etrônico

Estratégia
CONCURSOS

Aul

Passo de Tecnologia da Informação p/ TRF 3ª Região (Analista Judiciária - Informática) Pós-Edital

Professor: Equipe Informática e TI, Thiago Rodrigues Cavalcanti

Apresentação do Professor 1
1 – Introdução 2
2 – Análise Estatística 2
3 – Análise de Questões 3
4 – Orientações de Revisão e Pontos a Destacar 9
Conceitos de Engenharia de Software 9
Definição de Software 10
Princípios da Engenharia de Software 12
Ciclo de Vida do Software 14
Processo de Software 19
UML – Unified Modeling Language 21

THIAGO CAVALCANTI
PROFESSOR

APRESENTAÇÃO DO PROFESSOR

Olá Senhoras e Senhores,

Eu me chamo Thiago Cavalcanti. Sou funcionário do Banco Central do Brasil, passei no concurso em 2010 para Analista de Tecnologia da Informação (TI). Atualmente estou de licença, cursando doutorado em economia na UnB. Também trabalho como professor de TI no Estratégia e participo da equipe do Passo Estratégico como Analista de Informática.

Tenho graduação em Ciência da Computação pela UFPE e mestrado em Engenharia de Software. Já fui aprovado em diversos concursos tais como ANAC, BNDES, TCE-RN, INFRAERO e, claro, Banco Central. A minha trajetória como concurseiro durou pouco mais de dois anos. Neste intervalo, aprendi muito e vou tentar passar um pouco desta minha experiência ao longo deste curso.

A banca do concurso é a Fundação Carlos Chagas - FCC. O concurso em questão é do **Tribunal Regional Federal da 3ª Região**. Teremos muito trabalho pela frente, tendo em vista que o assunto é extenso e o prazo é curto, além de assuntos pouco convencionais que fazem do conteúdo programático do edital. Esteja atento, pois, a FCC consegue explorar o conteúdo com questões condizentes com o conhecimento exigido para o trabalho no órgão público.

1 – INTRODUÇÃO

Essa é a primeira aula do nosso curso e nela faremos uma apresentação teórica sobre os conceitos de engenharia de software. Tentei dividir o conteúdo da melhor forma possível para que você consiga entender cada tópico. Leia esta aula com atenção e caso haja alguma dúvida, não hesite em me perguntar no fórum.

2 – ANÁLISE ESTATÍSTICA

FIQUE ATENTO!

Realizamos a análise estatística dos assuntos mais cobrados pela banca nas últimas provas. Diante disso, foi possível concluir que a porcentagem de questões do conteúdo dessa aula nas últimas provas elaboradas pela banca é de **14,83%** de um total de 654 questões de todo conteúdo. Diante disso, podemos destacar que o assunto "**Conceitos de engenharia de software: Processo de desenvolvimento de software. Metodologias ágeis. Ciclos de vida do software. Análise de requisitos. Projeto de interfaces. Unized Modeling Language (UML) em orientação a objetos (notações, diagramas)**" possui grau de importância **médio**.

Como o percentual de cobrança de cada assunto pode sofrer grandes variações, vamos classificar a importância de cada tema nos seguintes grupos:

Percentual de cobrança	Grau de importância
até 10%	baixo
de 10% a 20%	médio
acima de 20%	alto

Não posso deixar de alertá-lo que apesar do grau de importância, nenhum assunto deve ser deixado de lado. Lembre-se que as bancas sigam padrões, elas podem inovar e surpreender.

3 – ANÁLISE DE QUESTÕES

Vamos agora fazer uma análise detalhadas das questões da FCC. Cada questão servirá de ponto de partida para expandir o conhecimento relacionado. Desta forma, leia com bastante cuidado os comentários das questões. Não esqueça que os comentários complementam a parte teórica da aula.

1. (FCC / TRT-23ª REGIÃO (MT) – 2016)

Para a contagem de pontos de função, um Técnico do Tribunal montou a seguinte tabela com base em seus levantamentos iniciais:

Tipo de Função	Complexidade Funcional		
		Qt	Val
ALI	Simples	1	7
	Média	2	10
	Complexa	0	15
AIE	Simples	2	5
	Média	0	7
	Complexa	0	10
Entrada	Simples	0	3
	Média	1	4
	Complexa	6	6
Saída	Simples	0	4
	Média	1	5
	Complexa	2	7
Consulta	Simples	1	3
	Média	3	4
	Complexa	1	6

Com estes dados, o total de pontos de função transacional corretamente contados por ele foi

a) 37.

b) 5.

c) 117.

d) 80.

e) 15.

Comentários

Abaixo da tabela temos a informação mais importante para responder essa questão: "o total de pontos de **<u>função transacional</u>**". São elas: Entrada, Saída e Consulta. O cálculo deve ser realizado apenas com os valores dessas funções.

EE = 0 x 3 + 1 x 4 + 6 x 6 = 40.

SE = 0 x 4 + 1 x 5 + 2 x 7 = 19.

CE = 1 x 3 + 3 x 4 + 1 x 6 = 21.

40 + 19 + 21 = 80.

Gabarito: alternativa D.

2. (FCC / Prefeitura de Teresina-PI – 2016)

A equipe de Analistas de Sistemas da PRODATER reuniu-se para escolher uma metodologia de desenvolvimento capaz de atender às seguintes características do projeto e do cliente:

– Satisfazer o cliente através da entrega contínua e adiantada de software, em períodos curtos de tempo.

– Permitir que haja mudanças nos requisitos, mesmo tardiamente ao desenvolvimento.

– Pessoas de negócio e desenvolvedores devem poder trabalhar juntos por todo o projeto.

– As equipes devem ser auto-organizáveis.

Uma escolha que atende a lista de características é a metodologia

a) RAD – Rapid and Agile Development, muito adequada para projetos que envolvem altos riscos técnicos, como novas tecnologias.

b) Espiral, mas somente se o projeto for de pequeno porte, pois o uso da prototipagem aumenta os riscos de fracasso.

c) de Desenvolvimento Concorrente, embora tenha etapas sequenciais muito próximas do modelo em cascata.

d) RUP – Rational Unisied Process, o framework ágil mais utilizado por ser o mais simples e de fácil adoção e adaptação.

e) Extreme Programming (XP), que tem aderência à orientação a objetos como um paradigma de desenvolvimento e segue o princípio KIS – Keep it Simple.

