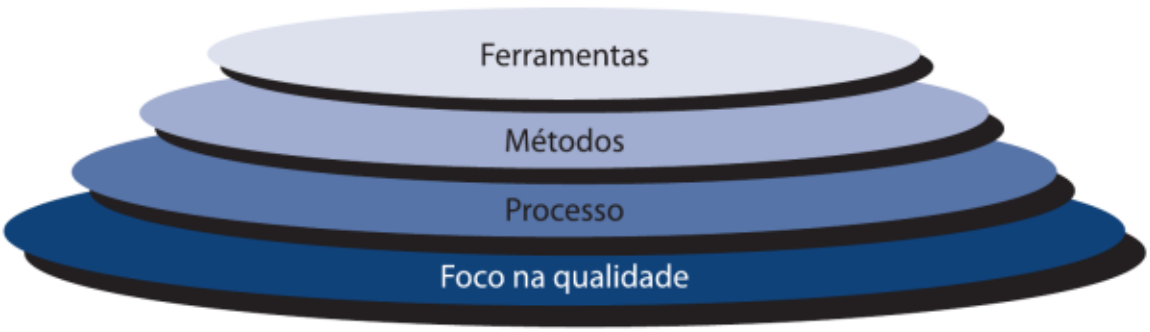
Vamos começar falando um pouquinho sobre Engenharia de Software! *Em primeiro lugar, o que é um software?* Bem, em uma visão restritiva, muitas pessoas costumam associar o termo software aos programas de computador. **No entanto, software não é apenas o programa, mas também todos os dados de documentação 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 era completamente fora de controle 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"? Quanto mais eu seguir o processo, o passo-a-passo, o que foi definido por várias pessoas a partir de suas experiências 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

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?* Pois é! Aqui a metáfora ainda vale...

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.

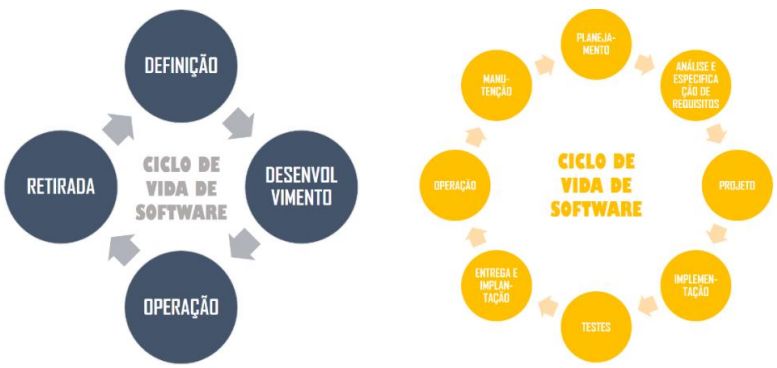
Dessa forma, nós podemos concluir que um ciclo de vida de software se **refere às fases pelas quais um sistema de software atravessa desde sua concepção até sua retirada de produção**. Bem, entender esse conceito não é o problema, esse é um conceito muito simples. *Concordam comigo? E qual é o problema,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

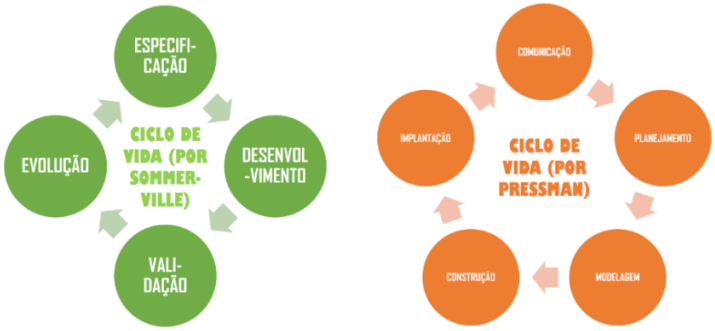
Sendo assim, percebam que há autores que descrevem as fases do ciclo de vida de software de maneira mais ampla e abstrata (com poucas e grandes fases) e autores que o fazem de maneira mais diminuta e detalhada (com muitas e pequenas fases). **Vamos começar a ver alguns ciclos de vida que existem por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

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).

> **(TCE/TO – 2009)** A escolha do modelo do ciclo de vida não depende de características específicas do projeto, pois o melhor modelo é sempre o mais usado pela equipe do projeto.
>
> _____
>
> **Comentários:** conforme vimos em aula, um processo de software não pode ser definido de forma universal. Eles devem ser definidos caso a caso, considerando-se diversos aspectos específicos do projeto em questão. Dessa forma, a afirmação não faz o menor sentido! A escolha depende das características do projeto; além disso, não existe um "melhor" modelo (Errado).

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 os requisitos esboçados, 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, bem como a funcionalidade e o comportamento esperados. 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, até que possam ser codificados e testados.
IMPLEMENTAÇÃO	O projeto deve ser traduzido para uma forma passível de execução pela máquina. A fase de implementação realiza esta tarefa, isto é, cada unidade de software do projeto detalhado é implementada.
TESTES	Inclui diversos níveis de testes, a saber, teste de unidade, teste de integração e teste de sistema. Inicialmente, cada unidade de software implementada deve ser testada e os resultados documentados. A seguir, os diversos componentes devem ser integrados sucessivamente até se obter o sistema. Finalmente, o sistema como um todo deve ser testado.
ENTREGA E IMPLANTAÇÃO	Uma vez testado, o software deve ser colocado em produção. Para tal, contudo, é necessário treinar os usuários, configurar o ambiente de produção e, muitas vezes, converter bases de dados. O propósito desta fase é estabelecer que o software satisfaz os requisitos dos usuários. Isto é feito instalando o software e conduzindo testes de aceitação. Quando o software tiver demonstrado prover as capacidades requeridas, ele pode ser aceito e a operação iniciada.

