

PROCESSOS DE SOFTWARE

Prof.ª Simone Cristina Aléssio

Prof.ª Neli Miglioli Sabadin

Prof. Pedro Sidnei Zanchett





Copyright © UNIASSELVI 2017

Elaboração:

Prof.^a Simone Cristina Aléssio

Prof.^a Neli Miglioli Sabadin

Prof. Pedro Sidnei Zanchett

Revisão, Diagramação e Produção:

Centro Universitário Leonardo da Vinci – UNIASSELVI

Ficha catalográfica elaborada na fonte pela Biblioteca Dante Alighieri
UNIASSELVI – Indaial.

005

A372p Aléssio; Simone Cristina

Processos de software / Simone Cristina Aléssio; Neli Miglioli
Sabadin; Pedro Sidnei Zanchett: UNIASSELVI, 2017.

235 p. : il.

ISBN 978-85-515-0046-0

1. Programação de Computador – Programas – Dados.
I. Centro Universitário Leonardo Da Vinci.

APRESENTAÇÃO

Prezado acadêmico, seja bem-vindo ao universo da disciplina de Processos de *Software*!

A Engenharia de *Software* pode ser vista como uma abordagem de desenvolvimento de *software* elaborada com disciplina e métodos bem definidos. O processo de *software* é visto por uma sequência de atividades que produzem uma variedade de documentos, resultando em um programa satisfatório e executável. O desenvolvimento de *software* tem-se caracterizado por uma sobreposição de atividades necessárias para especificar, projetar e testar retorno dos resultados do *software* que está sendo criado.

Hoje o desenvolvimento de *software* é independente e uma grande fonte de renda. Um setor que emprega muitas pessoas e seu lucro é surpreendente. Mas a principal mudança se observa no seu processo de desenvolvimento. O amadorismo inicial foi substituído por processos e padrões e cada vez mais organizados.

Entender bem cada fase e compreender a sua importância no processo de construção de *software* é nosso objetivo. Vamos começar nossos estudos conhecendo como tudo começou. E veremos a evolução do *software* através do tempo, entenderemos como ocorreu a crise do *software* e muitos mitos que cercavam essa área que estava dando seus primeiros passos rumo ao sucesso.

Além de conhecer o passado para entendermos como chegamos até aqui, vamos conhecer também os passos necessários para que cada vez mais se produza *software* com alto grau de qualidade. Veremos a importância dos usuários no processo de desenvolvimento e como suas necessidades são atendidas com requisitos bem compreendidos. Além de entender a importância de termos um bom gerenciamento de projetos para alcarmos além da satisfação do usuário, o lucro nos projetos.

Aproveitamos a oportunidade para destacar a importância de desenvolver as autoatividades, lembrando que essas atividades NÃO SÃO OPCIONAIS. E que objetivam a fixação dos conceitos apresentados. Em caso de dúvida na realização das atividades, sugiro que você entre em contato com seu Tutor Externo, ou com a tutoria da Uniasselvi, não prosseguindo as atividades sem ter sanadas todas as dúvidas que irão surgindo.

Bom estudo, sucesso na sua trajetória acadêmica e profissional!

Prof.^a Simone Cristina Aléssio
Prof.^a Neli Miglioli Sabadin
Prof. Pedro Sidnei Zanchett



Você já me conhece das outras disciplinas? Não? É calouro? Enfim, tanto para você que está chegando agora à UNIASSELVI quanto para você que já é veterano, há novidades em nosso material.

Na Educação a Distância, o livro impresso, entregue a todos os acadêmicos desde 2005, é o material base da disciplina. A partir de 2017, nossos livros estão de visual novo, com um formato mais prático, que cabe na bolsa e facilita a leitura.

O conteúdo continua na íntegra, mas a estrutura interna foi aperfeiçoada com nova diagramação no texto, aproveitando ao máximo o espaço da página, o que também contribui para diminuir a extração de árvores para produção de folhas de papel, por exemplo.

Assim, a UNIASSELVI, preocupando-se com o impacto de nossas ações sobre o ambiente, apresenta também este livro no formato digital. Assim, você, acadêmico, tem a possibilidade de estudá-lo com versatilidade nas telas do celular, tablet ou computador.

Eu mesmo, UNI, ganhei um novo *layout*, você me verá frequentemente e surgirei para apresentar dicas de vídeos e outras fontes de conhecimento que complementam o assunto em questão.

Todos esses ajustes foram pensados a partir de relatos que recebemos nas pesquisas institucionais sobre os materiais impressos, para que você, nossa maior prioridade, possa continuar seus estudos com um material de qualidade.

Aproveito o momento para convidá-lo para um bate-papo sobre o Exame Nacional de Desempenho de Estudantes – ENADE.

Bons estudos!



Olá acadêmico! Para melhorar a qualidade dos materiais ofertados a você e dinamizar ainda mais os seus estudos, a Uniasselvi disponibiliza materiais que possuem o código *QR Code*, que é um código que permite que você acesse um conteúdo interativo relacionado ao tema que você está estudando. Para utilizar essa ferramenta, acesse as lojas de aplicativos e baixe um leitor de *QR Code*. Depois, é só aproveitar mais essa facilidade para aprimorar seus estudos!

BATE SOBRE O PAPO ENADE!



Olá, acadêmico!

Você já ouviu falar sobre o ENADE?

Se ainda não ouviu falar nada sobre o ENADE, agora você receberá algumas informações sobre o tema.

Ouviu falar? Ótimo, este informativo reforçará o que você já sabe e poderá lhe trazer novidades.



Vamos lá!

Qual é o significado da expressão ENADE?

EXAME NACIONAL DE DESEMPENHO DOS ESTUDANTES

Em algum momento de sua vida acadêmica você precisará fazer a prova ENADE.



Que prova é essa?

É **obrigatória**, organizada pelo INEP – Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira.

Quem determina que esta prova é obrigatória... O **MEC – Ministério da Educação**.



O objetivo do MEC com esta prova é o de avaliar seu desempenho acadêmico assim como a qualidade do seu curso.



Fique atento! Quem não participa da prova fica impedido de se formar e não pode retirar o diploma de conclusão do curso até regularizar sua situação junto ao MEC.



Não se preocupe porque a partir de hoje nós estaremos auxiliando você nesta caminhada.

Você receberá outros informativos como este, complementando as orientações e esclarecendo suas dúvidas.



Você tem uma trilha de aprendizagem do ENADE, receberá e-mails, SMS, seu tutor e os profissionais do polo também estarão orientados.



Participará de webconferências entre outras tantas atividades para que esteja preparado para #mandar bem na prova ENADE.

Nós aqui no NEAD e também a equipe no polo estamos com você para vencermos este desafio.

