Projeto de Desenvolvimento de Software

Ivna Valença de Oliveira

Curso Técnico em Informática

Educação a Distância

2016

EXPEDIENTE

Professor Autor
Ivna Valença de Oliveira

***Design* Instrucional**
Deyvid Souza Nascimento
Maria de Fátima Duarte Angeiras
Renata Marques de Otero
Terezinha Mônica Sinício Beltrão

Revisão de Língua Portuguesa
Letícia Garcia

Diagramação
Izabela Cavalcanti

Coordenação
Anderson Elias

Coordenação Executiva
George Bento Catunda

Coordenação Geral
Paulo Fernando de Vasconcelos Dutra

Conteúdo produzido para os Cursos Técnicos da Secretaria Executiva de Educação Profissional de Pernambuco, em convênio com o Ministério da Educação (Rede e-Tec Brasil).

Setembro, 2016

O48p

Oliveira, Ivna Valença de.

Projeto de Desenvolvimento de Software: Curso Técnico em Informática: Educação a distância / Ivna Valença de Oliveira. – Recife: Secretaria Executiva de Educação Profissional de Pernambuco, 2016.

34 p.: il.

Inclui referências bibliográficas.

1. Educação a distância. 2. Informática – Desenvolvimento de software. 3. Qualidade de software. I. Oliveira, Ivna Valença de. II. Título. III. Secretaria Executiva de Educação Profissional de Pernambuco. IV. Rede e-Tec Brasil.

CDU – 004.41

Sumário

Introdução

Caro (a) estudante, seja bem-vindo(a) a esta nova disciplina. Fico feliz em poder compartilhar um pouco do meu conhecimento com você. Lembre-se de que um estudante sempre vai além. Use este caderno como guia, mas não fique preso a ele. Busque, pesquise, converse, e principalmente, questione. Nunca aceite tudo que é dito sem exercitar seu pensamento crítico. Todos nós temos bagagens de vida diferentes, que nos permitem ter visões distintas sobre o que é estudado.

Esta disciplina busca dar uma visão macro sobre o projeto de desenvolvimento de *softwares*. Teremos duas competências em que apresentaremos, primeiramente, uma introdução sobre o tema e, depois, falaremos sobre qualidade de *software*.

Espero que você aproveite a disciplina. Leia o caderno, assista aos vídeos, interaja no fórum, faça as atividades... estude! Como diria Albert Einstein, “a mente que se abre a uma nova ideia, jamais voltará ao seu tamanho original”.

Bons estudos!

Ivna Valença de Oliveira.

1.Competência 01 | Introdução a Projeto de Software

Imagine que você está conversando com um amigo. Nesta conversa, você fala que está fazendo um curso técnico de informática. O seu amigo fica muito animado e já começa a dizer como ele gostaria de ter um *software,* para auxiliar nas vendas do seu mercadinho. O seu amigo e potencial cliente descreve por alto como gostaria que funcionasse o *software*: “Quero colocar o meu mercadinho na Internet. Posso ter um mercado pequeno, mas, na Internet, quero ser grande!”. Em seguida, ele pergunta o prazo e o custo.

E agora? Como responder ao seu cliente? Em quanto tempo você irá desenvolver este projeto? Quanto você irá cobrar? Devo começar já codificando e pensar depois?

Você verá nesta disciplina que é necessário seguir algumas etapas, que chamaremos de processo, se quisermos desenvolver um *software*. Para explicar o que é um processo de desenvolvimento de *software,* temos que desvendar alguns mistérios antes. Vamos começar entendendo o que é *software*.

1.1 O que é Software?

Pare e pense: *software* é um programa de computador? O que é um programa? Um programa de computador é um conjunto de soluções algorítmicas, codificadas em uma linguagem de programação e executada em uma máquina real ou virtual.

Então, vamos voltar à pergunta inicial: um *software* é um programa de computador? Não! Segundo Pressman, o *software* é mais do que um programa, o *software* é mais do que apenas o código.

O *software* é um programa de computador associado a uma documentação.

Existem dois tipos de *software*: genéricos e personalizados. Os genéricos são desenvolvidos para o público geral e são também conhecidos como *softwares* de “prateleira”. Alguns exemplos de

software genéricos são: processadores de texto, *software* de desenho, ferramentas utilitárias etc. A outra categoria são os *softwares* personalizados, desenvolvidos sob encomenda para um cliente (interno ou externo) em particular.

As principais características dos *softwares* são:

- **Invisibilidade:** o *software* é invisível e transparente.
- **Complexidade:** o *software* é mais complexo do que qualquer outro produto construído por seres humanos.
- **Mutabilidade:** existe sempre uma pressão (de negócio e/ou necessidades técnicas) para se fazerem mudanças em um *software*.
- **Conformidade:** o *software* deve ser desenvolvido conforme o ambiente, seguindo seus padrões, métodos e técnicas. Não é o ambiente que deve se adaptar ao *software*.

Agora que você já sabe o que é um *software*, seus tipos e suas características, eu vou te explicar como aconteceu a evolução histórica do *software*.

1.2 Evolução do software

Você conhece alguém que nasceu antes da década de 50 ou 60? Se sim, tente conversar com essa pessoa e faça as seguintes perguntas:

Quando você era criança...

1. Como você estudava ao realizar tarefas para o colégio? Onde fazia as pesquisas?
2. Como você fazia para se comunicar com os parentes distantes?
3. Como você pesquisava diferentes preços de uma mercadoria?
4. Como você assistia a vídeos de outras pessoas?
5. Como você escutava música?

Depois de ter estas respostas, responda também este mesmo questionário, a partir de sua realidade. Você irá perceber que o número de *softwares* aumentou exponencialmente nas últimas décadas. Agora temos *softwares* em carros, smartphones, caixa eletrônico, eletrodomésticos, câmeras digitais, relógios, óculos, em robôs que mapeiam outros planetas etc. Quando for sair de casa, tente prestar atenção nisso e crie uma lista de dez *softwares* que você usa no dia a dia. Será uma boa experiência... acredite!

Abaixo, segue uma tabela sobre a evolução histórica do software, contendo seus marcos, o período e os acontecimentos (Tabela 1).

MARCO	PERÍODO	ACONTECIMENTOS
Os primeiros anos	1950 a início dos 60	Aplicações científicas e de engenharia
A segunda era	1960 a meados de 80	Aplicações comerciais em grande porte (sistemas de informação com banco de dados)
A terceira era	meados de 70 e década de 80	Aplicativos pessoais em microcomputadores (Personal Computer – PC).
A quarta era	meados de 80 a meados de 90	Aplicativos com interfaces gráficas, redes e arquitetura cliente-servidor
A quinta era	de meados de 90 a (sem fim definido)	Software Distribuídos, Internet, Groupwares e Intranets.
A sexta era (já existe ou existirá?)	(sem início e fim definidos)	Computação Pervasiva, Móvel e Ubíqua.

Tabela 1 – Evolução histórica do ***software***.
Fonte: próprio autor.

Agora me responda: como desenvolver tantos *softwares*, para diversas áreas, sem uma equipe preparada, sem muitos recursos, sem histórico prévio? Complicado, não é mesmo? Tenha certeza de que as pessoas envolvidas no desenvolvimento dos primeiros *softwares* também acharam! Vamos tentar entender o que aconteceu com eles?

1.3 Crise do software

O que aconteceu no fim dos anos 60? Você lembra? Foi nesse período que o primeiro homem pisou na lua, o festival de *Woodstock*[1] aconteceu nos Estados Unidos e a primeira *Polaroid*[2] com fotos instantâneas em 60 segundos foi criada. Certamente você não presenciou esses eventos. Ao mesmo tempo em que aconteciam, as pessoas perceberam que não era possível construir um *software* de que elas precisavam. Isto ocorreu por diversas razões e resultou no que chamamos "crise do *software*". Vamos ver as principais razões para que a crise fosse instalada:

- Aumento na demanda de *software*,
- Aumento do tamanho e complexidade do código,
- Problemas de prazo e custo,
- Baixa produtividade,
- Baixa qualidade,
- Difícil manutenção etc.

Qual foi a solução para a crise do *software*? Como em qualquer situação, antes de simplesmente resolver o problema, vamos parar para PENSAR! Foi com essa ideia que durante uma conferência da OTAN (Organização do Tratado do Atlântico Norte), em 1968, a solução para a crise foi chamada de "Engenharia de *Software*".

Vamos entender o que é?