Comentários

Analisando as alternativas temos:

a) ERRADA. RAD não é recomendado para projetos que envolvem altos riscos técnicos (Pressman)
b) ERRADA. Prototipagem é muito utilizado para elicitação de requisitos e não aumenta os riscos de fracasso
c) ERRADA. Cascata é sequencial; concorrente, como o próprio nome já diz, é executado em paralelo.
d) ERRADA. RUP é um framework verboso, muita documentação, artefato, não é de simples adoção
e) CORRETA.

Gabarito: alternativa E.

---

3. (FCC / ELETROBRAS-ELETROSUL – 2016)

Atualmente os softwares podem ser desenvolvidos utilizando-se métodos ágeis ou métodos tradicionais. A escolha da metodologia mais adequada vai depender de vários fatores, como

por exemplo, a característica de projeto, da empresa ou da gestão. Para fazer a escolha correta, é necessário ainda conhecer as características dos principais métodos e modelos de processo de desenvolvimento de software. Sobre estes métodos e modelos de processo é correto afirmar:

a) As metodologias ágeis são indicadas principalmente em casos em que os requisitos são bem compreendidos e provavelmente não sofrerão grandes alterações durante o desenvolvimento do sistema.

b) Os diagramas de Caso de Uso da UML são utilizados intensamente na fase de Elaboração do Rational Unified Process – RUP para criar um modelo de requisitos para o sistema.

c) Nos modelos em cascata os testes são desenvolvidos paralelamente aos requisitos, antes de iniciar o desenvolvimento, ajudando testadores e desenvolvedores a compreenderem os requisitos.

d) No Rational Unified Process – RUP o cliente participa do processo de desenvolvimento discutindo cenários com a equipe para gerar os cartões de estórias, que englobam as necessidades do cliente.

e) Sprinter e programação em pares são práticas descritas e amplamente utilizadas na eXtreme Programming – XP para agilizar o processo de desenvolvimento e reduzir a possibilidade de erros.

Comentários

Analisando as alternativas temos:

a) ERRADA. O indicado nesse caso é o modelo cascata.
b) CORRETA.
c) ERRADA. Não há paralelismo no modelo cascata. Todas as fases são sequenciais.
d) ERRADA. Cartões de estórias fazem parte do modelo XP.
e) ERRADA. Sprint é no modelo Scrum.

Gabarito: alternativa B.

---

4. (FCC / TRE-RR – 2015)

Dentre os princípios da Engenharia de Software NÃO se encontra:

a) Um sistema de software existe por uma razão: para fornecer valor aos seus usuários. Todas as decisões devem ser tomadas com este princípio em mente.

b) Keep it Simple: todo projeto deve ser tão simples quanto possível, mas simples não significa rápido e mal feito. Simplificar precisa de muito raciocínio e trabalho em várias interações.

c) Sempre especifique, projete e implemente sabendo que mais alguém terá de entender o que se está fazendo. Mais alguém irá usar, manter, documentar ou precisará entender o sistema que uma pessoa desenvolve.

d) Reuso sempre poupa tempo e esforço, assim, conseguir um alto nível de reuso é a principal meta a ser alcançada no desenvolvimento. O reuso torna mais barato o custo do sistema, pois produz componentes reusáveis.

e) Raciocinar clara e completamente antes da ação quase sempre produz os melhores resultados. Quando o raciocínio límpido é aplicado ao sistema, é mais provável que funcione adequadamente.

Comentários

Antes de responder a questão, vou apresentar um "macete" para lembrar os 7 princípios de Engenharia de Software, segundo Pressman:

1) a razão de existir - O Software de existir por um motivo
2) KISS (Keep It Simple, Stupid!, ou seja: não complique!)
3) mantenha a visão - Devendo ser clara e consistente
4) o que um produz outros consomem - Documentação e Clareza
5) esteja aberto para o futuro - Desenvolva pensando na Evolução
6) planeje com antecedência, visando a reutilização
7) pense! - Depois faça

Agora com essa informação fica mais fácil de responder à questão.

A alternativa que responde à questão é a letra D.

Gabarito: alternativa D.

---

5. (FCC / TRE-AP – 2015)

A fase de projeto de software possui duas atividades básicas: projeto da arquitetura e projeto detalhado. Nesta fase

a) o sistema é codificado, a partir da descrição computacional do sistema, em uma linguagem que torna possível a compilação e a geração do código-executável para o software.

b) em um processo de desenvolvimento orientado a objetos, são criadas as classes de objetos do sistema utilizando-se ferramentas CASE e bibliotecas de classes preexistentes para agilizar a implementação.

c) em um processo de desenvolvimento orientado a objetos, o projeto da arquitetura visa distribuir as classes de objetos relacionadas do sistema em subsistemas e seus componentes, distribuindo-os pelos recursos de hardware disponíveis.

d) os diversos módulos do sistema são integrados, resultando no produto de software.

e) o projeto de arquitetura realiza a modelagem das relações de cada módulo do sistema, com o objetivo de implantar as suas funcionalidades. Além disso, são implementados os projetos de interface com o usuário e de banco de dados.

Comentários

O projeto possui duas atividades básicas: projeto da arquitetura (ou projeto de alto nível), e projeto detalhado (ou projeto de baixo nível). Em um processo de desenvolvimento orientado a objetos, o projeto da arquitetura normalmente é realizado por um arquiteto de software. O projeto da arquitetura visa distribuir as classes de objetos relacionados do sistema em subsistemas e seus componentes, distribuindo também esses componentes pelos recursos de hardware disponíveis. Já no projeto detalhado, são modeladas as relações de cada módulo com o objetivo de realizar as funcionalidades do módulo. Além de desenvolver o projeto de interface com o usuário e o projeto de banco de dados. Portanto, a alternativa correta é a letra C.

Gabarito: alternativa C.

---

6. (FCC / TRE-PB – 2015)

Análise de Pontos de Função – APF é uma técnica para medir o tamanho funcional de um software cujo processo de medição envolve diversas etapas, dentre elas, a medição das funções de dados, que envolvem as funcionalidades fornecidas pelo sistema ao usuário para atender a suas necessidades de armazenamento de dados. Dentre as funções de dados estão

a) os Arquivos de Ponto de Controle – APC.

b) as Saídas Externas – SE.

c) as Entradas Externas – EE.

d) os Arquivos de Interface Externa – AIE.

e) as Consultas Externas – CE.