Conte sempre com a gente, para juntos mandarmos bem no ENADE! ✓✓



SUMÁRIO

UNIDADE 1 – PROCESSO, EXECUÇÃO, MODELAGEM E REUTILIZAÇÃO DE PROCESSOS DE SOFTWARE	1
TÓPICO 1 – PROCESSOS DE SOFTWARE	3
1 INTRODUÇÃO	3
2 CONCEITOS E CARACTERÍSTICAS DO PROCESSO DE SOFTWARE	3
2.1 NÍVEIS DE PROCESSO DE SOFTWARE	4
2.2 QUALIDADE DE PROCESSO DE SOFTWARE	7
2.3 INFLUÊNCIAS SOBRE A TECNOLOGIA DE PROCESSOS DE SOFTWARE	7
2.4 AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE ORIENTADOS AO PROCESSO	8
2.5 LIMITAÇÕES DAS TECNOLOGIAS ATUAIS	9
3 EXECUÇÃO DE PROCESSOS DE SOFTWARE	9
3.1 CONCEITOS DE EXECUÇÃO DE PROCESSOS DE SOFTWARE	10
3.2 DOMÍNIOS DE PROCESSOS DE SOFTWARE	12
3.3 A EXECUÇÃO NO CICLO DE VIDA DE PROCESSOS DE SOFTWARE	12
4 FORMALIDADES DE EXECUÇÃO	15
4.1 EXECUÇÃO PROCEDIMENTAL	15
4.2 EXECUÇÃO BASEADA EM REGRAS	15
4.3 EXECUÇÃO BASEADA EM REGRAS ECA (EVENTO-CONDIÇÃO-AÇÃO)	16
4.4 EXECUÇÃO BASEADA EM REDES DE PETRI	16
4.5 EXECUÇÃO BASEADA EM REDES DE TAREFAS	16
5 INTERAÇÃO COM USUÁRIOS	17
6 TIPOS DE ORIENTAÇÃO DE PROCESSO PARA DESENVOLVEDORES	19
7 PARADIGMAS DE INTERAÇÃO	19
8 FLEXIBILIDADE NA EXECUÇÃO DE PROCESSOS	20
8.1 ASPECTOS GERAIS DA FLEXIBILIDADE NA EXECUÇÃO DE PROCESSOS	20
RESUMO DO TÓPICO 1	22
AUTOATIVIDADE	25
TÓPICO 2 – EXECUÇÃO DE PROCESSO DE SOFTWARE	27
1 INTRODUÇÃO	27
2 REQUISITOS DE EXECUÇÃO DE PROCESSOS.....	27
2.1 REQUISITOS DE PRESCRIÇÃO	28
2.2 REQUISITOS DE INTERAÇÃO	28
2.3 REQUISITOS DE FLEXIBILIDADE	29
3 MODELOS DE CICLO DE VIDA OU MODELOS DE PROCESSO	30
3.1 MODELOS SEQUENCIAIS	31
3.2 MODELOS INCREMENTAIS	33
3.3 MODELOS EVOLUTIVOS OU EVOLUCIONÁRIOS	34
3.3.1 O modelo espiral	35
3.3.2 Prototipação	35

4 NORMAS E MODELOS DE QUALIDADE DE PROCESSO DE SOFTWARE	36
5 MELHORIA DO PROCESSO DE SOFTWARE BRASILEIRO	39
5.1 Descrição geral do MPS.BR	40
5.2 MR-MPS (Modelo de referência para melhoria do processo de <i>software</i>)	41
5.2.1 Níveis de maturidade	41
5.3 Processos	43
5.3.1 Capacidade do processo	43
5.3.2 MA-MPS – Método de avaliação para melhoria do processo de <i>software</i>	44
5.3.3 MN-MPS – Modelo de negócio para melhoria do processo de <i>software</i>	44
5.3.4 Processo de garantia da qualidade – GQA	44
RESUMO DO TÓPICO 2	46
AUTOATIVIDADE	48

TÓPICO 3 – TÉCNICAS DE MODELAGEM, MODELO EKD E REUTILIZAÇÃO DE PROCESSOS DE SOFTWARE	51
1 INTRODUÇÃO	51
2 ABORDAGENS DA MODELAGEM ORGANIZACIONAL	52
3 MÉTODO ERIKSSON – PENKER	54
3.1 RECURSOS	55
3.2 OBJETIVOS	55
3.3 REGRAS DE NEGÓCIOS	56
3.4 PROCESSOS	57
3.5 VISÃO DO NEGÓCIO	58
3.6 VISÃO DOS PROCESSOS DO NEGÓCIO	58
3.7 VISÃO DA ESTRUTURA DO NEGÓCIO	58
3.8 COMPORTAMENTO DO NEGÓCIO	59
3.9 INTEGRAÇÃO COM O PROCESSO DE SOFTWARE	59
3.9.1 Modelo de Dependências Estratégicas (SD – <i>Strategic Dependency Model</i>)	59
3.9.2 Modelo das Razões Estratégicas (SR – <i>Strategic Rationale Model</i>)	60
3.9.3 Notação de modelagem da técnica i*	61
4 MÉTODO EKD – ENTERPRISE KNOWLEDGE DEVELOPMENT	62
5 REUTILIZAÇÃO DE PROCESSOS DE SOFTWARE	66
5.1 DEFINIÇÃO DE COMPONENTES DE PROCESSO	72
5.2 DEFINIÇÃO DE CARACTERÍSTICAS DE PROCESSO	72
5.3 DEFINIÇÃO DE LINHAS DE PROCESSO	73
5.4 AVALIAÇÃO E APROVAÇÃO DOS ELEMENTOS REUTILIZÁVEIS	73
LEITURA COMPLEMENTAR	74
RESUMO DO TÓPICO 3	82
AUTOATIVIDADE	84

UNIDADE 2 – ENGENHARIA DE SOFTWARE E DE REQUISITOS E GERENCIAMENTO DE PROJETO DE SOFTWARE	85
--	-----------

TÓPICO 1 – NATUREZA DO SOFTWARE E ENGENHARIA DE SOFTWARE	87
1 INTRODUÇÃO	87
2 CRISE DO SOFTWARE	90
3 A NATUREZA DO SOFTWARE	92
4 APLICAÇÕES DO SOFTWARE	95
5 PROCESSO DE SOFTWARE	99
6 MITOS RELATIVOS AO SOFTWARE	101
RESUMO DO TÓPICO 1	104
AUTOATIVIDADE	106