1.4 Engenharia de software

Engenharia de *software* é uma disciplina de engenharia que se ocupa de todos os aspectos da produção de *software*, desde os estágios iniciais de especificação do sistema até a manutenção (PRESSMAN, 2011). Como a engenharia de *software* faz isso? Através da integração de métodos,

[1] www.youtube.com/watch?v=pJdTjFudoOs
[2] www.b9.com.br/31950/fotografia/livro-conta-a-incrivel-historia-da-polaroid/

ferramentas e procedimentos (processos), para o desenvolvimento de *software* (Figura 1).

Figura 1 - Engenharia de Software.
Fonte: próprio autor.
Descrição:A figura contém um círculo com o texto métodos, o símbolo de soma, um círculo com o texto ferramentas, o símbolo de igualdade e depois um círculo com o texto processos. A figura demonstra uma soma de métodos mais ferramentas resultando em processos.

Vamos falar isoladamente sobre estes tópicos nas subseções abaixo.

1.4.1 Métodos

Os métodos explicam "como fazer" para construir o *software* através de um amplo conjunto de tarefas. O principal objetivo da utilização de métodos é a produção de *software* de alta qualidade. Os métodos podem ser: orientados à função, orientados a objetos, orientados a aspectos. No momento, o mais utilizado e estudado é o método orientado a objetos. Esse método utiliza a linguagem UML para descrever os sistemas dos pontos de vista estrutural e comportamental. Não sabe o que é UML? Não se preocupe, falaremos sobre a linguagem UML já, já.

Para que servem os métodos? Os métodos permitem a criação de modelos do sistema que são representados graficamente (objetos, fluxo de dados, entidades, relacionamentos etc.). Talvez você já tenha feito um na disciplina de Banco da Dados! Caso não tenha visto, pesquise sobre o assunto. Na próxima competência falarei um pouco sobre estes modelos. Os métodos possuem regras que são o conjunto de restrições que se aplicam a modelos do sistema como, por exemplo, cada entidade ter um nome único. Os métodos têm recomendações, ou seja, padrões que o projeto deve seguir (exemplo: todo sistema deverá ter um ponto único de entrada). Os métodos também possuem diretrizes para o processo de *software*: descrição do conjunto de atividades que devem ser seguidas e como os modelos devem ser documentados. Para não esquecer, vamos resumir! As características dos métodos (Figura 2) de Engenharia de Software são: descrições gráficas, regras,

recomendações e diretrizes.

Figura 2 – Características dos métodos de Engenharia de Software.
Fonte: próprio autor.
Descrição: A figura contém quatro retângulos contendo em cada um os seguintes textos: descrições gráficas, regras, recomendações e diretrizes. Estes quatro retângulos apontam para um círculo que contém o texto características dos métodos.

1.4.2 Ferramentas

As ferramentas em Engenharia de *Software* dão suporte automatizado aos métodos. Existem atualmente ferramentas para sustentar cada um dos métodos. Quando as ferramentas são integradas, é estabelecido um sistema de suporte ao desenvolvimento de *software* chamado CASE (*Computer Aided Software Engineering*). As ferramentas CASE (Figura 3) são divididas em: *upper-case* e *lower-case*.

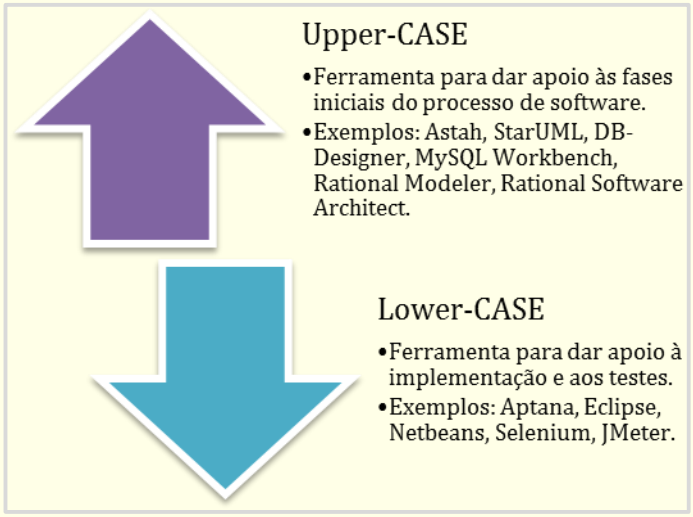

Figura 3 – Ferramentas CASE (Upper e Lower).
Fonte: próprio autor.
Descrição: A figura contém uma seta apontada para cima e ao lado desta o seguinte texto: Upper-CASE são ferramentas para dar apoio às fases iniciais do processo de software, temos como exemplos os softwares Astah, StarUML, DB-Designer, MySQL Workbench, Rational Modeler, Rational Software Architect. A figura também contém uma seta apontada para baixo e ao lado desta o seguinte texto: Lower-CASE são ferramentas para dar apoio à implementação e aos testes, temos como exemplos os softwares Aptana, Eclipse, Netbeans, Selenium, JMeter.

1.4.3 Processos

Os processos são elos de ligação entre os métodos e as ferramentas para desenvolvimento do *software*. Os processos definem:

- A sequência em que os métodos serão aplicados;
- Os produtos que serão entregues;
- Controles que ajudam assegurar a qualidade e coordenar as mudanças do *software*;
- Marcos de referência que possibilitam administrar o progresso do *software*.

Já sabemos que um processo é a soma de métodos mais ferramentas.

Agora, vamos explicar o que é um processo de *software*.

1.5 Processo de software

Processo de *software* (Figura 4) é uma forma de desenvolver ou produzir um *software*. Para isso, devemos seguir as atividades para: especificar, projetar, implementar e testar.

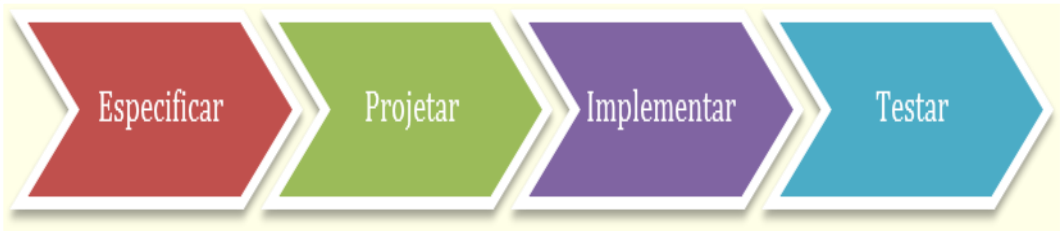

Figura 4 - Atividades do processo de software.
Fonte: próprio autor.
Descrição: A figura explicita que uma atividade só irá começar depois que a outra acabar.

Um processo de *software* irá definir quem faz o quê, quando e como. Todo processo é definido por um modelo. Estes modelos são utilizados para auxiliar o processo de produção. Por exemplo, quando uma planta (ou maquete) é criada por um engenheiro (ou arquiteto) para demonstrar a criação de uma casa. Dessa forma, temos uma visualização do que será feito. O modelo é uma simplificação da realidade. Quando utilizamos um modelo associado a um planejamento, temos um processo. Os modelos de processos de *software* são um conjunto de atividades fundamentais exigida para desenvolver um sistema de *software*.

Modelo de processo de *software* é uma representação abstrata de um processo.

As atividades básicas (Figura 5) são:

- Especificação;
- Projeto e implementação;
- Validação;
- Evolução.

Figura 5 - Atividades básicas do modelo de processo de software.
Fonte: próprio autor.
Descrição:A figura também explicita que uma atividade só irá começar depois que a outra acabar.

Os modelos de processo de *software* serão responsáveis por definir:

- Sequência de atividades;
- Fluxo de informação;
- Papéis das pessoas e as atividades pelas quais elas serão responsáveis.

Falarei a seguir sobre duas metodologias: RUP e Scrum.

1.5.1 Rational Unified Process – RUP

O RUP foi criado por Ivar Jacobson a partir do *Objectory Process* em 1987. Atualmente é um produto da IBM. Foi o primeiro processo a utilizar UML para modelagem de *software*.

Você já ouviu falar em UML? *Unified Modeling Language* (UML) é uma linguagem visual utilizada para modelar sistemas computacionais por meio do paradigma de Orientação a Objetos. A UML é composta por vários diagramas (Figura 6) com o objetivo de fornecer múltiplas visões do sistema a ser modelado, analisando-o e modelando-o sob diversos aspectos.