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

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.**

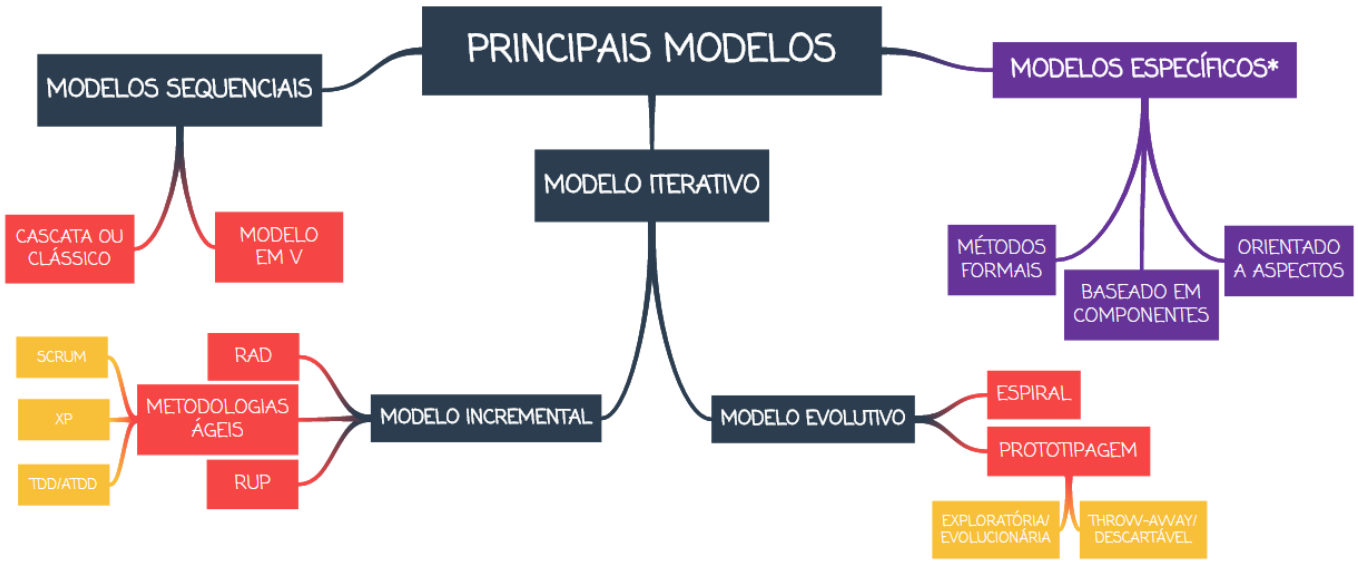
Também de acordo nosso querido autor, um modelo de processo de software é uma descrição simplificada desse processo de software que apresenta uma visão dele. Os modelos de processo incluem as atividades, que fazem parte do processo de software, e eventualmente os produtos de software e os papéis das 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...

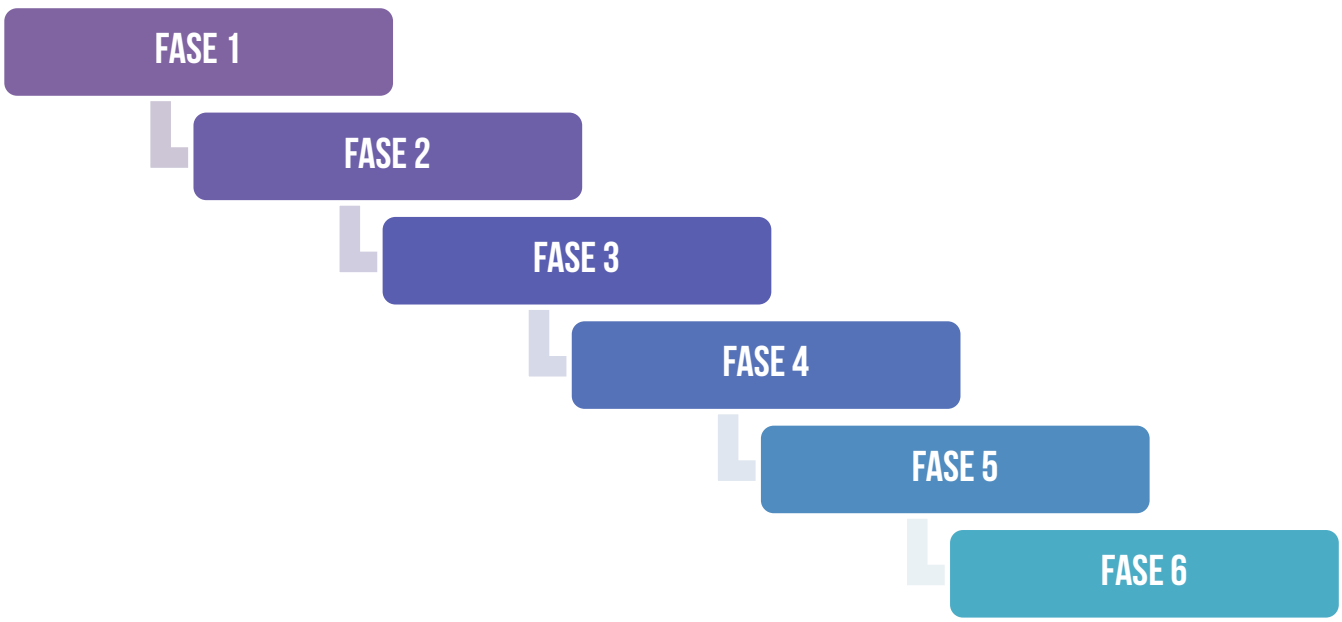
2.1 – Modelos Sequenciais

2.1.1 – Modelo em Cascata



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 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.