Comentários

Na aula expliquei que existem dois grupos de componentes de pontos de função. São eles: Função do tipo dado que inclui: os Arquivos Lógicos Internos (ALI) e os Arquivos de Interface Externa (AIE); e Função do tipo transação que inclui: Consulta Externa (CE), Saída Externa (SE) e Entrada Externa (EE). Logo as alternativas B, C e E que fazem referência a funções do tipo transação, são descartadas. Nossa resposta está na letra D - Arquivo de Interface Externa (AIE).

Gabarito: alternativa D.

---

7. (FCC / TRT-9ª REGIÃO (PR) – 2015)

A Análise por Pontos de Função é uma técnica paramétrica para estimativa de esforço para desenvolvimento de software. Esta técnica

a) é baseada no número de linhas de código produzidas.

b) é aplicável antes dos requisitos funcionais terem sido definidos.

c) não pode ser usada para calcular o custo-benefício de software comprado.

d) é normatizada internacionalmente pelo International Function Point Users Group – IFPUG.

e) não é adequada para medir a produtividade de uma equipe de desenvolvimento.

Comentários

A Análise por Pontos de Função foi proposta por Allan Albrecht em 1979 e formalizado em 1984 pela IBM, e em 1986 foi criado o IFPUG (International Function Points Users Group). Portanto, a alternativa correta é a letra D.

Gabarito: alternativa D.

---

8. (FCC / SABESP – 2014)

Um processo de engenharia de software é formado por um conjunto de passos parcialmente ordenados, relacionados com artefatos, pessoas, recursos, estruturas organizacionais e restrições, tendo como objetivo produzir e manter os produtos de software finais requeridos. Sobre estes processos é INCORRETO afirmar que

a) usualmente considera-se que a primeira grande divisão de um processo é a fase, que consiste em um período de tempo no qual determinadas atividades com objetivos bem específicos são realizadas.

b) a maioria dos processos de software é organizada em torno de tarefas (às vezes também chamadas de atividades). Toda atividade tem um objetivo principal estabelecido e visa criar ou produzir uma mudança de estado visível em um ou mais artefatos durante a execução de um projeto.

c) todos os modelos de processo têm fases cíclicas, ou seja, o desenvolvimento passa repetidamente de uma fase para a outra, formando um ciclo repetitivo de fases até a finalização do projeto.

d) na descrição de um processo, as atividades devem ser atribuídas a perfis ou cargos, e não a pessoas. Apenas quando o processo for usado em um projeto concreto é que deve haver atribuições de atividades a pessoas.

e) um modelo de processo é um conjunto de regras mais abstratas que especificam a forma geral de processos. Apresenta uma filosofia e uma forma geral de comportamento com base na qual processos específicos devem ser definidos.

Comentários

Entre as alternativas a única que não condiz com os processos de software é a letra C. Porque por exemplo o modelo em cascata é um exemplo de modelo sequencial e linear, invalidando assim a afirmativa de que “todos os modelos de processo têm fases cíclicas”.

Gabarito: alternativa C.

---

9. (FCC / TJ-AP – 2014)

Na Análise de Pontos de Função, são contados diversos componentes, dentre os quais NÃO se encontra(m):

a) Arquivos Lógicos Internos, que correspondem aos arquivos mantidos e utilizados pelo sistema sendo contado.

b) Arquivos de Interface Externa, que correspondem aos arquivos utilizados pelo sistema sendo contado, porém mantidos por outros sistemas.

c) Entradas Externas, que correspondem a transações cujo objetivo é a manutenção de arquivos ou a alteração do comportamento do sistema.

d) Consultas Externas, que correspondem a transações cujo objetivo é a apresentação de informações aos usuários, provenientes dos arquivos, sem a geração de dados derivados, atualização de arquivos ou a utilização de cálculos/fórmulas.

e) Esforço de Desenvolvimento, que corresponde ao tamanho funcional de um software e ao esforço gasto no seu desenvolvimento, medido em pessoas-hora, resultando na quantidade de horas e, consequentemente, no preço estimado de desenvolvimento do sistema.

Comentários

Entre as alternativas a única que apresenta componentes que não são contados na Análise de Pontos de Função é a letra E. Os requisitos não-funcionais (restrições) não são levados em conta em uma contagem não-ajustada justamente porque a APF mede as funcionalidades do software.

Gabarito: alternativa E.

---

4 – Orientações de Revisão e Pontos a Destacar

O Processo de Desenvolvimento de Software é abordado como um conceito dentro de Engenharia de Software. Por isso, vamos destacar tópicos dessa tema para termos base para nossa aula.

Conceitos de Engenharia de Software

O termo **Engenharia de Software** foi criado na década de 1960 e utilizado oficialmente em 1968 na *NATO Conference on Software Engineering* (Conferência sobre Engenharia de Software da OTAN). Ele foi criado na tentativa de contornar a dificuldade do desenvolvimento de softwares frente ao crescimento da demanda, conhecida na década de 1970 como crise do software, além de trazer uma abordagem de engenharia para o desenvolvimento de sistemas complexos.

Segundo Friedrich Ludwig Bauer[1], "Engenharia de software é a criação e a utilização de sólidos princípios de engenharia a fim de obter software de maneira econômica, que seja confiável e que trabalhe eficientemente em máquinas reais".

De acordo com o IEEE[2], "Engenharia de software é a aplicação de uma abordagem sistemática, disciplinada e quantificável, para o desenvolvimento, operação e manutenção do software; isto é, a aplicação de engenharia ao software.".

Engenharia de software é uma abordagem sistemática e disciplinada para o desenvolvimento de software (PRESSMAN[3], 2006).

Em suma, a Engenharia de Software é uma área do conhecimento da Computação que busca estruturar de forma racional e científica, através do uso de modelos matemáticos, a especificação, desenvolvimento e manutenção de sistemas de software aplicando tecnologias e métodos da Ciência da Computação, Gerência de projetos, das Engenharias e outros campos do conhecimento.

A partir destes conceitos, chegamos à conclusão que Engenharia de Software não é simplesmente programação.

