Pós-Graduação a Distância

Engenharia de Software

Elisamara de Oliveira
Claudinei Di Nuno

SUMÁRIO

SUMÁRIO

SUMÁRIO

APRESENTAÇÃO

Atualmente o software está presente de forma explícita, ou mesmo sem se fazer notar, em diversos aspectos da vida, inclusive nos sistemas críticos que afetam nossa saúde e bem-estar. Diferentemente de outras áreas da Engenharia, novas aplicações de software aparecem a cada dia e isso torna a área de desenvolvimento de software um desafio constante, necessitando de pessoas treinadas, capacitadas e em constante evolução para desempenhar adequadamente as funções necessárias para a produção de software de qualidade.

As dificuldades em se obter software de qualidade, obedecendo a prazos, custos e escopos estabelecidos, mostram-se presentes desde os primórdios da área de computação. Devido a isso, uma base sólida sobre a teoria e a prática da engenharia de software é essencial para sabermos como construir bons softwares e avaliarmos os riscos e as oportunidades que ele pode trazer para o bom desempenho dos negócios.

A engenharia de software trilhou um longo caminho desde o primeiro uso da expressão. Convidamos você a percorrer esse caminho. Ao longo desta disciplina, destacaremos aspectos-chave da engenharia de software, abordando, dentre outras, as seguintes questões:

- Qual a necessidade de uma disciplina de engenharia para o desenvolvimento de software?
- O que se entende por engenharia de software?
- O que se entende por software de computador?
- Por que lutamos para construir sistemas de alta qualidade?
- O que é um processo de software?
- Que modelos de processo podem ser aplicados ao desenvolvimento de software?
- Quais os pontos fortes e fracos de cada modelo de processo?
- Por que é tão complicado estabelecer os requisitos de um sistema?
- Quais técnicas podem nos auxiliar no estabelecimento dos requisitos?
- Para que servem os testes de software? Quais técnicas de teste podemos usar?
- Quais os prinicpais modelos de qualidade de software?
- O que é gerenciamento de projetos? O que é o PMBoK?

Quando essas questões forem respondidas, você estará melhor preparado para entender os conceitos de gestão e os aspectos técnicos da atividade de engenharia de software essenciais ao desenvolvimento Java, que serão abordados nas disciplinas subsequentes do nosso curso.

Elisamara de Oliveira

MÓDULO 1 – FUNDAMENTOS DA ENGENHARIA DE SOFTWARE

Para entendermos melhor os conceitos fundamentais que envolvem a área de engenharia de software, faremos um breve histórico dos principais fatos que ocorreram no mundo da tecnologia e justificaram a criação desse ramo da computação. É interessante notar que, há algumas décadas, não se tinha ideia da importância que os computadores e — em especial o software — iriam ter e como iriam afetar profundamente a vida de todos nós.

Aula 1 – Aspectos históricos da engenharia de software

1.1 A crise do software e a engenharia de software

A expressão *engenharia de software* foi criada no início da década de 1960, tendo sido utilizada oficialmente em 1968 na *NATO Conference on Software Engineering* (Conferência da OTAN sobre Engenharia de software), que aconteceu na cidade de Garmisch, Alemanha. Nesta ocasião, todos estavam preocupados em contornar os efeitos da crise do software e buscar maneiras de dar um tratamento de engenharia mais sistemático e controlado ao desenvolvimento de sistemas de software complexos.

Sistemas de software complexos apresentam, de forma geral, componentes heterogêneos e em grande número, com alto grau de interconexões, relações e dependência entre si. Esta heterogeneidade pode surgir, em parte, como resultado da evolução

do sistema e sua adaptação a novas condições de contorno. Os sistemas também podem se tornar complexos na medida em que novas versões são criadas.

A crise do software

A crise do software foi uma expressão criada para descrever as dificuldades enfrentadas no desenvolvimento de software no final da década de 1960. O aumento da complexidade dos problemas aliado à inexistência de técnicas bem estabelecidas e à crescente demanda por novas aplicações indicavam que medidas sérias deveriam ser tomadas. Essa crise teve como origem a introdução de computadores "mais poderosos", e o advento desse hardware com mais recursos tornava viáveis softwares bem maiores e mais complexos que os sistemas existentes.

A experiência inicial de construção desses sistemas mostrou que uma abordagem informal de desenvolvimento de software não era suficiente. Projetos importantes sofriam atrasos (às vezes, de alguns anos) e os custos eram muito maiores do que os inicialmente projetados. As soluções de software não eram confiáveis e os programas eram de difícil manutenção. Novas técnicas e novos métodos eram necessários para controlar a complexidade inerente aos grandes sistemas de software (LUIZ, 2010).

Como o principal objetivo dessa conferência foi estabelecer práticas mais maduras para o processo de desenvolvimento de software, ela é considerada como o evento que deu origem à disciplina conhecida como engenharia de software (IMASTERS, 2010).

Figura 1. *NATO Software Engineering Conference*, 1968.
Fonte: <http://goo.gl/YVicsj>. Acesso em: 29/01/2014.

Spector e Gifford (1986, p. 268-273) realizaram um famoso estudo no qual comparavam a construção de pontes ao desenvolvimento de software. Os autores partiam da premissa que as pontes normalmente eram construídas no tempo planejado, no orçamento e nunca caíam. Por outro lado, os softwares nunca ficavam prontos dentro do prazo e do orçamento, e, além disso, quase sempre apresentavam problemas.

1.2 O Chaos Report

Uma década mais tarde, em 1995, a organização *The Standish Group* (1995) publicou um estudo analisando as estatísticas sobre sucesso e fracasso dos projetos de desenvolvimento de software: o Chaos Report. Nesse estudo foi revelado que:

- quase 84% dos projetos de software eram mal-sucedidos, sejam por serem cancelados ou apresentarem falhas críticas (dentre elas conclusão fora do tempo previsto, gastos fora do orçamento estipulado ou produto com menos funcionalidades do que o planejado);
- 31.1% dos projetos eram cancelados antes de serem concluídos;
- 52.7% dos projetos apresentavam custo real 189% maior que o estimado e o tempo de conclusão 222% maior do que o previsto;
- 16.2% dos projetos eram concluídos dentro de prazo, custo e escopo estimados;
- estimou-se que, naquele ano, as agências governamentais e companhias privadas americanas teriam gasto US$ 81 bilhões apenas em projetos cancelados, e mais US$ 59 bilhões em projetos concluídos fora do tempo previsto (THE STANDISH..., 1995).

O grupo continuou publicando regularmente seu relatório nos anos seguintes e, apesar de 39% dos projetos de software iniciados em 2012 terem obtido sucesso ("successful"), ainda é preocupante saber que 18% de todos eles fracassam, não sendo finalizados ou nunca usados ("failed"), e 43% sofrem com algum problema de atraso, extrapolam o orçamento e/ou não atendem aos requisitos. Estas são as estatísticas até 2012 publicadas pelo *The Standish Group* (2013), conforme pode ser visto no quadro a seguir.

RESOLUÇÃO

	2004	2006	2008	2010	2012
Bem-sucedidos	29%	35%	32%	37%	39%
Malsucedidos	18%	19%	24%	21%	18%
Déficits/ falhas	53%	46%	44%	42%	43%

Resultados da resolução do projeto da pesquisa CHAOS entre os anos 2004-2012.

THE CHAOS MANIFESTO 1

Quadro 1. Resultados do Chaos Manifesto até 2012.
Fonte: The Standish Group (2013).

Ao observar esses números, fica claro que mais de 50 anos de experiência no desenvolvimento de software não bastaram para melhorar efetivamente a qualidade do software, a despeito da evolução ocorrida na área de engenharia de software e do ferramental disponível. Grady Booch, um dos criadores da UML (*Unified Modeling Language*), resumiu a história dizendo que uma "doença" que dure tanto tempo quanto esta deveria ser chamada de "anormalidade" (BOOCH; RUMBAUGH; JACOBSON; 2006)

Grady Booch
Fonte: http://goo.gl/iGW9QB

Aula 2 – Conceitos fundamentais da engenharia de software

2.1 Definições e conceitos de software e engenharia de software

Engenharia de software é uma disciplina da Engenharia que cuida de diversos aspectos do trabalho de desenvolvimento de sistemas de software, envolvendo todas as etapas do ciclo do projeto, desde a definição de requisitos até a sua manutenção, que ocorre após a entrega do produto e o início de sua operação. Seu principal objetivo é fornecer uma estrutura metodológica para a construção de um software com alta qualidade.

Podemos definir engenharia de software como um processo que envolve a criação e a utilização de sólidos princípios de engenharia, a fim de obter softwares que sejam:

- de alta qualidade;
- produzidos de maneira econômica;
- confiáveis;
- de trabalho eficiente em máquinas reais;
- entregues no prazo;
- feitos gerando satisfação ao cliente.

De acordo com Pressman (2010), a engenharia de software é uma tecnologia em camadas, apoiada fundamentalmente em um compromisso organizacional com a qualidade, como mostra a Figura 2. Nesta abordagem, podemos observar que o alicerce da engenharia de software é a camada de processo, que funciona como um adesivo que mantém unidas as camadas ligadas à tecnologia.

Figura 2. Engenharia de software: uma tecnologia em camadas.
Fonte: baseado em Pressman (2010).

O **processo** forma uma base que mantém unidas as camadas de tecnologia, permitindo o desenvolvimento de software com qualidade. Além disso, ele define o arcabouço que garante a efetiva utilização das técnicas de Engenharia de Software.

Os **métodos** fornecem as técnicas de "como fazer" a construção de software. Abrangem um amplo conjunto de tarefas que incluem comunicação, análise de requisitos, modelagem de projeto, construção de programas, testes e manutenção.

As **ferramentas** fornecem apoio automatizado ou semiautomatizado para o processo e seus métodos (PRESSMAN, 2010).

Em função de sua indiscutível importância, as técnicas e métodos da engenharia de software são atualmente muito usadas, mas ainda não o são utilizadas por todos e nem da maneira correta.

Não podemos prosseguir falando da engenharia de software sem antes entendermos o software e seus fundamentos. Assim, os conceitos de software são apresentados aqui como uma referência inicial para o estudo do software, propriamente dito, e de seus processos de desenvolvimento.

Pressman (2006, p. 4) conceitua o software como:

> (1) Instruções (programas de computador) que, quando executadas, produzem a função e o desempenho desejados; (2) Estruturas de dados que possibilitam que os programas manipulem adequadamente a informação; (3) Documentos que descrevem a operação e o uso dos programas.

As normas de gestão de qualidade e garantia da qualidade apresentam definições de software e seus componentes e processos. A norma NBR ISO 9000-3 – que é uma interpretação da norma de garantia de qualidade ISO 9001 para aplicação aos produtos de software – traz as seguintes definições:

- Software: criação intelectual compreendendo os programas, procedimentos, regras e qualquer documentação correlata à operação de um sistema de processamento de dados.
- Produto de software: conjunto completo de programas de computador, procedimentos e documentação correlata, assim como dados designados para entrega a um usuário.
- Item de software: qualquer parte identificável de um produto de software em etapa intermediária ou na etapa final de desenvolvimento.
- Desenvolvimento: todas as atividades a serem realizadas para a criação de um produto de software.
- Fase: segmento definido do trabalho.

Como podemos observar, o conjunto de conceitos apresentados deixa claro que o software é um produto complexo que exige cuidados constantes. Dessa forma, o controle da qualidade não pode ser uma atividade secundária, mas deve estar presente desde o início de seu desenvolvimento até a análise final de entrega.

2.2 Software x hardware

De acordo com Pressman (2010), comparar o software com produtos de hardware auxilia na compreensão das diferenças existentes entre eles e enfatiza as características inerentes a um software. Ainda segundo o autor, o processo de desenvolvimento do software apresenta diferenças fundamentais em relação ao hardware:

- O processo criativo do hardware gera algo físico (por exemplo, placas de circuitos). O desenvolvimento de software resulta em um elemento pertencente a um sistema lógico, intangível;
- O software geralmente é desenvolvido sob medida, ao contrário do hardware, no qual o projetista tem acesso a componentes existentes que executam tarefas definidas. O projetista do software nem sempre terá acesso a módulos prontos para utilização. Quando o faz, pode elevar o risco do produto devido a questões de segurança;
- Os custos do software estão concentrados no desenvolvimento e não no processo de manufatura. Isso significa que não pode ser gerido como projeto de manufatura;
- Ao longo do tempo, o produto de software não se desgasta, mas se deteriora em função da introdução de erros oriundos de atividades de manutenção ou evoluções implícitas no processo, que devem ser reconsideradas no modelo original.

Dessa forma, o software sofre deterioração ocasionada por diversos fatores, sendo uma característica peculiar do produto. Pressman (2010) relata que, no caso do hardware, temos um alto índice de falhas no início do seu ciclo de vida, ocasionadas por defeitos de fabricação e projeto. Posteriormente os defeitos são corrigidos, dando estabilidade nas falhas ou mantendo-a em um nível muito baixo e suportável para a estrutura.

No final do ciclo de vida do produto podem surgir problemas relacionados ao envelhecimento, acúmulo de poeira, vibração, abuso, temperaturas extremas, entre outros. Este processo pode ser visto no Gráfico 1.

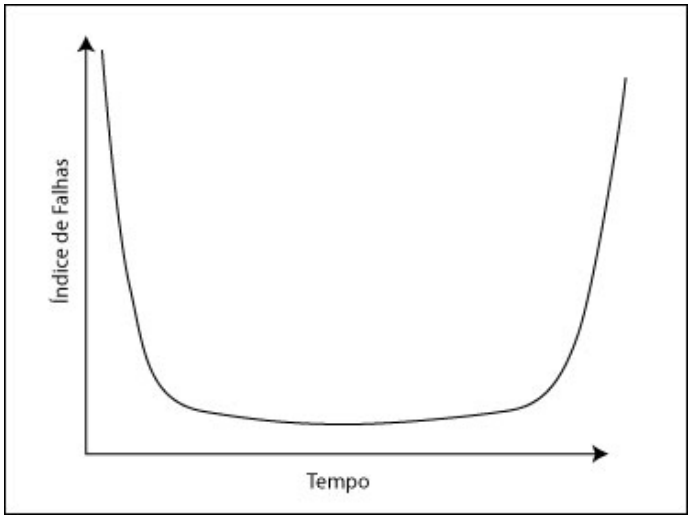


Gráfico 1. Curva de falhas do hardware.
Fonte: Pressman (2010).

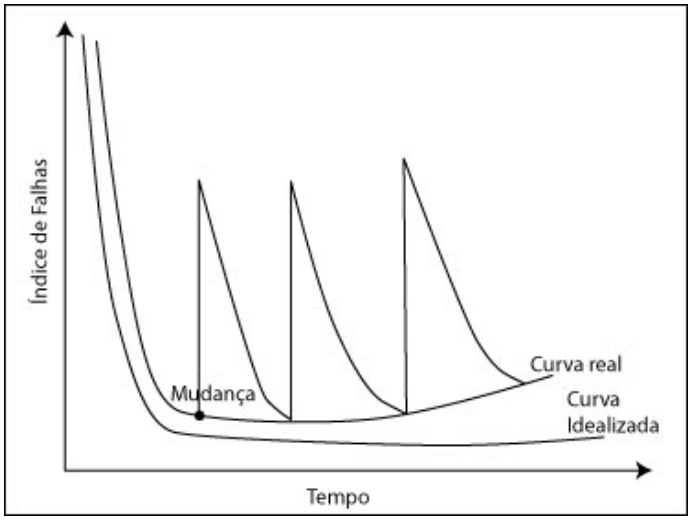


Gráfico 2. Curva de falhas do software.
Fonte: Pressman (2010).

Diferentemente da curva teórica de falhas do hardware, a de software leva em conta que este não sofre processos de envelhecimento como o hardware, pois o software não é algo físico. No início do ciclo de vida do software teremos problemas (*bugs*) que serão ajustados no decorrer do desenvolvimento e se estabilizarão, gerando uma tendência de achatamento da curva, conforme pode ser visto no Gráfico 2.

Notemos que esta é apenas uma teoria, já que a curva real do índice de falhas de um software considera o processo de manutenção e mudanças.

Durante o processo de refinamento do produto ou nas mudanças aumenta-se, consideravelmente, a probabilidade de inserção de novos erros, gerando picos na curva de falhas. As sucessivas alterações do software tendem a introduzir mais erros antes da estabilização de erros de alterações anteriores, ocasionando a tendência crescente do índice de falhas.

Isso nos remete a uma realidade bastante complicada em relação ao software: ele é um produto que quanto mais se conserta, pior fica. Um software em constante manutenção pode se tornar uma espécie de "colcha de retalhos", gerando pontos de falha diversos, difíceis de serem identificados e ajustados. O melhor modelo é o desenvolvimento de um novo produto após o tempo de vida útil do mesmo, considerando-se as novas necessidades e tecnologias disponíveis.

Quando o hardware é projetado e construído, os componentes digitais são inseridos em catálogos. A cada circuito integrado ao componente é assinalado um código, uma função definida e validada, uma interface especificada e um conjunto padrão de integração. Uma vez escolhido no catálogo, o componente desejado pode ser adquirido e utilizado em diferentes projetos de hardware, com alta confiabilidade.

No mundo do software, isso é algo que está apenas começando a ser utilizado numa escala mais ampla, apesar de existirem alguns casos antigos de reuso, como as bibliotecas de sub-rotinas científicas.

Atualmente, a visão de reuso foi ampliada para abranger não apenas algoritmos consagrados, mas também estruturas de dados, interfaces gráficas e diferentes classes e componentes orientados a objetos.

Exercícios do Módulo 1

1) Qual das questões abaixo não é mais uma das grandes preocupações de um engenheiro de software?

a) Por que normalmente se gasta tanto tempo para desenvolver software?
b) Por que, via de regra, o software custa tão caro?
c) Por que quase sempre não se consegue remover todos os erros do software antes de sua entrega?
d) Por que o hardware é tão caro?

2) O software é um produto que pode ser manufaturado usando as mesmas tecnologias utilizadas para outros artefatos de engenharia.

a) Verdadeiro.
b) Falso.

3) O software deteriora-se em vez de desgastar porque:

a) Ele "sofre" quando exposto a ambientes "hostis".
b) Defeitos são mais prováveis de surgir quando o software é usado várias vezes.
c) Mudanças frequentes aumentam a probabilidade de se introduzir erros no software.
d) Componentes de reposição de software são difíceis de encontrar no mercado.

4) Atividades "guarda-chuva" de engenharia de software são aplicadas somente durante as fases iniciais do desenvolvimento de software.

a) Verdadeiro.
b) Falso.

5) O que não está relacionado à chamada "crise do software"?

a) O ponto fundamental foi o fato de que os softwares eram entregues fora do prazo, com custos mais elevados do que os projetados inicialmente e com uma qualidade aquém do esperado.
b) Foi um termo criado para descrever as dificuldades enfrentadas no desenvolvimento de software no final da década de 1960.
c) A complexidade dos problemas, a ausência de técnicas bem estabelecidas e a crescente demanda por novas aplicações começavam a se tornar um problema sério.
d) Essa crise teve como origem a introdução de computadores pessoais na década de 1970.

6) A essência da prática da engenharia de software pode ser resumida em compreender o problema, planejar a solução, executar o plano e examinar o resultado.

a) Verdadeiro.
b) Falso.

7) Qual dos itens listados abaixo não se constitui numa das camadas da engenharia de software?

a) Processo.
b) Manufatura.
c) Métodos.
d) Ferramentas.

8) Complete as lacunas:

O processo criativo do hardware gera algo _____. O desenvolvimento de software resulta em um sistema lógico, _____. Os custos do software estão concentrados no _____ e não no processo de _____. Ao longo do tempo, o produto de software não se _____, mas se _____em função da introdução de erros oriundos de atividades de manutenção ou evoluções do sistema.

MÓDULO 2 – PROCESSO DE SOFTWARE

Neste módulo descreveremos o que é um processo de software, bem como os principais padrões de processo que apoiam o desenvolvimento de software: modelo em cascata, modelos evolucionários (modelo de prototipagem, espiral e concorrente), modelos incrementais (RAD e incremental), processo unificado e métodos ágeis de desenvolvimento.

Aula 3 – Fundamentos do processo de software

3.1 Processo de software

Todo projeto de software inicia-se a partir de alguma necessidade do negócio. Assim que essa necessidade é identificada, costuma ser expressa de maneira informal, por uma conversa. Mas tal informalidade deve parar por aí.

Até mesmo a especificação da necessidade do cliente é abrangida pelos métodos e técnicas da engenharia de software, e esse é um processo bastante complexo. Então, vamos começar a entendê-lo conhecendo exatamente do que se trata um processo de software.

Para que as necessidades da empresa ou de um cliente possam se transformar numa solução de software, todo o diálogo e a interação entre usuários, projetistas, ferramentas de desenvolvimento e tecnologias devem ser transformados num processo.

> Assim, processo de software é um arcabouço (framework) das tarefas requeridas para se construir um software de alta qualidade.

A partir daqui uma pergunta pode surgir: processo e engenharia de software são a mesma coisa? A resposta é "sim" e "não". O processo de software define a abordagem que é adotada quando o software é elaborado. A engenharia de software engloba também as tecnologias que constituem um processo, como métodos, técnicas e ferramentas de desenvolvimento. Assim, a engenharia de software engloba os processos de software.

Para que a engenharia de software possa ser aplicada como uma abordagem disciplinada para o desenvolvimento, operação e manutenção de um software, um processo deve ser definido. De uma forma geral, um processo é caracterizado por fases:

1) *Fase de definição:* esta fase se concentra no que o sistema de software irá realizar, isto é, identifica:

- que informação deve ser processada;
- que função e desempenho são desejados;
- que comportamento deve ser esperado do sistema;
- que interfaces devem ser estabelecidas;
- que restrições de projeto existem;
- que critérios de validação são necessários.

Nesta fase, os requisitos-chave do sistema e do software são identificados. Ela engloba três etapas importantes:

1) Engenharia de sistemas ou de informação;
2) Planejamento do projeto;
3) Análise de requisitos.

2) *Fase de desenvolvimento:* esta fase foca em como o desenvolvimento será realizado, definindo:

- como os dados devem ser estruturados;
- como as funções devem ser implementadas;
- como os detalhes procedimentais devem ser implementados;
- como as interfaces devem ser caracterizadas;
- como o projeto deve ser traduzido em uma linguagem de programação;
- como o teste vai ser realizado.

Nesta fase, três etapas técnicas específicas ocorrerão:

1) Projeto do software;
2) Geração de código;
3) Teste de software.

3) *Fase de manutenção:* esta fase tem como alvo as modificações e manutenções que o software sofrerá. Durante ela, quatro tipos de modificações são encontradas:

- Manutenção corretiva: modifica o software para corrigir defeitos;
- Manutenção adaptativa: modifica o software para acomodar mudanças em seu ambiente externo (processador, sistema operacional, etc.);
- Manutenção de aperfeiçoamento: aprimora o software além dos requisitos funcionais originais (cliente/usuário reconhece e solicita funcionalidades adicionais que trarão benefícios, à medida que o software é usado).
- Manutenção preventiva: faz modificações nos programas de modo que eles possam ser mais facilmente corrigidos, adaptados e melhorados.

Essas três fases são complementadas por atividades "guarda-chuva", que são aplicadas ao longo do processo de software:

- Controle e rastreamento do projeto;
- Gestão de riscos;
- Revisões técnicas formais;
- Garantia de qualidade;
- Gestão de configuração de software;
- Produção e preparação de produtos do trabalho (documentos);
- Gestão de reusabilidade;
- Medição.

Como vimos nesta aula, a engenharia de software é uma disciplina que integra o processo, métodos e ferramentas para o desenvolvimento de projetos de software. Há diversos modelos de processo, mas todos possuem algumas características em comum, definindo um conjunto de atividades de arcabouço, uma coleção de tarefas para que se consiga realizar as atividades, produtos de trabalho produzidos nas tarefas e um conjunto de atividades guarda-chuva que apoiam outras tarefas de todo o processo.

O *ciclo de vida* de um software descreve as fases pelas quais o software passa desde a sua concepção até a descontinuidade de seu uso. O conceito de ciclo de vida de um software é muitas vezes confundido com o de modelo de processo, embora sejam bem diferentes, como veremos a seguir.

Aula 4 – Modelos prescritivos de processo – Parte 1

Os processos prescritivos são uma categoria que engloba os processos que possuem pontos observáveis, ou seja, aqueles que a cada passo podem ser verificados e classificados como válidos ou sujeitos a ajustes.

Enquanto um modelo descritivo retrata como um processo é executado, um modelo prescritivo retrata como um processo deveria ser executado. Assim, um modelo prescritivo é uma recomendação que pode ser adaptada ou melhorada pela empresa/equipe de software que for adotá-la.

Os instrumentos prescritivos do processo de planejamento estratégico explicitam o que deve ser feito pela organização para se direcionar o esforço de desenvolvimento de software. Um modelo prescritivo de processo atua como complemento com conjuntos explícitos de tarefas explícitas para o desenvolvimento. Cada modelo prescritivo de processo também prescreve um fluxo de trabalho ou maneira como os elementos se inter-relacionam.

Há vários modelos que são classificados nesta categoria:

- Modelo em cascata
- Modelos evolucionários
  - Modelo de prototipagem
  - Modelo espiral
  - Modelo concorrente
- Modelos incrementais
  - Modelo RAD
  - Modelo incremental
- Modelo baseado em componentes
- Modelo de métodos formais
- Processo unificado

4.1 Modelo em cascata

No modelo em cascata, também conhecido como ciclo de vida clássico, o processo de desenvolvimento de software é visto como uma abordagem sistemática e sequencial, que começa com a especificação dos requisitos do cliente e progride seguindo as etapas de planejamento, modelagem, construção e implantação do sistema, culminando na manutenção progressiva do produto entregue, conforme ilustra a Figura 3.

Figura 3. Modelo em cascata.
Fonte: baseado em Pressman (2010).

Esse modelo é o paradigma mais antigo da engenharia de software, sendo bastante criticado em função dos problemas encontrados nos projetos em que é aplicado. A realidade tem mostrado que num projeto raramente se segue o fluxo sequencial que o modelo propõe, gerando problemas futuros que oneram os custos e prazos. Uma das causas mais comuns deste problema é a dificuldade do cliente em declarar claramente todas as suas necessidades e expectativas, ou seja, de definir todos os requisitos inicialmente.

O foco incorreto ou não claro pode gerar uma distorção, que reflete diretamente na percepção de qualidade por parte do próprio cliente. Isso pode levar a entregas parciais do produto, o que exige que o cliente tenha muita paciência durante os aceites parciais que são formalizados em um Documento de Encerramento do Projeto, com observações no campo Restrições "Entrega Parcial de Projeto", que contém detalhes quanto à aceitação condicional do projeto por parte do cliente.

Os trabalhos de desenvolvimento de software atuais seguem ritmos muito rápidos e sujeitos a diversas modificações, o que torna o modelo em cascata inadequado para esses tipos de projeto. Cumpre ressaltar que, embora o modelo em cascata ou ciclo de vida clássico tenha fragilidades, ele é significativamente melhor do que uma abordagem casual para o desenvolvimento de software.

4.2 Modelos evolucionários de processo

Modelos evolucionários de processo são aqueles que consideram a natureza evolutiva do software. Os modelos evolucionários são iterativos, sendo implementados de forma a permitir o desenvolvimento de versões cada vez mais completas do software. Citamos a seguir algumas de suas características:

- São usados quando o deadline (limite de tempo) não é adequado para o desenvolvimento do software, ou seja, a data de término não é realística (por exemplo, prazos reduzidos de mercado face à competitividade).
- Uma versão limitada pode ser introduzida para atender a essa competitividade e às pressões do negócio.
- São liberados produtos core (núcleo dos produtos) ao longo do desenvolvimento.
- Os detalhes e extensões do projeto são definidos ao longo do desenvolvimento.

Os modelos que são agrupados nesta categoria são: prototipação, modelo espiral e modelo concorrente, que serão apresentados a seguir.

4.2.1 *Prototipação*

É um modelo de processo que possibilita que o desenvolvedor crie um modelo do software que deve ser construído. Idealmente, o modelo (protótipo) serve como um mecanismo para identificar os requisitos de software. É indicado quando o cliente estabelece um conjunto de objetivos gerais para o software, sem identificar detalhadamente requisitos de entrada, processamento e saída.