Galera, na prática, esses estágios do modelo em cascata não são rigorosamente sequenciais, isto é, eventualmente eles se sobrepõem e trocam informações entre si. Na teoria, são fases rigorosamente sequenciais, sendo que o resultado de cada fase consiste em um ou mais documentos aprovados ou não, dependendo dos problemas. Por exemplo: durante o projeto, podem ser identificados problemas com os requisitos da fase anterior.

Em suma: o projeto segue uma série de passos ordenados em que, ao final de cada fase, a equipe do projeto finaliza uma revisão. Além disso, **o desenvolvimento não continua até que o cliente esteja satisfeito com os resultados alcançados**. *Vocês conseguem perceber como essas restrições engessam o desenvolvimento?* Se não, vocês vão entender em breve! De modo geral, grande parte dos modelos possuem as seguintes fases:

a) **Planejamento:** faz-se o esboço do escopo e dos requisitos do software, além de levantar estimativas razoáveis sobre recursos, custos e prazos.

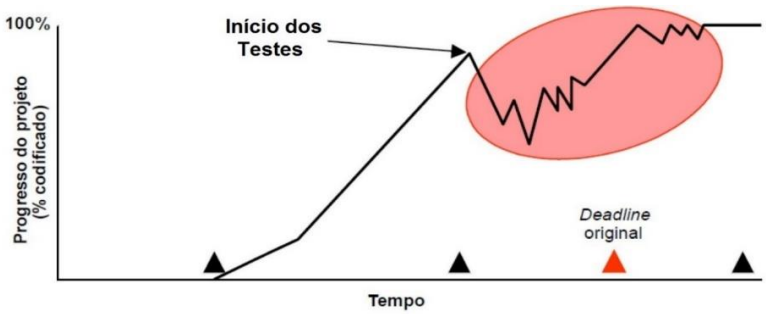
b) **Análise e Especificação de Requisitos:** durante essa fase, refinam-se os requisitos e o escopo, e desenha-se o problema a ser resolvido pelo software.

c) **Projeto:** durante essa fase, incorporam-se requisitos tecnológicos aos requisitos essenciais do sistema e projeta-se a arquitetura do sistema.

d) **Implementação:** durante essa fase, codifica-se o software como um conjunto de programas executáveis pela máquina.

e) **Teste:** o programa é testado como um sistema completo para garantir que os requisitos de software foram atendidos.

f) **Implantação, Operação e Manutenção:** o sistema de software é liberado para o cliente, treinam-se usuários, gerenciam-se os serviços e realizam-se manutenções.



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) 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.

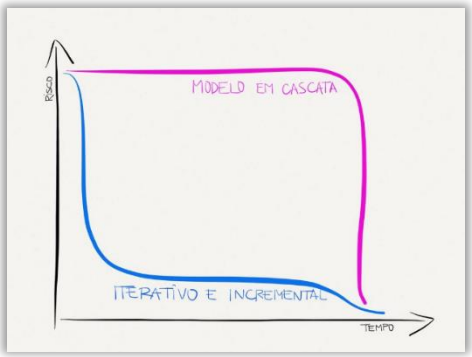
Entenderam essa parte direitinho? **Um erro na fase de requisitos, por exemplo, que não foi corrigido e foi descoberto no final do processo de desenvolvimento, terá um custo de correção altíssimo**, visto que provavelmente terá que se refazer tudo novamente. *Vocês*

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 de desenvolvimento de software, 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...

2.2 – Modelo Iterativo e Incremental

2.2.1 – Conceitos Básicos

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!