TÓPICO 2 – ENGENHARIA DE REQUISITOS	107
1 INTRODUÇÃO	107
2 ENGENHARIA DE REQUISITOS	108
3 ESTUDO DE VIABILIDADE	113
4 ELICITAÇÃO DE REQUISITOS	116
5 TÉCNICAS DE LEVANTAMENTO	118
6 ENTREVISTA	118
7 ETNOGRAFIA – OBSERVAÇÃO PESSOAL	119
8 QUESTIONÁRIO	119
9 BRAINSTORMING	119
10 JAD (JOINT APPLICATION DESIGN)	120
11 CRITÉRIOS PARA VALIDAÇÃO E ACEITAÇÃO DE REQUISITOS	120
12 GERENCIAMENTO DE REQUISITOS	122
13 PLANEJAMENTO DO GERENCIAMENTO DE REQUISITOS	125
14 GERENCIAMENTO DE MUDANÇAS DE REQUISITOS	125
RESUMO DO TÓPICO 2	127
AUTOATIVIDADE	129
TÓPICO 3 – GERENCIAMENTO DE PROJETOS E SOFTWARE	131
1 INTRODUÇÃO	131
2 GERENCIAMENTO DE PROJETOS	132
3 POR QUE A GESTÃO DE PROJETOS É ESSENCIAL?	134
4 GESTÃO DE PROJETOS E PMI	136
5 GERÊNCIA DE RISCOS	139
6 GERENTE DE PROJETOS – TAMBÉM UM LÍDER	141
LEITURA COMPLEMENTAR	142
RESUMO DO TÓPICO 3	152
AUTOATIVIDADE	153
UNIDADE 3 – MODELAGEM ESTRUTURADA DE SISTEMAS, MÉTRICAS DE PROCESSO DE SOFTWARE, FUNDAMENTOS DE MELHORIA DE PROCESSO DE SOFTWARE, FERRAMENTAS DE GESTÃO DE PROCESSOS DE SOFTWARE BPMN E EPF	155
TÓPICO 1 – INTRODUÇÃO À MODELAGEM ESTRUTURADA DE SISTEMAS E MÉTRICAS DE PROCESSO DE SOFTWARE	157
1 INTRODUÇÃO	157
2 MODELAGEM ESTRUTURADA DE SISTEMAS	157
2.1 DIAGRAMA DE FLUXO DE DADOS (DFD)	160
2.2 DICIONÁRIO DE DADOS (DD)	164
2.3 DIAGRAMA DE ENTIDADE E RELACIONAMENTO (DER)	166
3 MÉTRICAS DE PROCESSO DE SOFTWARE	174
RESUMO DO TÓPICO 1	178
AUTOATIVIDADE	180
TÓPICO 2 – FUNDAMENTOS DE MELHORIA DE PROCESSO DE SOFTWARE E FERRAMENTA DE GESTÃO DE PROCESSOS DE SOFTWARE	181
1 INTRODUÇÃO	181
2 MELHORIA DE PROCESSO DE SOFTWARE	181
3 FERRAMENTAS DE GESTÃO DE PROCESSOS DE SOFTWARE	188
3.1 FERRAMENTA CASE: BIZAGI MODELER	189
3.2 FERRAMENTA CASE: ARIS EXPRESS	191

3.3 FERRAMENTA CASE: BPMN.io	193	
3.4 FERRAMENTA CASE: DRAW.io	194	
3.5 FERRAMENTA CASE: YAOQIANG BPMN EDITOR.	195	
3.6 FERRAMENTA CASE: HEFLO! DOCUMENTAÇÃO	197	
3.7 FERRAMENTA CASE: MODELIO	198	
3.8 FERRAMENTA CASE: SYDLE	199	
3.9 FERRAMENTA CASE: MICROSOFT VISIO®	201	
3.10 FERRAMENTA CASE: EPF-COMPOSER	202	
LEITURA COMPLEMENTAR	205	
RESUMO DO TÓPICO 2	207	
AUTOATIVIDADE	210	
 TÓPICO 3 – BUSINESS PROCESS MANAGEMENT NOTATION (BPMN)		
E UM ESTUDO DE CASO DE UM PROCESSO SOFTWARE		
COM O AUXÍLIO DO EPF		211
1 INTRODUÇÃO	211	
2 BUSINESS PROCESS MANAGEMENT NOTATION (BPMN)	211	
3 ESTUDO DE CASO DE UM PROCESSO SOFTWARE COM O AUXÍLIO		
DO EPF		216
RESUMO DO TÓPICO 3	224	
AUTOATIVIDADE	225	
 REFERÊNCIAS	227	



PROCESSO, EXECUÇÃO, MODELAGEM E REUTILIZAÇÃO DE PROCESSOS DE SOFTWARE

OBJETIVOS DE APRENDIZAGEM

Esta unidade tem por objetivos:

- compreender a importância de um Processo de *Software*;
- conhecer os diversos componentes de um Processo de *Software*;
- conhecer as técnicas de modelagem de Processos de *Software*;
- entender o funcionamento da reutilização de Processos de *Software*.

PLANO DE ESTUDOS

Esta unidade está dividida em três tópicos. Em cada um deles, você encontrará atividades que contribuirão para a apropriação dos conteúdos apresentados.

TÓPICO 1 – PROCESSO DE SOFTWARE

TÓPICO 2 – EXECUÇÃO DE PROCESSO DE SOFTWARE

TÓPICO 3 – TÉCNICAS DE MODELAGEM, MODELO EKD E
REUTILIZAÇÃO DE PROCESSOS DE SOFTWARE

PROCESSOS DE SOFTWARE

1 INTRODUÇÃO

O desenvolvimento de *software* é uma atividade complexa que depende de muitos fatores, entre eles, o relacionamento e o processo comunicacional adotado pela equipe de desenvolvimento, principalmente, gerentes, analistas de sistemas e desenvolvedores. Neste sentido, ganha destaque a área de Tecnologia de Processos de *Software*, que permeia desde a modelagem até a simulação, testes e toda a evolução dos processos de desenvolvimento de sistemas computacionais.

Podemos entender o processo de *software* como um conjunto de atividades que envolve desde o entendimento e esboço da solução de um problema específico, até a sua implementação e posterior liberação para uso. Todas as atividades são aninhadas dentro de um modelo de processo de *software*, que permite desde a sua análise, compreensão e execução. Neste contexto, um ambiente de desenvolvimento de *software* que tem como prática a modelagem e execução de processos é denominado de ADS (Ambiente de Desenvolvimento de *Software*) orientado a processos (PRESMANN, 2011).

2 CONCEITOS E CARACTERÍSTICAS DO PROCESSO DE SOFTWARE

Podemos entender o processo de *software* como um conglomerado de atividades, políticas, normas, padrões, processos, tecnologias, procedimentos e conhecimentos utilizados em conjunto para desenvolver um sistema computacional (SOMMERVILLE, 2007).