Envolve o desenvolvimento de uma versão inicial do sistema, baseada no atendimento dos requisitos ainda pouco definidos. Esse processo permite a descoberta de falhas difíceis de serem encontradas na comunicação verbal, servindo de apoio à fase de levantamento de requisitos e prevenindo as possíveis falhas no sistema.

O objetivo principal de um protótipo é simular a aparência e funcionalidade do software, permitindo que os clientes, analistas, desenvolvedores e gerentes compreendam plenamente os requisitos do sistema, interagindo, avaliando, alterando e aprovando as características mais relevantes que o produto deva ter. A Figura 4 ilustra este processo.

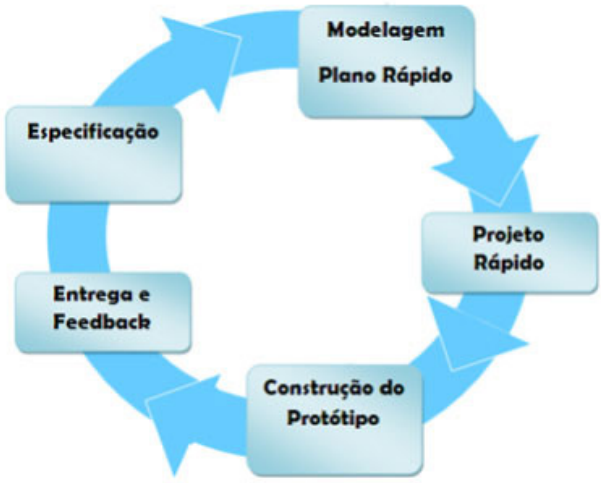


Figura 4. Modelo de prototipação.
Fonte: baseado em Pressman (2010).

Certamente, a redução de custos no desenvolvimento é um dos grandes ganhos de prototipação, pois envolve diretamente o usuário final, permitindo um desenvolvimento mais próximo dos desejos do cliente e priorizando a facilidade de uso. Assim, pode-se obter um nível de satisfação maior, em função do menor número de erros ou falhas de desenvolvimento comuns na passagem de informação entre o analista (que fez o levantamento de requisitos) e o desenvolvedor (equipe de desenvolvimento).

Um dos riscos envolvidos nesse modelo é o descomprometimento com a análise do produto, visto que os envolvidos podem se apoiar totalmente no modelo prototipado, gerando uma expectativa muitas vezes irrealista de desempenho em função do protótipo ser muito mais enxuto que o produto final e estar em um ambiente controlado.

Além disso, o cliente não sabe que o software que ele vê não considerou, durante o desenvolvimento, a qualidade global e a manutenibilidade em longo prazo. Ele pode, também, não aceitar facilmente a ideia de que a versão final do software está sendo construída e tentar forçar a utilização do protótipo como produto final.

Outro risco desse modelo é que o desenvolvedor frequentemente faz uma implementação comprometida (utilizando partes de programas existentes, geradores de relatórios e de janelas) com o objetivo de produzir rapidamente um protótipo executável. Depois de um tempo ele se familiariza com essas escolhas, e pode se esquecer que elas não são apropriadas para o produto final.

Embora não seja livre de problemas, a prototipação é um modelo eficiente, pois cada protótipo é construído a partir de um acordo entre o cliente e o desenvolvedor. Deve estar claro para ambos que o protótipo é utilizado como um mecanismo que permite a definição dos requisitos do projeto. A figura 5 traz uma outra visão deste processo.

Figura 5. Outra visão do modelo de prototipação.
Fonte: baseado em Pressman (2010).

4.2.2 *Modelo espiral*

O modelo espiral foi desenvolvido para abranger as melhores características do modelo em cascata e da prototipação, acrescentando, ao mesmo tempo, um novo elemento: a análise de riscos. Foi desenvolvido por Barry Boehm (1988) em seu artigo intitulado A Spiral Model of Software Development and Enhancement. Esse modelo foi o primeiro a explicar o porquê do modo iterativo e a elencar suas vantagens.

As iterações têm uma duração típica de seis meses a dois anos. Cada fase inicia-se com um objetivo esperado e termina como uma revisão do progresso pelo cliente, que deve ser interna, e assim por diante. Esforços de análise e engenharia são aplicados em cada fase, tendo sempre o foco no objetivo do projeto. A Figura 6 ilustra este processo.

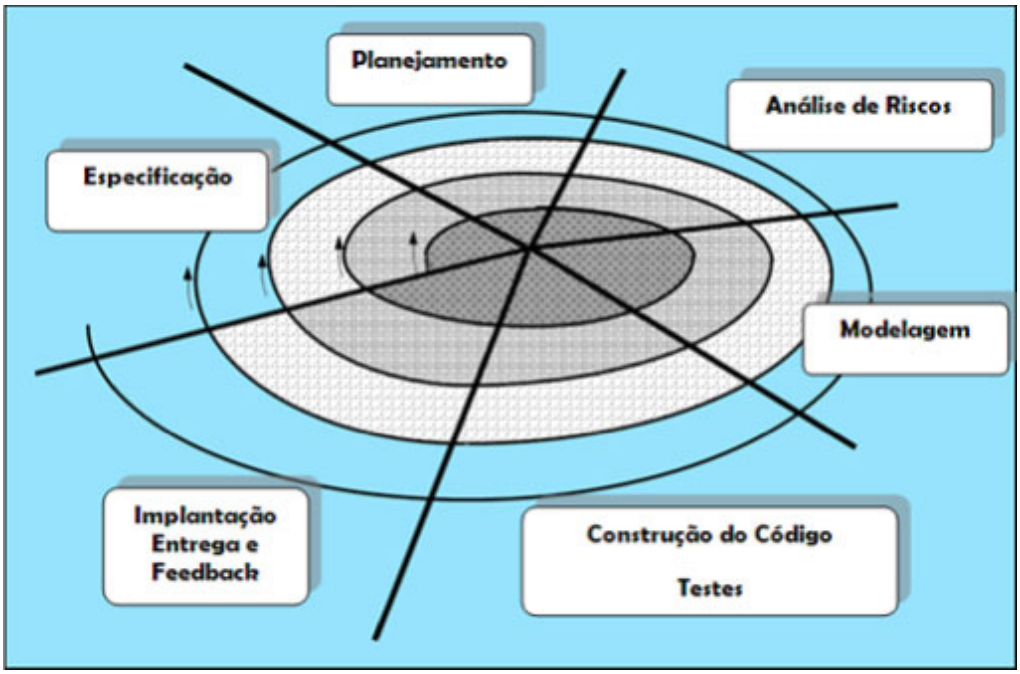


Figura 6. Modelo espiral.
Fonte: baseado em Pressman (2010).

As principais características desse modelo são:

- Engloba a natureza iterativa da prototipação e os aspectos sistemáticos e controlados do modelo em cascata.
- Fornece potencial para o desenvolvimento rápido de versões incrementais do software.
- O processo se inicia com a equipe de desenvolvimento movendo-se em volta da espiral, no sentido horário, a partir do centro.
- O primeiro circuito em torno da espiral pode resultar na especificação do produto.
- Nas primeiras iterações, a versão incremental pode ser um modelo em papel ou um protótipo.
- Nas iterações mais adiantadas são produzidas versões incrementais mais completas e melhoradas.

Trata-se de uma abordagem realística para o desenvolvimento de software de grande porte. Como o software evolui na medida em que o processo avança, o cliente e o desenvolvedor entendem melhor e reagem aos riscos em cada nível evolucionário. Para pequenos projetos, o conceito de desenvolvimento de software ágil torna-se uma alternativa mais viável.

Como vantagens desse modelo, podemos citar as estimativas realísticas dadas à identificação de problemas importantes logo no início do processo, a versatilidade para lidar com mudanças (quando inevitáveis), o desenvolvimento antecipado por parte dos engenheiros de software — que têm visibilidade das necessidades por fases — e o uso

da prototipagem (em qualquer estágio de evolução do produto) como mecanismo de redução de risco.

No entanto, o uso do modelo espiral exige experiência significativa para determinar os riscos, sendo dependente dessa experiência para obter sucesso. Além disso, pode ser difícil convencer os clientes que uma abordagem "evolutiva" é controlável.

4.2.3 *Modelo de desenvolvimento concorrente*

De acordo com Pressman (2010), o modelo de desenvolvimento concorrente, também chamado de engenharia concorrente, pode ser representado, esquematicamente, como uma série de atividades de arcabouço, ações e tarefas da engenharia de software e seus estados associados.

Um exemplo do uso desse modelo pode ver visto na Figura 7, que traz a atividade "modelagem" do modelo Espiral (mostrado na Figura 6), espelhada no modelo concorrente.

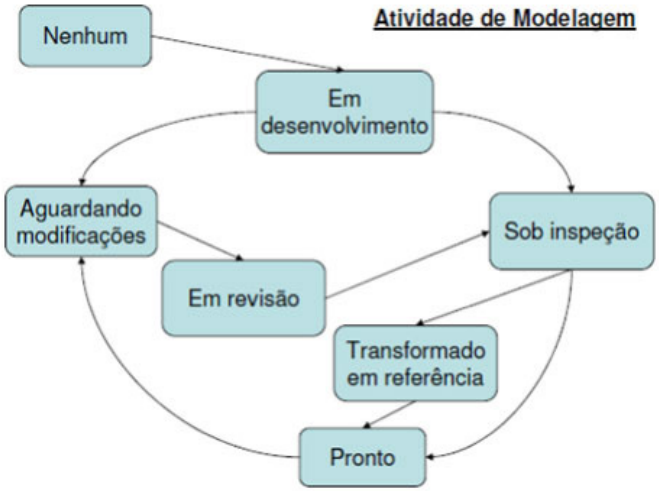


Figura 7. Transições de estado no modelo de desenvolvimento concorrente.
Fonte: baseado em Pressman (2010).

No modelo concorrente, uma atividade pode estar em qualquer um dos estados apresentados na Figura 7 (em desenvolvimento, sob inspeção, aguardando modificações, etc.) a qualquer tempo. O modelo define uma série de eventos que vão disparar transições de um estado para outro, para cada uma das atividades, ações ou tarefas da engenharia de software.

Por exemplo, suponha que durante os estágios da etapa de projeto, descobre-se uma inconsistência no modelo de análise. Isso geraria o evento "correção no modelo de análise", que, por sua vez, implicaria na passagem da atividade de análise do estado "pronto" para o estado "aguardando modificações".

De forma geral, as principais características desse modelo são:

- Todas as atividades ocorrem em paralelo, mas estão em diferentes estados.
- O modelo define uma série de eventos que vão disparar transições de estado a estado, para cada uma das atividades.
- Em vez de usar uma sequência como o modelo em cascata, ele define uma rede de atividades.
- Eventos gerados dentro de certa atividade — ou em algum outro lugar da rede de atividades — disparam transições entre estados de uma atividade
- Pode ser aplicado a todo tipo de desenvolvimento de software e fornece uma visão exata de como está o estado do projeto.
- Em vários projetos pode existir uma simultaneidade (concorrência) entre as várias atividades de desenvolvimento e de gestão de projetos.
- É representado como uma série de grandes atividades técnicas, tarefas e seus estados associados (fornece um panorama preciso do estado atual do projeto).

O software moderno é caracterizado por modificações contínuas, prazos muito curtos e por uma necessidade premente de satisfazer o usuário ou cliente. Os modelos evolucionários de processo foram criados com o objetivo de resolver esses problemas; contudo, também têm suas fragilidades. Uma delas é que a prototipagem pode trazer problemas para o planejamento do projeto, em função do número impreciso de ciclos necessários para se concluir o produto. Outra dificuldade se refere ao foco dado à flexibilidade e extensibilidade, em vez da alta qualidade. Se o foco for centrado na qualidade, o projeto pode resultar em atraso, o que pode comprometer a competitividade do empreendimento.

Aula 5 – Modelos prescritivos de processo – Parte 2

5.1 Modelos incrementais de processo

Há diversas situações em que os requisitos iniciais do software estão razoavelmente bem definidos, mas o escopo global do processo de desenvolvimento claramente elimina uma abordagem puramente linear ou sequencial.

Adicionalmente, pode haver a necessidade de se fornecer rapidamente um conjunto limitado de funcionalidades do software aos usuários e depois refinar, melhorar e expandir aquela funcionalidade em versões mais avançadas do software. Nesses casos, os modelos de processo que produzem software em incrementos são os mais indicados.

Os processos incrementais que discutiremos aqui incluem o modelo incremental e o modelo RAD — *Rapid Application Development* (Desenvolvimento Rápido de Aplicação).

5.1.1 *Modelo incremental*

O modelo incremental aplica elementos do modelo em cascata, mas de forma interativa. Este modelo, assim como a prototipagem, é incremental, mas o que o diferencia da prototipagem é que tem como objetivo apresentar um produto operacional a cada incremento realizado. Esse processo pode ser visto na Figura 8.

Tal modelo é muito útil quando a empresa não possui mão de obra disponível num dado período para uma implementação completa, dentro do prazo estipulado.

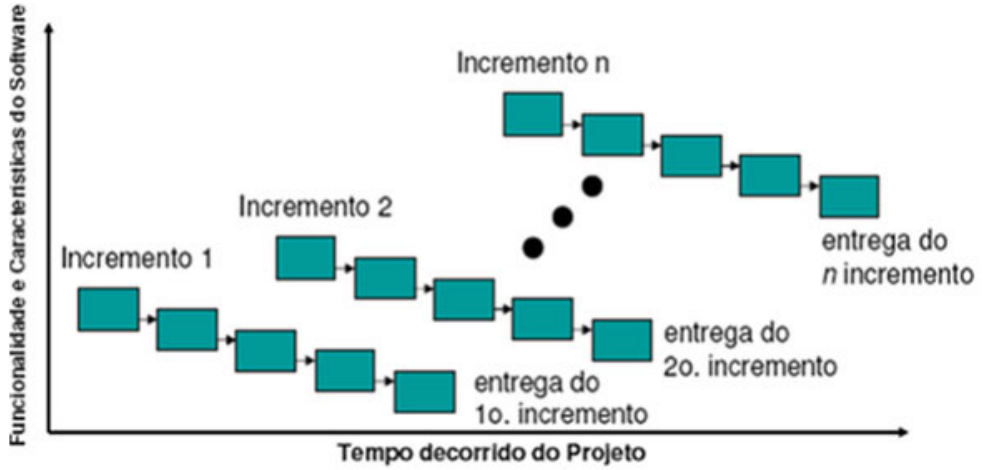


Figura 8. Modelo de processo incremental.
Fonte: baseado em Pressman (2010).

De uma forma geral, o modelo incremental apresenta as características:

- Combina elementos do modelo em cascata (aplicado repetitivamente) com a filosofia iterativa da prototipação.
- Aplica sequências lineares de uma forma racional à medida que o tempo passa.
- Cada sequência linear produz um incremento do software e pode gerar uma entrega parcial do produto.
- Os primeiros incrementos são versões simplificadas do produto final.
- O primeiro incremento é chamado de "núcleo do produto" (core).

Um exemplo clássico de aplicação do modelo incremental é o desenvolvimento de um processador de texto. Para esse projeto as etapas incrementais poderiam ser assim definidas:

1) Primeiro incremento: poderia efetuar as funções de controle de versões de arquivos, edição e produção de documentos.
2) Segundo incremento: adicionaria capacidade de edição e de produção de documentos mais sofisticados.
3) Terceiro incremento: incluiria a verificação sintática e gramatical
4) Quarto Incremento: adicionaria a capacidade avançada de disposição de página.

5) Note que todo o processo pode se repetir até que um produto completo seja produzido.

5.1.2 *Modelo RAD — Rapid Application Development*

O modelo RAD é uma adaptação do modelo em cascata, mas que enfatiza um desenvolvimento extremamente rápido. A alta velocidade é conseguida por uma abordagem de construção baseada em componentes, ou seja, o sistema é modularizado.

Se os requisitos forem bem definidos e o objetivo do projeto for restrito, a equipe pode criar um sistema plenamente funcional em pouco tempo.

O RAD enquadra-se no modelo de atividades de arcabouço do modelo em cascata:

- Especificação: atividade em que se entende o problema de negócio, as características das informações e é realizado o levantamento de requisitos.
- Planejamento: atividade essencial. Várias equipes trabalham em paralelo em diferentes funções.
- Modelagem: estabelece representações de projeto que servem como base para a atividade de construção. Abrange três fases:
  - » Modelagem de negócio;
  - » Modelagem de dados;
  - » Modelagem de processo.
- Construção: faz uso de componentes de softwares preexistentes e geração de códigos.
- Implantação: estabelece a base para iterações subsequentes, se necessárias.

A Figura 9 apresenta a representação esquemática do modelo RAD em relação ao modelo sequencial tradicional.

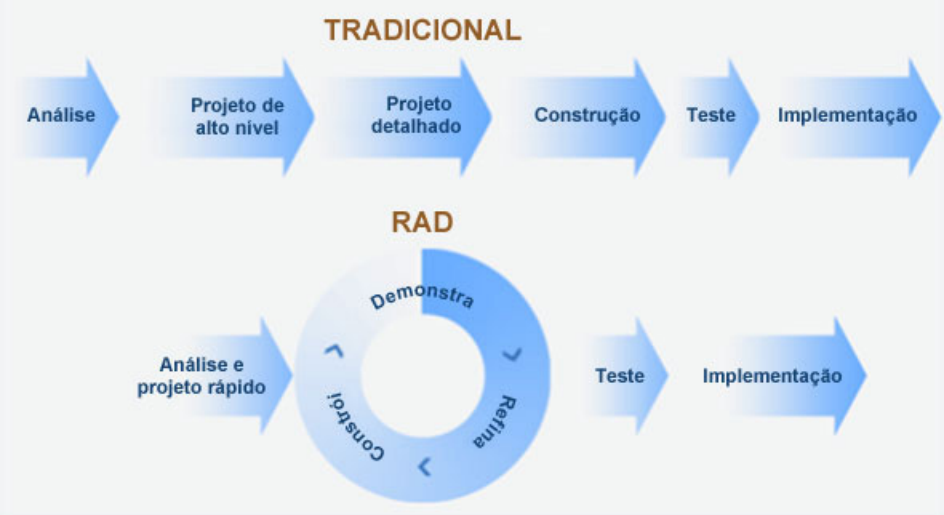


Figura 9. Modelo RAD x modelo tradicional.

As situações de desenvolvimento mais adequadas para se utilizar o modelo RAD incluem:

- Projetos em que os requisitos estão bem definidos, o objetivo do sistema é restrito e se deseja criar um "sistema plenamente funcional" dentro de períodos muito curtos (por exemplo, de 60 a 90 dias);
- Projetos em que há fortes restrições de tempo impostas pelo cliente;
- Aplicações que podem ser modularizadas de forma que cada função principal possa ser completada em menos de três meses;
- Projetos em que cada função principal possa ser alocada para uma equipe distinta e depois integrada para formar o todo do produto.

Mas a utilização do modelo RAD também pode implicar em problemas. Para projetos grandes, porém mensuráveis, o RAD requer um número elevado de recursos humanos que sejam suficientes para criar um número adequado de equipes. Além disso, o RAD requer um comprometimento entre desenvolvedores e clientes, para que as atividades possam ser realizadas rapidamente e o sistema seja concluído em um curto espaço de tempo. Se o comprometimento for abandonado por qualquer das partes, o projeto falhará. O uso do RAD também não é apropriado quando os riscos técnicos são grandes (por exemplo, quando a aplicação faz uso de uma nova tecnologia).

5.2 Desenvolvimento baseado em componentes

O desenvolvimento baseado em componentes (ou CBD — *Component-based Development*), também é conhecido como *Component-based Software Engineering* (CBSE) ou simplesmente como componente de software.

Atualmente, é muito comum que os desenvolvedores utilizem componentes de software que são encontrados em bibliotecas de uso gratuito ou mesmo disponíveis para compra. Esses componentes são conhecidos como COTS (*Commercial off-the-shelf*), ou software comercial de prateleira.

Geralmente esses componentes oferecem funcionalidades com interfaces bem definidas, que podem ser facilmente integradas no software que estiver sendo desenvolvido.

O método de desenvolvimento baseado em componentes incorpora as características de construção de componentes de biblioteca ao modelo espiral. De natureza evolucionária, adota uma abordagem iterativa para a criação de software. Assim, o modelo constrói aplicações a partir de componentes de software previamente preparados.

As atividades de modelagem e construção começam com a identificação de componentes candidatos e estes, por sua vez, podem ser projetados como módulos de software convencional ou como classes ou pacotes de classes orientadas a objeto.

No paradigma orientado a objetos, uma classe encapsula dados e algoritmos, que também podem ser utilizados para manipular os dados. Por meio dessa abordagem, uma biblioteca de classes pode ser construída com as classes identificadas no desenvolvimento do software e, a partir de então, toda iteração da espiral deverá verificar o conteúdo da biblioteca que pode ser reutilizado. A Figura 10 ilustra esse processo.

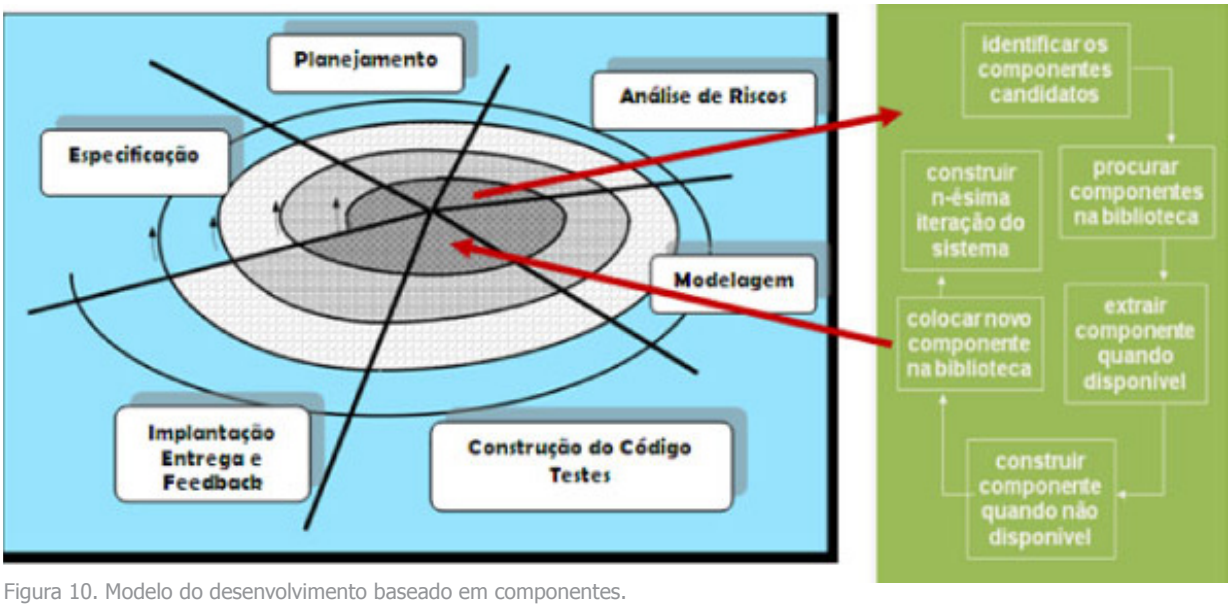


Figura 10. Modelo do desenvolvimento baseado em componentes.
Fonte: baseado em Pressman (2010).

De acordo com Pressman (2010), o modelo de desenvolvimento baseado em componentes leva ao reuso de software, e a reusabilidade fornece aos engenheiros vários benefícios. Com base nisso, estudos relatam uma redução de 70% no prazo do ciclo de desenvolvimento e de 84% no custo dos projetos desenvolvidos nesse modelo de processo, com índices de produtividade em torno de 26.2 — superior ao índice padrão de 16.9 da indústria.

O **reuso de software** se refere à utilização de software existente para o desenvolvimento de um novo software. A decisão quanto à utilização de componentes reutilizáveis envolve comparações entre o investimento necessário para sua aquisição e os gastos para o desenvolvimento de uma solução customizada. Devem ser estimados os custos e benefícios líquidos no investimento em reuso de software e avaliar os ganhos com a adoção do reuso.

Muitos métodos para o reuso estão em prática no mercado atual, sendo sua escolha determinada pela adequação ao modelo de negócio ou às tecnologias utilizadas.

O RiSE (*Reuse in Software Engineering*) tem como objetivos principais a redução dos custos do processo de desenvolvimento e o aumento da produtividade, sem perdas na qualidade do software.

O CRUISE (*Component Reuse in Software Engineering*) é apresentado no formato de livro on-line, com acesso livre e foco em reuso de software. Cada assunto do livro é revisado por especialistas, sejam da indústria ou da área acadêmica. É um dos primeiros esforços em mapear o reuso de software em diversos aspectos, com desenvolvimento baseado em componentes, reengenharia, ferramentas, qualidade de componentes de software, métricas para reuso, etc.

O RAD (*Rapid Application Development*), mencionado anteriormente, é um modelo de processo de desenvolvimento de software iterativo e incremental que enfatiza um ciclo de desenvolvimento extremamente curto (entre 60 e 90 dias), tendo um foco considerável no reuso para se atingir um prazo consideravalemente justo, tornando-o uma técnica de quarta geração que reutiliza componentes de programas existentes ou cria componentes reutilizáveis.

Aula 6 – Modelos prescritivos de processo – Parte 3

6.1 Modelo de métodos formais

Os métodos formais são técnicas baseadas em formalismos matemáticos que podem ser usados para a especificação, desenvolvimento e verificação de sistemas de software. Seu uso para o desenvolvimento de software é motivado pela base que fornece, garantindo confiabilidade e robustez a um projeto — uma vez que executam análises matemáticas apropriadas.

Exemplo de sistema crítico.
Fonte: http://www.apolo11.com/spacenews.php?posic=dat_20061206-111600.inc

No entanto, o alto custo do uso de métodos formais restringe seu uso ao desenvolvimento de sistemas de alta integridade, no qual há alta probabilidade das falhas conduzirem à perda de vidas ou sérios prejuízos (como nas missões espaciais e no controle de voos, por exemplo).

No modelo de métodos formais, também conhecido como engenharia de software cleanroom (sala limpa), uma questão fundamental é garantir que o software é realmente uma solução para o problema proposto. Conforme consta em PUCRS (2012), para realizar essa tarefa, é preciso primeiramente construir um modelo da solução (especificação) utilizando uma linguagem formal. Tendo esse modelo formal como base, o método permite:

- Realizar provas matemáticas que garantam que tal modelo possua as propriedades requisitadas (verificação);
- Analisar se a solução proposta é aceitável do ponto de vista de desempenho, além de indicar quais as melhores estratégias de implementação que deve-se seguir;
- Validar o modelo por meio de simulações;
- Realizar o desenvolvimento do software de forma que seja possível provar que a implementação esteja correta (geração de código correto).

Dentre os vários métodos de especificação formal existentes, destacam-se o método Z, que já foi utilizado em várias aplicações práticas, e o método de gramáticas de grafos, que possui uma linguagem visual de representação e descreve com naturalidade fenômenos de sistemas concorrentes (PUCRS, 2012).

De uma maneira mais abrangente, o modelo de métodos formais pode ser caracterizado da seguinte forma:

- Permite especificar, desenvolver e verificar um software por meio de uma rigorosa notação matemática.
- Fornece um mecanismo para eliminar muitos problemas encontrados nos outros modelos, como por exemplo, ambiguidade, incompletude e inconsistência, que podem ser descobertos e corrigidos mais facilmente por análise matemática.
- Promete o desenvolvimento de software livre de defeitos.
- Consome muito tempo de desenvolvimento e é muito caro.

Como poucos desenvolvedores possuem o conhecimento necessário para utilizá-lo, são requeridos muitos cursos e treinamentos. É difícil usar tais modelos matemáticos formais como meio de comunicação com a maioria dos clientes. Assim, como já dissemos, sua área de aplicação é muito restrita, abrangendo, principalmente, aplicações críticas.

Para mais informações sobre os métodos formais, consulte o link:
http://www.tiosam.org/enciclopedia/index.asp?q=Métodos_formais.