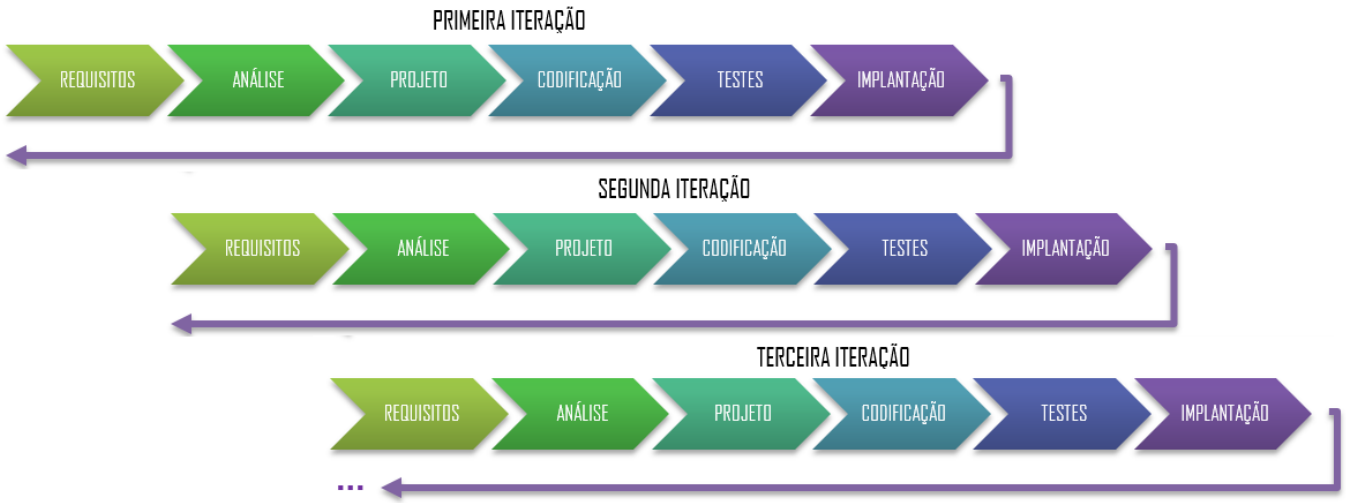
Foi quando surgiu o Modelo Iterativo e Incremental como uma tentativa de resolver esse problema de acúmulo de riscos. Vejamos as diferenças fundamentais:

- **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 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.

IMPORTANTE

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.

Professor, mas e se cair essa diferença em prova? Ora, caso caia em prova, saibam que 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...

2.2.2 – Rapid Application Development (RAD)

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. As fases são mostradas na imagem abaixo:

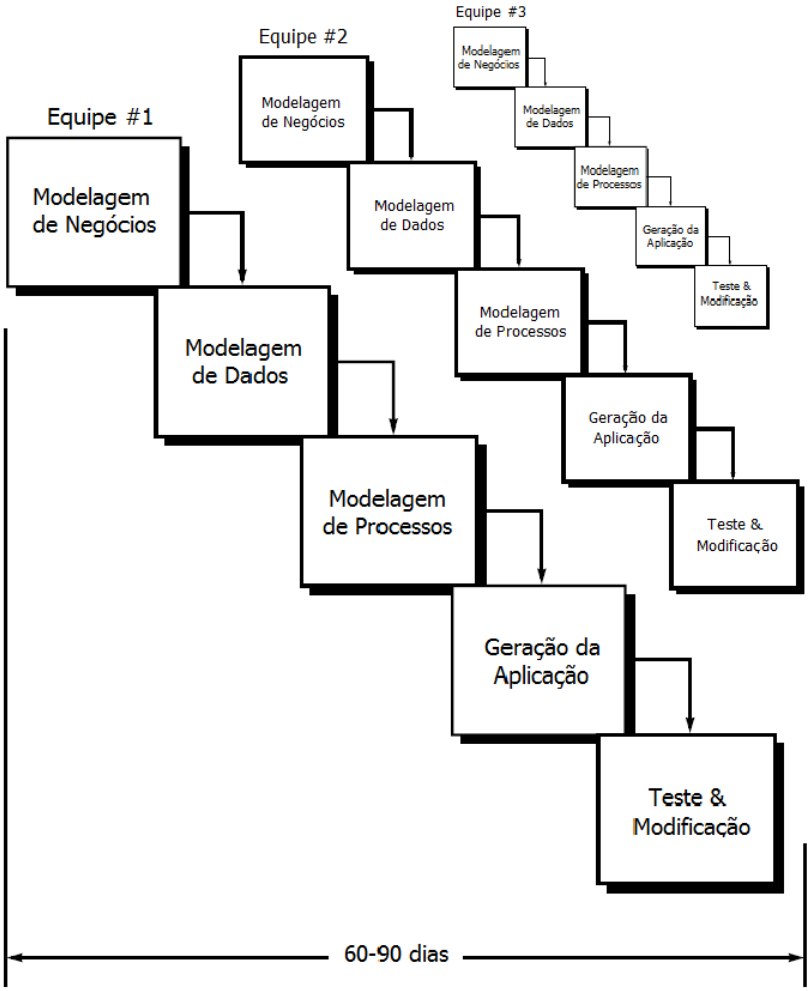
- **Modelagem de Negócio:** o fluxo de informações entre as funções de negócio é modelado de modo a responder que informação direciona o processo de negócio; que informação é gerada; quem gera essa informação; para onde vai a informação gerada; e, por fim, quem processa a informação.

- **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: Modelagem de Negócio, Dados e Processos são frequentemente condensados na etapa de Modelagem e Geração da Aplicação, Teste e Modificação são frequentemente condensados na etapa de Construção. **Lembrando que, como pode ser observado pela imagem acima, 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. **Galera, ele não pode ser utilizado em qualquer situação. Recomenda-se utilizá-lo quando**:

- a aplicação não necessita de softwares auxiliares (standalone);
- é possível fazer uso de classes pré-existentes;
- a performance não é o mais importante;
- o risco técnico é reduzido;
- a distribuição do produto no mercado é pequena;
- o escopo do projeto é restrito;
- o sistema pode ser dividido em vários módulos;
- 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)

(IPEA/2003) O RAD (Rapid Application Development) é um modelo de processo de software incremental que assume um ciclo de desenvolvimento curto e utiliza uma abordagem de construção com base em componentes.

Comentários: conforme vimos em aula, ele realmente é um modelo de processo incremental e ele – de fato – assume um ciclo de desenvolvimento curso com uma abordagem baseada em componentes (Correto).