DEFINIÇÃO DE SOFTWARE

Em uma visão restritiva, o termo software é associado aos programas / aplicativos de computador. Entretanto, seu conceito vai muito mais além. O software são códigos em linguagem natural ou não, entendíveis por um sistema computacional, assim como qualquer dado que faça com que opere corretamente. Ou seja, ele não é apenas o programa, mas sim todos os dados e configurações necessários para que o programa funcione corretamente.

O software é o conjunto de vários artefatos e não apenas o código fonte (SOMMERVILLE, 2003).

De acordo Pressman (2006) os softwares estão divididos nas seguintes categorias:

- **Software de sistema**

São programas que apoiam outros programas, como por exemplo o software que realiza a comunicação com o hardware (sistema operacional) e software que ajuda na construção de outro software (compiladores).

---

[1] Cientista da computação alemão, autor da primeira definição de engenharia de software.

[2] Instituto de Engenheiros Eletricistas e Eletrônicos (http://www.ieee.org.br/) - sociedade técnico-profissional internacional, dedicada ao avanço da teoria e prática da engenharia nos campos da eletricidade, eletrônica e computação.

[3] Roger S. Pressman é um engenheiro de software, escritor e consultor, norte-americano. Autoridade reconhecida internacionalmente nas tecnologias em melhoria de processos de software e engenharia de software.

- **Software de aplicação**

São os programas desenvolvidos para auxiliar na execução de funções específicas para o negócio de uma empresa, por exemplo o processamento de dados comerciais ou técnicos.

- **Software cientifico e de engenharia**

São programas que auxiliam no processamento de números em massa, por exemplo na astronomia, meteorologia, biologia molecular.

- **Software embutido (embarcado)**

São programas desenvolvidos para serem executados dentro de produtos específicos, como por exemplo o sistema de teclas digitais de um forno micro-ondas.

- **Software para linhas de produtos**

São conhecidos como software de prateleira, e projetados para prover capacidade específica de utilização por muitos clientes.

- **Aplicações Web / aplicativos móveis**

São programas voltados para as redes e abrangem uma série de variedades, contemplando aplicativos para dispositivos móveis.

- **Software de inteligência artificial**

São softwares que utilizam algoritmos não numéricos para solucionar problemas complexos, não passíveis de computação ou análise direta. Este tipo software está presente na robótica, no reconhecimento de voz e imagem, nos jogos.

- **Software legado**

São os sistemas de uma organização que, apesar de serem bastante antigos, fornecem serviços essenciais e geralmente utilizam bancos de dados obsoletos. Esses sistemas podem sofrer com dois extremos, ou são continuamente modificados para se adequar às mudanças, ou nunca são alterados quando atendem as necessidades do cliente. A grande preocupação em relação a eles está na baixa qualidade, pois muitas vezes não existem documentações, ou se existem são pobres de detalhes.

Software Livre

Também existe o conceito de software livre, que remete ao programa que dá liberdade ao utilizador, permitindo que ele o estude, modifique e compartilhe com outras pessoas. Para tanto, é necessário que o utilizador tenha acesso ao código-fonte, para altera-lo de acordo com as suas necessidades. Por conta da confusão em relação a este termo, onde empresas proprietárias de softwares utilizam

para associa-lo a cópia disponibilizada sem nenhum custo, o software livre também é dividido em categorias:

- **Software com copyleft**

É um software livre cujos termos de distribuição não permitem que os redistribuidores incluam restrições adicionais quando redistribuem ou modificam o software. Isso significa que toda cópia, mesmo modificada, precisa ser software livre.

- **Software livre sem copyleft**

É diferente da categoria anterior, pois o autor permite que a redistribuição e modificação possa conter restrições adicionais.

- **Domínio público**

Software não sujeito ao copyright. Este é um caso especial de software livre sem copyleft, onde as cópias ou versões modificadas podem não ser livres.

- **Software com licença GPL**

A licença GPL (*General Public License*) é a definição de licença de software idealizada por Richard Stallman em 1989, onde os softwares derivados de um produto originalmente licenciado pela GPL só podem ser distribuídos se utilizarem a mesma licença.

ESCLARECENDO

Copyleft significa o direito de permissão de cópia de uma obra por outros usuários, dando a liberdade de copiar, modificar e redistribuir, exigindo que esse direito seja mantido em todas as versões modificadas.

PRINCÍPIOS DA ENGENHARIA DE SOFTWARE

Os princípios da Engenharia de Software constituem a base dos métodos, tecnologias, metodologias e ferramentas adotadas que norteiam a prática de desenvolvimento de software. Os princípios se aplicam ao processo e ao produto de software através da adoção de métodos e técnicas. Os princípios-chave são:

- **Rigor e Formalidade**

Ao considerar que a engenharia de software deve ser realizada de maneira sistemática, o rigor é um complemento necessário a criatividade, que visa aumentar a confiança dos desenvolvedores. A formalidade é o rigor no seu nível mais elevado. Nesse princípio estão incluídas por exemplo a documentação rigorosa dos passos de desenvolvimento e gerenciamento, a análise sistemática dos dados de testes, a avalição dos prazos de entrega.

- **Separação de Interesses**

Abrange o domínio sobre a complexidade do software, separando os problemas principais e investindo esforços para solução de um de cada vez, dividindo as responsabilidades. Como exemplo temos o desenvolvimento por fases, separando os interesses por atividades e respeitando o tempo estipulado.

- **Modularidade**

Na modularidade um sistema complexo pode ser dividido em peças mais simples, denominadas módulos. Esse princípio depende do suporte da separação de interesses, para que ao lidar com um módulo específico seja possível ignorar os detalhes dos outros módulos. Como característica desse princípio, os módulos devem ter um alto nível de coesão, onde existe uma forte relação entre eles, mas com um baixo acoplamento, onde a interação entre eles seja baixa, possibilitando que eles sejam compreendidos como unidades em separado.

- **Abstração**

Esse conceito visa a identificação de aspectos importantes, ignorando os detalhes. Por exemplo, os botões de um relógio que são sua interface, entretanto eles são a abstração em relação ao ajuste da hora.

- **Antecipação a Mudanças**

Esse princípio este diretamente relacionado ao suporte a evolução de um software, considerando aspectos relacionados ao processo de evolução e compatibilidade com mudanças futuras.

- **Generalidade**

É o princípio que visa durante a resolução de um problema descobrir se ele é apenas uma instância de um problema mais abrangente, onde a solução pode ser reutilizada em outros casos.