Para entender como funciona o controle de tráfego aéreo (projeto crítico), assista à reportagem do Jornal da Globo, disponível em:
http://youtu.be/7wsrs3a-y3M.

6.2 Processo unificado

O processo unificado (PU) (ou *unified process* — UP) surgiu como um processo para o desenvolvimento de software visando à construção de sistemas orientados a objetos (o RUP — *Rational Unified Process* — é um refinamento do processo unificado). Trata-se de um processo iterativo e adaptativo de desenvolvimento que vem ganhando cada vez mais adeptos, devido à maneira organizada e consistente que oferece na condução de um projeto.

O PU utiliza uma abordagem evolucionária paro o desenvolvimento de software. O ciclo de vida iterativo é baseado em refinamentos e incrementos sucessivos, a fim de convergir para um sistema adequado. Em cada iteração, procura-se incrementar um pouco mais o produto, baseando-se na experiência obtida nas iterações anteriores e no *feedback* do usuário. Cada iteração pode ser considerada um miniprojeto de duração fixa, sendo que cada um desses inclui suas próprias atividades de análise de requisitos, projeto, implementação e testes.

O PU foi modelado usando o *Software Process Engineering Model* (SPEM), que é um padrão para modelagem de processos baseado em *Unified Modeling Language* (UML). O processo tem duas estruturas, ou duas dimensões, a saber:

- Estrutura dinâmica: representa a dimensão do tempo no processo.
- Estrutura estática: descreve como elementos do processo são agrupados em disciplinas.

A estrutura que representa o tempo é denominada fase e a estrutura que representa os elementos do processo são as disciplinas, que são descritas a seguir, baseado em IBM (2006). A Figura 11 ilustra tais estruturas.

Fases

Uma fase é definida como a dimensão de tempo entre duas maiores marcas (major milestones) de processos, durante a qual um conjunto bem definido de objetivos é encontrado, artefatos são completados e a decisão é tomada no sentido de ir para a próxima fase ou não.

As maiores marcas, por sua vez, podem ser definidas como eventos de grandes sistemas organizados no final de cada fase de desenvolvimento para fornecer visibilidade aos seus problemas, sincronizar o gerenciamento com as perspectivas de engenharia e verificar se os objetivos de cada fase foram obtidos. Esse conceito está intimamente relacionado ao risco que deve ser considerado na medida em que o sistema estiver evoluindo. Para cada fase os maiores milestones buscam identificar e antecipar os riscos.

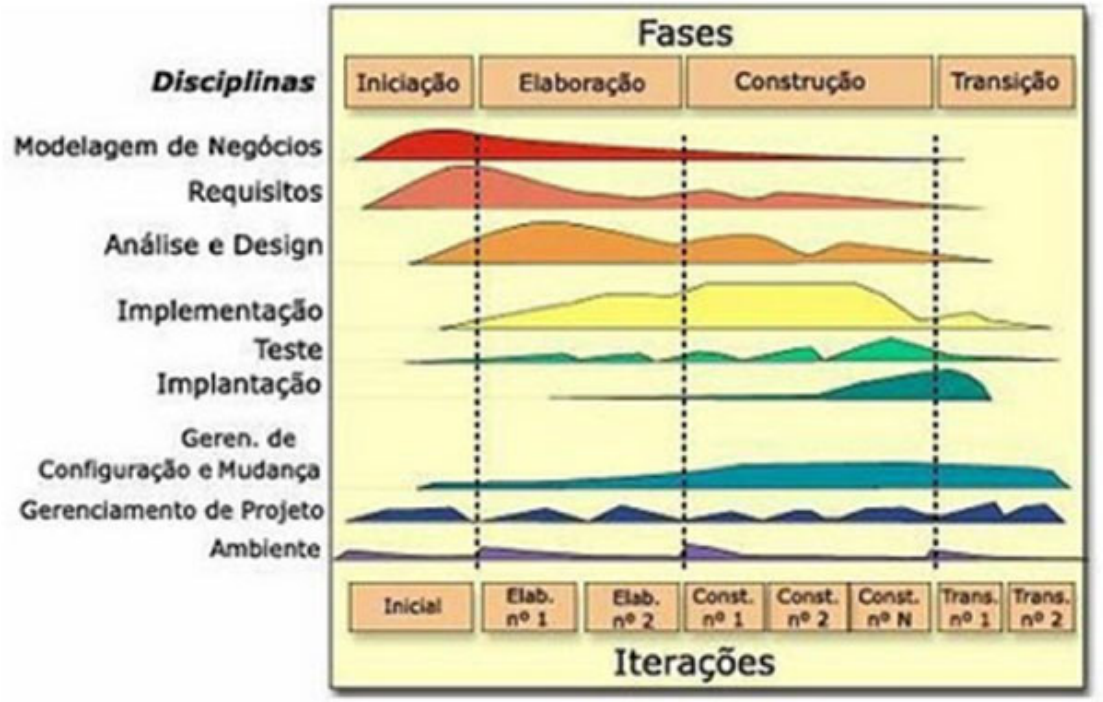


Figura 11. Fases e disciplinas do processo unificado.
Fonte: http://isosoftware.blogspot.com.br/2010/02/processo-unificado-pu-unified-process.html.

O processo unificado organiza suas iterações em quatro fases principais:

1) **Concepção ou iniciação:** o objetivo desta fase é levantar, de forma genérica e pouco precisa, o escopo do projeto. Não deve existir aqui a pretensão de especificar de forma detalhada os requisitos; a ideia é ter uma visão inicial do problema, estimar de forma vaga o esforço e os prazos e determinar se o projeto é viável e merece uma análise mais profunda.

2) **Elaboração:** na fase de elaboração todos (ou a grande maioria dos requisitos) são levantados em detalhes. Numa primeira iteração, um ou dois requisitos (os de maior risco e valor arquitetural) são especificados em detalhes. Estes são implementados e servem como base de avaliação junto ao usuário e desenvolvedores para o planejamento da próxima iteração. Em cada nova iteração na fase de elaboração pode haver um seminário de requisitos, em que requisitos antigos são melhor esclarecidos e os novos são detalhados. Ao fim dessa fase, deseja-se que 90% dos requisitos tenham sido levantados em detalhes, que o núcleo do sistema tenha sido implementado com alta qualidade, que os principais riscos tenham sido tratados e que haja condições para se fazer estimativas mais realistas.

3) **Construção:** visa à implementação iterativa dos elementos restantes de menor risco e mais fáceis, além da preparação para a implantação.

4) **Transição:** testes finais e implantação.

Disciplinas

Disciplina é uma coleção de atividades relacionadas que estão ligadas à maior área de interesse dentro do processo em geral. Cada disciplina possui, resumidamente, os seguintes objetivos específicos:

- **Modelagem de negócios:** entender a estrutura e a dinâmica da organização para a qual o produto será desenvolvido; identificar problemas correntes na organização e possíveis aperfeiçoamentos; assegurar que o cliente, o usuário final e os desenvolvedores possuam a mesma compreensão da empresa; produzir os requisitos de sistemas necessários para suportar os objetivos da organização.
- **Requisitos:** estabelecer e manter o consentimento entre clientes e stakeholders sobre o que o sistema deve fazer; fornecer uma melhor compreensão dos requisitos aos desenvolvedores; definir os limites do sistema; fornecer as bases para o planejamento

das iterações, estimativas de custo e tempo de desenvolvimento; definir as interfaces do sistema baseado nas necessidades e objetivos dos usuários.

• **Análise e design:** transformar os requisitos dentro de um projeto do que será o sistema; desenvolver uma arquitetura robusta para o sistema; adaptar o projeto para ajustá-lo ao ambiente de implementação.

• **Implementação:** preparar a organização do código em termos de implementação de subsistemas, organizados em camadas; implementar classes e objetos em termos de seus componentes; testar os componentes desenvolvidos como unidades; integrar os resultados obtidos por implementadores individuais (ou equipes) em um sistema executável.

• **Teste:** verificar a interação entre os objetos; verificar a integração de todos os componentes de software; verificar se todos os requisitos foram implementados corretamente; verificar os defeitos e assegurar que eles foram tratados antes da entrega do produto.

• **Implantação:** descrever as atividades associadas à verificação para que o produto esteja disponível ao usuário final.

• **Gerenciamento de configuração e mudança:** identificar itens de configuração; restringir alterações para aqueles itens; auditar as alterações neles feitas; definir e gerenciar as alterações daqueles itens.

• **Gerenciamento de projeto:** fornecer uma estrutura para o gerenciamento de projeto de software; fornecer um guia prático para planejamento, recrutamento, execução e monitoramento de projeto; fornecer uma estrutura para o gerenciamento de riscos.

• **Ambiente:** identificar as atividades necessárias para configurar o processo para o projeto; descrever as atividades requeridas para desenvolver as guias mestras no suporte ao projeto; fornecer para a organização de desenvolvimento de software o ambiente de processos e ferramentas que suportarão a equipe de desenvolvimento.

De uma maneira geral, podemos caracterizar o processo unificado da seguinte forma:

• É um framework genérico de um processo de desenvolvimento;
• É baseado em componentes;
• Utiliza toda a definição da UML;
• É dirigido pelos use cases, centrado na arquitetura, iterativo e incremental (conceitos-chave).

O processo unificado foi criado para ser um processo ágil de desenvolvimento e propõe uma abordagem realística para a condução de um projeto. Ao contrário do modelo em cascata, em que cada etapa do ciclo de vida deve ser realizada integral e sequencialmente, no PU as atividades são repetidas quantas vezes forem preciso, em ciclos organizados. Não há um plano detalhado para todo um projeto. Há um plano de alto nível (chamado plano de fases) que estima a data de término do projeto e outros marcos de referência principais, mas ele não detalha os passos de granularidade fina para se atingir tais marcos. Um plano detalhado (chamado plano de iterações) somente planeja a iteração a ser feita em seguida. O planejamento detalhado é feito de forma adaptativa, de iteração para iteração.

Aula 7 – Processo ágil de desenvolvimento

As definições modernas de desenvolvimento de software ágil evoluíram a partir de meados de 1990, como parte de uma reação contra métodos caracterizados por uma regulamentação complexa e rígida e contra a regimentação e sequenciamento usados no modelo em cascata. O objetivo era desburocratizar os procedimentos que tornavam as etapas dos processos lentas, o que era contrário ao modo de trabalho usual dos engenheiros de software.

Inicialmente, os métodos ágeis foram denominados de "métodos leves". Em 2001, Kent Beck e dezesseis outros profissionais notáveis da área de engenharia de software se reuniram, adotaram o nome de *métodos ágeis*, e publicaram o *Manifesto Ágil*, documento que reúne os princípios e práticas dessa metodologia de desenvolvimento. Esses veteranos formaram a *Agile Alliance* (ou Aliança Ágil), uma organização não lucrativa que promove e fomenta o desenvolvimento ágil (BECK, 2001).

Os membros da Aliança Ágil declararam os princípios da metodologia em seu manifesto da seguinte forma:

"Estamos descobrindo maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo. Por meio desse trabalho, passamos a valorizar:

— Indivíduos e interação entre eles, mais que processos e ferramentas;
— Software em funcionamento, mais que documentação abrangente;
— Colaboração com o cliente, mais que negociação de contratos;
— Responder a mudanças, mais que seguir um plano.

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda" (BECK, 2001).

Os principais modelos baseados na concepção de desenvolvimento ágil incluem o Scrum, criado em 1986; o ASD (*Adaptive Software Development*); o FDD (*Feature Driven Development*); o DSDM (*Dynamic Systems Development Method*), de 1995; o Crystal e a XP (*eXtreme Programming* ou Programação Extrema), criados em 1996.

O Desenvolvimento Ágil de Software (*Agile Software Development*) envolve uma metodologia que visa minimizar os riscos, por meio de desenvolvimentos em curtos períodos ou iterações, como mostra a Figura 12. A iteração típica envolve o desenvolvimento em fases curtas, de uma a quatro semanas, envolvendo todas as tarefas necessárias para implantar uma funcionalidade.

Considerando-se o período curto de cada iteração, a comunicação mantida entre os stakeholders é em tempo real sendo, preferencialmente, tratada por meio verbal (embora documentada), ou face a face, visando minimizar entendimentos parciais ou errôneos. Para tanto, é necessário estabelecer-se um local físico (uma sala) em que toda a equipe necessária ficará focada no projeto.

Figura 12. Etapas do desenvolvimento ágil.
Fonte: http://www.sstecnologia.com.br/.

Alguns críticos do processo ágil afirmam que o maior risco desse tipo de abordagem é a baixa qualidade ou mesmo inexistência de documentação do projeto devido à interação humana face a face que, se por um lado traz agilidade na comunicação, por outro traz a informalidade nas definições. Assim sendo, é necessário um bom acompanhamento do gestor do projeto para garantir a qualidade dos trabalhos, independente do processo utilizado.

De forma geral, os processos ágeis atendem aos projetos de software que, normalmente, apresentam:

- Dificuldade em prever com antecedência quais requisitos vão persistir e quais serão modificados, bem como quais prioridades dos clientes sofrerão mudanças ao longo do projeto;
- Intercalação das etapas de projeto e construção, que vão sendo realizadas juntas, de modo que os modelos de projeto vão sendo comprovados na medida em que são criados;
- Falta de previsibilidade da análise, projeto, construção e testes do ponto de vista do planejamento, como seria desejado.

A metodologia ágil vem gerando grande debate entre desenvolvedores apaixonados e críticos ferrenhos. Contudo, deve-se manter a imparcialidade profissional e questionar o que realmente tem que ser observado, como:

- Qual é a melhor forma de se alcançar a agilidade nos processos e práticas do desenvolvimento de software?
- Como construir softwares que satisfaçam às necessidades do cliente e possuam características de

qualidade que permitam sua extensão e ampliação para satisfazer às necessidades do cliente em longo prazo?

Para saber mais sobre o assunto, leia a seguir o artigo publicado na revista INFO Exame Online, em 30/06/2009:

Desenvolvimento Ágil Funciona?

(...) Scrum e *Extreme Programming* são algumas metodologias ágeis que estão em voga no momento. Elas refletem o espírito deste manifesto ágil?

Com a nossa cultura de *fast food*, de querer achar uma receita mágica para tudo, o mundo ágil ganhou força dentro de um certo nicho de desenvolvimento. A gente ouve falar muito hoje de Scrum, de *Extreme Programming*, mas se usam essas ferramentas apenas como ferramentas. Isso não funciona. Tem gente que até discute se o pico recente de Scrum não foi mais negativo que positivo, porque muita gente está aplicando e não atinge os resultados, simplesmente porque usa o raciocínio antigo com ferramentas novas. Os valores não foram entendidos. Scrum é um *template*. Ele te diz que você tem este papel chamado *product owner*, você tem este tempo chamado *sprint*, tem esta atividade chamada scrum diário, e estas coisas chamadas retrospectiva e *backlog*. Ele te dá nomes e elementos que você aplica. Mas não é só aplicar e ponto. Isso é só o começo. A grande mensagem da forma ágil de pensar é que o procedimento não é importante. É irrelevante se eu chamo o meu tempo de trabalho de *sprint* ou o dono do projeto de *product owner*. É absolutamente irrelevante, se eu não entendi os valores que são entregar valor ao cliente, criar uma organização de aprendizado, acreditar nas pessoas e dar suporte a elas. Esses princípios é que são importantes, mas não são explícitos. Em vez disso eu tenho papéis, cargos e atividades. Não se explicam os porquês e não entendendo os porquês você vai fazer exatamente a mesma coisa, mas em vez de chamar o sujeito de gerente de projeto você vai chamá-lo de *product owner*. Mas nada mudou.

Disponível em: <http://info.abril.com.br/noticias/ti/desenvolvimento-agil-funciona-30062009-3.shl>.

Para saber mais sobre Desenvolvimento Ágil, leia o artigo Exemplo de Desenvolvimento Ágil SCRUM, disponível em: http://www.devin.com.br/modelo-scrum/.

Exercícios do Módulo 2

1) Processos de software podem ser construídos a partir de modelos pré-existentes, objetivando adequar-se às necessidades de um determinado projeto.

a) Verdadeiro.
b) Falso.

2) Os tipos de manutenção que um software pode sofrer são:

• Manutenção _____: faz modificações nos programas de modo que eles possam ser mais facilmente corrigidos, adaptados e melhorados.
• Manutenção _____: modifica o software para corrigir defeitos.
• Manutenção _____: aprimora o software além dos requisitos funcionais originais (cliente/usuário reconhece e solicita funcionalidades adicionais que trarão benefícios à medida que o software é usado).
• Manutenção _____: modifica o software para acomodar mudanças no seu ambiente externo (processador, sistema operacional, etc.).

3) Em relação ao modelo em cascata, podemos afirmar que ele é:

a) Uma abordagem razoável quando os requisitos estão bem estabelecidos.
b) Uma boa abordagem quando se tem pressa em entregar o software.
c) A melhor abordagem para se usar quando se tem uma grande equipe de desenvolvimento.
d) Um modelo ultrapassado que raramente é utilizado atualmente.

4) Quais são as cinco atividades gerais da engenharia de software, também caracterizadas no modelo em cascata?

a) Especificação, gerenciamento de risco, medida, produção, revisão.
b) Análise, projeto, implementação, testes, manutenção.
c) Especificação, planejamento, modelagem, construção, implantação (entrega).
d) Análise, planejamento, projeto, programação, análise de risco.

5) Podemos dizer que o modelo incremental é:

a) Uma abordagem razoável quando os requisitos estão bem estabelecidos.
b) Uma boa abordagem quando um produto de trabalho "núcleo" é requerido rapidamente.
c) A melhor abordagem para se usar quando se tem uma grande equipe de desenvolvimento.
d) Um modelo revolucionário que normalmente não é utilizado para desenvolver produtos comerciais.

6) Podemos afirmar que os modelos de processo evolucionários:

a) São iterativos por natureza.
b) Podem facilmente acomodar mudanças de requisitos.
c) Quase sempre não produzem produtos descartáveis.
d) Todas as alternativas anteriores estão corretas.

7) O modelo de prototipagem é:

a) Uma abordagem razoável quando os requisitos estão bem definidos.
b) A melhor abordagem para usar em projetos com grandes equipes de desenvolvimento.
c) Uma abordagem razoável quando o cliente não consegue definir os requisitos claramente.
d) Um modelo de alto risco que raramente produz software de qualidade.

8) O modelo espiral de desenvolvimento de software:

a) Finaliza com a entrega do software.
b) É mais caótico do que o modelo incremental.
c) Inclui avaliação de riscos a cada iteração.
d) Todas as afirmativas anteriores estão corretas.

9) O modelo de desenvolvimento concorrente:

a) É outro nome para a engenharia concorrente.
b) Define eventos que podem desencadear transições de estados dentro de cada fase de desenvolvimento.
c) Somente é usado para desenvolver sistemas distribuídos.
d) É usado sempre que um grande número de solicitações de alteração de requisitos é previsto.
e) Somente a e b são verdadeiras.

10) Podemos dizer que o modelo de desenvolvimento baseado em componentes:

a) Somente é apropriado para projetos ligados ao hardware.
b) Não é capaz de suportar o desenvolvimento de componentes reusáveis.
c) Depende da tecnologia de objetos como suporte.
d) Não se mostra "rentável" quando quantificado por métricas de software.

11) O modelo de métodos formais faz uso de métodos matemáticos para:

a) Definir a especificação de requisitos.
b) Desenvolver sistemas livres de defeitos.
c) Verificar a corretude de sistemas baseados em computador.
d) Todas as alternativas anteriores são corretas.

12) Qual dos nomes abaixo não representa uma das fases do modelo de processo unificado?

a) Fase de iniciação.
b) Fase de validação.
c) Fase de elaboração.
d) Fase de construção.

13) Qual a diferença entre um modelo descritivo e um modelo prescritivo de processo?

Enquanto um modelo _____ retrata como um processo é executado, um modelo _____ retrata como um processo deveria ser executado. Assim, um modelo _____ é uma recomendação que pode ser adaptada ou melhorada pela empresa/equipe de software que for adotá-la.

14) O que não caracteriza os modelos evolucionários de processo?

a) São implementados de forma a permitir o desenvolvimento de versões cada vez mais completas do software.
b) São usados quando o deadline (limite de tempo) não é adequado para o desenvolvimento do software, ou seja, a data de término não é realística (por exemplo, prazos reduzidos de mercado face à competitividade).
c) São liberados produtos core ("núcleo dos produtos") ao longo do desenvolvimento.
d) Os detalhes e extensões do projeto são definidos logo no início, antes do desenvolvimento.

15) Assinale a alternativa FALSA:

a) Os modelos incrementais são mais indicados em situações em que os requisitos iniciais do software estão razoavelmente bem definidos, mas o escopo global do processo de desenvolvimento claramente elimina uma abordagem puramente linear ou sequencial.
b) O modelo de processo incremental é interativo como a prototipagem e ambos têm como objetivo apresentar um produto operacional a cada incremento realizado.
c) No PU a estrutura dinâmica representa a dimensão do tempo no processo.
d) No PU a estrutura estática descreve como elementos do processo são agrupados em disciplinas.

16) Assinale a alternativa que não é característica do processo unificado:

a) é um framework de propriedade da IBM que é usado como um processo de desenvolvimento.
b) é baseado em componentes.
c) utiliza toda a definição da UML.
d) é dirigido pelos use cases, centrado na arquitetura, iterativo e incremental.

17) Os princípios:

• Valorizar indivíduos (e a interação entre eles) mais do que processos e ferramentas;
• Valorizar o software em funcionamento mais do que a documentação abrangente;
• Valorizar a colaboração com o cliente mais do que a negociação de contratos;
• Valorizar responder a mudanças mais que seguir um plano.

Referem-se a _____.

18) Nos modelos de processos ágeis o único produto de trabalho gerado é o programa de trabalho.

a) Verdadeiro.
b) Falso.

MÓDULO 3 – ENGENHARIA DE REQUISITOS (ER)

Neste módulo definiremos o que são requisitos de software e a sua importância dentro da engenharia de requisitos. As etapas e principais conceitos e definições ligadas à engenharia de requisitos são apresentadas a seguir.

Aula 8 – Requisitos e engenharia de requisitos

A engenharia de requisitos (ER) pode ser vista como uma subárea da engenharia de software, cujo principal objetivo é a obtenção de uma especificação correta e completa dos requisitos de um sistema de software.

A engenharia de requisitos tem se tornado cada vez mais necessária para resolver os problemas encontrados nas organizações com relação à definição de sistemas. De acordo com Pressman (2010), na perspectiva do processo de software, a engenharia de requisitos é uma ação de engenharia de software que começa durante a atividade de especificação e continua durante a atividade de modelagem.

Para que o processo de desenvolvimento de software seja bem sucedido é fundamental que haja uma compreensão completa dos requisitos de software. Tanto o desenvolvedor como o cliente desempenham um papel ativo na análise e especificação dos requisitos. O desenvolvedor age como indagador, consultor e solucionador de problemas e o cliente tenta traduzir os conceitos relativos ao desempenho do software que ele tem concebido apenas em sua mente, tarefa que é, às vezes, bastante complicada e nebulosa, em detalhes concretos.

Mas, afinal, o que são os requisitos?

> De acordo com Sommerville (2007), os requisitos para um sistema de software estabelecem o que o sistema deve fazer e definem restrições sobre sua operação e implementação.

Os requisitos têm por objetivo:

- Estabelecer e manter concordância com os clientes e outros envolvidos sobre o que o sistema deve fazer;
- Oferecer aos desenvolvedores uma compreensão melhor do sistema a ser desenvolvido;
- Delimitar o sistema;
- Planejar o desenvolvimento do sistema;
- Fornecer uma base para estimar o custo e o tempo de desenvolvimento do sistema.

De acordo com Sommerville (2007), os requisitos são classificados como funcionais, não funcionais, de domínio, do usuário e do sistema. Mas algumas dessas classes ainda apresentam uma subclassificação. Uma breve descrição de tais categorias é apresentada na sequência.

8.1 Requisitos funcionais

São requisitos que descrevem a funcionalidade (funções que o sistema deve realizar) ou os serviços que se espera que o sistema faça. Alguns exemplos de requisitos funcionais são o cadastro de cliente, a emissão de nota fiscal, a consulta ao estoque e a geração de pedido.

8.2 Requisitos não funcionais

São requisitos que não dizem respeito diretamente à funcionalidade do sistema, mas expressam propriedades do sistema e/ou restrições sobre os serviços ou funções por ele providas. Sua classificação é mostrada na Figura 13. Essa classe de requisitos pode ser classificada em:

8.2.1 *Requisitos de produto*

São requisitos que especificam o comportamento do produto, como por exemplo:

- Requisitos de facilidade de uso ou de usabilidade: referem-se ao esforço para utilizar ou aprender a utilizar o produto. Estão relacionados com a interação do usuário junto ao sistema.
- Requisitos de confiabilidade: referem-se à frequência de ocorrência de falhas e recuperabilidade em caso de falha, como: tempo médio de falhas; probabilidade de indisponibilidade; taxa de ocorrência de falhas; tempo de reinício após falha; percentual de eventos causando falhas e probabilidade de corrupção de dados após falha.
- Requisitos de eficiência: referem-se ao desempenho do sistema e estão associados à eficiência, uso de recursos e tempo de resposta da aplicação, como: transações

processadas/segundo; tempo de resposta do usuário/evento, entre outros.

• Requisitos de portabilidade: são relativos à capacidade de transferir o produto para outros ambientes.

8.2.2 *Requisitos organizacionais*

São requisitos derivados das políticas organizacionais do cliente e do desenvolvedor, como por exemplo:

• Requisitos de implementação: ligados às regras de codificação e restrições de software e hardware usados para desenvolver ou executar o sistema.

• Requisitos de padrões: referem-se à definição da linguagem de programação e às normas que devem ser seguidas pelo sistema ou no processo de desenvolvimento.

• Requisitos de entrega: referem-se ao modo como será implantada a solução como configurações necessárias e ordem de instalação dos pacotes.

8.2.3 *Requisitos externos*

São requisitos procedentes de fatores externos ao sistema e ao seu processo de desenvolvimento, como por exemplo:

• Requisitos éticos.

• Requisitos legais (como política de privacidade e direitos autorais).

• Requisitos de interoperabilidade.

Exemplos: o sistema não deverá revelar aos operadores nenhuma informação pessoal sobre os clientes, além de seus nomes e o número de referência (legislação de privacidade); em razão das restrições referentes aos direitos autorais, alguns documentos devem ser excluídos imediatamente ao serem fornecidos.

8.2.4 *Requisitos de domínio*

Trata-se de requisitos derivados do domínio da aplicação do sistema. Em vez de serem obtidos a partir das necessidades específicas dos usuários do sistema, eles podem se transformar em novos requisitos funcionais, ou serem regras de negócios específicos do domínio do problema. Esses requisitos podem consistir, por exemplo, na utilização de uma interface padrão usando a norma Z39.50 ou na disponibilização de arquivos somente para leitura, devido aos direitos autorais

8.2.5 *Requisitos do usuário*

São os requisitos funcionais e não funcionais do sistema sob o ponto de vista do usuário. Em geral apresentam problemas como falta de clareza, confusão e fusão, ou seja, têm-se requisitos diferentes escritos como um único requisito.

8.2.6 *Requisitos do sistema*

São descrições mais detalhadas dos requisitos do usuário. São a base para que os engenheiros de software possam fazer o projeto do sistema. Servem como base para um contrato destinado à implementação do sistema.