Caso tenha curiosidade, você pode acessar esses vídeos que mostram os diagramas na prática: www.youtube.com/watch?v=aRVYgbiULgE e www.youtube.com/watch?v=teN4n-vhID0.

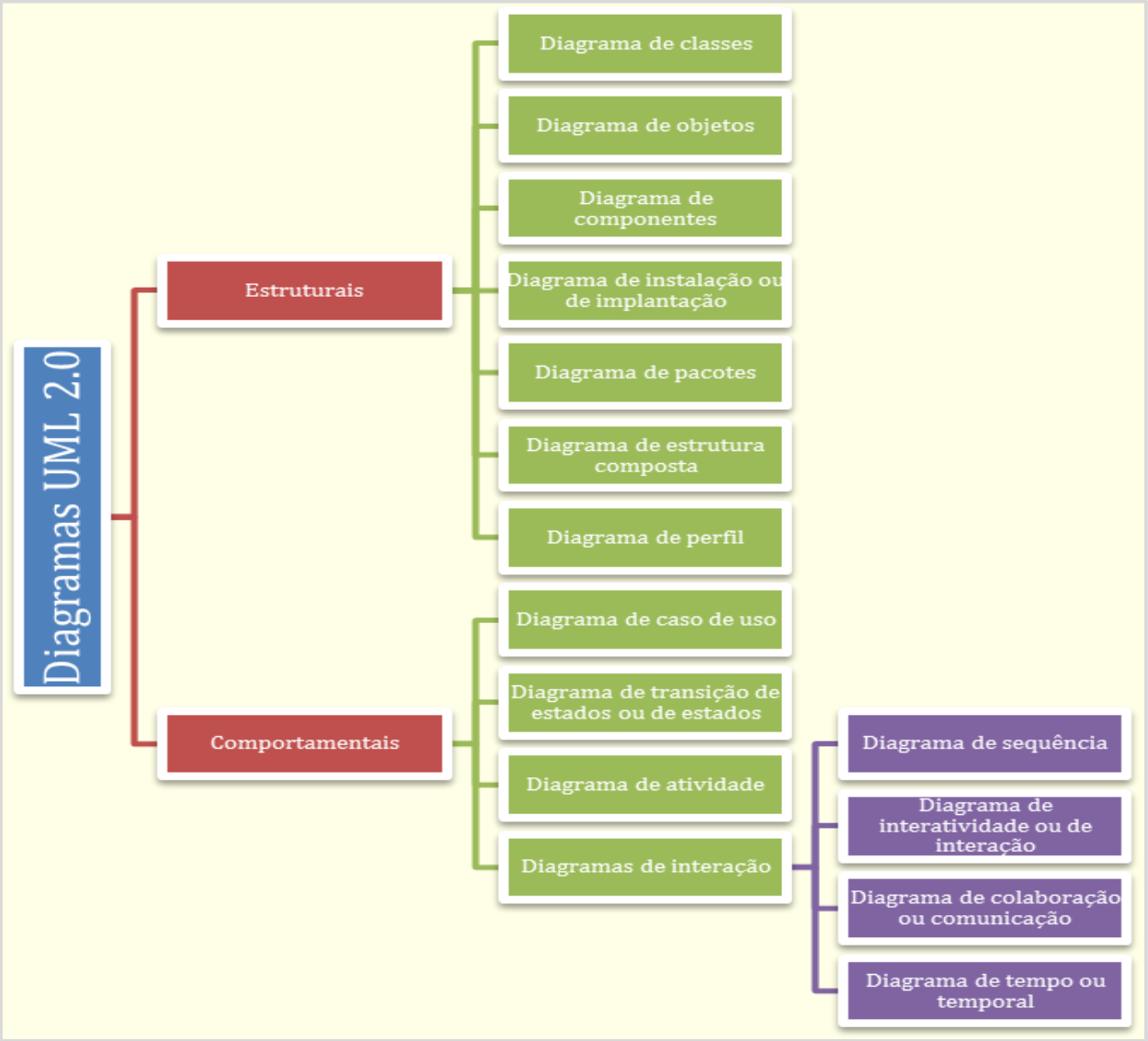

Figura 6 – Diagramas da UML 2.0.
Fonte: próprio autor.
Descrição: A figura contém a hierarquia dos diagramas UML 2.0. Os diagramas se dividem em estruturais e comportamentais. Nos diagramas estruturais temos os seguintes diagramas: diagrama de classes, diagrama de objetos, diagrama de componentes, diagrama de instalação ou de implantação, diagrama de pacotes, diagrama de estrutura composta e diagrama de perfil. Nos diagramas comportamentais temos os seguintes diagramas: diagrama de caso de uso, diagrama de transição de estados ou de estados, diagrama de atividade e diagramas de interação. O diagrama de interação se subdivide em: diagrama de sequência, diagrama de interatividade ou de interação, diagrama de colaboração ou comunicação e diagrama de tempo ou temporal.

O RUP possui três princípios (Figura 7): centrado na arquitetura, interativo e incremental, orientados por casos de uso.

Figura 7 – Princípios do RUP.
Fonte: próprio autor.
Descrição: A figura contém três retângulos contendo cada um os textos: centrado na arquitetura; iterativo e incremental; e orientado por casos de uso. No primeiro retângulo referente à centrado na arquitetura existem cinco outros retângulos com os textos: a arquitetura é a organização fundamental do sistema; inclui elementos estáticos e dinâmicos e sua interação; mostra a visão geral do sistema; facilita reuso.; seleciona a funcionalidade relevante. No segundo retângulo referente à iterativo e incremental existem quatro outros retângulos com os textos: acomoda mudanças de requisitos; integração não é mais feita no final do projeto. Riscos descobertos e tratados durante as integrações iniciais; defeitos podem ser encontrados e corrigidos ao longo das iterações. No terceiro retângulo referente à orientado por casos de uso existem quatro outros retângulos com os textos: os casos de uso (requisitos) guiam todo o desenvolvimento; são expressos sob a perspectiva do usuário; são expressos em linguagem natural; oferecem um meio simples de decompor os requisitos em partes (módulos, pacotes).

A arquitetura do RUP segue o gráfico de “baleias” exibido abaixo (Figura 8). O gráfico de “baleias” mostra a intensidade das disciplinas através das iterações e fases. As fases do processo são: iniciação, elaboração, construção e transição.

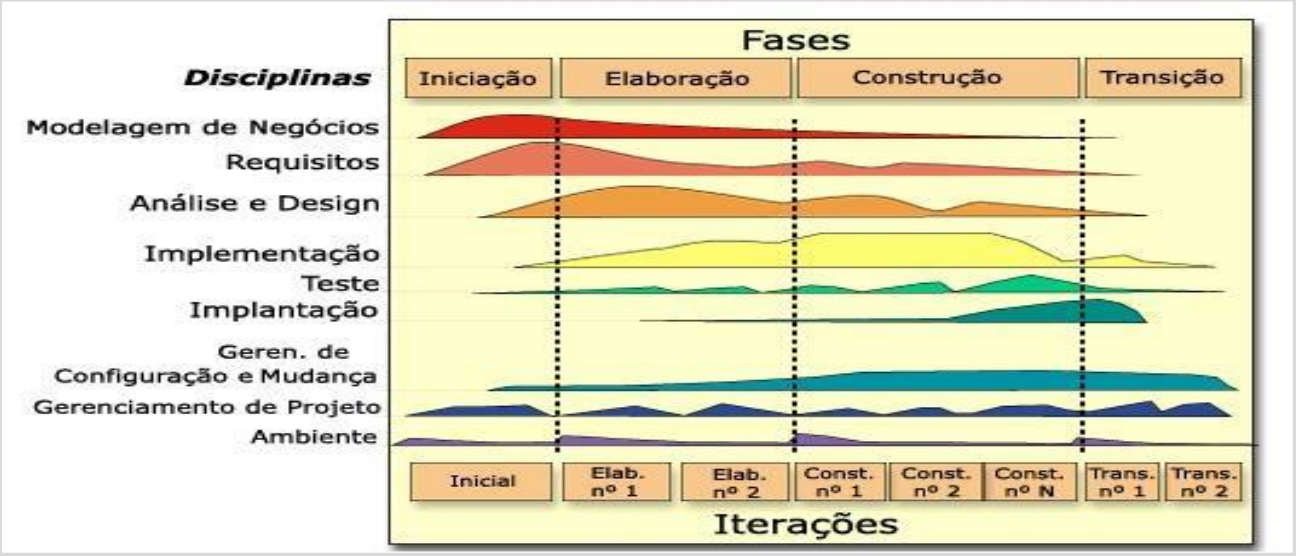

Figura 8 - Gráfico de "baleias".
Fonte: https://pt.wikipedia.org/wiki/IBM_Rational_Unified_Process
Descrição: A figura contém do lado esquerdo as disciplinas do RUP, na parte superior as fases e abaixo as iterações. Para cada disciplina é inserido um gráfico que simboliza o fluxo de informação de acordo com a fase e a iteração. Esses fluxos se assemelham ao formato de uma baleia.