- **Incrementação**

É relacionada a evolução de um software através de incrementos estruturados. Pode ser realizado através da entrega de subconjuntos de um sistema desde cedo, visando coletar o feedback dos usuários e adicionar funcionalidades de forma incremental.

CICLO DE VIDA DO SOFTWARE

Agora vamos iniciar nosso estudo sobre o ciclo de vida. Esse termo surgiu no contexto da biologia, onde trata do ciclo de vida de um ser vivo, ou seja, do conjunto de transformações pelas quais passam os indivíduos de uma espécie durante a vida.

Na imagem acima temos o exemplo do ciclo de vida do homem. Dessa forma, podemos extrair o conceito que o ciclo de vida trata das fases que um ser vivo passa desde o seu nascimento até a sua morte e aplicar esse conceito em diversos contextos.

Na Engenharia de Software esse termo normalmente é aplicado a sistemas artificias com o significado de mudanças que acontecem na vida de um produto de software e segue o mesmo princípio do conceito de ciclo de vida de um ser vivo.

Steve McConnell afirma que um modelo de ciclo de vida do software é uma representação que descreve todas as atividades que tratam da criação de um produto de software.

O ciclo de vida é a estrutura contendo processos, atividades e tarefas envolvidas no desenvolvimento, operação e manutenção de um produto de software, abrangendo a vida do sistema, desde a definição de seus requisitos até o término de seu uso.

Portanto, 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. Esse conceito é simples de entender, o problema em relação ao ciclo de vida do software, de acordo com os autores, está nas dezenas e diferentes fases para os ciclos de vida.

Infelizmente não há consenso entre os autores e além disso, alguns deles com o passar do tempo mudam de opinião. Portanto, aqui chegamos a minha sugestão após analisar as últimas provas da FCC em relação ao custo/benefício de estudar uma vasta lista de vários modelos de ciclos de vida do software, é muito alto. Sugiro que você guarde espaço na memória para assuntos que são mais recorrentes. De toda forma vou citar alguns modelos de ciclos de vida do software para você compreender como eles funcionam e as diferenças entre eles.

O modelo de ciclo de vida é a primeira escolha a ser feita no processo de software. A partir desta escolha será definida desde a maneira mais adequada de obter as necessidades do cliente, até quando e como o cliente receberá sua primeira versão operacional do sistema.

Diante disso, vou apresentar os seguintes modelos:

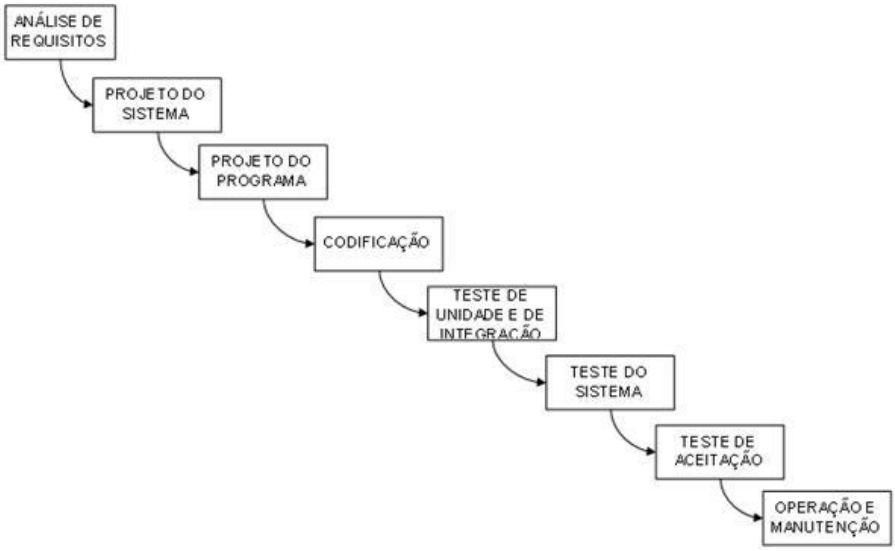
- Cascata
- Modelo em V
- Incremental
- RAD - *Rapid Application Development*
- Prototipagem

Modelo em Cascata

Formalizado por Royce em 1970, é o modelo mais antigo. Suas atividades fundamentais são:

- análise e definição de requisitos;
- projeto;
- implementação;
- teste;
- integração.

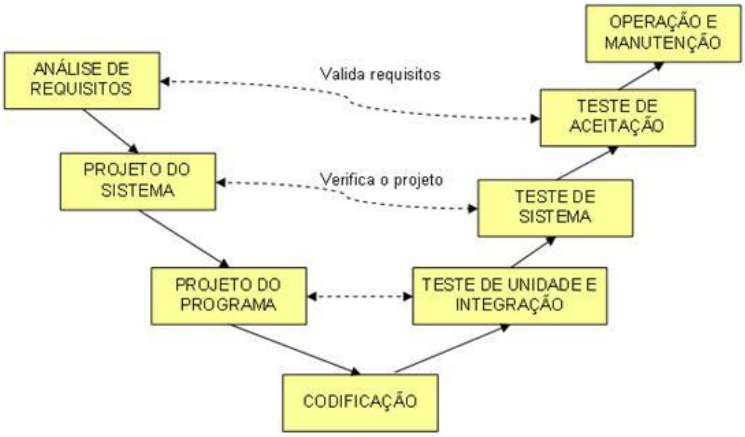
O modelo em cascata tem o grande mérito de ser o primeiro a impor o planejamento e o gerenciamento ao processo de software, que antes era casual. O nome "cascata" foi atribuído em razão da sequência das fases, onde cada fase só começa quando a anterior termina; e da transmissão do resultado da fase anterior como entrada para a fase atual (o fim de cada fase resulta em um documento aprovado). Nesse modelo, portanto, é dada muita ênfase às fases de análise e projeto antes de partir para a programação, a fim de que o objetivo do software esteja bem definido e que sejam evitados retrabalhos, conforme podemos observar na imagem abaixo.



Devido à sua simplicidade, o modelo em cascata é fácil de ser entendido pelo cliente. É um modelo que supõe um início e fim claro e determinado, assim como uma estimativa precisa de custo logo no início, fatores importantes na conquista do cliente.