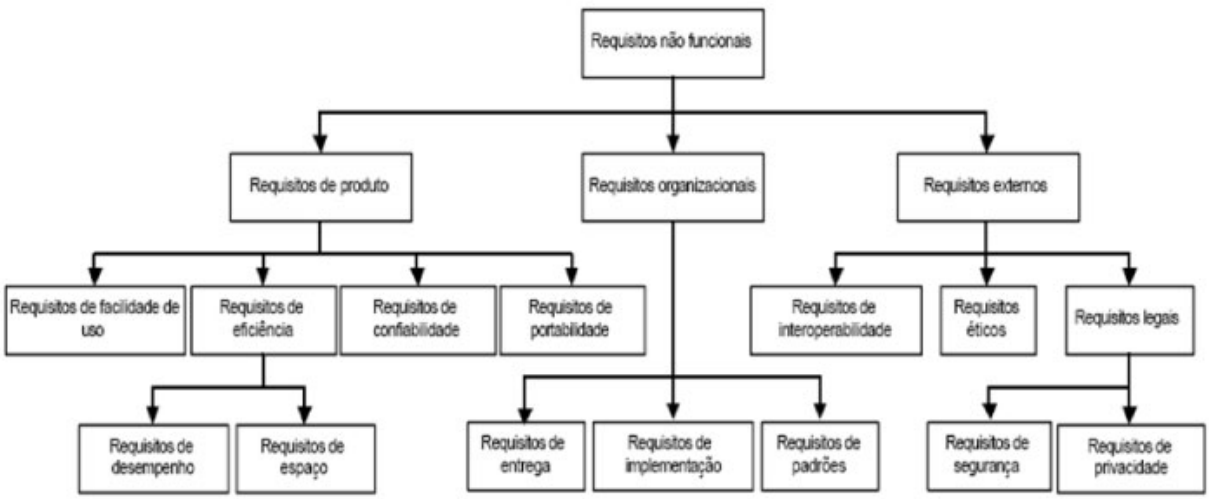


Figura 13. Classificação dos requisitos não funcionais.
Fonte: http://evertongomede.blogspot.com/2010/09/taxonomiados-requisitos-nao-funcionais.html.

Stakeholders (influenciadores/envolvidos): são as partes interessadas no projeto, ou seja, pessoas e organizações envolvidas no projeto ou que possuem interesses que podem ser afetados pelo projeto, podendo exercer influência sobre os resultados deste.

Baseline: conjunto de artefatos de software que servem de base para seu desenvolvimento.

DERS (Documento de Especificação de Requisitos de Software): De acordo com o IEEE esse documento deve ser completo e não ambíguo, sendo responsável por auxiliar os clientes de software, para que descrevam com precisão o que desejam obter, e desenvolvedores de software, para que entendam exatamente o que o cliente deseja. Para os clientes, desenvolvedores e outros indivíduos ligados ao projeto, um bom DERS deve prover diversos benefícios, tais como:

- Estabelecer a base de acordo entre os clientes e a empresa fornecedora sobre o que o software irá fazer;
- Reduzir o esforço de desenvolvimento;
- Prover uma base para estimativas de custo e prazos;
- Prover uma base para validação e verificação do produto;
- Facilitar a manutenção do produto de software final.

Um DERS conta com alguns padrões indicados para sua elaboração. O IEEE elaborou um documento que contém padrões e práticas recomendadas para a criação do DERS, chamado *IEEE Recommended Practice for Software Requirements Specification* (em português, Práticas Recomendadas para a Especificação de Requisitos de Software). Esse documento está disponível em: http://www6.conestogac.on.ca/~set/courses/year1/sef/documents/IEEE830-1998Standard.pdf.

Aula 9 – Etapas da engenharia de requisitos

A engenharia de requisitos reúne as primeiras atividades a serem realizadas nos diversos modelos de processo de software. Seu objetivo é definir o que deve ser feito, e não a forma como será implementado o sistema. Esse campo do saber serve como ponte entre o projeto e a construção do software, definindo as bases do contrato entre o desenvolvedor e o cliente.

Além disso, a engenharia de requisitos realiza a aquisição, o refinamento e a verificação das necessidades do sistema. A meta é sistematizar o processo de definição dos requisitos, obtendo uma especificação correta e completa do sistema para elaboração do DERS.

A engenharia de requisitos fornece um mecanismo adequado para entender o que o cliente deseja, analisar suas necessidades, negociar uma solução adequada ao seu problema, validar a especificação e administrar os requisitos, na medida em que eles são transformados em um sistema em operação. Para isso, o processo de ER é realizado por meio de sete etapas distintas, que, com base em Pressman (2006) e Sommerville (2007), serão explicadas na sequência desta seção.

9.1 Concepção ou estudo de viabilidade do sistema

De acordo com Sommerville (2007), o estudo de viabilidade é um estudo breve, direcionado, que se destina a responder a algumas perguntas:

- O sistema contribui para os objetivos gerais da empresa?
- O sistema pode ser implementado com a utilização de tecnologia atual dentro das restrições de custo e prazo?
- O sistema pode ser integrado com outros sistemas já em operação?

Após responder a essas questões é necessário questionar as fontes de informação (stakeholders). Para isso, são recomendadas algumas perguntas, tais como:

- Como a empresa se comportaria se esse sistema não fosse implementado?
- Quais são os problemas com os processos atuais e como um novo sistema ajudaria a diminuir esses problemas?
- Que contribuição direta o sistema trará para os objetivos da empresa?
- As informações podem ser transferidas para outros sistemas e também ser recebidas a partir deles?
- O sistema requer tecnologia que não tenha sido utilizada anteriormente na empresa?
- O que precisa e o que não precisa ser compatível com a empresa?
- Quem vai usar o sistema?

Após obter essas respostas, deve-se preparar um relatório de viabilidade.

9.2 Levantamento dos requisitos

Após os estudos iniciais de viabilidade, o próximo passo é o levantamento dos requisitos. Nesta etapa deve-se descobrir mais informações sobre o domínio da aplicação, que serviços o sistema deve oferecer, qual o desempenho exigido, dentre outros fatores. O levantamento de requisitos pode envolver diferentes tipos de pessoas na empresa, ou seja, os *stakeholders*.

Frequentemente os clientes não sabem na realidade o que querem do sistema computacional, a não ser em termos muito gerais. Eles costumam expressar os requisitos utilizando seus próprios termos e com o conhecimento implícito de sua área de atuação.

Diferentes *stakeholders* têm em mente diferentes requisitos e podem expressá-los de maneiras distintas. Além disso, fatores políticos podem influenciar os requisitos do sistema. O ambiente econômico e de negócios, no qual a análise de requisitos ocorre, é dinâmico; inevitavelmente, o ambiente se modifica durante o projeto. Como consequência, a importância dos requisitos específicos pode mudar. Assim, novos requisitos podem surgir por parte dos novos stakeholders, que não haviam sido consultados inicialmente.

9.3 Análise de requisitos

As informações obtidas do cliente durante a concepção e o levantamento são expandidas e refinadas durante a análise (ou elaboração) de requisitos. Esta atividade da ER tem por objetivo desenvolver um modelo técnico refinado das funções, características e restrições do software, sendo guiada pela criação e refinamento de cenários do usuário, que descrevem como o usuário final (e outros atores) vão interagir com o sistema.

Segundo Sommerville (2007), as etapas de levantamento e análise de requisitos são conhecidas em conjunto como elicitação de requisitos, e envolvem:

- Compreensão do domínio: os analistas devem desenvolver sua compreensão do domínio da aplicação. Por exemplo, se for um sistema para uma loja, o analista deverá descobrir como operam as lojas de varejo.
- Coleta de requisitos: é o processo de interagir com os stakeholders do sistema para descobrir seus requisitos. Certamente a compreensão do domínio tem um maior desenvolvimento durante essa atividade.
- Classificação: essa atividade considera o conjunto não estruturado dos requisitos e os organiza em grupos coerentes.
- Resolução de conflitos: quando múltiplos stakeholders estão envolvidos, os requisitos podem apresentar conflitos. Essa atividade tem o papel de encontrar e solucionar esses possíveis conflitos.
- Definição das prioridades: em qualquer conjunto de requisitos, alguns serão mais importantes do que outros. Esse estágio envolve a interação com os stakeholders para descobrir os requisitos mais importantes.
- Verificação de requisitos: os requisitos são verificados, a fim de se descobrir se eles são completos e consistentes e se estão em concordância com o que os stakeholders realmente desejam do sistema.

9.4 Negociação dos requisitos

A negociação categoriza e organiza os requisitos em subconjuntos relacionados. Neste passo as seguintes perguntas devem ser respondidas:

- Cada requisito está de acordo com o objetivo global do sistema?
- Todos os requisitos foram especificados no nível de abstração adequado?
- O requisito é realmente necessário ou representa uma característica adicional que pode não ser essencial para o objetivo do sistema?
- Cada requisito é limitado e não ambíguo?
- Cada requisito tem atribuição? Será utilizado por alguém?
- Algum requisito apresenta conflito com outro(s)?
- Cada requisito é realizável no ambiente técnico?
- Cada requisito pode ser testado quando estiver implementado?
- Qual é a prioridade de tal requisito?

9.5 Especificação dos requisitos

A especificação de requisitos geralmente é um documento escrito, que contém o detalhamento do sistema, os requisitos funcionais e não funcionais e os modelos do sistema que podem ser representados

por meio de diagramas, como o IDEF0 (Integration Definition for Function Modeling), os diagramas de casos de uso da UML (Unified Modeling Language), os diagramas de classes preliminar e os diagramas de sequência e prototipação.

9.6 Validação dos requisitos

A validação de requisitos examina a especificação de requisitos para garantir que todos os requisitos do sistema tenham sido identificados e detalhados, que as inconsistências, omissões e erros tenham sido detectados e corrigidos e que as prioridades tenham sido estabelecidas. Tem a função de mostrar que os requisitos realmente definem o sistema que o cliente deseja. Para Sommerville (2007), são realizadas diversas verificações, tais como:

- Verificações de validade: os usuários podem acreditar que um sistema é necessário para realizar certas funções, mas estudos e análises podem identificar outras funções adicionais ou diferentes necessárias.
- Verificações de consistência: os requisitos em um documento não devem ser conflitantes, ou seja, não devem existir restrições contraditórias ou descrições diferentes para uma mesma função do sistema.
- Verificações de completeza: o documento de requisitos (DERS) deve incluir requisitos que definam todas as funções e restrições exigidas pelo usuário do sistema.
- Verificações de realismo: utilizando o conhecimento da tecnologia existente, os requisitos devem ser verificados, a fim de assegurar que eles realmente possam ser implementados. Tais verificações devem também levar em conta o orçamento e os prazos para o desenvolvimento do sistema.
- Facilidade de verificação: para reduzir o potencial de divergências entre cliente e desenvolvedor, os requisitos do sistema devem sempre ser escritos de modo que possam ser verificados. Isso significa que um conjunto de verificações pode ser projetado para mostrar que o sistema entregue cumpre com esses requisitos, de forma que algumas questões devam ser feitas, tais como:
  - » O requisito é realmente passível de ser testado, como foi definido?
  - » O requisito pode ser adequadamente compreendido pelos compradores ou usuários finais do sistema?
  - » A origem do requisito é claramente definida?
  - » O requisito é adaptável?
  - » Ele pode ser modificado sem que isso provoque efeitos em grande escala em outros requisitos do sistema?

Um dos mecanismos de validação de requisitos é a RTF (Revisão Técnica Formal) (PRESSMAN, 2006). Em uma RTF com esse propósito, a equipe de desenvolvimento deve conduzir o cliente pelos requisitos do sistema, explicando as implicações de cada requisito. A equipe de revisão deve verificar cada requisito em termos de sua consistência e verificar os requisitos como um todo sob o ponto de vista de sua completeza. A RTF é uma atividade que garante a qualidade do software.

9.7 Gestão de requisitos

Gestão de requisitos é um conjunto de atividades que ajuda a equipe de projeto a identificar, controlar e rastrear requisitos e modificações de requisitos a qualquer momento no desenvolvimento do sistema.

Para auxiliar na gestão de requisitos, tabelas de rastreamento são criadas com o objetivo de verificar as relações entre requisitos e o impacto da mudança de requisitos.

Os requisitos mudam e há uma série de razões para isso; por exemplo, os problemas podem ser muito intricados; as especificações podem ficar incompletas; o aumento do entendimento do sistema pode ocorrer ao longo do projeto; como existem vários usuários e visões, quem paga pelo sistema geralmente é diferente de quem o usa; o fato de que a empresa e o ambiente se modificam e, na medida em que os usuários se familiarizam com o sistema, novos requisitos surgem, etc.

A gestão de requisitos precisa de apoio automatizado, como ferramentas CASE (*Computer Aided Software Engineering*) para:

- Armazenamento de requisitos: os requisitos devem ser mantidos em um depósito de dados seguro, gerenciado, que seja acessível por todos os envolvidos no processo de engenharia de requisitos.

• Gerenciamento de mudanças: esse processo pode ser simplificado se o apoio de ferramentas estiver disponível.
• Gerenciamento de facilidade de rastreamento: o apoio de ferramentas para a rastreabilidade permite que sejam descobertos requisitos relacionados.

Aula 10 – Levantamento de requisitos

De acordo com o PMBoK (*Project Management Body of Knowledge*), guia que reúne o conjunto de melhores práticas em gerenciamento de projetos produzido pelo PMI (2008), o processo de levantamento ou elicitação de requisitos envolve:

• **Coleta:** processo de definir e documentar as funções e funcionalidades necessárias do projeto e do produto para atender às necessidades e expectativas dos *stakeholders*. O sucesso do projeto é diretamente influenciado pela atenção na coleta e gerenciamento dos requisitos do projeto e do produto. Os requisitos incluem as necessidades quantificadas e documentadas e as expectativas do patrocinador, cliente e demais *stakeholders*. Tais requisitos precisam ser obtidos, analisados e registrados com detalhes suficientes para serem medidos, uma vez que a execução do projeto se inicie. Coletar os requisitos reúne as atividades de definir e gerenciar as expectativas do cliente.
• **Entrevistas:** uma entrevista é um meio, que pode ser formal ou informal, de descobrir informações dos stakeholders por meio de conversas diretas. Normalmente é feita com perguntas preparadas ou espontâneas e registro das respostas. As entrevistas são frequentemente conduzidas individualmente, mas podem envolver múltiplos entrevistadores e/ou entrevistados. Entrevistar participantes experientes, stakeholders e especialistas no assunto do projeto pode auxiliar na identificação e definição das características e funções das entregas desejadas.
• **Documentação:** os componentes da documentação podem incluir, mas não estão limitados: a) à necessidade do negócio ou oportunidade a ser aproveitada, descrevendo as limitações da situação atual e por que o projeto foi empreendido; b) aos objetivos do negócio e do projeto para permitir rastreamento; c) aos requisitos funcionais, descrevendo processos de negócio, informações e interação com o produto de forma apropriada a ser documentada textualmente; d) aos requisitos não funcionais, tais como nível de serviço, desempenho, cuidados, segurança, atendimento a leis e regulamentos, etc.; e) aos impactos em outras áreas organizacionais e f) aos impactos em outras entidades internas ou externas à organização.

A Figura 14 ilustra o fluxo ao qual é submetida uma análise de requisitos toda vez que o cliente manifesta uma (nova) necessidade.

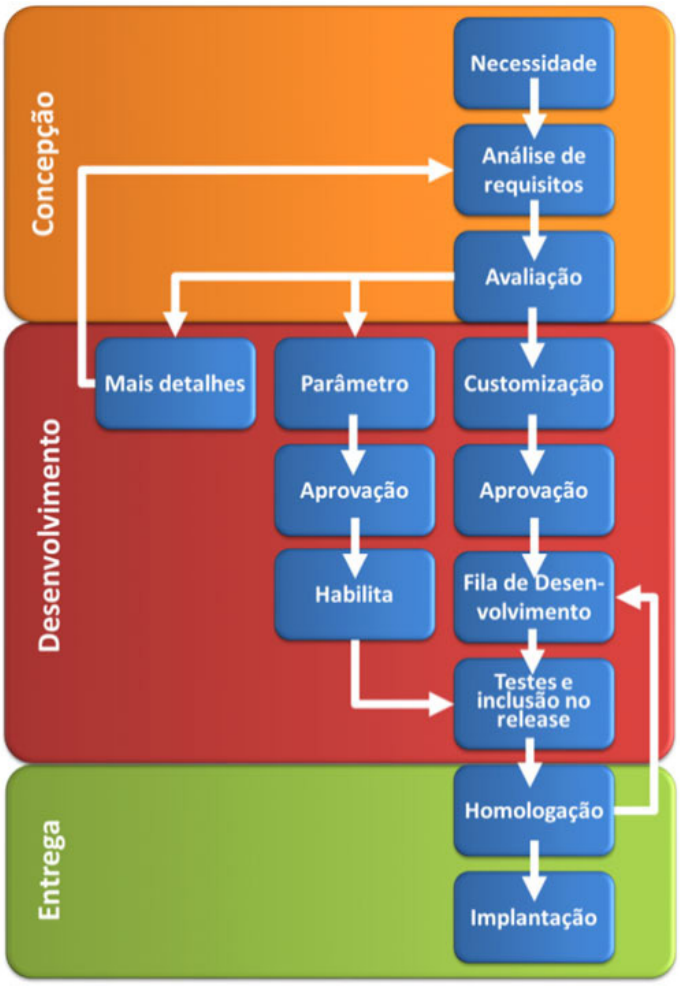


Figura 14. Fluxo da análise de requisitos.

Para que esse processo ocorra com sucesso, os requisitos devem ser claros, completos, sem ambiguidade, implementáveis, consistentes e testáveis, como mostra a Tabela 1. Os requisitos que não apresentem tais qualidades são problemáticos e devem ser revistos e renegociados com os clientes e usuários.

Tabela 1. Características dos requisitos.

Não ambíguo	Todo requisito tem apenas uma interpretação possível.
Correto	Todo requisito representa algo requerido ao sistema/funcionalidade a ser construído.
Completo	O requisito contém tudo que o software deve fazer e atender em todas as situações possíveis.
Compreensível	As pessoas podem facilmente compreender o significado de todos os requisitos com um mínimo de explicação.
Verificável	Os requisitos atendem ao conjunto de técnicas que podem ser usadas para verificar que todo requisito está implementado pelo sistema.
Internamente consistente	Não há requisitos conflitantes entre si.
Externamente consistente	Não há requisitos conflitantes com outra documentação de projeto já definida.
Realizável	Possibilita que um projeto seja implementado corretamente com todos os requisitos declarados.
Conciso	É tão pequeno quanto possível, sem afetar qualquer outra qualidade que deva atender.
Rastreável	É escrito de modo que facilite o "referenciamento" de cada requisito individualmente.
Modificável	Sua estrutura e estilo são projetadas de forma que qualquer mudança possa ser feita facil, completa e consistentemente.
Anotado por importância relativa	Permite que se possa facilmente determinar qual requisito é mais importante para os clientes.
Anotado por estabilidade relativa	Permite que se possa facilmente determinar qual requisito é mais suscetível à mudança.
Organizado	Permite que se possa facilmente localizar informações e relacionamentos lógicos entre seções adjacentes.

Fonte: Cruz e Gonsalez (2010).

Aula 11 – Técnicas de coleta de requisitos

A coleta ou extração de requisitos é feita por meio de técnicas. Nesta etapa, os requisitos são documentados à medida que são coletados e desse processo resulta um documento preliminar dos requisitos do sistema. A seguir são apresentadas as principais técnicas utilizadas.

Fonte: http://www.gruporizzato.com/wp-content/uploads/2011/10/entrevista-de-emprego.jpg

Entrevistas

Referem-se a encontros com clientes ou usuários que explicam o seu trabalho, ambiente

em que atuam, necessidades, dentre outros assuntos pertinentes. As entrevistas exigem do analista de requisitos habilidades sociais como saber ouvir, inferir e dirimir conflitos, sendo que essas habilidades são estendidas à equipe de desenvolvimento. Os quatro passos de uma entrevista são:

[1] Planejamento da Entrevista:

- Deve-se decidir quem será entrevistado.
- É necessário preparar os entrevistados (agendar data e hora, comentar sobre o assunto).
- É preciso preparar uma lista com diversos tipos questões (abertas: "explique como esse relatório é produzido"; fechadas: "quantos relatórios desse tipo são gerados?" e sequenciais: "por quê?", "dê um exemplo"). Deve-se preocupar também em dar continuidade a uma questão.
- Preparar mais de uma questão para um tópico a fim de confirmar a resposta e deixá-la mais completa.

[2] Condução da entrevista:

- Pirâmide: começa com questões fechadas e expande para questões abertas dirigidas. É uma abordagem importante quando o entrevistado parece relutante em falar do assunto.
- Funil: começa obtendo detalhes e dá continuidade obtendo respostas diretas. Muitas questões fechadas e sequenciais tornam-se desnecessárias.
- Diamante: combina as duas abordagens anteriores. A entrevista fica menos cansativa, pois varia o tipo de questão. Na entrevista não se deve induzir respostas (como uma questão do tipo: "o relatório deveria ser gerado semanalmente?").

[3] Finalização da entrevista:

Deve-se reservar um tempo, o menor possível, que seja adequado para sumarizar as informações recebidas. O analista deve explicar os próximos passos ao cliente e apresentar a importância da entrevista, agradecendo ao entrevistado.

[4] Análise de resultados:

Deve-se produzir um documento da entrevista e descobrir ambiguidades, conflitos e omissões. As informações devem ser consolidadas e documentadas, mas a técnica de entrevista apresenta diversos problemas:

- Observações divergentes: pessoas diferentes se concentram em diferentes aspectos e podem "ver" coisas diferentes.
- Interpretações diferentes: o entrevistador e o entrevistado podem estar interpretando palavras comuns de maneira diferente, tais como "pequena quantidade de dados" ou "caracteres especiais".
- Ambiguidades: ambiguidades existem em todas as formas de comunicação, especialmente em linguagem natural.
- Conflitos: entrevistador e entrevistado podem ter opiniões conflitantes sobre um determinado problema e a tendência é registrar o ponto de vista do entrevistador.

Questionários

Utilizar questionário é uma forma rápida de obter dados de uma grande quantidade de usuários, que podem estar em lugares geograficamente distintos. Os tipos de dados que podem ser coletados por meio de questionários são a) dados sobre a utilização do sistema atual; b) problemas e dificuldades que os usuários enfrentam em seu trabalho e c) expectativa dos usuários em relação ao novo sistema.

As questões devem ser claras e objetivas e deve-se preparar mais de uma questão para um tópico, a fim de confirmar a resposta e deixá-la mais completa.

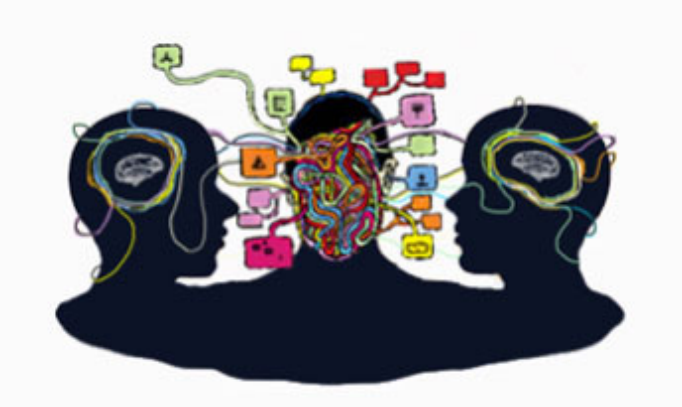
Os tipos de questões que compõem os questionários podem ser:

- **Abertas-dirigidas:** antecipam o tipo de resposta. Devem ser utilizadas quando não é possível criar alternativas. Por exemplo: "Por que você acha que os manuais do usuário do sistema financeiro não funcionam?"
- **Fechadas:** utilizadas quando é possível listar as possíveis alternativas. Por exemplo: "Os dados sobre vendas são entregues com que frequência?"; "Alternativas: diariamente, semanalmente, quinzenalmente, mensalmente, trimestralmente".

Na elaboração do questionário, deve-se tomar os seguintes cuidados: a) iniciar com as questões

mais importantes; b) questões com conteúdo semelhante e relacionadas devem estar próximas e c) questões que podem gerar controvérsias devem ser deixadas para depois.

Brainstorming



Fonte: http://community.vfs.com/oomph/tag/brainstorming/

Trata-se de uma técnica básica para a geração de ideias. Devem ser realizadas uma ou várias reuniões que permitam que as pessoas sugiram e explorem ideias sem que sejam criticadas ou julgadas. Deve existir um líder responsável por conduzir a reunião sem restringi-la. Essa técnica é especialmente útil no começo do processo de extração de requisitos, mas apresenta uma desvantagem: por não ser muito estruturada, pode não produzir a mesma qualidade ou nível de detalhe que se consegue com outras técnicas.

O brainstorming (ou tempestade cerebral) basicamente é dividido em duas etapas:

[1] Geração das ideias: reuniões que têm como objetivo fornecer ideias, sem discussões sobre o mérito delas. Existem quatro regras:

- É proibido criticar ideias.
- Ideias não convencionais ou estranhas são encorajadas.
- O número de ideias geradas deve ser bem grande.
- Os participantes devem ser encorajados a enriquecer as ideias dos outros participantes.

[2] Consolidação das ideias: as ideias geradas são discutidas, revisadas, organizadas, avaliadas, consolidadas, descartadas e priorizadas.

JAD (*Joint Application Development*)

Técnica utilizada para promover a cooperação, o entendimento e o trabalho em grupo entre usuários e desenvolvedores. É baseada em quatro princípios:

1) Dinâmica em grupo: utilização de sessões de grupo (encontros de usuários e desenvolvedores).
2) Uso de técnicas visuais: utilização de data-show, projetor, vídeos, etc.
3) Manutenção do processo organizado e racional: controle da sessão por um líder, que tem o objetivo final de identificar um conjunto preliminar de requisitos.
4) Utilização de documentação padrão: criação de um documento com os requisitos identificados.

Geralmente existem seis papéis (responsabilidades) divididas entre a equipe de desenvolvimento e usuários, que são: líder da sessão, engenheiro de requisitos, executor, representantes dos usuários, representantes de produtos de software e especialistas.

5W1H

A técnica 5W1H é mundialmente difundida e utilizada em diversos tipos de situações, sendo muito útil na coleta de requisitos. Veja na sequência o que são os 5 Ws e o único H, assim como algumas perguntas comumente aplicadas nessas fases:

- **Who (quem)** – refere-se às responsabilidades.
  - » Quem é o cliente/usuário do sistema?
  - » Quem executa?
  - » Quem gerencia?
  - » Quem fornece?
  - » Quem participa das decisões?

- **What (o que)** – refere-se às etapas.
  - » Quais são as entradas do sistema?
  - » Quais são as saídas?
  - » Quais são os indicadores?
  - » Quais são as metas?
  - » Quais são os recursos?
  - » Quais são os problemas?
  - » Quais são os métodos/tecnologias empregados?
  - » Que informações ou insumos são necessários para o trabalho? Quem as fornecem?
  - » Quais são as regras que determinam como o trabalho será feito?

» Há alguma coisa que possa parar, atrasar ou impedir o processo?
» Há espera para completar o processo?
» Quais são as exceções?
» Quais são as alternativas, caso o sistema não funcione conforme as expectativas?
» Qual ação é tomada quando uma etapa falha?

- **When (quando)** – refere-se ao tempo.
  » Quando é planejado o processo?
  » Quando é executado?
  » Quando é avaliado?
  » Quanto tempo leva o processo?
  » Com que frequência a atividade é executada?

- **Where (onde)** – refere-se aos locais.
  » Onde é planejado o processo?
  » Onde é executado?
  » Onde é avaliado?

- **Why (por quê?)** – refere-se às justificativas.
  » Por que/para que esse processo existe?
  » Quais são os fatores que determinam quando um produto é aceitável?

- **How (como)** – refere-se aos métodos.
  » Como é planejado o processo?
  » Como é executado?
  » Como é avaliado?
  » Como as informações são registradas e disseminadas?
  » Como é avaliada a satisfação do cliente?
  » Como são as medidas específicas associadas ao sistema, caso existam?

Fonte: acervo pessoal.

PIECES (Performance, Informação, Economia, Controle, Eficiência, Serviços)

Quando o desenvolvedor inexperiente apresenta dificuldades em determinar como começar e o que perguntar para extrair os requisitos do cliente, é interessante utilizar a técnica PIECES, pois ela ajuda a resolver esse problema. A técnica fornece um conjunto de categorias de problemas que ajudam o engenheiro de requisitos a estruturar o processo de extração de requisitos. Em cada categoria existem várias questões que o desenvolvedor deve explorar com os usuários, podendo a técnica ser adaptada para domínios de aplicação específicos.