Muito importante para a eficiência do processo é a escolha do modelo do ciclo de vida, que marca o início do processo de desenvolvimento do projeto e que para ser considerado eficaz deve considerar as relações entre atividades e envolvidos na execução das mesmas. O modelo de ciclo de vida organizará as atividades no sentido de estabelecer a dependência ou precedência de cada tarefa a ser desenvolvida.



Processos de *software* definem o conjunto de atividades conduzidas no contexto do projeto, tais como análise de requisitos, projeto, codificação, teste, instalação etc., os recursos (*software*, *hardware* e pessoas) necessários e os procedimentos a serem adotados na realização de cada uma das atividades. Sendo que essas atividades são compostas por outras atividades e podem se comunicar entre si e operam sobre artefatos de entrada para produzir artefatos de saída.

FONTE: Disponível em: <<http://www.linhadecodigo.com.br/artigo/1401/cmmi-para-novatos.aspx#ixzz472DPkwpg>>. Acesso em: 26 abr. 2016.

É importante ter em mente que o processo de *software* adotado muda de acordo com o projeto a ser desenvolvido. Isso significa que cada projeto exigirá ajustes específicos e direcionados à aplicação a ser desenvolvida. A cada novo projeto, os cenários mudam: tecnologias, pessoas, rotinas, processos, procedimentos, atividades. Logo, mudam também os processos de *software* adotados no desenvolvimento.

2.1 NÍVEIS DE PROCESSO DE SOFTWARE

O modelo CMMI como ferramenta no gerenciamento de projetos de *Software* é o que há de mais completo quando o assunto é qualidade de *software*. Mas, o que é CMMI? O **CMMI (Capability Maturity Model Integration)** é um modelo de referência que contém práticas (genéricas ou específicas) necessárias à maturidade em disciplinas específicas (*Systems Engineering (SE)*, *Software Engineering (SE)*, *Integrated Product and Process Development (IPPD)*, *Supplier Sourcing (SS)*). Desenvolvido pelo SEI (*Software Engineering Institute*), o CMMI é uma evolução do CMM e procura estabelecer um modelo único para o processo de melhoria corporativo, integrando diferentes modelos e disciplinas.

O CMMI é um modelo de maturidade de melhoria de processos para o desenvolvimento de produtos e serviços. Consiste de melhores práticas que endereçam desde atividades de desenvolvimento até a manutenção do produto e cobrem o ciclo de vida inteiro do projeto, desde sua concepção, passando pela entrega e posterior manutenção.

O CMMI tem cinco níveis de maturidade e 24 áreas de processos. Os níveis que determinam a maturidade da organização, sendo cada fase uma base para o próximo. É como uma escada que deve ser subida degrau a degrau, conforme a capacidade e a maturidade da empresa.

Já a área de processo é um conjunto de práticas relacionadas que, quando são executadas coletivamente, satisfazem um conjunto de objetivos considerados importantes para alcançar sensíveis melhorias nesta área. As áreas de processos estão organizadas por níveis de maturidade.

FONTE: Disponível em: <<http://www.linhadecodigo.com.br/artigo/1401/cmmi-para-nicantes.aspx#ixzz472DPkpg>>. Acesso em: 26 abr. 2016.

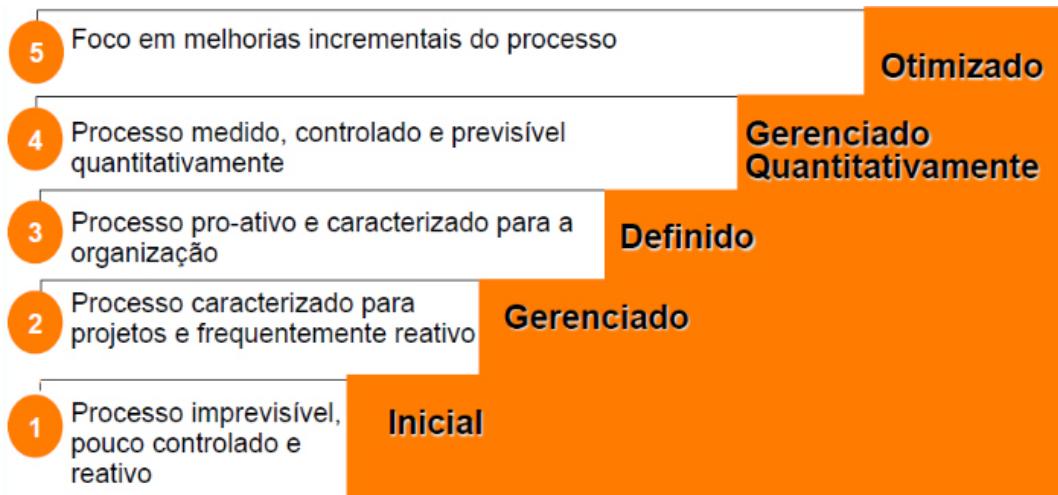
A seguir podemos destacar os cinco níveis de maturidade do CMMI e o que, de forma sucinta, cada degrau representa:

FIGURA 1 – NÍVEIS DE MATURIDADE DO CMMI



FONTE: Disponível em: <<http://www.tiegestao.com.br/2013/03/31/cmmi-capability-maturity-model-integration-modelo-integrado-de-maturidade-e-de-capacidade/>>. Acesso em: 26 abr. 2016.

FIGURA 2 – CARACTERÍSTICAS DOS NÍVEIS DE MATURIDADE DO CMMI



FONTE: Disponível em: <<http://docplayer.com.br/2650672-Processo-de-desenvolvimento-de-software.html>>. Acesso em: 26 abr. 2016.

As principais características dos níveis citados são:

- **Nível 1 – Inicial:** imaturidade organizacional; os processos são improvisados e geralmente não são seguidos; compromissos de prazo e custo não são cumpridos; o planejamento não é feito com base em estimativas; as qualidades, procedimentos e conhecimentos pertencem às pessoas e não aos projetos; as chances de sucesso dependem das habilidades pessoais dos gerentes e desenvolvedores.
- **Nível 2 – Gerenciado:** políticas e procedimentos para gerenciar o desenvolvimento de *software* estão definidas e são obedecidas; o planejamento é baseado em estimativas e na experiência anterior de outros projetos; os projetos utilizam processos definidos, usados, disseminados, documentados, medidos e fiscalizados com rotinas de melhoria; os processos afetados são puramente gerenciais (não técnicos) e pertencem aos projetos e não às pessoas.
- **Nível 3 – Definido:** os processos utilizados são estabelecidos e padronizados em toda a organização; processos técnicos passam a ser considerados ao lado dos processos gerenciais; tanto os processos gerenciais quanto os técnicos passam a ser repetidos; os processos pertencem à organização e não mais aos projetos.
- **Nível 4 – Quantitativamente gerenciado:** são estabelecidas metas quantitativas para os processos e produtos, medidas de qualidade e produtividade são coletadas em todos os projetos; é estabelecido controle estatístico de processos; a gestão passa a ser feita com bases quantitativas.