Modelo em V

Neste modelo, do Ministério de Defesa da Alemanha, 1992, o modelo em cascata é colocado em forma de "V". Do lado esquerdo do V ficam da análise de requisitos até o projeto, a codificação fica no vértice e os testes, desenvolvimento, implantação e manutenção, à direita, como podemos observar na imagem abaixo.



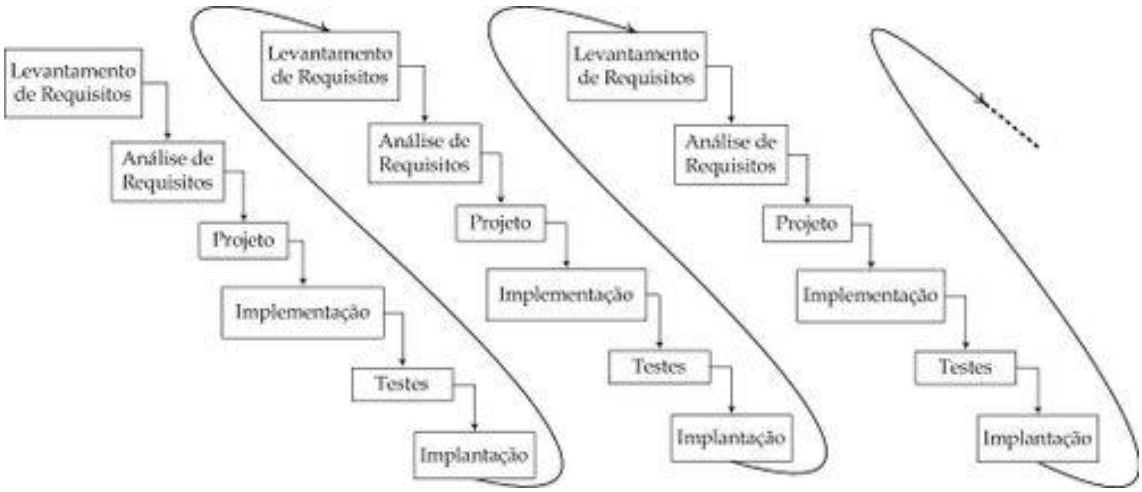
A característica principal desse modelo, que o diferencia do modelo em cascata, é a ênfase dada à verificação e validação: cada fase do lado esquerdo gera um plano de teste a ser executado no lado direito.

Da mesma forma que o modelo em cascata, o cliente só recebe a primeira versão do software no final do ciclo, mas apresenta menos risco, devido ao planejamento prévio dos testes nas fases de análise e projeto.

Modelo Incremental

Neste modelo, de Mills em 1980, 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 passará por todas as fases "cascata" de projeto, conforme se observa na imagem abaixo, e será entregue ao cliente um software operacional. Assim, o cliente receberá parte do produto final em menos tempo.



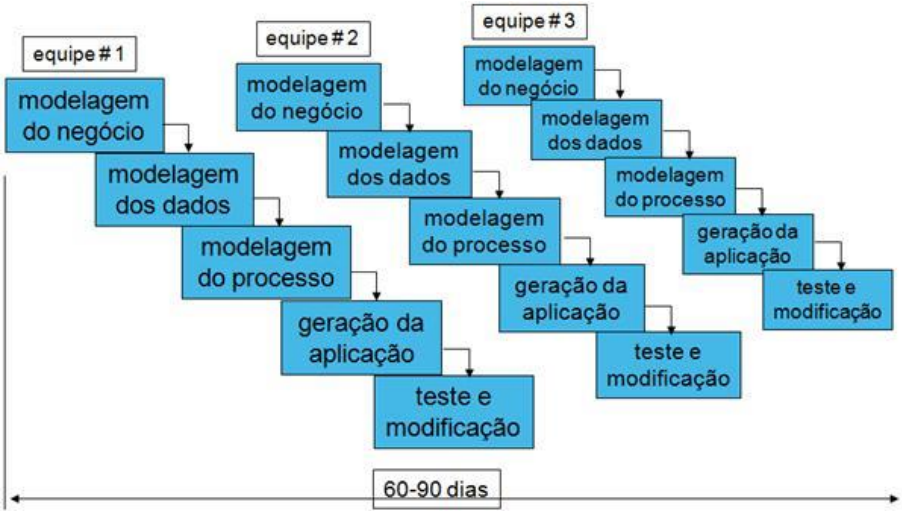
Esse ciclo de vida não exige uma equipe muito grande, pois a modularização diminui o escopo de cada incremento, e não há um paralelismo nas atividades. Haverá, por outro lado, uma dificuldade em manter a documentação de cada fase atualizada devido às melhorias no sistema e aos ajustes de requisitos solicitados pelos clientes.

Modelo RAD - Rapid Application Development

Este modelo formalizado por James Martin em 1991, como uma evolução da "prototipagem rápida", destaca-se pelo desenvolvimento rápido da aplicação. O ciclo de vida é extremamente comprimido, de forma a encontrarem-se exemplos, na literatura, de duração de 60 e 90 dias. É ideal para clientes buscando lançar soluções pioneiras no mercado.

É um ciclo de vida incremental, iterativo, onde é preferível que os requisitos tenham escopo restrito. A diferença principal do ciclo anterior é o forte paralelismo das atividades, requerendo, assim, módulos bastante independentes. Aqui os incrementos são desenvolvidos ao mesmo tempo, por equipes diferentes.

Além do paralelismo, a conquista do baixo tempo se dá graças à compressão da fase de requisitos e da fase de implantação. Isso significa que, na obtenção dos requisitos, costumam-se optar por metodologias mais dinâmicas e rápidas, como workshops ao invés de entrevistas. Permite-se também um desenvolvimento inicial no nível mais alto de abstração dos requisitos visto o envolvimento maior do usuário e visibilidade mais cedo dos protótipos.



Os sistemas desenvolvidos no ciclo RAD tendem a ter uma padronização de telas muito forte, devido a bibliotecas reutilizáveis e templates, porém tendem a perder em desempenho do sistema e na análise de risco (atividades estas que demandam tempo em qualquer projeto). Assim, é preferível seu uso para softwares de distribuição pequena.