As seis categorias de questões são:

1) **Performance:** reflete o que usuário espera;
2) **Informação (e dados):** refere-se ao tipo de acesso às informações (relatórios, funções on-line) e inclui a quantidade de informação oferecida pelo software) na medida certa, no tempo propício e em forma utilizável);
3) **Economia:** relaciona-se ao custo de usar um produto de software (processadores, armazenagem, conexões de rede, etc.);
4) **Controle:** inclui as restrições de acesso ao sistema, acesso a algumas informações e habilidade de monitorar o comportamento do sistema;
5) **Eficiência:** procura evitar coletar o mesmo dado mais de uma vez e armazená-lo em espaços múltiplos;
6) **Serviços:** refere-se a que tipo de serviços os usuários necessitam que o software realize.

Prototipação

Técnica que tem como objetivo extrair, entender e validar os requisitos. É baseada no conceito do processo de desenvolvimento de software de prototipação, já estudado anteriormente. Um protótipo do produto pode ser construído e, por meio do protótipo, os usuários podem descobrir quais são as suas reais necessidades. Tal técnica traz benefícios somente se o protótipo puder ser construído mais rápido do que o sistema real. É especialmente útil para superar dificuldades de comunicação por parte dos usuários e transmitir suas necessidades.

Na geração do documento preliminar dos requisitos extraídos em qualquer técnica, pode-se utilizar o modelo de caso de uso da UML para se ter uma ideia geral do escopo do produto de software. Geralmente esse não é o modelo final, mas serve como subsídio para a próxima etapa da engenharia de requisitos, referente à análise e negociação de requisitos.

Exercícios do Módulo 3

1) *Stakeholder* é qualquer pessoa que vá comprar o software em desenvolvimento.

a) a) Verdadeiro.
b) b) Falso.

2) É relativamente comum para diferentes usuários propor requisitos conflitantes, cada qual argumentando que sua versão é a mais adequada ou melhor.

a) a) Verdadeiro.
b) b) Falso.

3) Na validação de requisitos, o modelo de requisitos é revisto para assegurar sua viabilidade técnica.

a) a) Verdadeiro.
b) b) Falso.

4) Faça a associação e assinale a resposta correta sobre a técnica PIECES:

i-Performance ii-Informação iii-Economia iv-Controle v-Eficiência vi-Serviços

a) *relaciona-se ao custo de usar um produto de software: processadores, armazenagem, conexões de rede, etc.*
b) *inclui as restrições de acesso ao sistema, acesso a algumas informações e habilidade de monitorar o comportamento do sistema.*
c) *reflete o que usuário espera.*
d) *refere-se a que tipo de serviços os usuários necessitam que o software realize.*
e) *refere-se ao tipo de acesso às informações: relatórios, funções on-line; inclui a quantidade de informação oferecida pelo software: na medida certa, no tempo propício e em forma utilizável.*
f) *procura evitar coletar o mesmo dado mais de uma vez e armazená-lo em espaços múltiplos.*

a) i-c ii-e iii-a iv-b v-f vi-d
b) i-f ii-c iii-a iv-e v-d vi-b
c) i-e ii-d iii-b iv-f v-c vi-a
d) i-a ii-e iii-f iv-c v-b vi-d

5) Assinale a afirmativa ERRADA:

a) Requisitos funcionais: são requisitos que descrevem a funcionalidade (funções que o sistema deve realizar) ou os serviços que se espera que o sistema faça.
b) O principal objetivo da engenharia de requisitos é a obtenção de uma especificação correta e completa dos requisitos de um sistema de software.
c) Requisitos não-funcionais: são requisitos que não dizem respeito diretamente à funcionalidade do sistema, mas expressam suas propriedades e/ou restrições sobre os serviços ou funções por ele providas. Exemplos de requisitos não funcionais: cadastro de cliente; emissão de nota fiscal; consulta ao estoque; geração de pedido.
d) Requisitos segundo o IEEE: é uma condição ou capacidade que um usuário necessita para resolver um problema ou alcançar um objetivo. Uma condição ou capacidade que deve ser satisfeita por um sistema para satisfazer um contrato ou um padrão.

6) Quais os benefícios esperados de um documento de especificação de requisitos?

- Que estabeleça _____ entre os clientes e a empresa fornecedora sobre o que o software irá fazer;
- Que _____ o esforço de desenvolvimento;
- Que forneça uma base para estimativas de _____;
- Que forneça uma base para _____ do produto;
- Que facilite a _____ do produto de software final.

MÓDULO 4 – TESTES DE SOFTWARE

Pensar em teste de software nos remete, normalmente, à avaliação das linhas de código para encontrar falhas. Mas não se engane: testar um software tem um objetivo muito mais amplo.

O software precisa gerar o resultado esperado e, por isso, testá-lo envolve planejar e controlar a aplicação, além de validar seus resultados. O principal objetivo sempre será a busca de qualidade.

Neste módulo você entenderá esses conceitos, o que ajudará a conhecer as técnicas, estratégias e a importância dos testes de software.

Aula 12 – Fundamentos dos testes de software

12.1 Defeito, falha e erro

Para começarmos a falar sobre testes, precisamos entender primeiro a diferença entre defeito, falha e erro (veja Figura 15).

- **Defeito:** ocorre quando há uma instrução ou comando incorreto. Um veículo, por exemplo, apresenta defeito quando não obedece ao comando do motorista.
- **Falha:** é um comportamento inconsistente. No exemplo do veículo, quando ocorre uma falha, ele continua funcionando de forma precária, mas não para.
- **Erro:** é um desvio da especificação. Em um veículo seria semelhante a quando ocorre uma montagem indevida de algum componente.

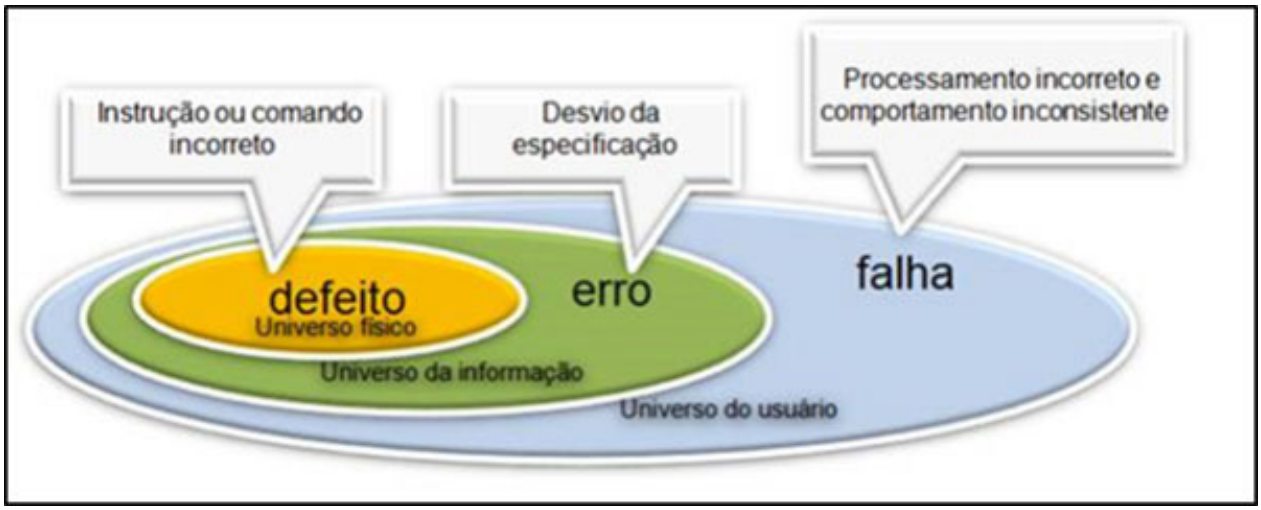


Figura 15. Defeito, falha e erro.
Fonte: http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035#ixzz27Pk4sNNA.

Mas por que é importante termos bem definidas essas diferenças? Considere a seguinte frase: "A melhor maneira de tentar construir a confiança é tentar destruí-la".

Parece um paradoxo, concorda? Pois saiba que não apenas parece, mas é um paradoxo.

Na tentativa de destruir a confiança para encontrar defeitos, falhas ou erros, na verdade estamos construindo uma confiança cada vez mais sólida no produto desenvolvido.

Pense, por exemplo, nos exaustivos testes automobilísticos, quando as montadoras literalmente destroem seus veículos para provar a

qualidade de seus bens produzidos. Você concorda que estão provando com isso a segurança que se espera em um momento crítico, como um acidente numa estrada em alta velocidade?

Test Crash
Fonte: http://just4x4sales.com.au/images/crash%20test.jpg.

Os projetos de software geralmente são muito complexos, então é fácil deduzirmos que será praticamente impossível que eles não possuam alguns defeitos — ainda mais porque em um grande projeto temos várias pessoas envolvidas, com formação e lógica diferentes, trabalhando juntos para atingir os objetivos.

Vários poderiam ser os motivos para alegar a impossibilidade de criação de um software sem qualquer tipo de erro, falha ou defeito, mas não queremos contribuir nem para a busca da perfeição (que não existe), nem para o relaxamento relacionado à inexistência de perfeição. Assim, vamos mostrar a importância dos testes, estratégias e testes para que você as aplique.

Qualidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade são atributos buscados incansavelmente no desenvolvimento de software e o que nos propomos a fazer na consideração deste assunto é analisar como o testador pode contribuir para essa busca quase inalcançável.

O processo de testes é efetivo quando encontra defeitos e é eficiente quando encontra os defeitos com a menor quantidade de recursos possível.

12.2 Teste de software e estratégia de testes

De acordo com Pressman (2010), um software é testado para descobrir erros que foram introduzidos inadvertidamente no momento em que foi projetado e construído.

> Teste é um conjunto de atividades que podem ser planejadas antecipadamente e conduzidas sistematicamente. Um gabarito de testes de software, que é um conjunto de passos no qual é possível incluir técnicas de projetos de casos de teste e métodos de teste específicos, deve ser definido para o processo de software, sendo utilizado no projeto (PRESSMAN, 2010).

O autor lista um conjunto de perguntas e respostas que apresentam os principais fundamentos que embasam os testes de software, quais sejam:

• **Quem faz?** O gerente do projeto, os engenheiros de software e especialistas em testes devem desenvolver uma estratégia de testes, de forma que o projeto tenha um plano formal de testes.

Fonte: http://www.techspotters.co.uk/blog/amdocs-named-top-vendor-for-telecom-software-testing-services/

• **Porque é importante?** O teste é a atividade que requer maior esforço de projeto dentro do ciclo de desenvolvimento de software. Se for conduzido ao acaso pode implicar em desperdício de tempo e esforço, além de abrir espaço para a infiltração de erros. Assim, é muito importante que se estabeleça uma estratégia sistemática para o teste de software.

• **Quais são os passos?** O teste começa nos componentes do software e vai aumentando seu campo de ação, de forma a abarcar todo o projeto. Depois que os componentes são testados eles precisam

ser integrados até que todo o sistema seja construído. A partir desse ponto uma série de testes de alto nível é executada para descobrir erros relativos aos requisitos do cliente. À medida que erros forem encontrados, é necessário se fazer um diagnóstico e buscar sua correção pelo processo de depuração do sistema.

• **O que é uma especificação de testes?** É um documento que relata a abordagem da equipe que realiza os testes, definindo um plano que descreve a estratégia global e um procedimento que define os passos específicos dos testes, bem como quais testes serão realizados.

• **Como ter certeza que os testes foram feitos corretamente?** Para certificar-se sobre esse aspecto deve-se realizar uma revisão na especificação do teste antes de realizá-lo, de forma a avaliar a completeza dos casos e das tarefas elencadas. Um plano e procedimento de testes efetivos vão levar à construção coordenada do software e à descoberta de erros em cada fase da construção do projeto.

De acordo com IBM (2006), uma estratégia para o teste de um projeto descreve a abordagem geral e os objetivos das atividades de teste. Ela inclui os estágios de teste (unidade, integração e sistema) que devem ser abordados e os tipos de teste (de função, desempenho, carga, estresse, etc.) que devem ser executados. A estratégia deve definir as ferramentas e técnicas de teste a serem empregadas e os critérios de conclusão e êxito que serão usados.

> Uma estratégia de testes de software integra métodos de projeto de casos de teste em uma série bem planejada de passos, que resultam na construção bem sucedida de software. A estratégia fornece um roteiro que descreve os passos a serem conduzidos como parte do teste, quando esses passos são planejados e depois executados, além do quanto de esforço, tempo e recursos serão necessários (PRESSMAN, 2010).

Além de enfatizar a importância da estratégia de testes, Pressman (2010) afirma que esta deve ser suficientemente flexível para criar uma abordagem de teste sob medida para o projeto em desenvolvimento, mas ao mesmo tempo ser rígida para promover um planejamento adequado e permitir um acompanhamento gerencial na medida em que o projeto progride.

O plano de teste deve definir conjuntos de critérios de conclusão para os testes unitários, de integração e de sistema. Pode haver diferentes conjuntos de critérios de conclusão definidos para iterações individuais. Uma estratégia para testes de software pode ser vista no contexto da espiral mostrada na Figura 16.

Figura 16. Espiral de estratégia de testes.
Fonte: Pressman (2010).

O teste de unidade começa no centro da espiral e concentra-se em cada componente (trecho de código-fonte) do software. O teste progride movendo-se para fora ao longo da espiral, indo em direção ao teste de integração, que foca no projeto e na construção da arquitetura do software.

Seguindo para fora da espiral, há o *teste de validação*, no qual os requisitos são validados, ou seja, a especificação dos requisitos é confrontada com o software que acabou de ser construído. Finalmente, chega-se ao teste de sistema, em que os outros elementos do software são testados como um todo.

Sommerville (2007) e Pressman (2010) descrevem, de forma complementar, esses estágios de teste da seguinte forma:

• **Teste de unidade:** é também conhecido como teste unitário. Avalia a menor unidade do código e seu objetivo é verificar falhas de funcionamento em partes pequenas independentes do software. Possibilita uma análise mais profunda e específica de uma função independente do resto do código, facilitando a descoberta de erros nos limites dos módulos. Esse teste tem como base estratégica o teste de caixa branca. Nesse teste, os casos são projetados para descobrir erros devido a computações errôneas, comparações incorretas ou fluxo de controle impróprio.

• **Teste de integração:** avalia diferentes componentes que são desenvolvidos separadamente mas trabalham em conjunto. Ao avaliar esses componentes de forma isolada, consegue-se encontrar possíveis falhas no resultado apresentado em consequência do mau funcionamento ou erro de algum dos componentes.

• **Teste de validação:** avalia o software em um ambiente específico, considerando os requisitos definidos pelo cliente em uma situação próxima à realidade. Tem como objetivo provar ao cliente que o software atende às solicitações desejadas.

• **Teste de sistema:** tenta impor a visão do cliente, dando normalmente uma perspectiva diferente ao testador. Normalmente executa-se em um ambiente que se assemelha ao ideal. Seu objetivo é pôr o sistema desenvolvido completamente à prova, com todos os elementos adequadamente integrados e realizando suas funções corretamente. Em testes de sistemas encontramos pelo menos quatro tipos de testes:

» Teste de recuperação: esse procedimento avalia a recuperação de uma falha dentro de um tempo especificado em caso de falha. Podem ocorrem várias falhas forçadas para verificar a recuperação.

» Teste de segurança: nesse caso o testador tenta penetrar no sistema provocando ações que prejudiquem alguém. Ele usa formas ilegais ou impróprias simulando condições extremas de violação das informações.

» Teste de estresse: esse procedimento cria ambientes extremos para utilização do software. Volumes anormais e frequência irregular têm como objetivo forçar falhas por sobrecarga, verificando as possíveis interrupções e consequências de paradas anormais por estresse. Os dados, então, são avaliados para ver se não foram corrompidos ou perdidos.

» Teste de desempenho: todo sistema desenvolvido tem em sua especificação de requisitos o desempenho adequado de funcionamento e após a integração dos componentes uma avaliação é indispensável para se obter o seu desempenho real.

De acordo com Pressman (2010), do ponto de vista procedimental, o teste no contexto da engenharia de software é uma série de quatro passos que são implementados sequencialmente, conforme mostra a Figura 17.

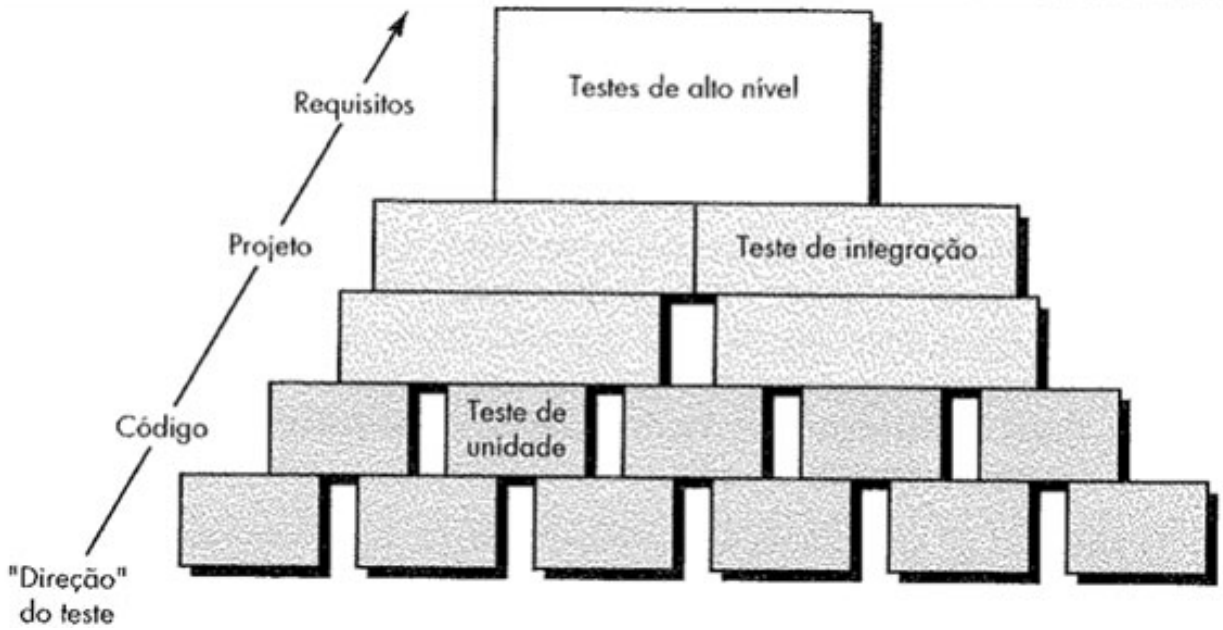


Figura 17. Passos de teste de software.
Fonte: Pressman (2010).

Inicialmente o teste focaliza cada componente individualmente, garantindo que ele funcione adequadamente como uma unidade. Em seguida, os componentes devem ser montados ou integrados para formar um pacote de software completo. Depois que o software tiver sido integrado, um conjunto de testes de alto nível é conduzido, de forma que critérios de validação de requisitos sejam avaliados. O último passo avança para fora dos limites da engenharia de software, indo em direção ao sistema de computação. O teste de sistema verifica se todos os elementos estão

ajustados de forma que a função e o desempenho global do sistema sejam alcançados.

Uma estratégia de teste deve acomodar testes de baixo nível, que atuem na identificação de problemas em um pequeno segmento do código-fonte e testes de alto nível, que validam as principais funções do sistema levando em consideração os requisitos do cliente. A estratégia fornece diretrizes aos profissionais e um conjunto de referências para o gestor.

Existem testes rápidos e alguns mais demorados para serem realizados. A definição de prioridades é essencial para o sucesso na execução dos testes.

Pressman (2010) garante que o teste de software é um fator decisivo para garantir a qualidade de um programa. Ainda de acordo com o autor, alguns testes, por sua simplicidade, podem ser executados a todo momento, como os testes de longevidade. Há outros que devem ser programados, como os testes de segurança e desempenho, os quais geram invariavelmente lentidão das ferramentas que necessitam utilizar. Outros tipos de testes, como carga e estresse, não precisam ser executados frequentemente, pois avaliam a infraestrutura e esta normalmente não é alterada com frequência.

Saff e Ernst (2003) afirmam que o uso de metodologia ágil incentiva a criação do ambiente de integração contínua, no qual se mantém o código-fonte sempre testado e confiável, diminuindo a alocação de recursos humanos em atividades de teste, pois esse framework atua com entregas imediatas — o que exige validação imediata.

É importante destacar que executar testes com frequência é fundamental. É mais vantajoso executar baterias pequenas e frequentes de testes do que uma rara e grande bateria. Além disso, as baterias de testes que não são executadas com frequência podem se tornar obsoletas, correndo o risco de ficarem desatualizadas e, consequentemente, não acompanharem a evolução natural dos componentes relacionados.

Ressalta-se ainda que quanto mais obsoleto se torna um teste, mais caro fica, comprometendo recursos e podendo, no momento mais importante, não atender à solicitação de seus usuários. Sua falta de execução periódica causa também acúmulos de erros ou falhas, que podem tornar a implementação de melhorias incompatível com o cumprimento de prazos estabelecidos.

Aula 13 – Tipos de testes de software

13.1 Testes automatizados

De acordo com Bastos et al. (2007), quanto mais cedo se iniciarem os testes, mais barata será a correção dos erros encontrados e, para conquistar esse benefício, o processo de teste, assim como o processo de desenvolvimento, deve ter um ciclo de vida, que é definido em fases.

Ainda segundo os autores, o ciclo de vida dos testes de software compreende as etapas que envolvem: planejamento, preparação, procedimentos iniciais, especificação, execução e entrega, como mostra a Figura 18.

Figura 18. Ciclo de vida do processo de teste.
Fonte: Bastos et al. (2007).

Segundo Pressman (2010), o teste frequente responde por mais esforço de projeto do que qualquer outra atividade de engenharia de software. Se for conduzido ao acaso, há um desperdício de tempo e esforço e, mais do que isso, erros se infiltram sem serem descobertos. Assim, seria razoável estabelecer uma estratégia sistêmica para o teste de software.

Testes automatizados podem fazer a diferença no que diz respeito à confiança que um software desenvolvido pode alcançar. Como qualquer artefato, o código-fonte dos testes automatizados e os documentos gerados requerem qualidade. À medida que aumenta a complexidade do software,

os testes automatizados também aumentam sua abrangência.

Fonte: http://silvioqualidade.wordpress.com/

De acordo com Bernardo e Kon (2008), testes automatizados são programas ou *scripts* simples que exercitam funcionalidades do sistema sendo testado e fazem verificações automáticas nos efeitos colaterais obtidos. A grande vantagem dessa abordagem é que todos os casos de teste podem ser facil e rapidamente repetidos a qualquer momento e com pouco esforço.

Os autores afirmam que a reprodução de um teste automatizado inúmeras vezes, em situações específicas, garante que passos importantes não sejam esquecidos, e isso certamente evitará uma situação indesejável.

Além disso, como os casos para verificação são descritos por meio de um código interpretado por um computador, é possível criar situações de testes bem mais elaboradas e complexas do que as realizadas manualmente, possibilitando qualquer combinação de comandos e operações. A magnitude dos testes pode também ser facilmente alterada. Por exemplo, é trivial simular centenas de usuários acessando um sistema ou inserir milhares de registros em uma base de dados, o que não é factível com testes manuais.

Uma prática muito utilizada até a década de 1990 era criar uma função de teste em cada módulo ou classe do sistema que continha algumas simulações de uso da unidade. Esta prática implicava em problemas, já que a legibilidade do código ficava comprometida em função de códigos de testes serem adicionados ao próprio código do aplicativo. Este problema motivou o surgimento dos *frameworks* (como o JUNIT), que auxiliam e padronizam a escrita de testes automatizados, facilitando o isolamento do código de teste do código da aplicação.

De acordo com Lages (2010), a escolha dos testes automatizados candidatos, ou seja, os mais críticos, deve ser realizada com base no planejamento de um projeto de automação. O autor afirma que apesar de não existir uma categorização amplamente difundida, a experiência tem mostrado que os testes candidatos são normalmente agrupados nestas quatro áreas:

- *Smoke tests*: um conjunto mínimo de testes é selecionado com o objetivo de validar um *build* ou liberação antes do início de um ciclo de testes;
- Testes de regressão: os testes são selecionados com o objetivo de executar o reteste de uma funcionalidade ou da aplicação inteira;
- Funcionalidades críticas: os testes são selecionados com o objetivo de validar as funcionalidades críticas que podem trazer riscos ao negócio;
- Tarefas repetitivas: os testes são selecionados com o objetivo de reduzir o envolvimento dos testadores em atividades manuais repetitivas e suscetíveis a erros, tais como cálculos matemáticos, simulações, processamentos, comparações de arquivos ou dados, etc.

Cada tipo de teste possui restrições próprias relacionadas com o ambiente, que podem ter como dependência a flexibilidade e portabilidade das tecnologias utilizadas. Fatores externos de software ou hardware também podem influenciar na definição de onde executar os testes.

Um ambiente compartilhado de testes ou ambiente de homologação, que é um ambiente criado para replicar todas as condições do ambiente de produção, contribui para a eficiência na execução dos testes e qualidade final da entrega. Não é fácil a criação e manutenção desse ambiente, então, quando ele já está em uso, deve ser usado de forma abrangente.

Com o tempo, é comum essa infraestrutura do ambiente de homologação correr o risco de ficar desatualizada; caso isso ocorra, a equipe deve buscar mostrar aos responsáveis da área técnica a importância da manutenção de um ambiente de homologação atualizado. Esse ambiente deve espelhar o máximo possível do ambiente de produção, de forma a buscar garantir sucesso na execução dos testes.

Existem várias alternativas para criar ou montar esse ambiente de homologação, entre elas encontramos, por exemplo, a virtualização — que quando usada eficazmente pode gerar um ambiente adequado e de baixo custo. Quando há uma grande quantidade de testes sendo executados nesse ambiente simultaneamente, pode haver uma sensível queda de desempenho do sistema, o que normalmente não compromete a confiabilidade dos testes.

De acordo com Delamaro, Maldonado e Jino (2007), os testes de desempenho, de carga, de estresse e de longevidade requerem um ambiente de testes similar ao ambiente de produção, porque os resultados obtidos estão diretamente relacionados com o ambiente. Os testes de segurança normalmente requerem, também, uma infraestrutura similar ao do ambiente de produção, pois se espera que tais testes exponham as vulnerabilidades do sistema no ambiente adequado. Os testes de correção (unidade, integração, aceitação, etc.) não necessitam dos mesmos ambientes, produção ou homologação; portanto, há maior flexibilidade, pois se pode usar até mesmo em um hardware local ou a máquina do desenvolvedor.

O custo da criação desses testes automatizados normalmente é bem maior do que o dos testes manuais, principalmente por causa da estrutura por trás desse ambiente.

Há softwares gratuitos que podem dar uma base ou referência para testes integrados de software. Veja alguns exemplos:

- JMeter: http://jakarta.apache.org/jmeter
- Selenium-IDE: http://selenium-ide.openqa.org
- Selenium: http://selenium.openqa.org
- Selenium RC: http://selenium-rc.openqa.org
- SUnit: http://sunit.sourceforge.net
- JUnit: http://www.junit.org
- TestNG: http://testng.org
- JSUnit: http://www.jsunit.net
- CppTest: http://cpptest.sourceforge.net
- csUnit: http://www.csunit.org

13.2 Testes não automatizados

De acordo com Caetano (2008), a automação de testes tem o objetivo de reduzir o envolvimento humano em atividades manuais repetitivas. Entretanto, isto não significa que a automação de testes deve se limitar apenas a fazer o trabalho repetitivo e entediante. Os casos de testes manuais são criados num contexto apropriado à sua execução de forma manual. Muito provavelmente, casos de testes manuais não conseguirão a mesma eficiência em um contexto em que os testes venham a ser executados de forma automática.

A automação de testes é pouco eficaz quando os testes são complexos e exigem interações intersistemas ou validações subjetivas. Assim, na prática, a automação de 100% dos testes manuais nem sempre é a melhor estratégia. Além disso, muitas vezes o custo e o tempo para automatizar os testes de um projeto são maiores que o custo e o tempo do próprio projeto de desenvolvimento.

Segundo Lages (2010), quando se deseja aplicar a automação na etapa de execução de testes das funcionalidades de um sistema, vários fatores devem ser avaliados para se identificar se é vantagem ou não se automatizarem os testes. A automação de testes deve ser feita somente quando uma rotina específica de teste for executada várias vezes, pois a automação também demanda tempo de criação e manutenção de *scripts*. Lages ressalta que cerca de 70% dos testes utilizados em empresas atuantes no mercado são manuais e que o teste automatizado é praticamente limitado ao Teste de Regressão. Segundo o autor, os testes feitos de forma manual são os que realmente detectam os erros e falhas nos sistemas.