Com este conhecimento você já consegue realizar uma breve pesquisa. Pare agora de ler o caderno e exercite:

1. Explique do que se trata cada disciplina.
2. Liste todas as atividades que são realizadas em cada fase.

Vamos lá! Coragem!

SAIBA MAIS
Caso queira ler mais sobre o RUP, veja este artigo:
www.boente.eti.br/publica/seget2008rup.pdf.

1.5.2 Scrum

O Scrum é um processo iterativo e incremental para o desenvolvimento de qualquer produto e gerenciamento de qualquer projeto. É mais um *framework*[3] do que uma metodologia, mais atitude

3 Um framework (ou biblioteca) em desenvolvimento de software é uma abstração que une códigos comuns entre vários projetos de software, provendo uma funcionalidade genérica. Um framework pode atingir uma funcionalidade específica, por configuração, durante a programação de uma aplicação. Fonte: Wikipédia.

do que um processo. Jeff Sutherland, John Scumniotales e Jeff McKenna conceberam, documentaram e implementaram o Scrum, em 1993. Em 1995, Ken Schwaber formalizou a definição de Scrum (Figura 9) e ajudou a implantá-lo no desenvolvimento de *softwares* em todo o mundo (Wikipédia).

Figura 9 – Origem do Scrum.
Fonte: próprio autor.
Descrição: A figura contém duas fotos com os nomes Jeff Sutherland, PhD e Ken Schwaber. Dois artigos com os textos: lean management e the new new product development game. Em seguida, o texto interativo e incremental. Por fim, um balão explosivo contendo o texto scrum.

Veja o manifesto ágil através deste link: http://agilemanifesto.org/iso/ptbr/manifesto.html e leia sobre os doze princípios do *software* ágil.

No Scrum temos três papéis: product owner (dono do produto), scrum master e development team (equipe de desenvolvimento). Na Tabela 2, constam as responsabilidades para cada papel.

PAPEL	RESPONSABILIDADES
Product Owner	Responsável por garantir o ROI (Retorno de Investimento). Responsável por conhecer as necessidades do(s) cliente(s). Proxy em ambientes com mais de um cliente.
Scrum Master	Responsável por remover os impedimentos do time. Responsável por garantir o uso de Scrum. Protege o time de interferências externas.
Team	Definir metas das iterações. Autogerenciamento. Produzir produto com qualidade e valor para o cliente.

Tabela 2 – Papéis do Scrum e suas responsabilidades.
Fonte: próprio autor.

O funcionamento do Scrum pode ser representado pela Figura 10.

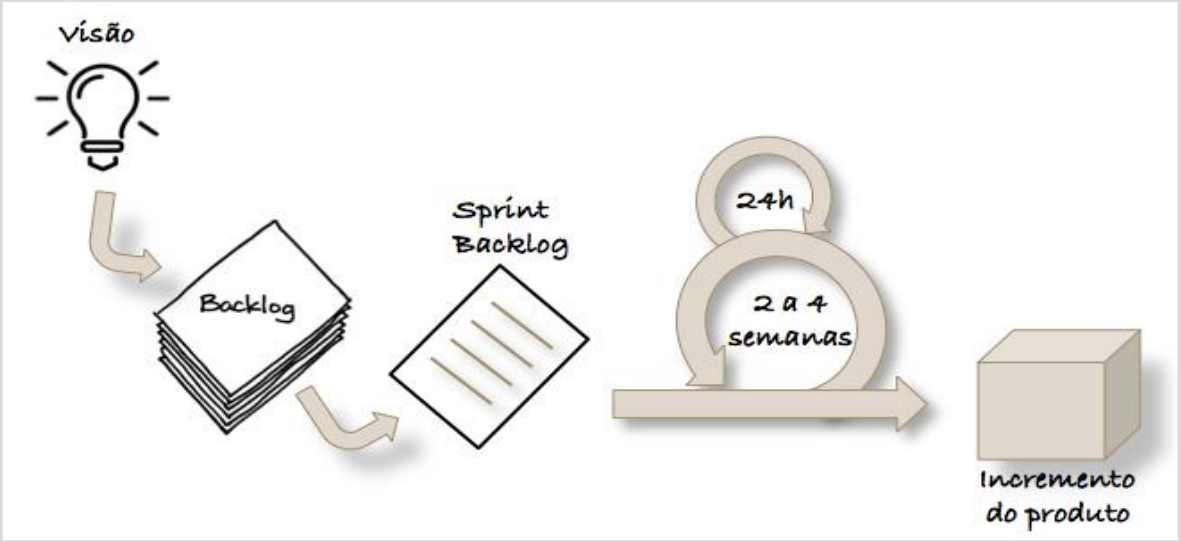

Figura 10 – Funcionamento do Scrum.
Fonte: próprio autor.
Descrição: A figura contém uma lâmpada com o texto visão acima dela, uma seta que direciona para um conjunto de papéis com o texto backlog, uma seta que direciona para um papel com o texto Sprint backlog, uma seta que direciona para um cubo com o texto incremento do produto.

Que tal realizar uma pesquisa sobre as etapas do processo do Scrum? Para melhor entendimento, veja o vídeo "Scrum - Aprenda Scrum em 9 minutos por Denisson Vieira" (www.youtube.com/watch?v=XfvQWnRgxG0). No final da pesquisa, tente responder as perguntas abaixo:

1. O que significa backlog? Qual a diferença entre product backlog e sprint backlog?
2. O que é uma Sprint?
3. Durante as reuniões scrum, quais são as perguntas levantadas?
4. O que é um incremento do produto? O que ele contém?

Já conhecemos duas metodologias de processo de desenvolvimento de *software*: RUP e Scrum. Qual devemos usar? Existem outros modelos? Caso tenha interesse, faça uma pesquisa sobre outros modelos, como: cascata, prototipação, XP, etc.

Na próxima seção iremos abordar uma tarefa muito importante do ciclo de vida de um processo de *software*: a engenharia de requisitos.

1.6 Engenharia de requisitos

O que é um requisito? Talvez você já tenha ouvido essa palavra e já saiba o que é. De qualquer forma, vamos definir! Requisitos são objetivos ou restrições estabelecidas por clientes e usuários que definem as propriedades do sistema. Ele descreve o que é requerido para que o sistema cumpra o seu objetivo. Para realizarmos a análise de requisitos devemos seguir um processo demonstrado na Figura 11.

Figura 11 – Atividades que envolvem a análise de requisitos (Descoberta, Refinamento, Modelagem, Especificação e Validação).
Fonte: próprio autor.
Descrição: A figura contém cinco retângulos contendo os textos: descoberta, refinamento, modelagem, especificação e validação. O primeiro retângulo se liga ao segundo, e assim por diante, até chegar no último.

Podemos classificar os requisitos baseado no detalhamento:

- Requisitos do usuário: declarações em linguagem natural e também em diagramas, sobre as

funções que o sistema deve fornecer e as restrições sob as quais deve operar. Descrição em alto nível das necessidades do usuário.

- Requisitos de sistema: documento estruturado com descrições detalhadas dos serviços de sistemas. Escrito como um contrato entre o cliente e contratante. Derivado dos requisitos do usuário. Definem o que o sistema deve fazer, e não como ele deve ser implementado. São organizados de acordo com os diferentes subsistemas que constituem o sistema.
- Especificação de projeto de *software*: descrição abstrata do projeto de *software*; que é uma base para um projeto ou implementação mais detalhada. Derivado dos requisitos do sistema.