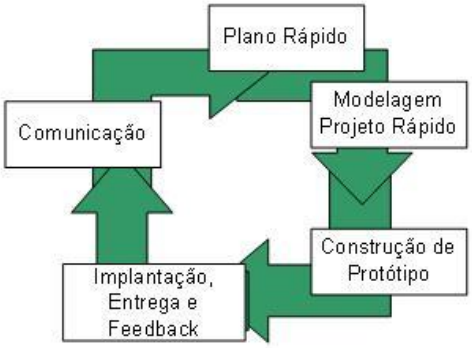
Modelo com Prototipagem

É a construção de um exemplar do que foi entendido dos requisitos capturados do cliente. Pode ser considerado um ciclo de vida ou pode ser usado como ferramenta em outros ciclos de vida.

Um protótipo em engenharia de software pode ser o desenho de uma tela, um software contendo algumas funcionalidades do sistema. São considerados operacionais (quando já podem ser utilizados pelo cliente no ambiente real, ou seja, em produção), ou não operacionais (não estão aptos para serem utilizados em produção). Os protótipos podem ser descartados, ou reaproveitados para evoluírem até a versão final.

No ciclo de vida de prototipagem, não é exigido um conhecimento aprofundado dos requisitos num primeiro momento. Isso é bastante útil quando os requisitos não são totalmente conhecidos, são muitos complexos ou confusos. Desta forma, se o cliente não sabe expressar o que deseja (o que ocorre bastante quando não é um sistema legado), a melhor maneira de evitar que se perca tempo e recursos com uma má interpretação é a construção de modelos, ou seja, de protótipos do que o software faria.

Assim, o cliente experimentará, na prática, como o sistema ou parte dele funcionará. A partir desse primeiro contato, o cliente esclarece o que não foi bem interpretado, aprofunda alguns conceitos e até descobre um pouco mais sobre o que realmente precisa. A partir deste feedback, novos requisitos são colhidos e o projeto ganha maior profundidade. Outro protótipo é gerado e apresentado ao cliente, que retorna com mais feedbacks. Ou seja, o cliente participa ativamente do início ao fim do processo.



É importante relembrar que conforme expliquei anteriormente, não existe um modelo ideal e na maioria dos softwares desenvolvidos são utilizados mais de um modelo de ciclo de vida. Abaixo inseri uma tabela de comparação entre os modelos de ciclo de vida do software apresentados.

Modelo	Foco	Requisitos	Primeira Versão para Cliente	Gerenciamento (escala de 1 a 5 de complexidade)
Cascata	Documento e artefato	Bem conhecido / congelado	Fim do ciclo	1
V	Planejamento de testes	Bem conhecido / congelado	Fim do ciclo	2
Incremental	Incrementos operacionais	Maior abstração / Tratado em módulos	Protótipos operacionais	3
RAD	Rapidez	Escopo restrito / Maior abstração / Tratado em módulos	Protótipos operacionais	4
Prototipagem	Dúvidas nos Requisitos	Abstratos	Protótipos não operacionais	5

PROCESSO DE SOFTWARE

O processo de software é o conjunto de atividades que constituem o desenvolvimento de um sistema computacional. Estas atividades são agrupadas em fases, como: definição de requisitos, análise, projeto, desenvolvimento, teste e implantação. Em cada fase são definidas, além das suas atividades, as funções e responsabilidades de cada membro da equipe, e como produto resultante, os artefatos. O que diferencia um processo de software do outro é a ordem em que as fases vão ocorrer, o tempo e a ênfase dados a cada fase, as atividades presentes, e os produtos entregues.

As atividades do processo de desenvolvimento de software incluem:

- **Análise de requisitos de software**

A extração dos requisitos de um desejado produto de software é a primeira tarefa na sua criação. Embora o cliente, provavelmente, acredite saber o que o software deva fazer, esta tarefa requer habilidade e experiência em Engenharia de Software para reconhecer a incompletude, ambiguidade ou contradição nos requisitos.

- **Especificação**

A especificação é a tarefa de descrever precisamente o software que será escrito, preferencialmente de uma forma matematicamente rigorosa. Na prática, somente especificações mais bem-sucedidas foram escritas para aplicações bem compreendidas e afinadas que já estavam bem desenvolvidas, embora sistemas de software de missão crítica sejam frequentemente bem especificados antes do desenvolvimento da aplicação. Especificações são mais importantes para interfaces externas que devem permanecer estáveis.

- **Arquitetura de Software**

A arquitetura de um sistema de software remete a uma representação abstrata daquele sistema. Arquitetura é concernente à garantia de que o sistema de software irá ao encontro de requisitos do produto, como também assegurar que futuros requisitos possam ser atendidos. A etapa da arquitetura também direciona as interfaces entre os sistemas de software e outros produtos de software, como também com o hardware básico ou com o sistema operacional.

- **Implementação (ou codificação)**

A transformação de um projeto para um código deve ser a parte mais evidente do trabalho da engenharia de software, mas não necessariamente a sua maior porção.

- **Teste**

Teste de partes do software, especialmente onde tenha sido codificado por dois ou mais engenheiros trabalhando juntos, é um papel da engenharia de software.

- **Documentação**

Uma importante tarefa é a documentação do projeto interno do software para propósitos de futuras manutenções e aprimoramentos. As documentações mais importantes são das interfaces externas.

- **Suporte e Treinamento de Software**

Uma grande porcentagem dos projetos de software falha pelo fato de o desenvolvedor não perceber que não importa quanto tempo a equipe de planejamento e desenvolvimento irá gastar na criação do software se ninguém da organização irá usá-lo. As pessoas ocasionalmente resistem à mudança e evitam aventurar-se em áreas pouco familiares. Então, como parte da fase de desenvolvimento, é muito importante o treinamento para os usuários de software mais entusiasmados, alternando o

treinamento entre usuários neutros e usuários favoráveis ao software. Usuários irão ter muitas questões e problemas de software os quais conduzirão para a próxima fase.

- **Manutenção**

A manutenção e melhoria de software lidam com a descoberta de novos problemas e requisitos. Ela pode tomar mais tempo que o gasto no desenvolvimento inicial do mesmo. Não somente pode ser necessário adicionar códigos que combinem com o projeto original, mas determinar como o software trabalhará em algum ponto depois da manutenção estar completa, pode requerer um significativo esforço por parte de um engenheiro de software. Cerca de 2/3 de todos os engenheiros de software trabalham com a manutenção, mas estas estatísticas podem estar enganadas. Uma pequena parte destes trabalha na correção de erros. A maioria das manutenções são para ampliar os sistemas para novas funcionalidades, as quais, de diversas formas, podem ser consideradas um novo trabalho. Analogamente, cerca de 2/3 de todos os engenheiros civis, arquitetos e construtores trabalham com manutenção de uma forma similar.