Existem diversos tipos de testes para detectar a ineficiência de um sistema, e antes de muitos deles serem automatizados são executados por várias

vezes em modo manual, para depois que houver a comprovação do sucesso de sua utilização serem automatizados. Citamos na sequência alguns desses testes:

- *Teste de desempenho*: teste que determina o desempenho de um sistema. As ferramentas que apoiam os testes de desempenho possuem duas facilidades, a saber, geração de carga e mensuração das transações do teste. De acordo com Silva (2012), a geração de carga pode simular múltiplos usuários ou grandes volumes de dados de entrada. Durante a execução, os tempos de resposta para determinadas transações são medidos e registrados. As ferramentas de teste de desempenho, normalmente, fornecem relatórios baseados nos registros dos testes e gráficos da carga em função dos tempos de resposta.
- *Teste de carga*: teste que mede o comportamento de um sistema com uma carga crescente, como por exemplo, número de usuários paralelos e/ou números de transações, para determinar qual a carga que ele pode suportar. Para Silva (2012), esses testes indicam também o tempo de resposta de transações e processos de negócios, com o intuito de determinar se a aplicação está de acordo com as expectativas documentadas e determinadas pelo cliente.
- *Teste de estresse*: teste que avalia a execução de um sistema ou componente no limite (ou acima) dos requisitos especificados. Esses testes determinam a carga necessária para que o sistema falhe e avaliam seu comportamento em condições anormais. Podem incluir, além de cargas de trabalho extremas, insuficiência de memória e recursos de processamento limitados. Avaliam também a capacidade que o software tem de se recuperar de uma falha mantendo a integridade dos dados (SILVA, 2012).
- *Teste de longevidade*: nesse teste é avaliado o funcionamento do sistema em médio e longo prazo. Seu objetivo é detectar influências externas ao software, principalmente de hardware ou sistema operacional. Fatores como grandes volumes armazenados, tamanho ou limpeza do cache podem ser decisivos na eficiência em médio e longo prazo de um software.
- *Teste de avaliação de desempenho*: muitos problemas de desempenho que podem surgir após o sistema ser submetido à carga extrema durante um longo período de tempo só podem ser detectados num ambiente controlado. De acordo com Silva (2012), existem tipos distintos de testes de desempenho, como os testes de configuração — que, da mesma forma que o teste de longevidade, visam identificar configurações otimizadas de hardware e software para a execução correta da aplicação — e os testes de contenção — que avaliam a forma como o sistema gerencia demandas para um mesmo recurso computacional (memória, dados, processador, etc.).
- *Teste de segurança*: os testes de segurança normalmente requerem uma infraestrutura similar à do ambiente de produção, pois se espera que esse tipo de teste exponha as vulnerabilidades do sistema no ambiente adequado. Testes de segurança garantem a confidencialidade, integridade e disponibilidade das informações. Um dos problemas relacionados a ele é o buffer overflow, ou a necessidade maior de capacidade de armazenamento.
- *Teste de correção*: os testes de correção (unidade, integração, aceitação, etc.) não necessitam das mesmas dependências. Têm como objetivo revelar a presença de erros, descobrindo o maior número possível deles.

Uma vez que o software tenha sido elaborado, é necessário fazer a implementação de testes manuais ou automatizados. Isso pode exigir conhecimento multidisciplinar, ou seja, conhecimento de programação, de testes de software, de linguagem de negócio e de ferramentas específicas. Portanto, o profissional responsável pelos testes automatizados normalmente é diferenciado.

Segundo Schach (2007), implementar refere-se ao processo de converter o projeto detalhado de um sistema de software em código. Se apenas uma única pessoa realizasse este trabalho, o processo seria mais fácil, mas a maioria dos produtos de software são muito grandes e complexos, exigindo uma equipe de desenvolvedores. Em cenários mais realistas, diversas pessoas trabalham simultaneamente e em diferentes componentes do sistema. Isso gera um fator complicador ao processo de testabilidade. Assim, desenvolvedores e testadores devem trabalhar de forma colaborativa e integrada.

Exercícios do Módulo 4

1) Uma falha:

a) Ocorre quando há uma instrução ou comando incorreto.
b) É um desvio de especificação.
c) É um comportamento inconsistente.
d) É uma informação inadequada fornecida.

2) Assinale a alternativa correta: um dos problemas relacionados a este teste é o buffer overflow, ou a necessidade de maior capacidade de armazenamento:

a) Segurança.
b) Longevidade.
c) Estresse.
d) Integração.
e) Sistema.

3) Assinale a alternativa correta: tem como objetivo revelar a presença de erros. Descobre o maior número possível deles, alguns dependendo do desenvolvimento previamente já identificado para comprovar a eficiência do teste.

a) Sistema.
b) Correção.
c) Estresse.
d) Integração.

4) São programas ou scripts simples que exercitam funcionalidades do sistema sendo testado e fazem verificações automáticas nos efeitos colaterais obtidos.

a) Testes oficiais.
b) Testes de equivalência.
c) Testes frequentes.
d) Testes manuais.
e) Testes automatizados.

5) Podemos afirmar sobre testes de baixo nível:

a) São testes realizados no fim do projeto.
b) São testes de hardware, por isso de baixo nível.
c) São testes completos do sistema.
d) Atuam na identificação de problemas em um pequeno segmento do código-fonte.
e) Atuam nos problemas complexos do programa.

6) Alguns testes, por sua simplicidade, podem ser executados frequentemente, como os testes _____. Há outros que devem ser programados, como os testes _____, os quais geram invariavelmente lentidão por causa das ferramentas que necessitam utilizar. Outros tipos de testes, como _____, não precisam ser executados a todo o momento, pois avaliam a infraestrutura e esta normalmente não é alterada com frequência.

MÓDULO 5 – QUALIDADE DE SOFTWARE

A aplicação de processos de qualidade à engenharia de software visa à construção de produtos com maior qualidade, pois padrões são seguidos durante todo o ciclo de vida do software. Esse processo é chamado de ciclo de vida da qualidade de software, pois inicia na concepção do software e segue até sua descontinuidade, visando ao controle do desenvolvimento, auxiliando na definição de prazos e tendendo a evitar imprevistos. Os modelos de qualidade, como o CMMI e MPS.BR, propõem a utilização de práticas e normatização de processos de desenvolvimento, objetivando a construção de um produto com qualidade. É isto que veremos neste módulo do curso.

Aula 14 – Qualidade de software e qualidade total

Como vimos, vários problemas ainda são comuns no desenvolvimento de software. Isso se deve principalmente pelo aspecto não repetitivo do desenvolvimento de produtos de software, o que torna a garantia da qualidade uma atividade difícil e, muitas vezes, imprevisível.

A delimitação do escopo de sistemas e/ou produtos de software também não é uma tarefa trivial. Muitas vezes o usuário não consegue definir com precisão todos os requisitos necessários ao projeto. Além disso, ainda existe a volatilidade dos requisitos, muito comum no desenvolvimento de software.

Todo esse cenário faz com que a importância da área de garantia da qualidade cresça continuamente nas organizações de desenvolvimento de software, pois a gerência de alto nível utiliza os resultados produzidos por essa área para obter visibilidade da qualidade dos processos executados e dos produtos entregues aos clientes.

Além disso, decisões estratégicas de negócio são tomadas com base em dados consolidados das atividades de garantia da qualidade. Esses e outros fatores aumentam a complexidade e a relatividade do conceito de qualidade de software, devido à sua forte dependência com a perspectiva de quem está avaliando determinado produto ou serviço.

> Segundo Pressman (2010), a garantia da qualidade de *software* está diretamente relacionada às características de qualidade do processo de desenvolvimento e de seus produtos intermediários, bem como aos esforços de melhoria de processos das organizações. Além disso, as atividades de garantia da qualidade devem estar presentes ao longo de todo o ciclo de vida de desenvolvimento do software, a fim de assegurar que o projeto, o desenvolvimento e a disponibilização de uma aplicação aconteçam de maneira bem-sucedida.

Em virtude disso, normalmente as organizações definem padrões, processos e procedimentos que devem ser seguidos para assegurar a uniformidade e o controle com relação ao desenvolvimento e à manutenção de software.

Assim, o principal objetivo da garantia da qualidade é assegurar que esses padrões e políticas utilizados durante o desenvolvimento do software sejam adequados para prover o nível de confiança requerido para o processo ou produto de trabalho.

No entanto, tal nível de confiança varia de acordo com os diferentes tipos de usuários dos produtos de software, bem como o grau esperado de adequação do produto aos propósitos para os quais foi desenvolvido. Portanto, deve-se considerar que usuários diferentes provavelmente terão propósitos distintos para o desenvolvimento de um mesmo produto.

No contexto de desenvolvimento de software, a qualidade pode ser entendida como um conjunto de características a serem satisfeitas em um determinado grau, de modo que o produto de software atenda às necessidades explícitas e implícitas de seus usuários. De acordo com Pressman (2010), qualidade é a conformidade com requerimentos e com características implícitas que são esperadas de um software profissionalmente desenvolvido.

Atualmente, a qualidade de software vem ganhando um grande foco nas empresas de TI, pois essas perceberam que a qualidade não é um gasto e sim um investimento. Além disso, com a evolução constante da tecnologia, os clientes estão cada vez mais exigentes, o que também exige dos desenvolvedores muito mais cuidado na criação dos produtos de software.

A preocupação das organizações é garantir qualidade em cada atividade realizada no processo de produção e evitar erros, de modo a produzir de forma correta da primeira vez e até eliminar a necessidade de inspeções, melhorando sistematicamente os processos, de modo a aumentar sempre sua qualidade. É desse conceito que nasce a qualidade total.

Qualidade total é a preocupação com a qualidade em todas as atividades da empresa, buscando sistematicamente o nível "zero defeito", por meio da melhoria contínua dos processos de produção.

O termo TQM (*Total Quality Management* ou Gerenciamento da Qualidade Total), amplamente usado nas organizações, descreve uma abordagem para a melhoria da qualidade. Os quatro elementos-chave do TQM podem ser vistos na Figura 19 e são descritos na sequência.

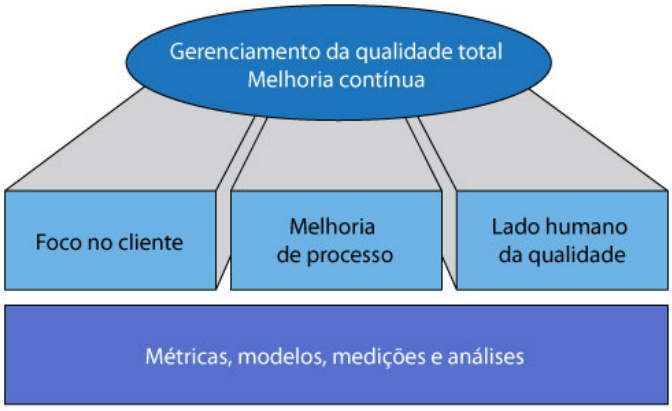


Figura 19. Elementos do gerenciamento da qualidade total.

- *Customer focus* (foco do cliente): o objetivo é atingir a satisfação total do cliente. O foco do cliente inclui o estudo das necessidades e vontades do cliente, coleta de requisitos do mesmo e a medição e gerenciamento da satisfação do cliente.
- *Process improvement* (melhoria de processo): o objetivo é reduzir as variações de processo e atingir a melhoria da qualidade contínua. Esse elemento inclui ambos os processos de negócio e o processo de desenvolvimento do produto. Por meio da melhoria de processo, a qualidade do produto será reforçada.
- *Human side of quality* (lado humano da qualidade): o objetivo é criar a cultura de qualidade por toda a empresa. As áreas de foco incluem liderança, apoio da alta gerência, participação total de todos os colaboradores da empresa e outros fatores humanos, sociais e psicológicos.
- *Metrics, models, measurement and analysis* (métricas, modelos, medições e análises): o objetivo é direcionar a melhoria contínua em todos os parâmetros da qualidade por um sistema de medição orientado a metas.

Na organização moderna, portanto, qualidade significa simultaneamente adequação ao uso, conformidade às especificações e qualidade total no processo. Chega-se, assim, ao ponto que nos interessa. As organizações têm de produzir produtos e serviços de qualidade, não mais como uma estratégia de diferenciação de mercado, mas como uma condição de preexistência. As empresas devem ter em mente a importância de juntar os conceitos de qualidade de processo e de projeto com qualidade total, entre outros, a fim de obter a qualidade total do produto, utilizando padrões e um bom planejamento.

Aula 15 – O modelo de qualidade CMMI

A abordagem de qualidade conhecida como CMM (*Capability Maturity Model*) pode ser definida como um conjunto de melhores práticas para diagnóstico e avaliação de maturidade do desenvolvimento de software em uma empresa.

O CMM é uma marca registrada do SEI (*Software Engineering Institute*), sediado na Universidade Carnegie Mellon, em Pittsburgh, EUA. Esse modelo é construído a partir do conceito de processo. O SEI tem por missão aprimorar o desenvolvimento e a adoção das melhores práticas de Engenharia de Software.

O CMMI (*Capability Maturity Model Integration*) pode ser usado como um guia de melhores

práticas cujo objetivo é melhorar os processos organizacionais e sua habilidade para gerenciar o desenvolvimento, a aquisição e a manutenção de produtos e serviços de software.

Fonte: http://www.totalbanco.com.br/cmmi.shtml 1

Como outros modelos CMM, os modelos CMMI fornecem um guia a ser usado para o desenvolvimento de processos. Os processos usados em uma organização dependem de muitos fatores, incluindo domínios de aplicação e estrutura e tamanho da organização.

15.1 Visão geral do modelo CMMI

O CMMI está organizado em três modelos (veja Figura 20) chamados de constelação, cada um contendo práticas para áreas de desenvolvimento, serviços e de aquisição:

- *CMMI for development* (CMMI-DEV): voltado ao processo de desenvolvimento de produtos e serviços.
- *CMMI for services* (CMMI-SVC): voltado aos processos de empresas prestadoras de serviços.
- *CMMI for acquisition* (CMMI-ACQ): voltado aos processos de aquisição e terceirização de bens e serviços.

A organização pode usar um modelo CMMI para ajudar a estabelecer objetivos e prioridades do melhoramento de processos, obtendo um guia para garantir estabilidade, processos estáveis e maduros.

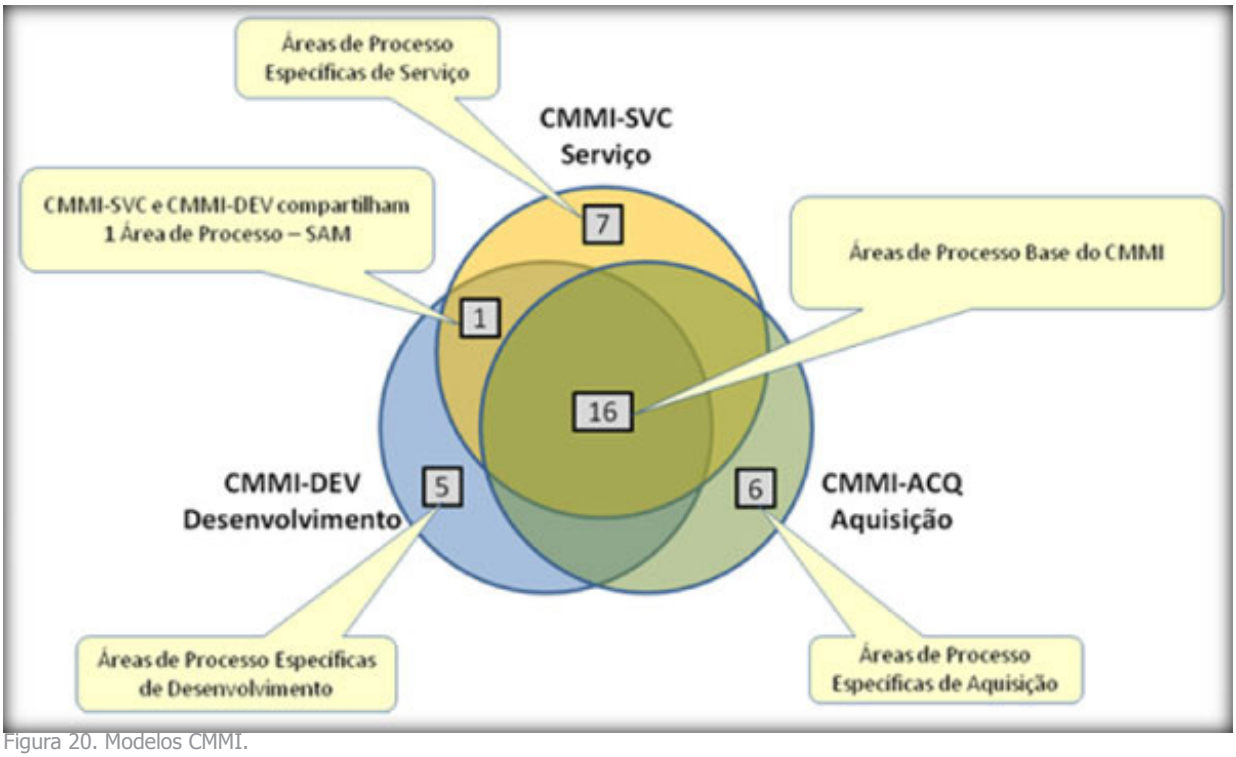


Figura 20. Modelos CMMI.
Fonte: http://www.teclogica.com.br/blog/?p=508.

A organização pode usar o modelo CMMI para ajudar a estabelecer objetivos e prioridades do melhoramento de processos, obtendo um guia para garantir estabilidade, processos estáveis e maduros. Mas o CMMI não é um modelo simples de ser implantado, pois exige uma mudança de cultura voltada para o planejamento, a qualidade e o controle dos processos de desenvolvimento dos softwares.

O CMMI possui duas representações: contínua ou por estágios. Estas representações permitem

à organização utilizar diferentes caminhos para a melhoria de seus processos de acordo com seu interesse. Estas representações serão detalhadas a seguir.

15.2 Representação contínua e por estágios

O propósito do CMMI é fornecer um guia para melhorar processos de organizações e sua habilidade de gerenciar o desenvolvimento, aquisição e manutenção de produtos ou serviços de software. O CMMI, por meio de sua estrutura, ajuda a organização a avaliar sua maturidade organizacional ou sua capacidade na área de processos, estabelecendo prioridades para melhoramentos e sua implementação.

Como vimos, o modelo CMMI apresenta dois caminhos a serem seguidos:

- **Contínuo:** permite que a organização evolua de forma incremental aos processos correspondentes a uma área de processo (*Process Area* – PA) (individual) ou a um grupo de área de processos selecionado pela empresa.
- **Por estágios:** a evolução é feita em um grupo de processos relacionados que são endereçados ao se implementar grupos de áreas de processos pré-determinados sucessivos.

Para a representação contínua, usa-se o termo *nível de capacidade* ou ainda capacidade da área de processo, sendo que existem seis níveis de capacidade. Um nível de capacidade está relacionado a apenas uma área de processo. Exemplo: nível de capacidade 3 na área de planejamento de projetos.

Para a representação por estágios, usa-se o termo nível de maturidade. São cinco os níveis de maturidade. Um nível de maturidade está relacionado a um grupo de áreas de processo. Exemplo: nível de maturidade 2 significa que a empresa implementou as práticas das áreas de processo (PP, PMC, REQM, SAM, MA, PPQA e CM).

A Figura 21 mostra os dois tipos de representação. A coluna da esquerda é a representação contínua e a da direita é a representação por estágios. A primeira diferença que vemos é a quantidade de níveis: na contínua são seis níveis (de capacidade) e a contagem começa do nível 0, enquanto na representação por estágios são cinco níveis (de maturidade) e começa-se do nível 1. Os níveis da esquerda (contínua) representam a capacidade de um processo, ao passo que os da direita (por estágios) representam a maturidade da organização.

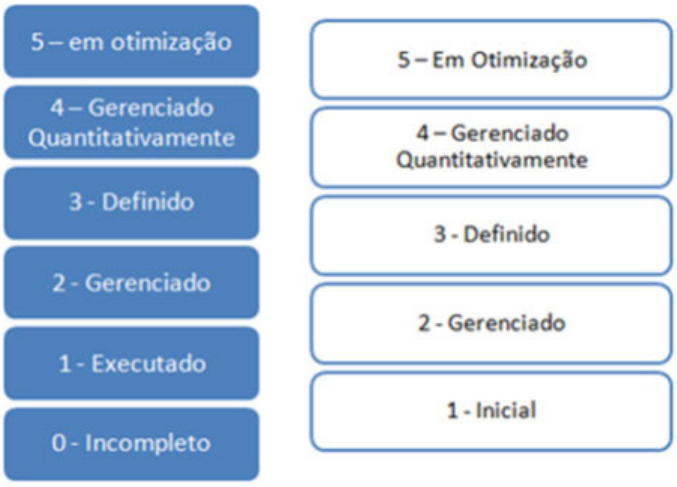


Figura 21. Níveis de capacidade (contínua) e de maturidade (por estágios)
Fonte: http://tudoqueeugostoeoutrosassuntos.blogspot.com.br/2012/07/cmmi-comparacao-dos-niveis-das.html.

Assim, os níveis de maturidade são usados para caracterizar a melhoria organizacional relativa a um conjunto de áreas de processos, enquanto os níveis de capacidade caracterizam a melhoria organizacional relativa a uma única área de processo. Mas, caro aluno, é importante frisar que são duas formas de se enxergar a mesma coisa. Independente da representação adotada, os níveis caracterizam a melhoria e a evolução de um estado desorganizado ou imaturo até um estado que usa informações quantitativas para determinar e gerenciar as melhorias a serem implementadas e que irão satisfazer as necessidades de negócios da organização.

Há muitas razões para se selecionar uma representação ou outra. Talvez a organização escolha usar a representação com qual é mais familiarizada. Se usados para melhoria de processos ou avaliações, ambas as representações são projetadas para oferecer resultados equivalentes.

15.3 Níveis de maturidade

O nível de maturidade de uma organização oferece uma maneira de prever seu desempenho futuro. Conforme mostram as experiências, as organizações têm resultados melhores quando direcionam seus esforços de melhoramento de processos em um número gerenciável de áreas de processo, que demandam um esforço crescente com o aperfeiçoamento da organização.

Um nível de maturidade pode ser concebido como um patamar evolutivo de processo de melhoramento. Dessa forma, cada nível de maturidade estabelece uma parte importante dos processos da organização. Os níveis de maturidade são medidos pelo alcance de metas específicas e genéricas que aplicam para cada conjunto predefinido de áreas de processo.

De acordo Quintella, Rocha e Motta (2005, p. 6-7), os cincio níveis de maturidade do CMMI podem ser descritos como se segue.

Nível de maturidade 1: inicial

Poucos processos são definidos e o sucesso depende de esforços individuais. A organização geralmente não utiliza práticas de gestão bem estabelecidas, implicando no desenvolvimento de produtos com planejamento ineficiente e sistemas nos quais os compromissos são reativos.

Nível de maturidade 2: gerenciado

São estabelecidos processos básicos de gerenciamento de projeto para controle de custos, prazos e escopo, permitindo que projetos anteriores bem sucedidos possam ser repetidos em aplicações similares. O *status* dos produtos e serviços são visíveis para a gerência em pontos específicos (*milestones*). As práticas existentes adotadas pela organização são mantidas durante os momentos de crise, assegurando que os projetos sejam executados e gerenciados conforme os planos documentados. Compromissos são estabelecidos entre os *stakeholders*, quando necessário, e os produtos são por eles revistos para a validação de requisitos, padrões e objetivos.

Nível de maturidade 3: definido

Um processo de Engenharia de Software é documentado, padronizado e integrado em um processo padrão da organização. Todos os projetos utilizam uma versão aprovada e adaptada deste processo organizacional para desenvolvimento e manutenção de produtos e serviços. Os processos são bem caracterizados e compreendidos, sendo descritos conforme padrões, procedimentos, ferramentas e métodos, além de serem melhorados ao longo do tempo. A gerência da organização estabelece objetivos de processo e busca assegurar que esses objetivos sejam seguidos de forma apropriada. Um programa de treinamento é implementado para garantir que colaboradores e os gerentes obtenham conhecimentos e habilidades necessárias para cumprir seus papéis.

Nível de maturidade 4: gerenciado

Tanto os processos como os projetos são quantitativamente medidos e controlados. A organização, além de estabelecer metas quantitativas de qualidade para os produtos e performance de processos, as utiliza como critério de gerenciamento. A produtividade e a qualidade são medidas para as atividades importantes do processo gerenciamento em todos os projetos, com os processos instrumentalizados com medições consistentes e bem definidas. Os projetos conseguem o controle sobre seus produtos e processos, reduzindo a variação no desempenho dos seus processos, que recaem em limites aceitáveis. Os riscos envolvidos na introdução de um novo domínio de aplicação são conhecidos e gerenciados com cuidado.

Nível de maturidade 5: otimizado

A organização inteira está com foco na melhoria contínua da performance de processo, tanto por melhoria incremental como por inovações tecnológicas. Objetivos mensuráveis de melhoria de processos são estabelecidos, continuamente revisados para refletir mudanças nos objetivos de negócio e utilizados como critério para a melhoria do processo de gerenciamento. São realizadas

análises de custo/benefício das novas tecnologias e das mudanças propostas com base nos dados sobre a efetividade dos processos. Lições aprendidas em projetos bem sucedidos e causas de fracassos são disseminadas para outros projetos. A participação e empowerment da força de trabalho, alinhada com os objetivos e valores da organização e seus negócios, impulsiona a otimização de processos, que passam a ser velozes e inovadores. A melhoria de processos passa a ser parte da atividade de todos, levando a um ciclo de melhoria contínua.

Caro aluno, assista à animação sobre CMMI, disponível em: http://www.youtube.com/watch?v=kF8sxDDoRns&feature=youtu.be

Aula 16 – O modelo de qualidade MPS.BR

16.1 Visão geral do modelo MPS.BR

O **MPS.BR** (**M**elhoria de **P**rocesso de **S**oftware.**Br**asileiro) foi criado em dezembro de 2003, coordenado pela SOFTEX (Associação para Promoção da Excelência do Software Brasileiro). O programa tem por objetivo a melhoria de processo para o desenvolvimento de software brasileiro, visando aumentar a competitividade da indústria brasileira de software nos mercados interno e externo, por meio de programas de qualificação de profissionais nessa área e de melhoria e avaliação de processos e produtos de software brasileiros a um custo acessível às empresas de menor porte.

O MPS.BR conta com a participação de representantes de universidades, instituições governamentais, centros de pesquisa e de organizações privadas, os quais contribuem com suas visões complementares que agregam qualidade ao empreendimento. Além disso, o MPS.BR recebe investimentos de empresas privadas e tem o apoio do Ministério da Ciência, Tecnologia e Inovação (MCTI), Financiadora de Estudos e Projetos (FINEP), Serviço Brasileiro de Apoio às Micro e Pequenas Empresas (SEBRAE) e Banco Interamericano de Desenvolvimento (BID/FUMIN).

Segundo o SOFTEX (2012), as iniciativas desse programa buscam que ele seja adequado ao perfil de empresas com diferentes tamanhos e características, públicas e privadas, embora com especial atenção às micro, pequenas e médias empresas. Adicionalmente, outro objetivo do projeto é replicar o modelo na América Latina, incluindo o Chile, Argentina, Costa Rica, Peru e Uruguai.

O modelo MPS segue os modelos e normas internacionais mais aceitos no mercado. Está em conformidade com as normas internacionais ISO/IEC 12207 e ISO/IEC 15504, é compatível com o modelo CMMI, baseado nas melhores práticas da engenharia de software e é adequado à realidade das empresas brasileiras. Assim, o modelo é compatível com os padrões de qualidade aceitos internacionalmente e tem como pressuposto o aproveitamento de toda a competência existente nos padrões e modelos de melhoria de processo já disponíveis.