Também podemos classificar os requisitos baseados no tipo:

- Requisitos funcionais: declarações de funções que o sistema deve fornecer, como o sistema deve reagir a entradas específicas e como deve se comportar em determinadas situações. Exemplos:
  - O sistema exibirá uma lista de disciplinas disponíveis no semestre.
  - Os alunos poderão selecionar até quatro disciplinas por semestre.
  - O sistema mostrará uma mensagem se o limite de disciplinas no semestre for ultrapassado.
- Requisitos não-funcionais: expressam qualidade e restrições sobre os serviços ou as funções oferecidos pelo sistema. Envolve restrições de conformidade (padrões), funcionalidade, confiabilidade, desempenho, segurança, usabilidade, etc. Exemplo:
  - Desempenho: o sistema suportará até 2000 usuários simultâneos.
  - Segurança: o sistema deverá prevenir o acesso não autorizado de um aluno às informações de outros usuários.

Alguns exemplos de requisitos funcionais (RF) e não funcionais (RNF):

1. O sistema deverá mostrar os pré-requisitos para um aluno se matricular em uma disciplina. ▫ RF
2. Ao finalizar a matrícula, o sistema exibe um comprovante. ▫ RF

3. O comprovante deverá ser gerado através de um arquivo em PDF, com o formato PDF/A-1b. ▫ RNF

4. O tempo máximo de consulta ao banco de dados não poderá ser superior a 5 segundos. ▫ RNF

5. Para que uma disciplina possa ser oferecida é obrigatória a matrícula de no máximo, 20, e no mínimo, 3 alunos. ▫ RF

6. Uma disciplina com menos de 3 alunos será cancelada pelo sistema. ▫ RF

1.6.1 Processo de Análise de Requisitos

O processo de análise de requisitos (Figura 12) consiste em quatro etapas:

1. Entendimento do domínio: os desenvolvedores devem entender o domínio da aplicação e definir um glossário do domínio.
2. Extração e análise de requisitos: atividades que levam à descoberta, revelação e entendimento dos requisitos, através de interação com o(s) usuário(s) e desenvolvedores.
3. Especificação: armazenamento dos requisitos em um ou mais formatos (linguagem formal ou semiformal; representações simbólicas ou gráficas). Deverá descrever o uso do sistema pelo usuário.
4. Validação: verificação formal dos requisitos, visando determinar se estão em conformidade com as necessidades do usuário (normalmente requer a aprovação formal).

Figura 12 – As quatro etapas do processo de análise de requisitos.
Fonte: próprio autor.
Descrição: A figura contém quatro retângulos contendo os textos: entendimento do domínio, extração e análise dos requisitos, especificação e validação. O primeiro retângulo se liga ao segundo, e assim por diante, até chegar no último.

Como podemos extrair os requisitos? Lembra-se do seu cliente que precisa de um *software* para vender os produtos do mercadinho online? Como você faria para elicitar[4] os requisitos funcionais e não-funcionais deste *software*? Temos algumas técnicas (Figura 13) para fazer a extração: de modo formal ou informal.

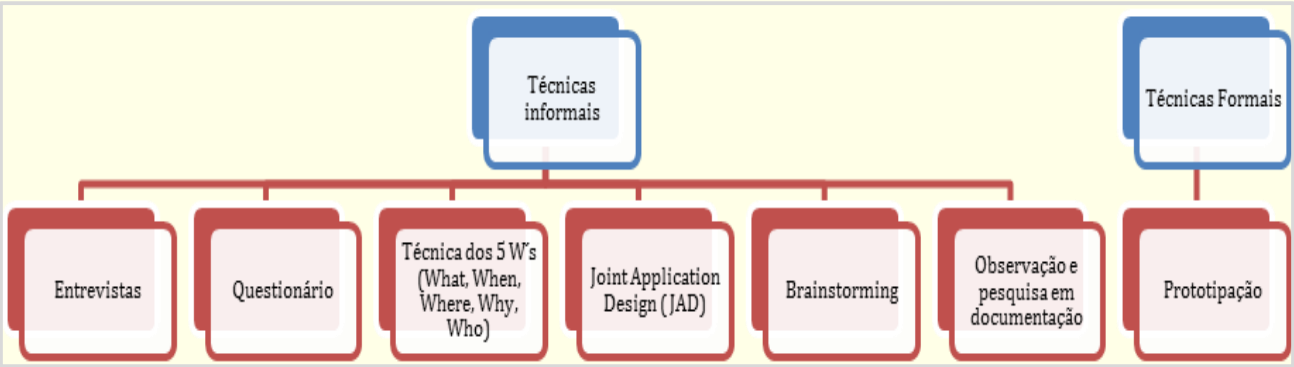

Figura 13 – Principais técnicas de extração de requisitos.
Fonte: próprio autor.
Descrição: A figura contém uma hierarquia das técnicas de extração de requisitos. No primeiro nível temos: técnicas informais e técnicas formais. Em técnicas informais, temos: entrevistas, questionário, técnica dos 5 W´s (What, When, Where, Why, Who), Joint Application Design (JAD), brainstorming, observação e pesquisa em documentação. Em técnicas formais, temos: prototipação.

4 Elicitar significa descobrir, tornar explícito, obter o máximo de informações para o conhecimento do objeto em questão. Diz respeito à elicitação, à tarefa de identificar os fatos
relacionados aos requisitos do sistema, de forma a prover o mais correto e mais completo entendimento do que é demandado daquele software.

Agora é com você! Faça uma pausa no caderno para pesquisar sobre as técnicas que estão na figura acima.

Chegamos ao final da primeira competência. Espero que tenham entendido todos os conceitos. Em caso de dúvidas, não se esqueça de entrar em contato via fórum com os tutores virtuais.

2.Competência 02 | Qualidade De Software

Nesta competência falaremos sobre qualidade de *software*. Vamos começar? Sempre que você usa um *software* e ocorre algum problema, você fica satisfeito? E quando aquele programa trava o seu computador? Ou aquele app (aplicativo) que trava o seu smartphone ou tablet? Imagine os *softwares* que são usados em foguetes, robôs que realizam cirurgias, controladores de voos, entre outros. Será que os engenheiros de *software* só se preocupam em seguir um processo sem um foco? Quase todos irão concordar que um *software* de alta qualidade é uma meta importante.

Então, sempre que pensarmos em processo, temos que pensar em um processo com foco na qualidade!

Ótimo! Mas pare para pensar... o que é qualidade? O que é qualidade de *software*? Podemos definir qualidade de *software* como sendo a satisfação de requisitos funcionais e de desempenho explicitamente declarados, normas de desenvolvimento explicitamente documentadas e características implícitas que são esperadas em todo o *software* desenvolvido profissionalmente (Pressman).

Qualidade de *software* é uma mistura complexa de fatores que diferem de acordo com os diferentes tipos de aplicação e de clientes. Segundo a norma ISO[5] 9126, existem seis atributos-chave de qualidade (Pressman):

- Funcionalidade ▫ Grau em que o *software* satisfaz as necessidades declaradas.
- Confiabilidade ▫ Período de tempo em que o *software* está disponível para uso.
- Usabilidade ▫ Grau em que o *software* é fácil de usar.
- Eficiência ▫ Grau em que o *software* faz uso otimizado dos recursos do sistema.
- Manutenabilidade ▫ Facilidade com a qual podem ser feitos reparos no *software*.
- Portabilidade ▫ Facilidade com a qual o *software* pode ser transposto de um ambiente para o outro.

[5] Na seção 2.2.1, explico o que é uma norma ISO.

Para cada fator de qualidade visto acima, tem-se subatributos, que são mostrados na Figura 14.