(CONAB/2006) O modelo de processo de desenvolvimento de software incremental que enfatiza um ciclo de desenvolvimento extremamente curto, que compreende as fases de modelagem do negócio, modelagem dos dados, modelagem do processo, geração da aplicação, além de teste e entrega, e que o desenvolvimento é conseguido pelo uso de construção baseada em componentes, é conhecido como modelo:

a) sequencial linear;
b) RAD (Rapid Application Development);
c) de prototipagem;
d) espiral;
e) de desenvolvimento concorrente.

Comentários: conforme vimos em aula, desenvolvimento extremamente curto, modelo incremental, baseado em componentes... só pode se referir ao RAD (Letra B).

EXERCÍCIOS COMENTADOS

ENGENHARIA DE SOFTWARE

1. **(FCC – 2010 – DPE/SP – Analista de Sistemas)** A Engenharia de Software:

I. não visa o desenvolvimento de teorias e fundamentações, preocupando-se unicamente com as práticas de desenvolvimento de software.

II. tem como foco o tratamento dos aspectos de desenvolvimento de software, abstraindo-se dos sistemas baseados em computadores, incluindo hardware e software.

III. tem como métodos as abordagens estruturadas para o desenvolvimento de software que incluem os modelos de software, notações, regras e maneiras de desenvolvimento.

IV. 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.

É correto o que se afirma em:

a) III e IV, apenas.
b) I, II, III e IV.
c) I e II, apenas.
d) I, II e III, apenas.
e) II, III e IV, apenas.

Comentários:

(I) Errado, Sommerville diz: "*Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software*". No entanto, ele não diz que a engenharia de software se preocupa **unicamente** com as práticas de desenvolvimento de software.

(II) Errado, Pressman diz: "*System engineering is concerned with all aspects of computer-based systems development including hardware, software, and process engineering*" – a questão trata da Engenharia de Sistemas.

(III) Correto, de fato ela tem como métodos as abordagens estruturadas para o desenvolvimento de software que incluem os modelos de software, notações, regras e maneiras de desenvolvimento.

(IV) Errado, o princípio da abstração ignora os detalhes; e o princípio da composição não existe – o que existe é o princípio da decomposição. E ele divide o problema em partes menores.

Em suma, nenhuma das opções nos atende! *Vocês sabem qual opção a banca marcou como correta?* A Letra D!!! *E ela voltou atrás com os recursos?* Não!!! Pois é, galera! Acostumem-se com isso :(

Gabarito: Letra D

---

2. **(CESPE – 2013 – TCE/RO – Analista de Sistemas)** Assim como a Engenharia de Software, existe também na área de informática a chamada Ciência da Computação. Assinale a alternativa que melhor apresenta a diferença entre Engenharia de Software e Ciência da Computação.

   a) A Ciência da Computação tem como objetivo o desenvolvimento de teorias e fundamentações. Já a Engenharia de Software se preocupa com as práticas de desenvolvimento de software.

   b) A Engenharia de Software trata da criação dos sistemas de computação (softwares) enquanto a Ciência da Computação está ligada ao desenvolvimento e criação de componentes de hardware.

   c) A Engenharia de Software trata dos sistemas com base em computadores, que inclui hardware e software, e a Ciência da Computação trata apenas dos aspectos de desenvolvimento de sistemas.

   d) A Ciência da Computação trata dos sistemas com base em computadores, que inclui hardware e software, e a Engenharia de Software trata apenas dos aspectos de desenvolvimento de sistemas.

   e) A Ciência da Computação destina-se ao estudo e solução para problemas genéricos das áreas de rede e banco de dados e a Engenharia de Software restringe- se ao desenvolvimento de sistemas.

Comentários:

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.

Essa questão parece contradizer o que diz Sommerville, mas não! A Engenharia de Software coloca em prática a teoria e fundamentação trazida pela Ciência da Computação. A Engenharia de Software aplica a teoria, mas - grosso modo - ela não tem a finalidade principal de elaborá-la; a finalidade principal é de colocá-la em prática. Já a Ciência da Computação é exatamente o contrário. Enfim, a primeira opção foi retirada literalmente do Sommerville (Pág. 5, 8ª Ed.).

Gabarito: Letra A

---

3. **(CESPE – 2009 – ANAC – Analista de Sistemas)** O termo engenharia pretende indicar que o desenvolvimento de software submete-se a leis similares às que governam a manufatura de produtos industriais em engenharias tradicionais, pois ambos são metodológicos.

Comentários:

Perfeito! A Engenharia busca os princípios mais consolidados de outras engenharias mais tradicionais – engenharia mecânica, civil, elétrica, etc.

Gabarito: Correto

---

4. **(CESPE - 2016 – TCE/PR – Analista de Sistemas – B)** A engenharia de software refere-se ao estudo das teorias e fundamentos da computação, ficando o desenvolvimento de software a cargo da ciência da computação.

Comentários:

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. Conforme vimos em aula, não se trata do estudo – trata-se da aplicação. Além disso, o desenvolvimento também fica a cargo da engenharia de software. Em geral, a ciência da computação trata da teoria e a engenharia de software trata da prática.

Gabarito: Errado

---

5. **(CESPE - 2016 – TCE/PR – Analista de Sistemas – E)** O conceito de software se restringe ao desenvolvimento do código em determinada linguagem e seu armazenamento em arquivos.