O modelo MPS, conforme definido pelo Softex (2012), baseia-se nos conceitos de maturidade e capacidade de processo para a avaliação, melhoria da qualidade e produtividade de software e serviços correlatos, bem como para a melhoria da qualidade e produtividade dos serviços prestados. Dentro desse contexto, o modelo MPS possui quatro componentes:

- Modelo de referência MPS para software (MR-MPS-SW);
- MPS para serviços (MR-MPS-SV);
- Método de avaliação (MA-MPS);
- Modelo de negócio para melhoria de processo de sotware e serviços.

Cada componente é descrito por meio de guias e/ou documentos do modelo MPS, como pode ser visto na Figura 22.

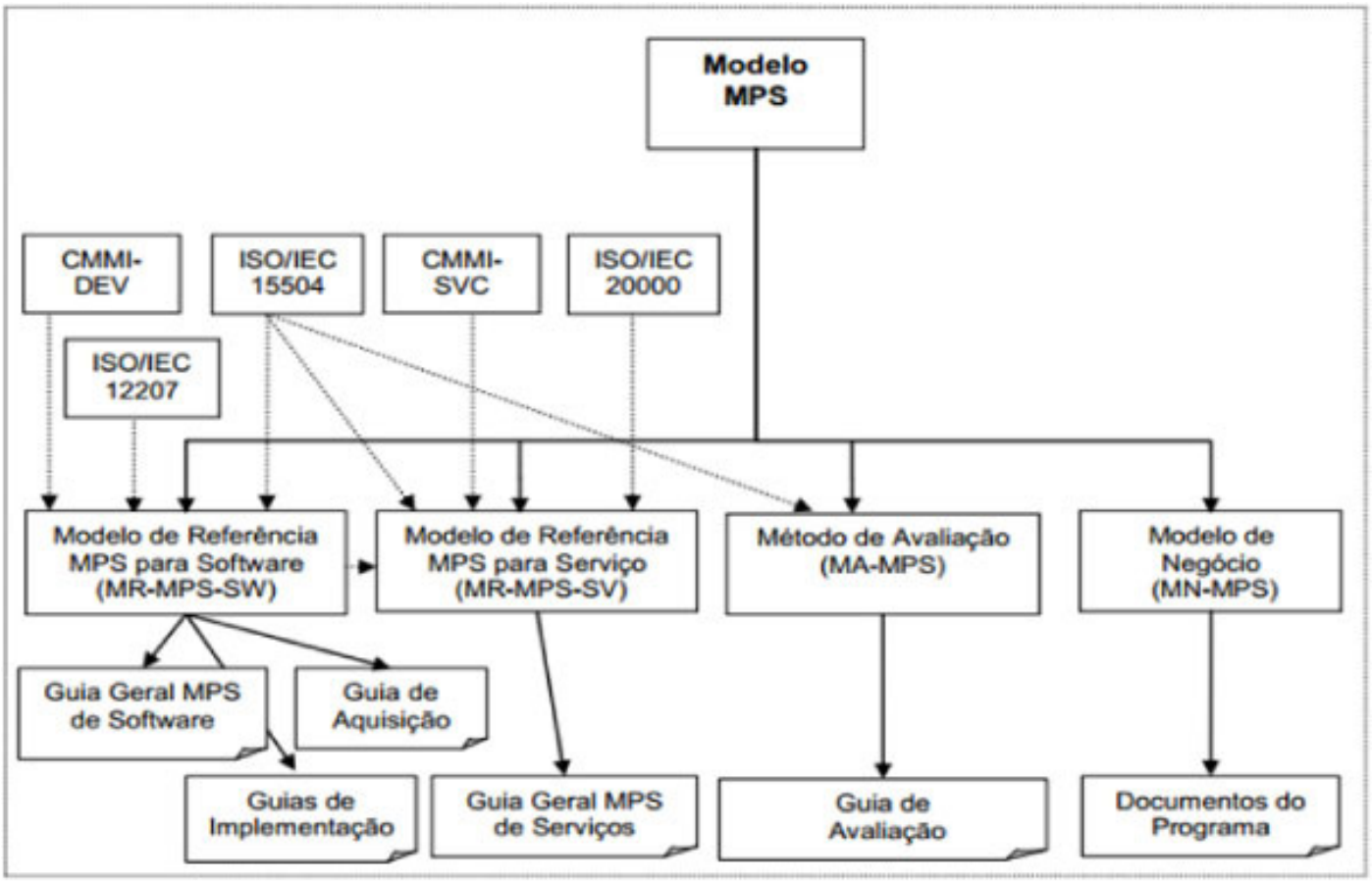


Figura 22. Modelo MPS.
Fonte: Softex (2012).

16.2 Níveis de maturidade do MPS.BR

De acordo com o Softex (2012), o modelo de referência MR-MPS do MPS.BR *define níveis de maturidade* que são uma combinação entre processos e sua capacidade, declarando o propósito e os resultados esperados de sua execução. Os níveis de maturidade estabelecem patamares de evolução de processos.

O nível de maturidade em que se encontra uma organização permite prever o seu desempenho futuro ao executar um ou mais processos. O MR-MPS define sete níveis de maturidade, a saber:

- [7] Nível A – Em otimização.
- [6] Nível B – Gerenciado quantitativamente.
- [5] Nível C – Definido.
- [4] Nível D – Largamente definido.
- [3] Nível E – Parcialmente definido.
- [2] Nível F – Gerenciado.
- [1] Nível G – Parcialmente gerenciado.

No modelo MPS.BR um nível é alcançado quando os propósitos e todos os resultados esperados dos respectivos processos são atendidos. Os níveis são acumulativos, ou seja, se a organização se encontra no nível F, isso significa que a mesma já passou pelo nível G, e os processos do nível G estão sendo executados junto com os processos do nível F. O nível mais alto é o nível A (em otimização), logo, o nível mais baixo é o nível G (parcialmente gerenciado).

Os processos do MR-MPS são descritos em termos de propósitos e resultados:

- O *propósito* descreve o objetivo geral que será atendido durante a execução do processo.
- Os *resultados* esperados do processo instituem os resultados a serem obtidos com a implementação efetiva do processo.

Na sequência, apresentamos o que é verificado em cada um dos sete níveis de maturidade do modelo MPS.BR, de acordo com o Softex (2012).

Nível G – parcialmente gerenciado

O nível de maturidade G é composto pelos processos gerência de projeto e gerência de requisitos.

Nível F – gerenciado

O nível F é composto pelo nível anterior (G) acrescido dos processos aquisição, gerência de configuração, garantia da qualidade e medição. Esse nível agrega processos que irão apoiar a gestão do projeto no que diz respeito à garantia da qualidade e medição, bem como àqueles que irão organizar os artefatos de trabalho por meio da gerência de configuração.

Nível E – parcialmente definido

O nível E é composto pelos processos dos níveis de maturidade anteriores (G e F), acrescido dos processos avaliação e melhoria do processo organizacional, definição do processo organizacional, gerência de recursos humanos e gerência de reutilização. O processo gerência de projetos sofre sua primeira evolução retratando seu novo propósito: gerenciar o projeto com base no processo definido para o projeto e nos planos integrados. Essa evolução se deu devido ao escopo do nível E do MPS.BR exigir maior consistência em relação aos processos-padrão da organização, pois até o nível F do modelo não há necessidade de os projetos executarem processos padronizados na organização como um todo.

Nível D – largamente definido

O nível de maturidade D é composto pelos processos de todos os níveis de maturidade que o antecedem (G ao E), acrescido dos processos desenvolvimento de requisitos, integração do produto, projeto e construção do produto, validação, e verificação.

Nível C – definido

O nível de maturidade C é composto por todos os processos dos níveis de maturidade anteriores (G ao D), acrescidos dos processos análise de decisão e resolução, desenvolvimento para reutilização e gerência de riscos.

Nível B – gerenciado quantitativamente

Esse nível de maturidade é composto pelos processos dos níveis de maturidade anteriores (G ao C), sendo que ao processo gerência de projetos são acrescentados novos resultados. A partir do nível B, a organização/unidade organizacional passa a ter uma visão quantitativa do desempenho de seus processos no apoio ao alcance dos objetivos de qualidade e de desempenho dos processos. É importante considerar que, ao se implementar os níveis anteriores, em especial o processo medição, já se deve preparar o caminho para a implementação do nível B, por meio de uma escolha adequada das medidas.

É a partir do nível de maturidade que a organização passa a gerenciar quantitativamente os projetos. Desse modo, o processo gerência de projetos passa a ser executado de forma quantitativa e alguns de seus resultados esperados são modificados para ter este enfoque, de forma que novos resultados serão acrescentados.

Nível A – em otimização

O nível de maturidade A é composto pelos níveis G, F, E, D, C e B acrescido do processo análise de causas de problemas e resolução.

16.3 MPS.BR versus CMMI

O modelo CMMI é proprietário e envolve um grande custo para a realização das avaliações do modelo para se obter a certificação. Geralmente o custo fica entre R$300 mil a mais de R$1 milhão, dependendo da complexidade do processo. Além disso, o processo é longo: geralmente, para chegar aos níveis de maturidade mais altos leva-se em média de 4 a 8 anos. (OLIVEIRA, 2008)

Essas dificuldades contrastam com a realidade das empresas brasileiras que não podem realizar um investimento tão alto para a obtenção da certificação. O alto custo da adaptação para obtenção da certificação e o longo prazo para alcançar os níveis mais altos de maturidade impossibilitavam as pequenas e médias empresas desenvolvedoras de software a aderirem ao programa do CMMI.

O MPS.BR surgiu como um movimento cujo objetivo era suprir a demanda das empresas

nacionais, que precisavam encontrar uma forma de adaptar à sua realidade, rapidamente, modelos para melhoria de processos de software como o CMMI níveis 2 e 3, a um custo mais acessível.

Os dois modelos apresentam níveis de maturidade que definem a capacidade da empresa para trabalhar com projetos grandes e complexos. Como vimos, o CMMI varia do 1 ao 5 e o MPS.BR varia do G ao A. Ao contrário do CMMI, o primeiro nível do MPS.BR já exige que a empresa tenha determinados processos definidos (ITS, 2014).

Os níveis do MPS.BR também são compostos por áreas de processos, a saber, os tópicos mais importantes para um processo de desenvolvimento de software. Assim, podemos criar uma equivalência dos níveis de maturidade do CMMI e do MPS.BR. Essa equivalência pode ser vista na Tabela 2 e melhor visualizada na Figura 23.

Tabela 2. Equivalência entre os níveis de maturidade CMMI x MPS.BR.

Comparação dos níveis de maturidade	
CMMI	**MPS.BR**
1	Não é definido
2	G
	F
3	E
	D
	C
4	B
5	A

Fonte: acervo pessoal.

Analisando a Tabela 2, podemos verificar que os níveis do MPS.BR permitem que a empresa implante processos de uma forma mais gradual. Essa estratégia aplicada ao mercado brasileiro de software permite que empresas de pequeno porte, que não possuem muito dinheiro para investir em metodologias e processos, possam tomar a iniciativa de definir processos.

O MPS.BR e o CMMI possuem níveis equivalentes de qualidade de software, mas a norma brasileira tem a vantagem de ser muito mais barata, além de existir financiamento do BID para grupos de empresas que desejem se certificar (ITS, 2014).

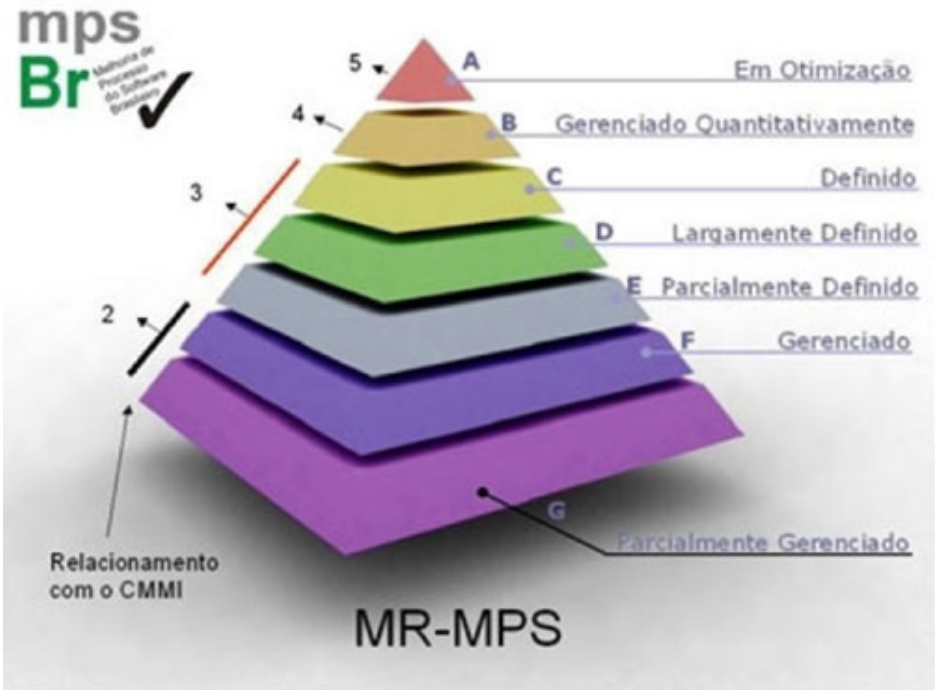


Figura 23. Níveis de maturidade do modelo MPS.BR e CMMI.
Fonte: http://bettawork.com.br/news/pagNoticia_6.html.

Uma experiência interessante já bastante comum entre as empresas brasileiras é que algumas delas utilizam o modelo MPS.BR como forma de se prepararem melhor para alcançar um nível do modelo CMMI, já que há equivalência entre os níveis do CMMI e do MPS.BR. Além disso, o custo de um processo de implantação do MPS.BR é menor que o do CMMI, e esse é um dos principais incentivos para algumas empresas optarem por isso. Uma empresa pode alcançar o nível A do MPS.BR e depois já tentar o nível 5 do CMMI.

Exercícios do Módulo 5

1) Qual destas não é uma característica para a qualidade de software?

a) Está fortemente relacionada à conformidade com os requisitos.
b) Caracteriza o grau de satisfação do cliente.
c) É responsabilidade de apenas uma área da empresa: a área de qualidade.
d) Deve estar presente desde o planejamento do software.

2) Assinale V-verdadeiro e F-falso:

( ) A qualidade de software ocorre quando um conjunto de características desejáveis, definido pela indústria, é incorporado em um produto, de modo a aprimorar seu desempenho durante sua existência.
( ) Pode-se afirmar que o teste de software é uma das atividades de controle da qualidade, ou seja, o teste de software é orientado ao produto e está dentro do domínio do controle da qualidade.
( ) Um dos modelos mais recentes de qualidade de software é o de James A. McCall, conhecido como fatores da qualidade, que avaliam o software em três pontos distintos: operação, transição e revisão do produto.
( ) Qualidade total é a preocupação com a qualidade em todas as atividades da empresa buscando sistematicamente o nível "zero defeito", por meio da melhoria contínua dos processos de produção.

3) Analise as afirmativas e assinale a resposta correta:

i. A avaliação de produtos de software é definida como uma operação técnica que consiste em elaborar um julgamento de uma ou mais características de um produto de software de acordo com um procedimento definido.
ii. As normas da ISO mais conhecidas que abordam a qualidade de produto de software são a norma ISO/IEC 12207 e a norma ISO/IEC 15504.
iii. O ANSI (American National Standards Institute) é o representante ISO dos Estados Unidos, e no Brasil a ISO é representada pela ABNT (Associação Brasileira de Normas Técnicas).

a) Apenas i e ii estão corretas.
b) Apenas ii e iii estão corretas.
c) Apenas i e iii estão corretas.
d) Todas estão corretas.

4) Assinale V-verdadeiro e F-falso:

( ) Uma distinção entre o nível 4 e o nível 5 do CMMI é o tipo de variação de processo com que se lida. No nível 4, processos ocupam-se com causas especiais de variação e fornecem estatística previsível de resultados. Já no nível 5 a preocupação é que os processos lidem com causas comuns de variação e mudem o processo para melhorar a performance, a fim de alcançar os objetivos quantitativos estabelecidos e obter o melhoramento do processo.
( ) Organizações em níveis de maturidade CMMI 2 geralmente produzem produtos e serviços que funcionam; entretanto, eles frequentemente excedem o prazo e o orçamento de seus projetos.
( ) No nível de maturidade CMMI 3, os processos são bem caracterizados e entendidos, sendo descritos como padrões, procedimentos, ferramentas e métodos. O conjunto de padrões de processos da organização é estabelecido e melhorado continuamente.

5) Complete as frases sobre representação no modelo CMMI:

Para a representação contínua, usa-se o termo _____ ou ainda _____ da área de processo. Ou seja, um _____ está relacionado a apenas uma área de processo.

Para a representação _____, usa-se o termo _____ ou ainda _____ da organização. Ou seja, um _____ está relacionado a um grupo de áreas de processo.

6) Assinale V-verdadeiro e F-falso nas afirmações sobre o MPS.BR:

( ) No modelo MPS.BR, um nível é alcançado quando os propósitos e todos os resultados esperados dos respectivos processos são atendidos. Os níveis são acumulativos.

( ) O nível mais baixo do MPS.BR é o nível A; logo, o nível mais alto é o nível G.

( ) O nível de maturidade A é composto pelos níveis G, F, E, D, C e B, acrescido do processo análise de causas de problemas e resolução.

( ) Adicionalmente, outro objetivo do MPS.BR é replicar o modelo na América do Norte, incluindo os Estados Unidos e Canadá, que apoiam o projeto.

( ) O modelo MPS segue os modelos e normas internacionais mais aceitos no mercado. Está em conformidade com as normas internacionais ISO/IEC 12207 e ISO/IEC 15504 e é compatível com o modelo CMMI.

( ) Apesar do MPS.BR ser compatível com os padrões de qualidade aceitos internacionalmente, não se compromete em aproveitar a competência existente nos padrões e modelos de melhoria de processo já disponíveis.

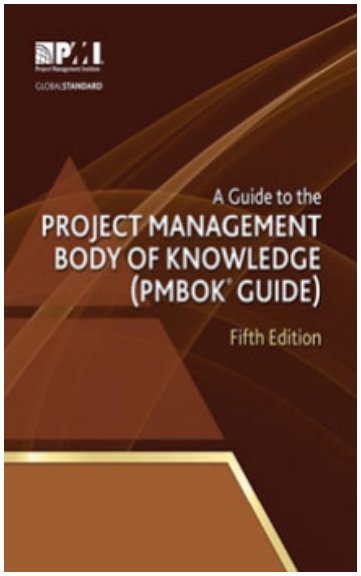
( ) O modelo MPS baseia-se nos conceitos de maturidade e capacidade de processo para a avaliação e melhoria da qualidade e produtividade de produtos de software e serviços correlatos.

MÓDULO 6 – O GERENCIAMENTO DE PROJETOS – PMBOK

Neste módulo abordaremos o Gerenciamento de projetos – PMBoK, com base no *Guide to Project Management Body of Knowledge* (PMBOK® Guia), publicado em 2012. O principal objetivo deste módulo é apresentar os conceitos básicos sobre a gestão de projetos e a sua aplicação no desenvolvimento de projetos de software.

Aula 17 – Conceitos básicos do gerenciamento de projetos

17.1 O Guia PMBoK – *Project Management Body of Knowledge*



Fonte: http://marketplace.pmi.org/Pages

O guia PMBoK apresenta conhecimentos sobre o gerenciamento de projetos e é editado pelo PMI (*Project Management Institute*) desde 1983, com o intuito de documentar os processos de gerenciamento de projetos. Atualmente, o guia PMBoK está na quinta edição (publicado em 2013) e apresenta atualizações importantes para a sua melhoria e qualidade. Destas, podemos destacar:

- Acréscimo de cinco processos, totalizando 47;
- Alteração de nomes de processos;
- Inclusão de uma nova área de conhecimento: "gerenciamento de partes interessadas", com quatro processos;
- Aumento do número de páginas;
- Acréscimo do padrão ISO 21500 ao apêndice;
- Remanejamento de área de dois processos.

O guia PMBoK é reconhecido internacionalmente e utilizado por muitos gerentes de projetos nas mais diversas organizações. As informações contidas nesse guia garantem melhor qualidade dos resultados.

De acordo com o PMI, o gerenciamento de projetos "é a aplicação de conhecimentos, habilidades e técnicas para a execução de projetos de forma efetiva e eficaz. Trata-se de uma competência estratégica para organizações, permitindo que elas unam os resultados dos projetos com os objetivos do negócio — e, assim, competindo melhor em seus mercados".

Fontes: http://marketplace.pmi.org/Pages
http://brasil.pmi.org/brazil/AboutUS/WhatIsProjectManagement.aspx

17.2 O gerente de projetos e os stakeholders

De acordo com o PMI (2008), um projeto é um empreendimento temporário, progressivo e executado para criar um produto ou serviço único. Seu gerenciamento é de responsabilidade de um gerente de projeto individual que mantém o andamento e a execução dos elementos envolvidos, minimizando riscos do projeto.

O profissional que lidera os projetos numa organização é chamado de gerente de projetos. O gerente de projetos é o responsável pelo planejamento e acompanhamento dos projetos até a sua finalização. Segundo o guia PMBoK (PMI, 2008), esse profissional deve ter as seguintes competências:

- Conjunto de conhecimentos em gerenciamento de projetos;
- Conhecimentos e habilidades de gerenciamento geral;
- Entendimento do ambiente do projeto;
- Habilidade interpessoais;
- Conhecimentos, normas e regulamentos da área de aplicação.

Assista à reportagem Profissão gerente de projetos, disponível em: http://youtu.be/NU5EmJIbDLA

Conforme descrito no Módulo 3, item 8, as partes interessadas do projeto (*stakeholders*) são pessoas e organizações que possuem interesses envolvidos no projeto, ou que podem ser afetados com o resultado e exercer algum tipo de influência no projeto.

De acordo com o PMBoK, as principais partes interessadas e presentes nos projetos são: clientes/usuários; membros da equipe do projeto; equipes de gerenciamento de projetos; gerentes de projetos; organização executora;, patrocinador (que fornece os recursos financeiros); influenciadores (pessoas ou grupos que não estão diretamente relacionados ao projeto, mas que, devido à posição na organização, podem influenciar o projeto) e o escritório de projetos.

17.3 O ciclo de vida de um projeto

O ciclo de vida determina as fases de desenvolvimento de um projeto e tem como principal objetivo garantir a qualidade dos seus resultados.

No Módulo 2, item 3, vimos a definição do ciclo de vida de desenvolvimento do software e sua importância. O gerenciamento de projetos está contido no ciclo de vida do software e permeia todo o processo de desenvolvimento.

Os ciclos de vida para o desenvolvimento de projetos são:

- Ciclo de vida preditivo – determina que todo o processo deve ser planejado com antecedência. É o ciclo de vida típico do gerenciamento tradicional;
- Ciclo de vida incremental e iterativo – as partes são geradas separadamente e integradas quando finalizadas. A cada integração das partes é realizada iterações, com o intuito de revisão e melhorias no projeto;
- Ciclo de vida adaptativo – é um ciclo de vida também incremental e iterativo. Entretanto, as iterações com custo e tempo são bem mais rápidas. Por isso, é conhecido como método ágil ou orientado a mudanças.

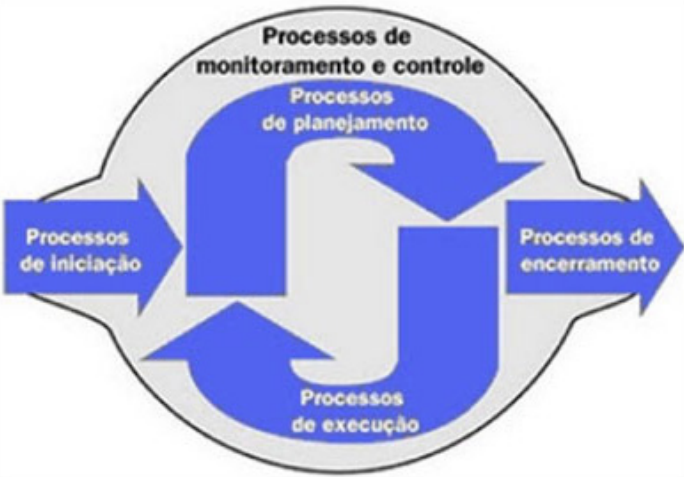
O ciclo de vida adaptativo é uma novidade na quinta edição do PMBoK e ressalta a importância e a utilização da abordagem dos métodos ágeis no gerenciamento de projetos. Trata-se de um ciclo de vida recomendável para ser utilizado em projetos de desenvolvimento de software.

17.4 Áreas do conhecimento e grupos de processos do PMBoK (5ª edição)

De acordo com PMI (2008), um processo é um conjunto de ações e atividades inter-relacionadas que são executadas para alcançar um objetivo. Cada processo é caracterizado por suas entradas, as ferramentas e as técnicas que podem ser aplicadas, além das saídas resultantes.

O PMBoK organiza os processos do gerenciamento de projetos em cinco grupos de processos. Não houve alteração dos grupos de processos na quinta edição do PMBoK. Assim, os grupos de processos são:

- Grupo de processos de iniciação: define e autoriza o projeto ou fase;
- Grupo de processos de planejamento: detalha e planeja o projeto;
- Grupo de processos de execução: executa o plano de projeto e reúne pessoas e recursos;
- Grupo de processos de monitoramento e controle: controla as mudanças, monitora e acompanha os riscos e o desempenho do projeto;
- Grupo de processos de encerramento: encerra o projeto, aquisições e libera os recursos.



Fonte: http://tiinteligente.blogspot.com.br/2010/06/PMBoK-grupos-de-processos-conceito.html

Os grupos de processos não são as fases do projeto, mas apenas um agrupamento dos processos que pertencem ao PMBoK. Cabe ao líder do projeto analisar cada um dos processos, de acordo com as necessidades do projeto, e aplicá-los na sua gestão.

Caro aluno,

No link http://dicasgp.pmtech.com.br/planilha-processos-pmbok5/ você encontrará uma planilha, elaborada por Mauro Sotille, com um resumo dos 47 processos e um detalhamento dos processos de cada uma das 10 áreas do conhecimento. É uma planilha utilizada por alunos que se preparam para os exames de certificação PMP® e CAPM®. Além dessa planilha, também está disponível um resumo (tabela) da integração entre as áreas de conhecimento e os grupos de processos.

Vale a pena conferir!

O guia PMBoK é organizado em áreas de conhecimento e, por sua vez, cada área de conhecimento é descrita por processos.

A 5ª edição do PMBoK acrescentou uma nova área de conhecimento, chamada "gerenciamento das partes interessadas", às nove áreas de conhecimento que caracterizam os principais aspectos envolvidos na gestão de projetos. Assim, a nova composição das áreas de conhecimento é formada por: integração, escopo, tempo, custos, qualidade, recursos humanos, comunicações, riscos, aquisições e partes interessadas (stakeholders).

A Tabela 3 apresenta as dez áreas de conhecimento e os seus respectivos processos e grupos de processos. Na última coluna encontra-se um breve resumo de cada uma das áreas de conhecimento:

Tabela 3. Áreas de Conhecimento – PMBoK (5ª Edição).