- **Nível 5 – Otimização:** a organização está engajada na melhoria contínua de seus processos; identificação de pontos fracos e defeitos; ações preventivas sobre causas; mudanças mais significativas de processos e/ou tecnologias são feitas a partir de análise de custo/benefício com base em dados quantitativos.

FONTE: Disponível em: <<http://www.linhadecodigo.com.br/artigo/1401/cmmi-parainiciantes.aspx#ixzz472DPkpg>>. Acesso em: 26 abr. 2016.

2.2 QUALIDADE DE PROCESSO DE SOFTWARE

Desenvolver *softwares* dentro do prazo estipulado, seguindo rigidamente as definições estabelecidas no escopo e sem comprometer a qualidade desejada, é sempre um desafio. Mas, é o mínimo que se espera ao considerarmos que o *software* é um produto como qualquer outro.

Os processos de *software* servem de alicerce para garantir a qualidade dos projetos. Se um *software* foi desenvolvido tendo como base processos de *software*, incluindo processos de qualidade, a probabilidade de falhas ou necessidade de rework é consideravelmente menor. Empresas de *software* que incorporam modelos de processos de *software* e seguem suas práticas tem maior chance de reconhecimento, uma vez que trabalham sob orientações e padrões de qualidade definidos e recomendados pela comunidade de especialistas do setor de desenvolvimento de *software*.

Desde 1980, os estudos têm se concentrado nas melhorias de processos de *software* com enfoque para a qualidade. Existem vários modelos no mercado, dos quais podemos destacar como principais normas e modelos de qualidade de *software*: **ISO 9000** (ISO, 2000), **ISO/IEC 12207** (ISO/IEC, 2008), **ISO/IEC 15504** (ISO/IEC, 2003), **CMMI** (SEI, 2006) e **MPS.BR** (SOFTEX, 2007). O Tópico 2 deste caderno explorará estas normas.

2.3 INFLUÊNCIAS SOBRE A TECNOLOGIA DE PROCESSOS DE SOFTWARE

As ferramentas CASE formam a base do suporte tecnológico para apoiar o desenvolvimento de *software* e tiveram origem nos primeiros ensaios de propor um ciclo de vida de desenvolvimento para estes projetos, fornecendo uma referência ou guia para todas as etapas de desenvolvimento de sistemas computacionais (NUNES, 1992).

Na sequência, houve a necessidade de se controlar os processos envolvidos no desenvolvimento. Neste sentido, o ADS (Ambiente de Desenvolvimento de *Software*) foi incrementado com os PSEEs (*Process-centred Software Engineering Environments / Ambiente de Engenharia de Software Centrado no Processo*), auxiliando e controlando atividades envolvidas no ciclo de vida do *software*, mais precisamente na produção e manutenção do *software* (LIMA et al., 2006).

Outra questão importante que afeta diretamente a tecnologia de processos de *software*: a rápida assimilação e disseminação de modelos acadêmicos como *Unified Process Programação Extrema* e SCRUM.

Cabe ressaltar que as ferramentas tecnológicas envolvidas no processo de *software* têm como intuito automatizar tarefas e facilitar a tomada de decisão por parte de gerentes e demais envolvidos.

2.4 AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE ORIENTADOS AO PROCESSO

Os ambientes de *software* orientados a processos surgiram com a intenção de fornecer um apoio mais efetivo aos desenvolvedores no sentido de auxiliar no aumento da produtividade, melhoria da qualidade e diminuição de custos, através da adoção de PSEEs (*Process-centred Software Engineering Environments / Ambiente de Engenharia de Software Centrado no Processo*) para automatizar a gerência dos processos. A tecnologia PSEEs possibilita uma melhor comunicação entre as pessoas envolvidas e torna mais confiável o que está sendo desenvolvido; realiza algumas tarefas automaticamente e desta forma libera colaboradores para outras atividades; permite a extração de informações dos processos sempre que necessário; permite a reutilização dos processos em projetos futuros, exigindo apenas a adequação em relação ao novo cenário da aplicação que será desenvolvida; também possibilita medições distintas, tão importante para o aperfeiçoamento dos processos e dos projetos de forma geral (LIMA et al., 2006).

Neste contexto podemos destacar três características importantes e que trabalham de forma conjunta: coordenação, colaboração e comunicação. A gerência está relacionada com a coordenação, que depende diretamente da colaboração e comunicação. A colaboração deve permitir o desenvolvimento de atividades bem como uma rotina de decisões compartilhadas, coordenadas pela gerência para que se alcancem os objetivos desejados. Um processo comunicacional eficiente é o elo entre os conceitos de coordenação e colaboração.

Para tornar isso viável, um modelo de processo deve ser adotado para descrever como a camada de coordenação deve trabalhar. Este modelo será construído através de uma linguagem de modelagem do processo (PML – *Process Modeling Language*).

2.5 LIMITAÇÕES DAS TECNOLOGIAS ATUAIS

Para Reis (2003) algumas limitações tecnológicas e atuais podem ser facilmente citadas e encontradas em relação aos controles de processos de *software*, uma vez que as atividades de desenvolvimento são orientadas às pessoas. Esta característica aumenta a complexidade do processo e da gerência na execução das atividades.

A seguir, você pode entender melhor alguns problemas/limitações encontrados na tecnologia PSEEs:

- o ambiente de desenvolvimento de *software*, na maioria dos casos, é estático: os modelos são rígidos e as mudanças são difíceis de serem implementadas, uma vez que os desenvolvedores seguem modelos que não consideram a natureza criativa de suas tarefas;
- os PSEEs, tais como os *Workflow management systems* (WfMS), são sistemas de coordenação e desempenham bem suas atividades. O problema é que geralmente precisam alojar tarefas às pessoas que já foram direcionadas para outras atividades. Desta forma, competem por recursos humanos limitados, sendo que a execução de processos depende da disponibilidade destas pessoas.
- outra limitação corresponde à falta de mecanismos para auxiliar a escolha de desenvolvedores para tarefas específicas. A tecnologia ainda não tem a inteligência de associar a pessoa mais adequada para determinada tarefa.

3 EXECUÇÃO DE PROCESSOS DE SOFTWARE

A execução de processos de *software* é uma das etapas que compreendem o ciclo de vida do desenvolvimento de *software*. A execução é uma atividade executada por pessoas ou processos automáticos controlados pelo próprio sistema, e contempla questões como planejamento, controle e até a segurança do processo (HUMPHREY, 1993).