Comentários:

Em uma visão restritiva, muitas pessoas costumam associar o termo software aos programas de computador. Software não é apenas o programa, mas também todos os dados de documentação e configuração associados, necessários para que o programa opere corretamente. O software

não é apenas o programa, mas também todos os dados de documentação e configuração associados, necessários para que o programa opere corretamente.

Gabarito: Errado

PROCESSOS DE DESENVOLVIMENTO

6. **(CESPE – 2011 – TJ/ES – Analista de Sistemas)** Entre as etapas do ciclo de vida de software, as menos importantes incluem a garantia da qualidade, o projeto e o estudo de viabilidade. As demais atividades do ciclo, como a implementação e os testes, requerem maior dedicação da equipe e são essenciais.

Comentários:

Para a definição completa do processo, cada atividade descrita no modelo de ciclo de vida deve ser decomposta e para suas subatividades, devem ser associados métodos, técnicas, ferramentas e critérios de qualidade, entre outros, formando uma base sólida para o desenvolvimento. Adicionalmente, outras atividades, tipicamente de cunho gerencial, devem ser definidas, como controle e garantia da qualidade. Não existe uma hierarquia de etapas mais importantes e menos importantes!

Gabarito: Errado

---

7. **(CESPE - 2016 – TCE/PR – Analista de Sistemas – A)** A engenharia de software está relacionada aos diversos aspectos de produção de software e inclui as atividades de especificação, desenvolvimento, validação e evolução de 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. Dessa forma, a questão está correta.

Gabarito: Correto

---

8. **(CESPE - 2016 – TCE/PR – Analista de Sistemas – D)** Um processo de software é composto por quatro atividades fundamentais: iniciação, desenvolvimento, entrega e encerramento.

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. Dessa forma, a questão está incorreta.

Gabarito: Errado

---

9. **(INSTITUTO CIDADE – 2012 – TCM/GO – Analista de Sistemas)** De acordo com a engenharia de software, como todo produto industrial, o software possui um ciclo de vida. Cada fase do ciclo de vida possui divisões e subdivisões. Em qual fase avaliamos a necessidade de evolução dos softwares em funcionamento para novas plataformas operacionais ou para a incorporação de novos requisitos?

a) Fase de operação;
b) Fase de retirada;
c) Fase de definição;
d) Fase de design.
e) Fase de desenvolvimento;

Comentários:

A fase retirada é um grande desafio para os tempos atuais. Diversos softwares que estão em funcionamento em empresas possuem excelente níveis de confiabilidade e de correção. No entanto, eles precisam evoluir para novas plataformas operacionais ou para a incorporação de novos requisitos.

A retirada desses softwares legados em uma empresa é sempre uma decisão difícil: *como abrir mão daquilo que é confiável e ao qual os funcionários estão acostumados, após anos de treinamento e utilização?* Portanto, processos de reengenharia podem ser aplicados para viabilizar a transição

ou migração de um software legado para um novo software de forma a proporcionar uma retirada mais suave.

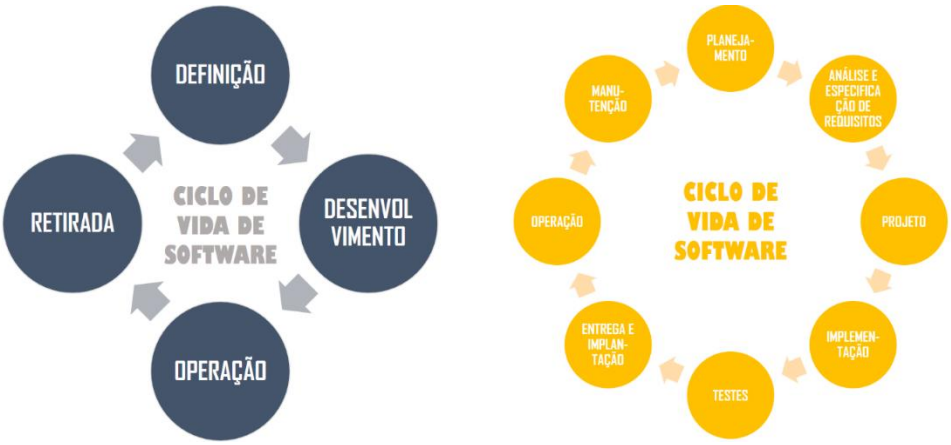
A avaliação da necessidade de evolução do software em funcionamento para novas plataformas operacionais ou para a incorporação de novos requisitos realmente ocorrem na fase de retirada.

Gabarito: Letra B

---

10. **(CESPE - 2010 – DETRAN/ES – Analista de Sistemas)** Quando um aplicativo de software desenvolvido em uma organização atinge, no fim do seu ciclo de vida, a fase denominada aposentadoria, descontinuação ou fim de vida, todos os dados por ele manipulados podem ser descartados.

Comentários:



Essa questão é um absurdo! Observem que ela afirma "*podem ser descartados*". Ora, é evidente que podem ser descartados. No entanto, o gabarito oficial é errado!

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

MODELO ITERATIVO E INCREMENTAL

16. **(CESPE – 2011 – TJ/ES – Análise de Sistemas)** O modelo de processo incremental de desenvolvimento de software é iterativo, assim como o processo de prototipagem. Contudo, no processo incremental, diferentemente do que ocorre no de prototipagem, o objetivo consiste em apresentar um produto operacional a cada incremento.