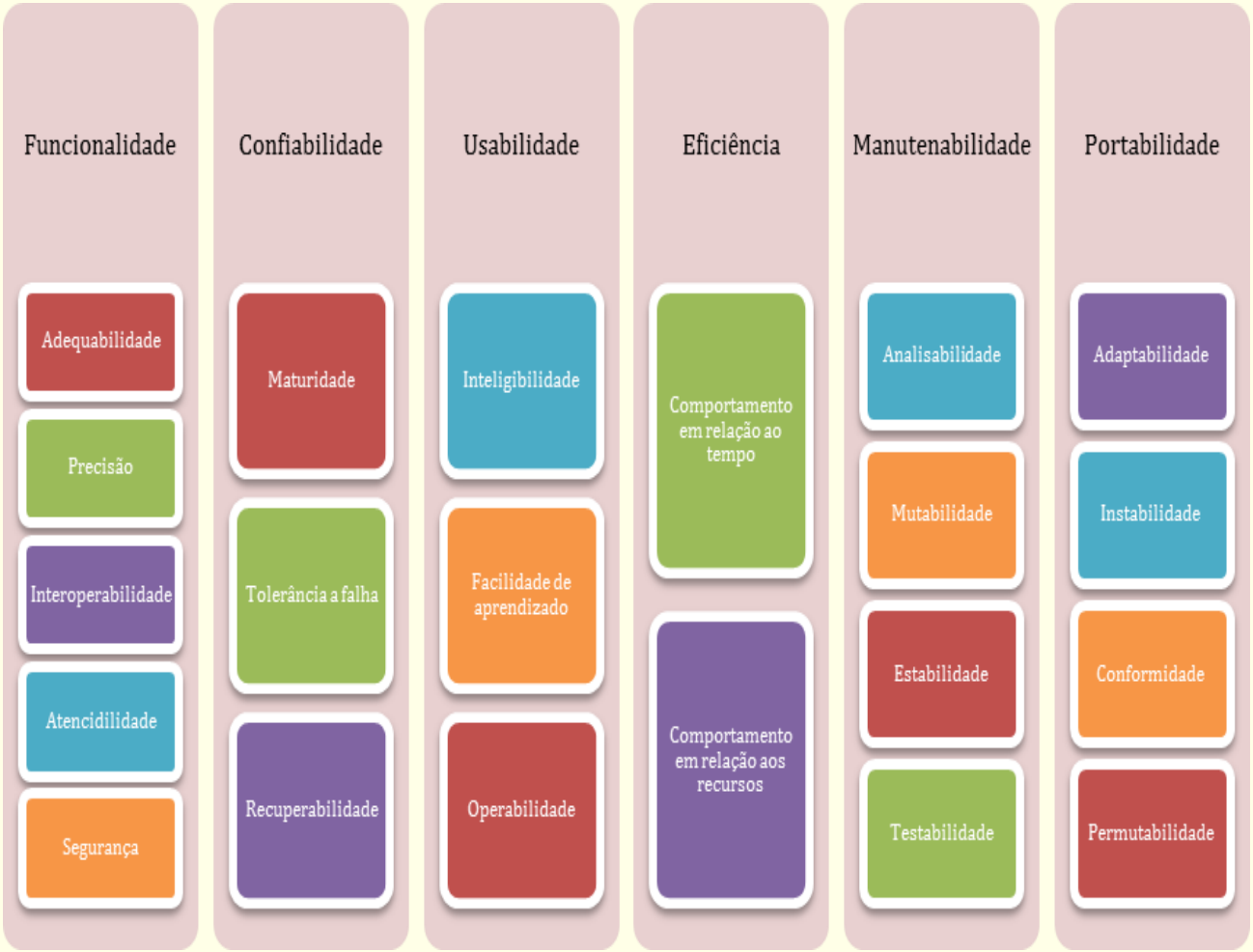

Figura 14 - Fatores de qualidade ISO 9126 e os seus subatributos.
Fonte: próprio autor.
Descrição: A figura contém seis retângulos com os textos: funcionalidade, confiabilidade, usabilidade, eficiência, manutenabilidade e portabilidade. Em funcionalidade, temos mais cinco retângulos: adequabilidade, precisão, interoperabilidade, atencidilidade e segurança. Em confiabilidade, temos três retângulos: maturidade, tolerância a falha e recuperabilidade. Em usabilidade, temos três retângulos: inteligibilidade, facilidade de aprendizado e operabilidade. Em eficiência, temos dois retângulos: comportamento em relação ao tempo e comportamento em relação aos recursos. Em manutenabilidade, temos quarto retângulos: analisabilidade, mutabilidade, estabilidade e testabilidade. Em portabilidade, temos mais quarto retângulos: adaptabilidade, instabilidade, conformidade e permutabilidade.

Como há diferentes conceitos do que é qualidade de *software*, temos algumas normas/modelos que foram criados, como o ISO 9126 acima. Porém, antes de falar sobre alguns deles, vamos entender o que é *Software Quality Assurance*.

2.1 Software Quality Assurance – SQA

O processo de garantia de Qualidade de *Software* (SQA) é utilizado para garantir que os processos e produtos estejam em conformidade com os requisitos especificados e referentes ao plano estabelecido. É visível a relação entre qualidade de produto e qualidade de processo, através da SQA. Estas são as principais atividades:

- **Implementação do processo:** define, implementa e mantém os padrões de qualidade, metodologias, procedimentos, ferramentas, revisão e coordenação de contratos, recursos, cronogramas, responsabilidades e tarefas dos outros processos de apoio.
- **Garantia do produto:** garante que os planos estejam de acordo com o contratado, incluindo o produto de *software* e sua documentação.
- **Garantia do processo:** garante que os processos do ciclo de vida estejam em conformidade com o contrato e planos do projeto. Garante que as práticas de engenharia de *software*, os ambientes de desenvolvimento e testes estejam de acordo com o contrato. Garante a medição do produto e do processo e a capacitação da equipe.

2.2 Normas/Modelos de qualidade de software

Um processo representa, dentro da área de *software*, um conjunto de atividades cujo objetivo é atingir uma meta previamente estipulada. Já por capacidade e maturidade de um processo, deve-se ter a noção do grau de qualidade com o qual um processo atinge um resultado esperado. Veremos agora uma norma e dois modelos de qualidade de *software*: ISO/IEC – 25010, CMMI e MPS.BR.

2.2.1 ISO/IEC-25010

Uma norma é um documento aprovado por um organismo reconhecido que prevê validar conhecimentos comuns para utilização determinada de processos, regras ou características para produtos cuja correspondência seja satisfatória. A Organização Internacional de Normalização (*International Organization for Standardization*), popularmente conhecida como ISO, atua em vários

países e possui normas em várias áreas de interesse econômico e técnico. A Comissão Eletrotécnica Internacional (*International Electrotechnical Commission*, IEC) é uma organização internacional de padronização de tecnologias elétricas, eletrônicas e relacionadas. Alguns dos seus padrões são desenvolvidos juntamente com ISO.

A norma ISO/IEC – 25010, substituiu a ISO/IEC – 9126, aumentando a lista de fatores de qualidade. A norma ISO/IEC 25010 contém duas características (Figura 15) que foram acrescentadas: segurança e compatibilidade. Dessa forma, a nova norma totaliza oito características: seis antigas e duas novas.

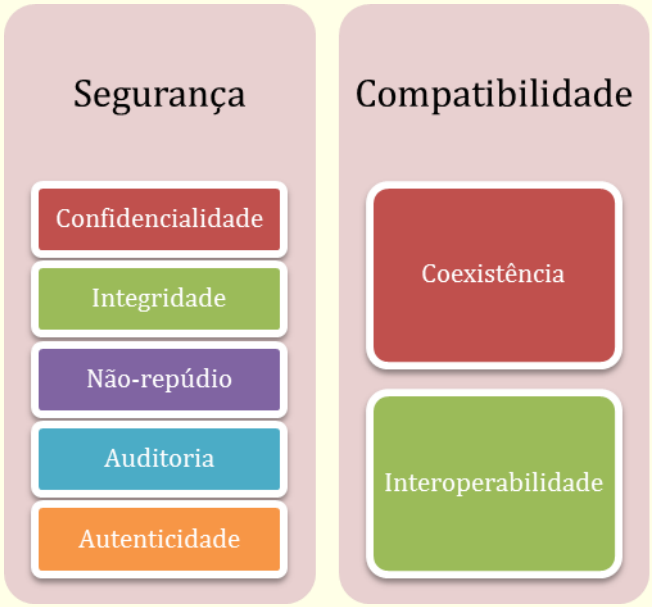

Figura 15 – As características e subcaracterísticas adicionadas na ISO/IEC 25010.
Fonte: próprio autor.
Descrição: A figura contém dois retângulos com os textos: segurança e compatibilidade. Em segurança, temos mais cinco retângulos com os textos: confidencialidade, integridade, não-repúdio, auditoria e autenticidade. Em compatibilidade, temos mais dois retângulos: coexistência e interoperabilidade.

Existem outras normas ISO que podem ser aplicadas no projeto de desenvolvimento de um *software*. Que tal pesquisá-las?

2.2.2 Capability Maturity Model Integration – CMMI

O CMMI (*Capability Maturity Model Integration*), criado pelo SEI (*Software Engineering Institute*), trata-se de um modelo de referência que define práticas necessárias para o desenvolvimento e avaliação de maturidade de *software* em uma organização. Este modelo possui cinco níveis (Figura 16): inicial, gerenciado, definido, quantitativamente gerenciado e otimização.