3.1 CONCEITOS DE EXECUÇÃO DE PROCESSOS DE SOFTWARE

A execução de processos de *software* é praticada sempre que o modelo adotado está pronto para ser executado. Pode-se dizer que o processo está pronto para execução quando há interação entre tudo o que está envolvido nas atividades de desenvolvimento: desenvolvedores, ferramentas e processos. A fase de execução dos processos não é totalmente automatizada. Ela apenas depende de um processo automatizado para orientar os desenvolvedores a coordenarem melhor os trabalhos em desenvolvimento, e também para tornar possível a execução automática de tarefas repetitivas (HUMPHREY, 1993).

Com relação aos modelos de processos de *software*, podemos entendê-los como um conceito abstrato do processo de *software*, que define quem, como, quando e por que as atividades devem ser realizadas. As linguagens PML (Linguagem de Modelagem de Processos) são utilizadas para descrever estes modelos específicos, e possuem características específicas como o fato de detalhar as atividades que necessitam da intervenção humana para serem colocadas em prática.

Os modelos de processos instanciados relacionam-se ao cenário da organização de forma geral, principalmente em questões específicas no desenvolvimento de *software*, bem como os prazos envolvidos em cada projeto. Para que um processo possa ser instanciado ele depende da adequada alocação de recursos (humanos, financeiros e tecnológicos) para cada atividade, bem como o prazo de desenvolvimento estipulado em cada customização. É importante ter em mente que um processo abstrato pode dar origem a vários processos instanciados (SOMMERVILLE, 2007).

Os processos instanciados são interpretados pelas **máquinas de execução**, conhecidas conceitualmente como *Process Engine*. Nesta etapa ocorre o gerenciamento e orientação aos desenvolvedores de acordo com o modelo de processo definido. Esta fase tem o objetivo de alinhar a execução das atividades conforme definições estabelecidas pelo modelo.

O *feedback* é um ponto importantíssimo na fase da execução dos processos. É importante que ocorra e pode se manifestar de formas distintas: os envolvidos podem avisar quando da conclusão de tarefas sob sua responsabilidade; informações podem ser extraídas do banco de dados para averiguar o andamento do projeto; podem ser criados eventos específicos, como relatórios ou consultas para fazer apontamentos pontuais sobre os projetos, obtendo-se assim um *feedback* mais consistente e confiável. Entretanto, seja qual for a forma de *feedback*, este sempre terá origem em uma decisão ou posicionamento por parte das pessoas envolvidas no desenvolvimento (PRESSMANN, 2011).

A figura a seguir demonstra os relacionamentos do mecanismo de execução com outros elementos de um PSEE, sendo que a **Interação com o Usuário** remete

às diferentes visões percebidas pelos usuários dos sistemas. Isso é importante para que os desenvolvedores possam dar um correto *feedback* acerca da conclusão/início das atividades, facilitando o trabalho dos gerentes na coordenação, realocação e distribuição correta dos trabalhos pendentes.

O item Ferramentas remete à interpretação de processos executáveis, onde ferramentas distintas podem ser necessárias a fim de propor um melhor entendimento do processo. O **Repositório** serve para armazenar os resultados produzidos que são decorrentes das mudanças/ajustes ocorridas na fase de execução dos processos. A **Alocação de Recursos** concentra tudo o que está relacionado a todos os tipos de recursos necessários para a execução dos processos e as **Métricas/Histórico** são necessárias para avaliar e manter informações dos processos a fim de fazer um diagnóstico geral de desempenho dos processos organizacionais (SOMMERVILLE, 2007).

FIGURA 3 – INTERAÇÃO DO MECANISMO DE EXECUÇÃO COM OUTROS COMPONENTES DO PSEE



FONTE: Reis (2003)

3.2 DOMÍNIOS DE PROCESSOS DE SOFTWARE

Definir processos é diferente de executar processos. A execução depende das pessoas, e por este motivo a imprevisibilidade é adicionada nesta fase. Esta característica pode gerar discrepâncias em relação entre o que está definido e o que é realmente foi executado. Por causa disso, três domínios distintos podem ser apontados nos processos de *software* (REIS, 2003).

Desta forma, existem três diferentes domínios de processos de *software*:

- Domínio das definições de processo: são modelos ou partes de modelos de processos descritos com uso de alguma notação padronizada.
- Domínio da realização do processo: contém descritivos de atividades realizadas por pessoas ou programas computadorizados.
- Domínio da execução da definição do processo: concentra-se no que é necessário para executar uma definição de processo em um ambiente de desenvolvimento de *software*.

3.3 A EXECUÇÃO NO CICLO DE VIDA DE PROCESSOS DE SOFTWARE

Os processos de *software* são dinâmicos e evolutivos por natureza, devido à instabilidade do ambiente operacional e das constantes necessidades de ajustes e melhorias dos sistemas (REIS, 2003).

Com relação ao ciclo de vida, existe um ciclo de vida para processos de *software* e um ciclo de vida de produtos de *software*. Neste caso, atividades são chamadas de meta-atividades e o ciclo de vida em si é conhecido como metaprocesso de *software*. A figura a seguir mostra uma representação de um ciclo de vida de processos de *software* que inclui a fase de execução de processos.

FIGURA 4 - CICLO DE VIDA DOS PROCESSOS DE SOFTWARE



FONTE: Reis (2003)

No ciclo de vida exibido na figura anterior, podemos destacar:

- **Provisão de Tecnologia**: aborda as tecnologias envolvidas nos processos e atividades de desenvolvimento de *software* (línguagens de modelagem de processo, modelos de processo prontos para reutilização e ferramentas para aquisição, modelagem, análise, projeto, simulação, evolução, execução e monitoração de modelos de processo).
- **Análise de Requisitos do Processo**: é a etapa de identificação de requisitos: tanto para novos processos quanto para processos já existentes.
- **Projeto do Processo**: através de línguagens de modelagem de processos, define-se a arquitetura que será o alicerce dos processos.
- **Instanciação do Processo**: otimiza e altera especificações do processo promovidas por atividades anteriores incorporando informações mais detalhadas dos prazos e recursos envolvidos.
- **Simulação do Processo**: permite verificação e validação dos processos definidos antes da execução.

- **Execução do Modelo de Processo:** através de ferramentas específicas para executar o processo instanciado. Aqui ocorre o *feedback* e a coleta de informações para gerenciamento da execução dos processos.
- **Avaliação do Processo:** avalia de forma quantitativa e qualitativa o desempenho dos que estão sendo executados. Essa avaliação ocorre de forma paralela à execução dos processos.