Comentários:

De fato, no modelo iterativo e incremental, apresenta-se sempre um produto a cada incremento. Já na prototipação, não. Idealmente, ele serve apenas para identificar requisitos.

Gabarito: Correto

---

17. **(CESPE – 2008 – TJ/DF – Análise de Sistemas)** No modelo de desenvolvimento incremental, embora haja defasagem entre os períodos de desenvolvimento de cada incremento, os incrementos são desenvolvidos em paralelo.

Comentários:

Questão perfeita! Os incrementos são codificados não exatamente em paralelo – há uma pequena defasagem.

Gabarito: Correto

---

18. **(CESPE - 2009 – UNIPAMPA - Análise de Sistemas)** No modelo de desenvolvimento incremental, a cada iteração são realizadas várias tarefas. Na fase de análise, pode ser feito o refinamento de requisitos e o refinamento do modelo conceitual.

Comentários:

Perfeito, é a fase seguinte à fase de requisitos e busca refiná-los.

Gabarito: Correto

---

19. **(CESPE - 2016 – TCE/PR – Analista de Sistemas – C)** No modelo iterativo de desenvolvimento de software, as atividades são dispostas em estágios sequenciais.

Comentários:

A questão trata do modelo em cascata de desenvolvimento de software. No modelo iterativo e incremental, as atividades são dispostas ao longo de diversas iterações

Gabarito: Errado

---

20. **(COMPERVE - UFRN - 2018)** Considere as afirmativas apresentadas abaixo a respeito dos modelos de processos de software cascata (waterfall) e incremental.

I Uma das vantagens do modelo de processo cascata é que ele antecipa eventuais correções a serem feitas nos requisitos do software.

II O modelo de processos cascata é recomendado quando os requisitos são estáveis e claros.

III No desenvolvimento incremental, a arquitetura e o projeto do software tendem a manter-se estáveis.

IV No desenvolvimento incremental, o acompanhamento e o progresso das atividades são avaliados pela entrega de artefatos.

Estão corretas as afirmativas:

a) II e IV.
b) I e IV.
c) I e III.
d) II e III.

Comentários:

(I) Errado. Pelo contrário, ele atrasa a redução de riscos e eventuais correções; (II) Correto. O modelo em cascata é realmente recomendado quando os requisitos são estáveis e claros; (III) Errado. No desenvolvimento incremental, a arquitetura e o projeto tendem a se manter instáveis; (IV) Correto. O acompanhamento e o progresso das atividades são – de fato – avaliados pela entrega dos artefatos a cada iteração.

Gabarito: Letra A

RAPID APPLICATION DEVELOPMENT

21. (CESPE - ANS - 2005) O modelo Rapid Application Development (RAD) é uma adaptação do modelo em espiral para atender a projetos de software fundamentados em componentes.

Comentários:

Na verdade, ele é uma adaptação de alta velocidade do modelo em cascata.

Gabarito: Errado

---

22. (COPESE - UFPI - 2014) O modelo RAD (Rapid Application Development) é um modelo incremental que enfatiza um ciclo de desenvolvimento curto, sendo construído baseado em componentes.

Comentários:

Perfeito! *Incremental? Sim! Ciclo de desenvolvimento curto?* Sim! *Baseado em componentes?* Sim!

Gabarito: Correto

---

23. **(UPANET - JUCEPE - 2012)** O Desenvolvimento Rápido de Aplicações (RAD – Rapid Application Development) pode fazer uso do processo de desenvolvimento conjunto de aplicações (JAD – Joint Application Development) para coletar dados e analisar requisitos.

Comentários:

Questão estranha! *Pode usar o JAD?* Claro, É uma técnica para levantar requisitos. *Por que não poderia?*

Gabarito: Correto

---

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

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

Comentários:

No Modelo RAD a modelagem abrange três das principais fases - modelagem de negócio, modelagem de dados e modelagem de processos - e estabelecem representações de projeto que servem com base para a atividade de construção do RAD.

Gabarito: Letra E

---

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

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

Comentários:

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

Gabarito: Letra D

LISTA DE EXERCÍCIOS

ENGENHARIA DE SOFTWARE

1. **(FCC – 2010 – DPE/SP – Analista de Sistemas)** A Engenharia de Software:

I. não visa o desenvolvimento de teorias e fundamentações, preocupando-se unicamente com as práticas de desenvolvimento de software.

II. tem como foco o tratamento dos aspectos de desenvolvimento de software, abstraindo-se dos sistemas baseados em computadores, incluindo hardware e software.

III. tem como métodos as abordagens estruturadas para o desenvolvimento de software que incluem os modelos de software, notações, regras e maneiras de desenvolvimento.

IV. 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.

É correto o que se afirma em:

a) III e IV, apenas.
b) I, II, III e IV.
c) I e II, apenas.
d) I, II e III, apenas.
e) II, III e IV, apenas.

2. **(CESPE – 2013 – TCE/RO – Analista de Sistemas)** Assim como a Engenharia de Software, existe também na área de informática a chamada Ciência da Computação. Assinale a alternativa que melhor apresenta a diferença entre Engenharia de Software e Ciência da Computação.

a) A Ciência da Computação tem como objetivo o desenvolvimento de teorias e fundamentações. Já a Engenharia de Software se preocupa com as práticas de desenvolvimento de software.

b) A Engenharia de Software trata da criação dos sistemas de computação (softwares) enquanto a Ciência da Computação está ligada ao desenvolvimento e criação de componentes de hardware.

c) A Engenharia de Software trata dos sistemas com base em computadores, que inclui hardware e software, e a Ciência da Computação trata apenas dos aspectos de desenvolvimento de sistemas.

d) A Ciência da Computação trata dos sistemas com base em computadores, que inclui hardware e software, e a Engenharia de Software trata apenas dos aspectos de desenvolvimento de sistemas.

e) A Ciência da Computação destina-se ao estudo e solução para problemas genéricos das áreas de rede e banco de dados e a Engenharia de Software restringe- se ao desenvolvimento de sistemas.