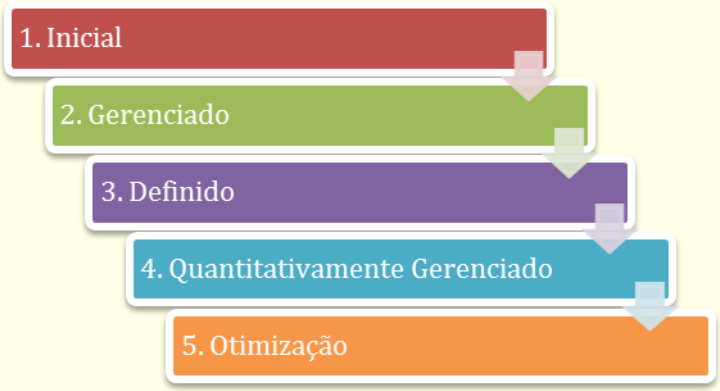

Figura 16 - Os cinco níveis de maturidade do CMMI.
Fonte: próprio autor.
Descrição: A figura contém cinco retângulos com os seguintes textos: inicial, gerenciado, definido, quantitativamente gerenciado e otimização.

Por que uma empresa deveria seguir esse modelo?

Para melhor entendimento, que tal assistir aos vídeos disponibilizados pela ISD Brasil sobre os níveis de maturidade do CMMI?! Estes são os vídeos:
1 - ISD BRASIL - o que é o Nível 1 de Maturidade do CMMI
www.youtube.com/watch?v=kF8sxDDoRns
2 - ISD BRASIL - implementando o CMMI Nível 3 de Maturidade
www.youtube.com/watch?v=PiQh_bzoJoc
3 - ISD BRASIL - implementando e refletindo sobre os Níveis 2 e 3 de Maturidade do CMMI
www.youtube.com/watch?v=C2BFOEvtu3w
4 - ISD BRASIL - implementando o CMMI Nível 5 de Maturidade
www.youtube.com/watch?v=tKT3XO_CS7M

Após assistir aos vídeos, faça uma tabela comparativa (Figura 17) entre uma empresa no nível 1 do CMMI (problemas enfrentados) e uma empresa após implantação do CMMI (soluções encontradas). Vou te ajudar a começar...

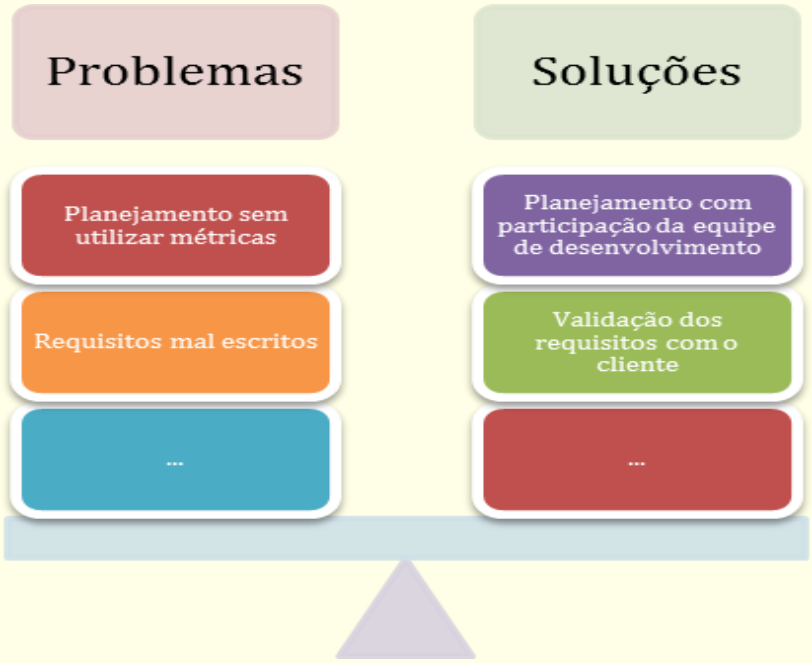

Figura 17 – Problemas e soluções da empresa FazSite.
Fonte: próprio autor.
Descrição: A figura contém uma balança. Do lado esquerdo constam os problemas e do lado direito as soluções. Em problemas, temos: requisitos mal escritos; planejamento sem utilizar métricas. Em soluções, temos: validação dos requisitos com o cliente; planejamento com participação da equipe de desenvolvimento.

.

2.2.3 Melhoria de Processo do Software Brasileiro – MPS.BR

A base utilizada para a criação do MPS.BR foi proveniente das normas NBR ISO/IEC, o CMMI, entre outras. Observe a Figura 18. A construção deste modelo é um esforço conjunto entre a Softex[6], o governo e universidades, para a aplicação em empresas brasileiras.

[6] A Associação para Promoção da Excelência do Software Brasileiro (Softex) executa, desde 1996, iniciativas de apoio, desenvolvimento, promoção e fomento para impulsionar a Indústria Brasileira de Software e Serviços de TI, uma das maiores em todo o mundo, conhecida por sua criatividade, competência e fonte de talentos. Texto retirado do site da Softex (http://www.softex.br/a-softex/).

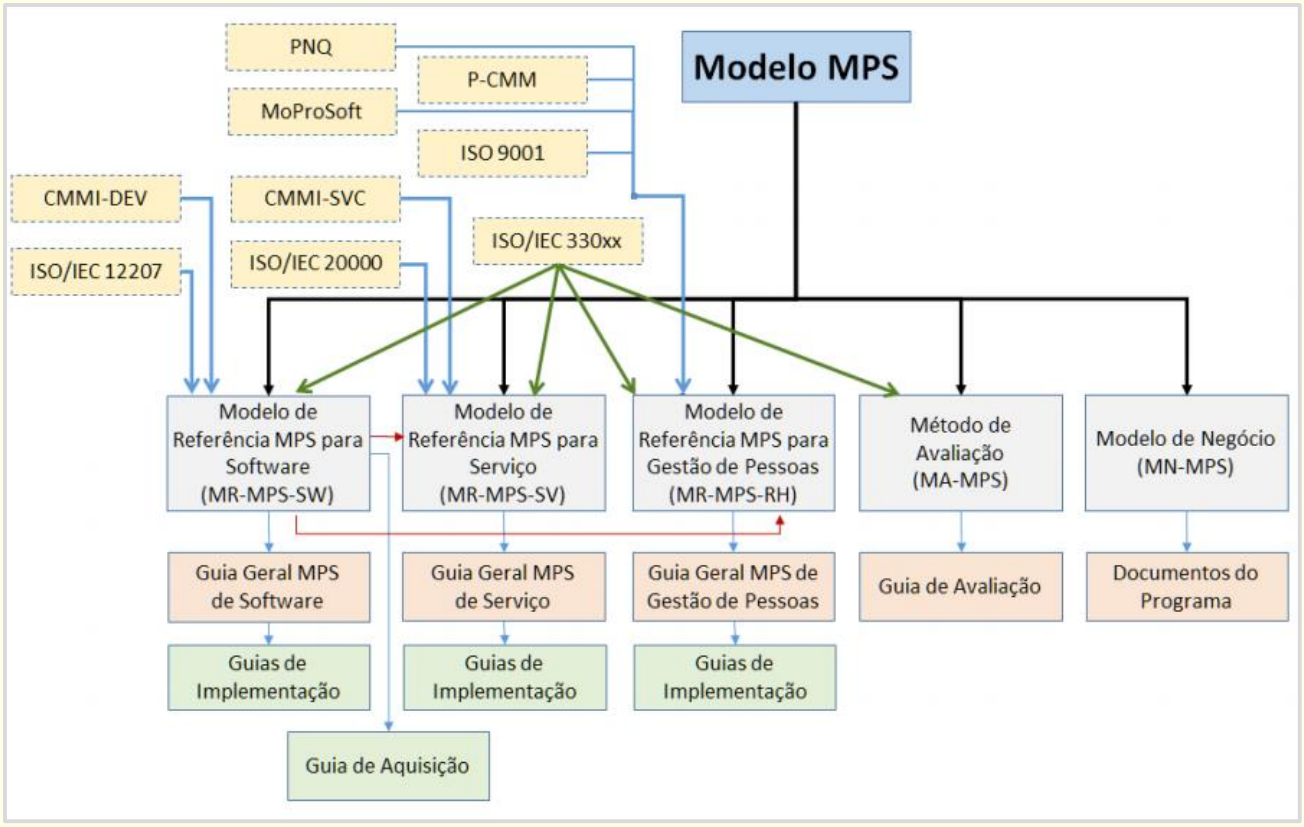

Figura 18 - Componentes do MPS.BR.
Fonte: Guia Geral MPS de Software de Janeiro 2016.
Descrição: A figura contém a hierarquia do Modelo MPS. Identificando os cinco componentes descritos no texto e seus guias.