Para Lima (2006) alguns aspectos são fortemente relatados na literatura de exceções da execução de processos de *software*:

- **Automação de processo:** dar suporte às atividades que podem ser automatizadas, excluindo-se a intervenção humana.
- **Trabalho cooperativo:** orientar as pessoas a trabalharem de forma cooperativa durante todo o ciclo de vida de desenvolvimento do *software*.
- **Monitoração:** monitorar as atividades em todas as fases do processo, a fim de permitir uma gerência mais eficiente dos projetos;
- **Registro da história do processo:** manter as informações de evolução dos processos para que melhorias sejam efetuadas, permitindo a melhora contínua dos modelos de processos de desenvolvimento de *software* adotados.

Outras exceções devem entrar na pauta de exceções de processos, como:

- A forma de tratar a influência humana na execução de processos, a preservação da consistência e recuperação de inconsistências.
- A capacidade de permitir modificação dinâmica (reflexão da linguagem de modelagem).
- A integração com ferramentas do ambiente.
- O controle de acesso a artefatos do processo.
- O tratamento das transações longas e das versões de modelos de processos.

Conheça a seguir, os formalismos necessários e adotados na Execução de Processos de *software*.

4 FORMALIDADES DE EXECUÇÃO

A execução dos processos de *software* depende diretamente da linguagem adotada na etapa da modelagem dos processos, sendo que na maioria das vezes o nível de formalidade adotado é baixo. Geralmente, a modelagem orientada a processos usa o mesmo nível de formalismo para a execução dos processos (REIS, 2003).

A modelagem de processos de *software* é bem específica e tem evoluído significativa e paralelamente à própria evolução tecnológica. Essa evolução caminha para aspectos mais direcionados ao processo de desenvolvimento de *software*, cujas atividades são consideradas como de alta complexidade.

Acompanhe e conheça os formalismos de execução (e modelagem) mais conhecidos e utilizados em ambientes de desenvolvimento de *software*.

4.1 EXECUÇÃO PROCEDIMENTAL

A execução procedural é um processo de baixo nível e considera que os processos de *software* devem ser modelados seguindo conceitos da programação convencional, pois considera que um processo de *software* também é *software*. Nesta situação, todas as questões do processo devem ser especificadas, detalhadas e documentadas antes da execução iniciar. Podemos considerar, neste caso, como execução do processo, a própria execução de um programa (REIS, 2003).

4. 2 EXECUÇÃO BASEADA EM REGRAS

Na execução baseada em regras, os processos são modelados como regras que possuem pré e pós-condições, como ocorre no desenvolvimento de sistemas especialistas.

Uma desvantagem deste tipo de execução é que o gerente consegue monitorar a execução das regras, mas não consegue coordenar e gerenciar o desenvolvimento geral (SOMMERVILLE, 2007).

4.3 EXECUÇÃO BASEADA EM REGRAS ECA (EVENTO-CONDIÇÃO-AÇÃO)

A execução ECA permite identificar eventos antes da regra ser executada, podendo disparar as regras que satisfazem a condição do evento. Isso pode gerar novos eventos a serem tratados (SOMMERVILLE, 2007).

Algumas desvantagens:

- o formalismo é difícil para entendimento humano;
- há dificuldade de controlar a execução;
- faltam conceitos de alto nível como processo, espaço de trabalho, planejamento, entre outros;
- há dificuldade em responder a questões simples como: Onde estamos? Qual é a próxima tarefa?

4.4 EXECUÇÃO BASEADA EM REDES DE PETRI

Este tipo de execução se apoia no formalismo matemático das Redes Petri, que se utilizam de grafos focados em dois nodos específicos: lugares e transições. Lugares representam os objetos ou recursos e as transições, as etapas do processo.

O comportamento desta rede é definido pelas regras adotadas em suas transições (PRESSMANN, 2011).

4.5 EXECUÇÃO BASEADA EM REDES DE TAREFAS

Este modelo apresenta um grafo cuja função é representar a estrutura do processo, sendo que os nodos representam atividades e os arcos apresentam o fluxo de controle/dados entre as diversas atividades (PRESSMANN, 2011).

5 INTERAÇÃO COM USUÁRIOS

As pessoas devem ser sempre orientadas e instruídas durante a execução dos processos de *software*, e isso depende na maioria das vezes do tipo de interação escolhida para o ambiente do processo que está em execução. É necessário que ocorra a interação entre os envolvidos; do contrário, coloca-se o processo todo em risco.

Podemos citar como exemplo de interação, duas funções básicas envolvidas em processos de *software*: gerente e desenvolvedor. O gerente tem como responsabilidade, coordenar, organizar e distribuir o projeto adequadamente. Os desenvolvedores têm como atribuição principal executar as tarefas para a construção dos aplicativos (REIS, 2003).

O quadro a seguir sintetiza os tipos de interação durante a execução, os objetivos e as ferramentas utilizadas na interação do gerente e dos desenvolvedores.

QUADRO 1 – TIPOS DE INTERAÇÃO COM USUÁRIOS

	Interação com o Gerente	Interação com o Desenvolvedor
Objetivos e papéis dos usuários	<ul style="list-style-type: none"> -Visão macro dos eventos que ocorrem durante a execução de processos. -Acompanhamento e monitoração dos eventos (desempenhados por desenvolvedores), prazos e recursos. -Manipulação do processo (ajustes durante a execução) 	<ul style="list-style-type: none"> -Recebem orientação acerca das tarefas a serem desempenhadas. -Fornecem <i>feedback</i> sobre a performance
Ferramentas utilizadas	<ul style="list-style-type: none"> -Ferramentas para monitoração do processo -Ferramentas para modelagem de processos e planejamento de atividades 	<ul style="list-style-type: none"> -Ferramentas CASE -Ferramentas de interação do PSEE (por exemplo, agendas, espaço de trabalho, etc.)

FONTE: Adaptado de Souza, Reis e Reis (2001)

Os principais requisitos relacionados à interação com gerentes coincidem com os requisitos gerais de PSEEs encontrados em alguns estudos como os de Young (1998) e Gimenez (1998):

- **Permitir diferentes visões de processos:** permitir que o gerente visualize os processos por ângulos diferentes, sob as perspectivas: funcional, comportamental, organizacional, informacional e tecnológica (CURTIS; KELLNER; OVER, 1992).
- **Mostrar o estado atual e o histórico do processo:** permite a análise do processo em relação ao que se pratica no momento e também ao histórico de execuções do processo.
- **Permitir modificação dinâmica do processo durante a execução:** permite aos gerentes modificar os processos mesmo que estejam em execução.
- **Fornecer independência do formalismo de modelagem de processos:** permite representar de formas diferentes o processo independentemente do nível de formalismo adotado.
- **Fornecer monitoração de eventos e tratamento de exceções na execução de processos:** facilitar a percepção do gerente em relação às exceções dos processos, com o intuito de prover um tratamento automático, disparado na maioria das vezes por triggers de banco de dados.