UML – Unified Modeling Language

A necessidade de um técnica-padrão que visa cobrir todas as etapas de desenvolvimento de um sistema de informação. O espectro que vai da análise de requisitos, passando pela modelagem, projeto, implementação até chegar a implantação. Um destas técnicas é conhecida como UML – *Unified Modeling Laguage*. Ela oferece um mecanismo na forma de notação diagramática e sintaxe de linguagem associada para cobrir todo o ciclo de vida.

A UML combina conceitos comumente aceitos de muitos métodos e metodologias orientados a objetos (O-O). Ela é genérica independente de linguagem e plataforma. A UML tem muitos tipos diagramas que podem ser divididos em duas categorias:

> **Diagramas estruturais**. Estes descrevem os relacionamentos estruturais ou estáticos entre objetos de esquema, objetos de dados componentes de software. Incluem os diagramas de classes, de objetos, de componentes e de implementação.
> **Diagramas comportamentais**. Sua finalidade é descrever o comportamento ou relacionamento dinâmico entre componentes. Incluem diagramas de caso de uso, de seq uência, de colaboração, de estados e de atividades.

Pense que um sistema tem aspectos estáticos e dinâmicos, esses dois tipos de diagrama descrevem essas características. Vamos dar um exemplo de cada um dos tipos de diagrama apenas para ajudar você na fixação do conteúdo. O **Diagrama de Classes** oferece um ótimo exemplo do tipo de diagrama estrutural e fornece um conjunto inicial de elementos de notação que todos os outros diagramas de estrutura usam. O propósito do diagrama de classes é mostrar os tipos que estão sendo modelados no sistema.

Uma classe é representada na forma de um retângulo, contendo duas linhas que separam 3 partes. A primeira contém no nome da classe, a segunda os atributos da classe e a última os métodos da mesma. Vejamos uma figura que ajude a entender melhor esses conceitos:

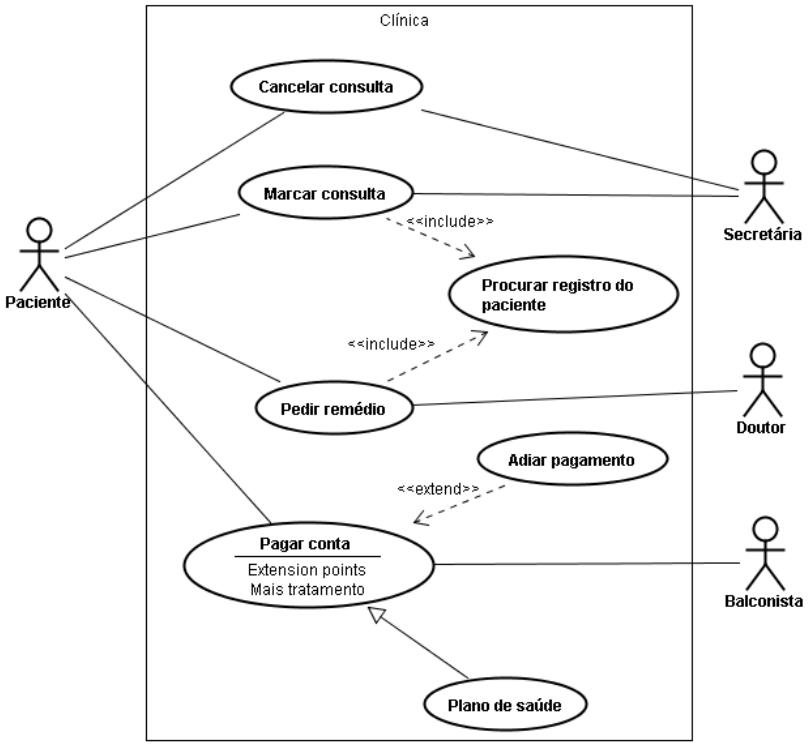
Concurseiro
nome: String concursoAlvo: String identidade: Integer dataDeNascimento: Date
getConcursoAlvo() : String getIdade(): Integer

Observe a classe acima, temos a classe Concurseiro com os atributos nome, concursoAlvo, indentidade e dataDeNascimento. Veja que cada atributo tem um tipo de dado associado. Por fim, o retângulo mostra as operações que podem ser executadas com os dados desta classe, conhecidos como métodos. Um detalhamento maior sobre o que seria um diagrama de classe pode ser visto neste artigo[4].

O diagrama comportamental que usaremos como exemplo é o diagrama de Casos de Uso. O diagrama de casos de uso tem o objetivo de **auxiliar a comunicação entre os analistas e o cliente**. Ele descreve um cenário que mostra as funcionalidades do sistema do ponto de vista do usuário. O cliente deve ver no diagrama de casos de uso as principais funcionalidades de seu sistema.

O diagrama de Caso de Uso é representado por atores, casos de uso e relacionamentos entre estes elementos. Um **ator** é representado por um boneco e um rótulo com o nome do ator. Um ator é um usuário do sistema, que pode ser um usuário humano ou um outro sistema computacional. Um **caso de uso** é representado por uma elipse e um rótulo com o nome do caso de uso. Um caso de uso define uma grande função do sistema. Os relacionamentos ajudam a descrever casos de uso. A figura abaixo descreve um diagrama de caso de uso. Nela temos os atores Paciente, Secretária, Doutor e Balconista que fazem acesso a diferentes macro funcionalidades do sistema, ou seja, os casos de uso.

[4] https://www.ibm.com/developerworks/br/rational/library/content/RationalEdge/sep04/bell/index.html



Antes de você pergunte o que são os termos <<include>> e <<extends>> na figura acima, deixa eu tentar explicar de uma forma bem simples. O <<include>> afirma que um caso de uso depende do outro, tente observar na figura, para marcar uma consulta é necessário ter os dados do paciente. Já o <<extends>> incluí uma funcionalidade extra que não é obrigatória, por exemplo, adiar pagamento é uma possibilidade na ação de pagar conta.

Forte abraço e bons estudos.

THIAGO CAVALCANTI
PROFESSOR

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.