O MPS.BR está dividido em cinco componentes:

- Modelo de Referência para *Software* (MR-MPS-SW);
- Modelo de Referência para Serviço (MR-MPS-SV);
- Modelo de Referência para Gestão de Pessoas (MR-MPS-RH);
- Método de Avaliação (MA-MPS);
- Modelo de Negócio (MN-MPS).

Vamos entender do que se trata o componente MR-MPS-SW?!

2.2.3.1 Modelo de Referência para Software – MR-MPS-SW

O MR-MPS-SW define os níveis de maturidade que são uma combinação entre processos e sua

capacidade. Existem sete níveis, começando do G (Parcialmente Gerenciado) até o A (Em Otimização), conforme a Figura 19. Através do nível de maturidade no qual encontra-se uma empresa, é possível prever o seu desempenho no futuro ao executar um ou mais processos.

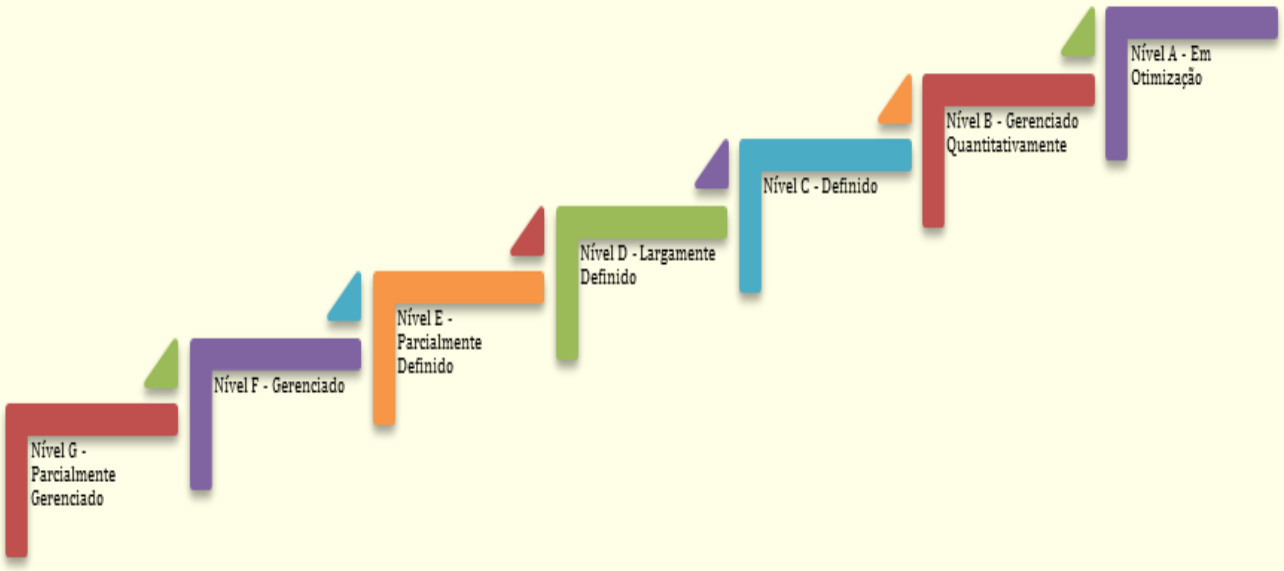

Figura 19 – Os sete níveis de maturidade do MPS.BR.
Fonte: próprio autor.
Descrição: A figura contém uma escada contendo os sete níveis: Nível G - Parcialmente Gerenciado; Nível F – Gerenciado; Nível E - Parcialmente Definido; Nível D - Largamente Definido; Nível C - Definido; Nível B - Gerenciado Quantitativamente; Nível A - Em Otimização.

A capacidade do processo é representada por um conjunto de atributos de processo (AP) descrito em termos de resultados esperados e identifica o grau de refinamento e institucionalização com que o processo é executado na organização. A capacidade do processo no MPS são descritos por nove atributos de processos (AP) que são:

- AP 1.1 – O processo é executado;
- AP 2.1 – O processo é gerenciado;
- AP 2.2 – Os produtos de trabalho do processo são gerenciados;
- AP 3.1 – O processo é definido;
- AP 3.2 – O processo está implementado;
- AP 4.1 – O processo é medido;
- AP 4.2 – O processo é controlado;

- AP 5.1 – O processo é objeto de melhorias incrementais e inovação;
- AP 5.2 – O processo é otimizado continuamente.

A Tabela 3 exibe os sete níveis de maturidade, os processos e os atributos de processo correspondente a cada nível do G (Parcialmente Gerenciado) até o A (Em Otimização).

NÍVEL	PROCESSOS	ATRIBUTOS DE PROCESSO
A	(sem processo específico)	AP 1.1, AP 2.1, AP 2.2, AP 3.1, AP 3.2, AP 4.1, AP 4.2, AP 5.1 e AP 5.2
B	Gerência de Projetos – GPR (evolução)	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP 3.2, AP 4.1 e AP 4.2
C	Gerência de Riscos – GRI Desenvolvimento para Reutilização – DRU Gerência de Decisões – GDE	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP 3.2
D	Verificação – VER Validação – VAL Projeto e Construção do Produto – PCP Integração do Produto – ITP Desenvolvimento de Requisitos – DRE	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP 3.2
E	Gerência de Projetos – GPR (evolução) Gerência de Reutilização – GRU Gerência de Recursos Humanos – GRH Definição do Processo Organizacional – DFP Avaliação e Melhoria do Processo Organizacional – AMP	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP 3.2
F	Medição – MED Garantia da Qualidade – GQA Gerência de Portfólio de Projetos – GPP Gerência de Configuração – GCO Aquisição – AQU	AP 1.1, AP 2.1 e AP 2.2
G	Gerência de Requisitos – GRE Gerência de Projetos – GPR	AP 1.1 e AP 2.1

Tabela 3 - Níveis de maturidade do MR-MPS-SW.
Fonte: próprio autor.

Para mais detalhes, consulte o Guia Geral MPS de *Software* em www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_Geral_Software_2016.pdf.

Conclusão

Caro (a) estudante, chegamos ao final da nossa disciplina. Ao longo dessas duas semanas, entendemos a necessidade de seguir um processo para o desenvolvimento de um *software*, estudamos alguns conceitos relacionados à disciplina de Engenharia de Software e agora sabemos a importância em desenvolver um projeto de *software* com qualidade. Espero que você tenha aproveitado ao máximo todo o material compartilhado. O mundo da engenharia de *software* é enorme e este aqui foi o seu primeiro e o mais importante passo: o inicial. Continue os seus estudos e aprimore o seu pensamento crítico. Muito obrigada pela oportunidade de compartilhar um pouco do meu conhecimento para você.

Abraços,

Ivna Valença de Oliveira.

Referências

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *ISO/IEC 25010:2011*. Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models. Vernier, Geneva Switzerland, 2011.

ISD BRASIL. O que é CMMI? In: PERGUNTAS frequentes. São Paulo, 2016. Disponível em: <http://www.isdbrasil.com.br/o-que-e-cmmi.php>. Acesso em: 10 set. 2016.

PRESSMAN, Roger S. MAXIM, Bruce. *Engenharia de Software:* Uma Abordagem Profissional. 7.ed. São Paulo: McGraw Hill Brasil, 2011.

SOFTEX. Projetos para o setor. Brasília, 2016. Disponível em: < http://www.softex.br/mpsbr/>. Acesso em: 10 set. 2016.

SOMMERVILLE, Ian. *Engenharia de Software*. 8.ed. São Paulo: Pearson, 2007.

Minicurrículo do Professor

Olá, meu nome é **Ivna Valença de Oliveira**. Eu fiz graduação em Análise de Sistemas na Universo, especialização em Testes de *Software* na Uninassau e mestrado em Ciências da Computação com foco em Inteligência Computacional no Centro de Informática (UFPE). Desde 2012, leciono algumas disciplinas no curso de Ciências da Computação da Faculdade Nova Roma, entre elas: Programação Imperativa, Introdução à Computação, Programação Orientada a Objetos, Laboratório de Programação, Engenharia de *Software*, Estruturas de Dados, Testes e Configuração de S*oftware* e Inteligência Artificial.

Abraços,

Ivna :)