3. **(CESPE – 2009 – ANAC – Analista de Sistemas)** O termo engenharia pretende indicar que o desenvolvimento de software submete-se a leis similares às que governam a manufatura de produtos industriais em engenharias tradicionais, pois ambos são metodológicos.

4. **(CESPE - 2016 – TCE/PR – Analista de Sistemas – B)** A engenharia de software refere-se ao estudo das teorias e fundamentos da computação, ficando o desenvolvimento de software a cargo da ciência da computação.

5. **(CESPE - 2016 – TCE/PR – Analista de Sistemas – E)** O conceito de software se restringe ao desenvolvimento do código em determinada linguagem e seu armazenamento em arquivos.

PROCESSOS DE DESENVOLVIMENTO

6. **(CESPE – 2011 – TJ/ES – Analista de Sistemas)** Entre as etapas do ciclo de vida de software, as menos importantes incluem a garantia da qualidade, o projeto e o estudo de viabilidade. As demais atividades do ciclo, como a implementação e os testes, requerem maior dedicação da equipe e são essenciais.

7. **(CESPE - 2016 – TCE/PR – Analista de Sistemas – A)** A engenharia de software está relacionada aos diversos aspectos de produção de software e inclui as atividades de especificação, desenvolvimento, validação e evolução de software.

8. **(CESPE - 2016 – TCE/PR – Analista de Sistemas – D)** Um processo de software é composto por quatro atividades fundamentais: iniciação, desenvolvimento, entrega e encerramento.

9. **(INSTITUTO CIDADE – 2012 – TCM/GO – Analista de Sistemas)** De acordo com a engenharia de software, como todo produto industrial, o software possui um ciclo de vida. Cada fase do ciclo de vida possui divisões e subdivisões. Em qual fase avaliamos a necessidade de evolução dos softwares em funcionamento para novas plataformas operacionais ou para a incorporação de novos requisitos?

a) Fase de operação;
b) Fase de retirada;
c) Fase de definição;
d) Fase de design.

e) Fase de desenvolvimento;

10. **(CESPE - 2010 – DETRAN/ES – Analista de Sistemas)** Quando um aplicativo de software desenvolvido em uma organização atinge, no fim do seu ciclo de vida, a fase denominada aposentadoria, descontinuação ou fim de vida, todos os dados por ele manipulados podem ser descartados.

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.

MODELO ITERATIVO E INCREMENTAL

16. **(CESPE – 2011 – TJ/ES – Análise de Sistemas)** O modelo de processo incremental de desenvolvimento de software é iterativo, assim como o processo de prototipagem. Contudo, no processo incremental, diferentemente do que ocorre no de prototipagem, o objetivo consiste em apresentar um produto operacional a cada incremento.

17. **(CESPE – 2008 – TJ/DF – Análise de Sistemas)** No modelo de desenvolvimento incremental, embora haja defasagem entre os períodos de desenvolvimento de cada incremento, os incrementos são desenvolvidos em paralelo.

18. **(CESPE - 2009 – UNIPAMPA - Análise de Sistemas)** No modelo de desenvolvimento incremental, a cada iteração são realizadas várias tarefas. Na fase de análise, pode ser feito o refinamento de requisitos e o refinamento do modelo conceitual.

19. **(CESPE - 2016 – TCE/PR – Analista de Sistemas – C)** No modelo iterativo de desenvolvimento de software, as atividades são dispostas em estágios sequenciais.

20. **(COMPERVE - UFRN - 2018)** Considere as afirmativas apresentadas abaixo a respeito dos modelos de processos de software cascata (waterfall) e incremental.

    I Uma das vantagens do modelo de processo cascata é que ele antecipa eventuais correções a serem feitas nos requisitos do software.

    II O modelo de processos cascata é recomendado quando os requisitos são estáveis e claros.

    III No desenvolvimento incremental, a arquitetura e o projeto do software tendem a manter-se estáveis.

    IV No desenvolvimento incremental, o acompanhamento e o progresso das atividades são avaliados pela entrega de artefatos.

Estão corretas as afirmativas:

a) II e IV.
b) I e IV.
c) I e III.
d) II e III.

RAPID APPLICATION DEVELOPMENT

21. **(CESPE - ANS - 2005)** O modelo Rapid Application Development (RAD) é uma adaptação do modelo em espiral para atender a projetos de software fundamentados em componentes.

22. **(COPESE - UFPI - 2014)** O modelo RAD (Rapid Application Development) é um modelo incremental que enfatiza um ciclo de desenvolvimento curto, sendo construído baseado em componentes.

23. **(UPANET - JUCEPE - 2012)** O Desenvolvimento Rápido de Aplicações (RAD – Rapid Application Development) pode fazer uso do processo de desenvolvimento conjunto de aplicações (JAD – Joint Application Development) para coletar dados e analisar requisitos.

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

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

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

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

GABARITO

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

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.