Áreas de conhecimento	Processos	Grupos de processo	Resumo
1. Gerenciamento de integração	1. Desenvolver o termo de abertura do projeto	Iniciação	Identificar, definir, combinar, unificar e coordenar os diversos processos e atividades de gerenciamento de projetos, que são essenciais para atender às necessidades dos clientes e de outras partes interessadas
	2. Desenvolver o plano de gerenciamento do projeto	Planejamento	
	3. Orientar e gerenciar a execução do projeto	Execução	
	4. Monitorar e controlar o trabalho do projeto	Monitoramento e controle	
	5. Realizar o controle integrado de mudanças	Monitoramento e controle	
	6. Encerrar o projeto ou fase	Encerramento	
2. Gerenciamento do escopo	7. Planejar o gerenciamento do escopo	Planejamento	Definir e controlar todos os processos necessários que devem estar ou não no projeto, com o intuito de garantir o sucesso do mesmo
	8. Coletar os requisitos		
	9. Definir o escopo		
	10. Criar a EAP		
	11. Validar o escopo	Monitoramento e controle	
	12. Controlar o escopo		

3. Gerenciamento de tempo	13. Planejar o gerenciamento do cronograma	Planejamento	Incluir os processos necessários para o cumprimento do término do projeto dentro do prazo estipulado
	14. Definir as atividades		
	15. Sequenciar as atividades		
	16. Estimar os recursos das atividades		
	17. Estimar as durações das atividades		
	18. Desenvolver o cronograma		
	19. Controlar o cronograma	Monitoramento e controle	
Áreas de conhecimento	**Processos**	**Grupos de processo**	**Resumo**
4. Gerenciamento de custos	20. Planejar o gerenciamento dos custos	Planejamento	Planejar, estimar, orçar e controlar os custos a fim de encerrar o projeto dentro do orçamento aprovado
	21. Estimar os custos		
	22. Determinar o orçamento		
	23. Controlar os custos	Monitoramento e controle	
5. Gerenciamento da qualidade	24. Planejar o gerenciamento da qualidade	Planejamento	Garantir a qualidade do projeto e a melhoria contínua dos processos do início ao fim
	25. Realizar a garantia da qualidade	Execução	
	26. Controlar a qualidade	Monitoramento e controle	
6. Gerenciamento de Recursos Humanos	27. Planejar o gerenciamento dos Recursos Humanos	Planejamento	Organizar e gerenciar a equipe do projeto
	28. Mobilizar a equipe do projeto	Execução	
	29. Desenvolver a equipe do projeto		
	30. Gerenciar a equipe do projeto		
7. Gerenciamento das comunicações	31. Planejar o gerenciamento das comunicações	Planejamento	Planejar, gerenciar e controlar as comunicações de forma adequada e bem-sucedida
	32. Gerenciar as comunicações	Execução	
	33. Controlar as comunicações	Monitoramento e controle	

8. Gerenciamento de riscos	34. Planejar o gerenciamento de riscos	Planejamento	Aumentar a probabilidade e o impacto dos eventos positivos e diminuir a probabilidade e o impacto dos eventos adversos
	35. Identificar os riscos		
	36. Realizar a análise qualitativa dos riscos		
	37. Realizar a análise quantitativa dos riscos		
	38. Planejar as respostas aos riscos		
	39. Monitorar e controlar os riscos	Monitoramento e controle	
Áreas de conhecimento	**Processos**	**Grupos de processo**	**Resumo**
9. Gerenciamento de aquisições	40. Planejar as aquisições	Planejamento	Gerenciar os processos de aquisições de produtos, serviços ou resultados para a realização do projeto
	41. Conduzir as aquisições	Execução	
	42. Administrar as aquisições	Monitoramento e controle	
	43. Encerrar as aquisições	Encerramento	
10. Gerenciamento das partes interessadas	44. Identificar as partes interessadas	Iniciação	Gerenciar os stakeholders durante todo o ciclo de vida do projeto
	45. Planejar o gerenciamento das partes interessadas	Planejamento	
	46. Gerenciar o engajamento das partes interessadas	Execução	
	47. Controlar o engajamento das partes interessadas	Monitoramento e controle	

Fonte: adaptado de PMI (2012).

Exercícios do Módulo 6

1) Os cinco grupos de processos do PMBoK são:

a) Custos, tempo, recursos, *stakeholders* e processos.
b) Requisitos, análise dos processos, modelagem de dados, testes e implantação.
c) Iniciação, planejamento, execução, monitoramento e controle e encerramento.
d) Processo de gestão, processo de planejamento, processo de controle, processo de execução e processo de finalização.

2) Os grupos de processos não são as fases do projeto.

a) Verdadeiro.
b) Falso.

3) Na quinta edição do PMBoK foi acrescida uma nova área de conhecimento, chamada de:

a) Escopo.
b) *Stakeholders*.
c) Integração.
d) Qualidade.

4) Sobre o PMBoK, é correto afirmar que "apresenta conhecimentos sobre o gerenciamento de projetos e foi realizado pelo PMI – Project Management Institute – desde 1983, com o intuito de documentar os processos de gerenciamento de projetos".

a) Verdadeiro.
b) Falso.

5) Leia as afirmações abaixo:

I - O ciclo de vida preditivo determina que os processos não necessitam ser realizados com antecedência.
II - Não há diferenças entre o ciclo de vida incremental e interativo com o ciclo de vida adaptativo.
III - O ciclo de vida incremental e o ciclo de vida interativo utilizam características da metodologia ágil.
IV - O ciclo de vida adaptativo é também conhecido como método ágil.

Está correta a afirmação:

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

6) Segundo o PMBoK, o gerente de projetos deve ter as seguintes competências: conhecer gerenciamento geral e gestão de projetos, ter habilidades interpessoais e habilidades de gerenciamento geral, entender do ambiente do projeto e conhecer as normas e regulamentos da área de aplicação.

a) Verdadeiro.
b) Falso.

7) Qual dos itens listados abaixo não se constitui uma área de conhecimento do PMBoK?

a) Comunicações.
b) Riscos.
c) Métodos.
d) Aquisições.

CONSIDERAÇÕES FINAIS

Esta disciplina introduziu muitos aspectos essenciais para a prática da engenharia de software. Vimos que o processo de desenvolvimento de software envolve atividades, recursos e produtos.

Vimos também que um modelo de processo é útil para orientar o comportamento quando você estiver trabalhando em grupo. Modelos de processo detalhados dizem como coordenar e colaborar com colegas enquanto se projeta e constrói um sistema. Os modelos também mostram a cada membro da equipe quais atividades ocorrem, quando e por quem elas são realizadas, de modo a tornar clara a divisão de tarefas.

Vimos que os requisitos podem ser modificados até mesmo enquanto analisamos o problema e desenvolvemos a solução, sendo que a solução deve estar bem documentada e ser flexível, devendo-se documentar as suposições e os algoritmos utilizados de tal modo que seja fácil mudá-los posteriormente.

Possivelmente, caro aluno, você fará grande parte do seu trabalho como membro de uma equipe de desenvolvimento maior. Como vimos nesta disciplina, o desenvolvimento de software envolve análise de requisitos, projeto, implementação, teste, gerência de configuração, garantia da qualidade, dentre outras atividades. Você e outras pessoas de sua equipe podem desempenhar vários papéis, e o sucesso do projeto depende em larga escala da comunicação e coordenação entre os membros da equipe.

Você pode ajudar no sucesso de seu projeto, selecionando um processo de desenvolvimento apropriado ao tamanho da equipe, ao nível de risco do projeto e ao domínio do aplicativo, bem como utilizando técnicas de testes, modelos de referência de qualidade como CMMI e MPS.BR, além de ferramentas bem integradas que apoiem o tipo de comunicação necessária ao seu projeto.

No andamento do nosso curso, vários aspectos fundamentais estudados nesta disciplina serão abordados, proporcionando a você oportunidades de colocar em prática os conceitos, métodos, técnicas e ferramentas da engenharia de software que auxiliam na obtenção de um produto de qualidade, respeitando os prazos, custos e escopo estabelecidos.

Profa. Elisamara

RESPOSTAS COMENTADAS DOS EXERCÍCIOS

Módulo 1

1) Qual das questões abaixo não é mais uma das grandes preocupações de um engenheiro de software?

d) Por que hardware é tão caro?

2) O software é um produto que pode ser manufaturado usando as mesmas tecnologias utilizadas para outros artefatos de engenharia.

b) Falso.

3) O software deteriora-se em vez de desgastar porque:

c) Mudanças frequentes aumentam a probabilidade de se introduzir erros no software.

4) Atividades "guarda-chuva" de engenharia de software são aplicadas somente durante as fases iniciais do desenvolvimento de software.

b) Falso

5) O que não está relacionado à chamada "crise do software"?

d) Essa crise teve como origem a introdução de computadores pessoais na década de 1970.

6) A essência da prática da engenharia de software pode ser resumida em compreender o problema, planejar a solução, executar o plano e examinar o resultado.

a) Verdadeiro

7) Qual dos itens listados abaixo não se constitui numa das camadas da engenharia de software?

b) Manufatura

8) O processo criativo do hardware gera algo *físico*. O desenvolvimento de software resulta em um sistema lógico, *intangível*. Os custos do software estão concentrados no *desenvolvimento* e não no processo de *manufatura*. Ao longo do tempo, o produto de software não se *desgasta*, mas se *deteriora* em função da introdução de erros oriundos de atividades de manutenção ou evoluções do sistema.

Módulo 2

1) Processos de software podem ser construídos a partir de modelos pré-existentes, objetivando adequar-se às necessidades de um determinado projeto.

a) Verdadeiro.

2) Os tipos de manutenção que um software pode sofrer são:

- Manutenção *preventiva*: faz modificações nos programas de modo que eles possam ser mais facilmente corrigidos, adaptados e melhorados.
- Manutenção *corretiva*: modifica o software para corrigir defeitos.
- Manutenção *de aperfeiçoamento*: aprimora o software além dos requisitos funcionais originais (cliente/usuário reconhece e solicita funcionalidades adicionais que trarão benefícios, à medida que o software é usado).
- Manutenção *adaptativa*: modifica o software para acomodar mudanças no seu ambiente externo (processador, sistema operacional, etc.).

3) Em relação ao modelo em cascata, podemos afirmar que ele é:

a) Uma abordagem razoável quando os requisitos estão bem estabelecidos.

4) Quais são as cinco atividades gerais da engenharia de software, também caracterizadas no modelo em cascata?

c) Especificação, planejamento, modelagem, construção, implantação (entrega).

5) Podemos dizer que o modelo incremental é:

b) Uma boa abordagem quando um produto de trabalho "núcleo" é requerido rapidamente.

6) Podemos afirmar que modelos de processo evolucionários:

d) Todas as alternativas anteriores estão corretas.

7) O modelo de prototipagem é:

c) Uma abordagem razoável quando o cliente não consegue definir os requisitos claramente.

8) O modelo espiral de desenvolvimento de software:

c) Inclui avaliação de riscos a cada iteração.

9) O modelo de desenvolvimento concorrente:

e) Somente a e b são verdadeiras.

10) Podemos dizer que o modelo de desenvolvimento baseado em componentes:

c) Depende da tecnologia de objetos como suporte.

11) O modelo de métodos formais faz uso de métodos matemáticos para:

d) Todas as alternativas anteriores são corretas.

12) Qual dos nomes abaixo não representa uma das fases do modelo de processo unificado?

b) Fase de validação.

13) Qual a diferença entre um modelo descritivo e um modelo prescritivo de processo?

Enquanto um modelo descritivo retrata como um processo é executado, um modelo prescritivo retrata como um processo deveria ser executado. Assim, um modelo prescritivo é uma recomendação que pode ser adaptada ou melhorada pela empresa/equipe de software que for adotá-la.

14) O que não caracteriza os modelos evolucionários de processo?

d) Os detalhes e extensões do projeto são definidos logo no início, antes do desenvolvimento.

O correto seria: os detalhes e extensões do projeto são definidos ao longo do desenvolvimento.

15) Assinale a alternativa FALSA:

b) O modelo de processo incremental é interativo como a prototipagem e ambos têm como objetivo apresentar um produto operacional a cada incremento realizado.

O correto seria: O modelo de processo incremental é interativo como a prototipagem, mas, diferentemente desta, o incremental tem como objetivo apresentar um produto operacional a cada incremento realizado.

16) Assinale a alternativa que não é característica do processo unificado:

a) é um framework de propriedade da IBM que é usado como um processo de desenvolvimento.

O correto seria: é um framework genérico de um processo de desenvolvimento.

17) Os princípios:

- Valorizar indivíduos (e a interação entre eles) mais do que processos e ferramentas;
- Valorizar o software em funcionamento mais do que a documentação abrangente;
- Valorizar a colaboração com o cliente mais do que a negociação de contratos;
- Valorizar responder a mudanças mais que seguir um plano.

Referem-se a *metodologias ágeis*.

18) Nos modelos de processos ágeis o único produto de trabalho gerado é o programa de trabalho.

b) Falso.

Módulo 3

1) Stakeholder é qualquer pessoa que vá comprar o software em desenvolvimento.

a) Falso.

2) É relativamente comum para diferentes usuários propor requisitos conflitantes, cada qual argumentando que sua versão é a mais adequada ou melhor.

a) Verdadeiro.

3) Na validação de requisitos, o modelo de requisitos é revisto para assegurar sua viabilidade técnica.

b) Falso.

4) Faça a associação e assinale a resposta correta sobre a técnica PIECES:

a) i-c ii-e iii-a iv-b v-f vi-d

5) Assinale a afirmativa ERRADA:

c) Requisitos não-funcionais: são requisitos que não dizem respeito diretamente à funcionalidade do sistema, mas expressam suas propriedades e/ou restrições sobre os serviços ou funções por ele providas. Exemplos de requisitos não funcionais: cadastro de cliente; emissão de nota fiscal; consulta ao estoque; geração de pedido.

O correto seria: Exemplos de requisitos funcionais: cadastro de cliente; emissão de nota fiscal; consulta ao estoque; geração de pedido.

6) Quais os benefícios esperados de um documento de especificação de requisitos?

- *Que estabeleça a base de acordo entre os clientes e a empresa fornecedora sobre o que o software irá fazer;*
- *Que reduza o esforço de desenvolvimento;*
- *Que forneça uma base para estimativas de custo e prazos;*
- *Que forneça uma base para validação e verificação do produto;*
- *Que facilite a manutenção do produto de software final.*

Módulo 4

1) Uma falha:

c) É um comportamento inconsistente.

2) Assinale a alternativa correta: um dos problemas relacionados a este teste é o buffer overflow, ou a necessidade de maior capacidade de armazenamento:

a) Segurança.

3) Assinale a alternativa correta: tem como objetivo revelar a presença de erros. Descobre o maior número possível deles, alguns dependendo do desenvolvimento previamente já identificado para comprovar a eficiência do teste.

b) Correção.

4) São programas ou scripts simples que exercitam funcionalidades do sistema sendo testado e fazem verificações automáticas nos efeitos colaterais obtidos.

e) Testes automatizados.

5) Podemos afirmar sobre testes de baixo nível:

d) Atuam na identificação de problemas em um pequeno segmento do código-fonte.

6) Alguns testes, por sua simplicidade, podem ser executados frequentemente, como os testes *de longevidade***. Há outros que devem ser programados, como os testes** *de segurança e desempenho***, os quais geram invariavelmente lentidão por causa das ferramentas que necessitam utilizar. Outros tipos de testes, como** *carga e estresse***, não precisam ser executados a o todo momento, pois avaliam a infraestrutura e esta normalmente não é alterada com frequência.**

Módulo 5

1) Qual destas não é uma característica para a qualidade de software?

c) É responsabilidade de apenas uma área da empresa: a área de qualidade.

O correto seria: é responsabilidade de todos.

2) Assinale V-verdadeiro e F-falso:

(*V*) A qualidade de software ocorre quando um conjunto de características desejáveis, definido pela indústria, é incorporado em um produto, de modo a aprimorar seu desempenho durante sua existência.
(*V*) Pode-se afirmar que o teste de software é uma das atividades de controle da qualidade, ou seja, o teste de software é orientado ao produto e está dentro do domínio do controle da qualidade.
(*F*) Um dos modelos mais recentes (o correto seria: um dos primeiros) de qualidade de software é o de James A. McCall, conhecido como fatores da qualidade, que avaliam o software em três pontos distintos: operação, transição e revisão do produto.
(*V*) Qualidade total é a preocupação com a qualidade em todas as atividades da empresa buscando sistematicamente o nível "zero defeito", por meio da melhoria contínua dos processos de produção.

3) Analise as afirmativas e assinale a resposta correta:

i. A avaliação de produtos de software é definida como uma operação técnica que consiste em elaborar um julgamento de uma ou mais características de um produto de software de acordo com um procedimento definido.
ii. As normas da ISO mais conhecidas que abordam a qualidade de produto (o correto seria: de processo) de software são a norma ISO/IEC 12207 e a norma ISO/IEC 15504.
iii. O ANSI (American National Standards Institute) é o representante ISO dos Estados Unidos, e no Brasil a ISO é representada pela ABNT (Associação Brasileira de Normas Técnicas).

c) Apenas i e iii estão corretas.

4) Assinale V-verdadeiro e F-falso:

(*V*) Uma distinção entre o nível 4 e o nível 5 do CMMI é o tipo de variação de processo com que se lida. No nível 4, processos ocupam-se com causas especiais de variação e fornecem estatística previsível de resultados. Já no nível 5 a preocupação é que os processos lidem com causas comuns de variação e mudem o processo para melhorar a performance, a fim de alcançar os objetivos quantitativos estabelecidos e obter o melhoramento do processo.
(*F*) Organizações em níveis de maturidade CMMI 2 (o correto seria: 1) geralmente produzem produtos e serviços que funcionam; entretanto, eles frequentemente excedem o prazo e o orçamento de seus projetos.
(*V*) No nível de maturidade CMMI 3, os processos são bem caracterizados e entendidos, sendo descritos como padrões, procedimentos, ferramentas e métodos. O conjunto de padrões de processos da organização é estabelecido e melhorado continuamente.

5) Complete as frases sobre representação no modelo CMMI:

Para a representação contínua, usa-se o termo nível de capacidade ou ainda capacidade da área de processo. Ou seja, um nível de capacidade está relacionado a apenas uma área de processo.

Para a representação por estágios, usa-se o termo nível de maturidade ou ainda maturidade da organização. Ou seja, um nível de maturidade está relacionado a um grupo de áreas de processo.

6) Assinale V-verdadeiro e F-falso nas afirmações sobre o MPS.BR:

(*V*) No modelo MPS.BR, um nível é alcançado quando os propósitos e todos os resultados esperados dos respectivos processos são atendidos. Os níveis são acumulativos.
(*F*) O nível mais baixo (o correto seria: alto) do MPS.BR é o nível A; logo, o nível mais alto (o correto seria: baixo) é o nível G.
(*V*) O nível de maturidade A é composto pelos níveis G, F, E, D, C e B, acrescido do processo análise de causas de problemas e resolução.
(*F*) Adicionalmente, outro objetivo do MPS.BR é replicar o modelo na América do Norte, incluindo os Estados Unidos e Canadá, que apoiam o projeto (o correto seria: América Latina).
(*V*) O modelo MPS segue os modelos e normas internacionais mais aceitos no mercado. Está em conformidade com as normas internacionais ISO/IEC 12207 e ISO/IEC 15504 e é compatível com o modelo CMMI.

(*F*) Apesar do MPS.BR ser compatível com os padrões de qualidade aceitos internacionalmente, não se compromete em aproveitar a competência existente nos padrões e modelos de melhoria de processo já disponíveis.
(*V*) O modelo MPS baseia-se nos conceitos de maturidade e capacidade de processo para a avaliação e melhoria da qualidade e produtividade de produtos de software e serviços correlatos.

Módulo 6

1) Os cinco grupos de processos do PMBoK são:

c) Iniciação, planejamento, execução, monitoramento e controle e encerramento.

2) Os grupos de processos não são as fases do projeto.

a) Verdadeiro.

3) Na quinta edição do PMBoK foi acrescida uma nova área de conhecimento, chamada de:

b) Stakeholders.

4) Sobre o PMBoK, é correto afirmar que "apresenta conhecimentos sobre o gerenciamento de projetos e foi realizado pelo PMI – Project Management Institute – desde 1983, com o intuito de documentar os processos de gerenciamento de projetos".

a) Verdadeiro.

5) Leia as afirmações abaixo:

I - O ciclo de vida preditivo determina que os processos não necessitam ser realizados com antecedência.
II - Não há diferenças entre o ciclo de vida incremental e interativo com o ciclo de vida adaptativo.
III - O ciclo de vida incremental e o ciclo de vida interativo utilizam características da metodologia ágil.
IV - O ciclo de vida adaptativo é também conhecido como método ágil.

Está correta a afirmação:

d) IV.

6) Segundo o PMBoK, o gerente de projetos deve ter as seguintes competências: conhecer gerenciamento geral e gestão de projetos, ter habilidades interpessoais e habilidades de gerenciamento geral, entender do ambiente do projeto e conhecer as normas e regulamentos da área de aplicação.

a) Verdadeiro.

7) Qual dos itens listados abaixo não se constitui uma área de conhecimento do PMBoK?

c) Métodos.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDRADE, A.; ROSSETTI, J.P. **Governança corporativa**. São Paulo: Atlas, 2007.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **ISO 9000-3**: Quality management and quality assurance standards – Part 3: Guidelines for the application of ISO 9001:1994 to the development, supply, installation and maintenance of computer software. Rio de Janeiro, 1997.

BASTOS, A. et al. **Base de conhecimento em testes de software**. Rio de Janeiro: Martins Editora, 2007.

BECK, K. et al. **Manifesto for agile software development**. Snowbird, Utah. 2001. Disponível em: <http://agilemanifesto.org/>. Acesso em: 24 out. 2013.

BERNARDO, Paulo C.; KON, Cristiano. **A importância dos testes automatizados**. 2008. Disponível em: <http://www.devmedia.com.br/artigo-engenharia-de-software-3-a-importancia-dos-testes-automatizados/9532>. Acesso em: 14 fev. 2014.

BOEHM, Barry W. A spiral model of software development and enhancement. **Computer**, Los Alamitos, v. 21, n. 5, p. 61-72, maio 1988.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML – guia do usuário**. Rio de Janeiro: Editora Campus, 2006.

CAETANO, Cristiano. **Melhores Práticas na Automação de Testes**. 2008. Disponível em: <http://www.devmedia.com.br/artigo-engenharia-de-software-5-melhores-praticas-na-automacao-de-testes/10249>. Acesso em: 14 fev. 2014.

COCKBURN, Alistair; HIGHSMITH, Jim. *Agile software development: the business of innovation*. **Computer**, Los Alamitos, v. 34, n. 9, p. 120-122, set. 2001.

CRUZ, Márcia; GONSALES, Samuel. **Análise de requisitos como ferramenta de planejamento**. 2010. Disponível em: <http://blogbohm.com/2010/10/18/analise-de-requisitos-como-ferramenta-de-planejamento/>. Acesso em: 24 out. 2013.

DELAMARO, M. E.; MALDONADO, J. C; JINO, M. **Introdução ao teste de software**. Rio de Janeiro: Elsevier, 2007.

FONSECA, Ijar M. **Princípios fundamentais da análise de requisitos**. 2010. Disponível em: <http://www2.dem.inpe.br/ijar/AnalEstr.html>. Acesso em: 24 out. 2013.

GASTALDO, Denise L.; MIDORIKAWA, Edson T. **Processo de engenharia de requisitos aplicado a requisitos não funcionais de desempenho – um estudo de caso**. 2002. Disponível em: <http://wer.inf.puc-rio.br/wer03/artigos/denise_gastaldo.pdf>. Acesso em: 30 nov. 2011.

IBM. **Rational unified process**. 2006. Disponível em: <http://www.wthreex.com/rup/v711_ptbr/index.htm>. Acesso em: 20. out 2013.

IMASTERS. **Sem boas práticas de engenharia não há agilidade**. 2010. Disponível em: < https://www.ibm.com/developerworks/community/blogs/fd26864d-cb41-49cf-b719-d89c6b072893/entry/sem_boas_pr_C3_A1ticas_de_engenharia_n_C3_A3o_h_C3_A1_agilidade2?lang=en>. Acesso em: 20 out. 2013.

ITS - Instituto de Tecnologia de Software. **Comparação do MPS.BR com o CMMI**. 2014. Disponível em: <http://its.org.br/o-que-e-mpsbr/comparacao-do-mps-br-com-o-cmmi/>. Acesso em: 15 fev. 2014.

LAGES, Daniel Scaldaferri. Automação dos testes: um lobo na pele de cordeiro?. **Engenharia de software Magazine**, Rio de Janeiro, n. 29, 2010. Disponível em: <http://www.devmedia.com.br/automacao-dos-testes-um-lobo-na-pele-de-cordeiro-engenharia-de-software-29/18207>. Acesso em: 25 out. 2013.

LEITE, Jair C. **Ciclo de vida de software**. 2007. Disponível em: <http://engenhariadesoftware.blogspot.com/2007/02/ciclo-de-vida-do-software-parte-1.html>. Acesso em: 11 nov. 2011.

LUIZ, Ricardo Fernandes. **Crise do software**. 2010. Disponível em: <https://www.ibm.com/

developerworks/community/blogs/fd26864d-cb41-49cf-b719-d89c6b072893/entry/sem_boas_pr_C3_A1ticas_de_engenharia_n_C3_A3o_h_C3_A1_agilidade2?lang=en>. Acesso em: 20. out 2013.

MOREIRA, Daniela. **Desenvolvimento ágil funciona?** 2009. Disponível em: < http://info.abril.com.br/noticias/ti/desenvolvimento-agil-funciona-30062009-3.shl>. Acesso em: 27 out. 2013.

OLIVEIRA, Camila S. **Comparando CMMi x MPS.BR: as vantagens e desvantagens dos modelos de qualidade no Brasil**. 2008. Disponível em: <http://www.camilaoliveira.net/Arquivos/Comparando%20CMMi%20x%20MPS.pdf>. Acesso em: 25 out. 2013.

PAULA FILHO, Wilson P. **Engenharia de software: fundamentos, métodos e padrões**. 3. ed. Rio de Janeiro: LTC, 2009.

PMI – Project Management Institute. **Um guia do conjunto de melhores práticas em gerenciamento de projetos (Guia PMBOK)**. 4. ed. Atlanta: PMI Book Service Center, 2008.

PMI – Project Management Institute. **A guide to the project management body of knowledge**. 5. ed. Atlanta: PMI Book Service Center, 2012.

PRESSMAN, R. S. **Engenharia de software**. 6. ed. São Paulo: McGraw-Hill, 2006.

PRESSMAN, R. S. **Engenharia de software**. 7. ed. São Paulo: McGraw-Hill, 2010.

PUCRS - Pontifícia Universidade Católica do Rio Grande do Sul. **Métodos Formais**. 2002. Disponível em: <http://www.inf.pucrs.br/~formos/metfomais.htm>. Acesso em: 14 fev. 2014.

QUINTELLA, Heitor L. M. M; ROCHA, Henrique M.; MOTTA, Wladimir. **Avaliação do nível de maturidade dos processos de desenvolvimento de produtos na indústria automotiva do sul fluminense com base nos critérios do CMMI**. 2005. Disponível em: <http://www.producao.uff.br/conteudo/rpep/volume52005/RelPesq_V5_2005_13.pdf>. Acesso em: 15 fev. 2014.

RUMBAUGH, James. et al. **Modelagem e projeto baseados em objetos**. Rio de Janeiro: Campus, 2006.

SAFF, David; ERNST, Michael D. Can continuous testing speed software development?. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING (ISSRE), 14., 2003. **Proceedings**. 2003. p. 281-292.

SCHACH, Stephen. E**ngenharia de software: os paradigmas clássico e orientado a objetos**. São Paulo: McGrawHill, 2008.

SHORE, James; WARDEN, Shane. **A arte do desenvolvimento ágil**. Porto Alegre: Alta Books, 2008.

SILVA, Dennis L. **Taxonomia dos Testes de Desempenho**. 2012. Disponível em: <https://www.ibm.com/developerworks/community/blogs/dennislopes/entry/taxonomia_dos_testes_de_desempenho?lang=en>. Acesso em: 14 fev. 2014.

SOFTEX. **MPS.BR – Melhoria de processo do software brasileiro**. 2012. Disponível em: <http://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_Geral_Software_20121.pdf>. Acesso em: 27 out. 2013.

SOMMERVILLE, Ian. **Engenharia de software**. São Paulo: Addison-Wesley, 2007.

SPECTOR, Alfred; GIFFORD, David. **A Computer Science Perspective on Bridge Design**. Communications of the ACM, Nova Iorque, volume 9, número 4, april. 1986. Disponível em: <http://www.ics.uci.edu/~andre/informatics121s2007/spectorgifford.pdf>. Acesso em: 14 fev. 2014.

THE STANDISH GROUP. **The Standish Group Report – Chaos**. 1995. Disponível em: <http://www.projectsmart.co.uk/docs/chaos-report.pdf>. Acesso em: 20 out. 2013.

THE STANDISH GROUP. **Chaos Manifesto 2013**. 2013. Disponível em: <http://versionone.com/assets/img/files/ChaosManifesto2013.pdf>. Acesso em: 20 out. 2013.