Do outro lado, a interação com desenvolvedores também é importante para manter o processo em execução sincronizado com a realização e objetivos pré-estabelecidos para o processo em questão. O *feedback* do desenvolvedor em relação ao processo deve ser pontual e consistente, e para que isso seja possível, ele não deve apresentar excesso de atividades sob sua responsabilidade. Cabe, neste caso, o bom senso do gerente no momento de realizar a distribuição dos trabalhos que deverão ser executados.

A seguir são listados alguns requisitos dos PSEEs durante a interação com desenvolvedores (REIS, 2003):

- Orientar desenvolvedores na realização de suas tarefas.
- Fornecer visualização adequada das tarefas do processo.
- Obter feedback do andamento do processo.
- Fornecer visualização dos estados do processo (atual e anteriores).

- Facilitar comunicação e cooperação com outros desenvolvedores.
- Flexibilizar a interação (adequação do paradigma de interação ao usuário).

O desenvolvedor necessita visualizar, selecionar e realizar ações sobre itens de trabalho.

6 TIPOS DE ORIENTAÇÃO DE PROCESSO PARA DESENVOLVEDORES

Dowson e Fernström (1994) identificaram os tipos de orientação aos desenvolvedores durante a execução de processos e os dividiram em quatro categorias:

- **Orientação passiva:** entrega aos desenvolvedores informações para que eles possam decidir corretamente em relação às ações a serem tomadas conforme definição dos processos.
- **Orientação ativa:** notifica os desenvolvedores acerca de suas tarefas ou eventos de maior importância.
- **Obrigação do processo:** controla os acessos aos dados e ferramentas de forma direta, obrigando os desenvolvedores a seguirem o processo de acordo com a modelagem especificada.
- **Automação de processo:** procura executar partes do processo de forma automática, sem intervenção humana.

7 PARADIGMAS DE INTERAÇÃO

São quatro os paradigmas de interação na execução de processos de *software*:

- **Interação orientada a tarefas:** na prática adota-se uma lista de tarefas que deverão ser executadas.
- **Interação orientada a documentos:** neste tipo de interação, o desenvolvedor visualiza o processo como um conjunto de documentos a serem criados ou manipulados.
- **Interação orientada a metas:** Este paradigma utiliza o conceito de metas a serem atingidas, sem distinguir quais tarefas devem ser realizadas ou quais documentos devem ser manipulados.

- **Interação orientada a ferramentas:** este paradigma depende diretamente das ferramentas de desenvolvimento. Porém, não existem ferramentas para controle de trabalho ou feedback do desenvolvedor. São consideradas apenas as ferramentas utilizadas para o desenvolvimento dos trabalhos;

Podemos entender então, que a interação dos desenvolvedores com o PSEE está diretamente associada ao nível de formalismo e tipo de orientação (ativa, passiva, obrigatória) utilizados no processo.

8 FLEXIBILIDADE NA EXECUÇÃO DE PROCESSOS

Processos de desenvolvimento de *software* são atípicos. Eles são diferentes de outros tipos de processos pois envolvem pessoas executando tarefas de alta complexidade, e por este motivo não é possível definir todo o processo de desenvolvimento de forma antecipada. É um ambiente envolto na incerteza. Poder escolher caminhos alternativos é uma prática recorrente na execução de processos afim de alinhar adequadamente os projetos em desenvolvimento. Esta característica de “escolha” pode aumentar consideravelmente a flexibilidade dos processos.

8.1 ASPECTOS GERAIS DA FLEXIBILIDADE NA EXECUÇÃO DE PROCESSOS

A flexibilidade na execução de processos necessita de:

- **Modificação dinâmica durante a execução:** deve ser permitido alterar todos os processos, inclusive os que já estão sendo executados, sem que ele apresente inconsistências.
- **Execução de processos incompletos:** deve ser possível iniciar um processo mesmo que este não esteja completo. Porém, deve-se prever que ele será detalhado posteriormente.
- **Instanciação das atividades do processo durante a execução:** A alocação de pessoas e recursos pode ser feita após o processo ter sido iniciado.
- **Escolhas entre caminhos alternativos:** o projetista do processo deve prever caminhos alternativos para que o processo possa mudar o trajeto sem ser necessário que alguém fique monitorando a situação.

- **Gerência e tratamento de eventos:** ao invés de monitorar exaustivamente a execução do processo, o gerente pode programar o mecanismo de execução para agir em determinadas situações de forma automática.

Conforme mencionado anteriormente, a modelagem pode acontecer durante a execução do processo. Então, podemos entender que qualquer aspecto que flexibilize a modelagem, também estará tornando mais flexível a modelagem do processo de forma geral (LERNER, 2000).

RESUMO DO TÓPICO 1

Neste tópico, você aprendeu que:

- O desenvolvimento de *software* é uma atividade complexa que depende de muitos fatores, entre eles, o relacionamento e o processo comunicacional adotado pela equipe de desenvolvimento, principalmente, gerentes, analistas de sistemas e desenvolvedores.
- Processo de *software* é um conjunto de atividades, políticas, normas, padrões, processos, tecnologias, procedimentos e conhecimentos utilizados em conjunto para desenvolver um sistema computacional.
- O CMMI é um modelo de maturidade de melhoria de processos para o desenvolvimento de produtos e serviços. Consiste de melhores práticas que endereçam desde atividades de desenvolvimento até a manutenção do produto e cobrem o ciclo de vida inteiro do projeto, desde sua concepção, passando pela entrega e posterior manutenção.
- Os ambientes de *software* orientados a processos surgiram com a intenção de fornecer um apoio mais efetivo aos desenvolvedores no sentido de auxiliar no aumento da produtividade, melhoria da qualidade e diminuição de custos, através da adoção de PSEEs para automatizar a gerência dos processos.
- A execução de processos de *software* é praticada sempre que o modelo adotado está pronto para ser executado. Pode-se dizer que o processo está pronto para execução quando há interação entre tudo o que está envolvido nas atividades de desenvolvimento: desenvolvedores, ferramentas e processos. A fase de execução dos processos não é totalmente automatizada. Ela apenas depende de um processo automatizado para orientar os desenvolvedores a coordenarem melhor os trabalhos em desenvolvimento, e também para tornar possível a execução automática de tarefas repetitivas.
- A execução dos processos de *software* depende diretamente da linguagem adotada na etapa da modelagem dos processos, sendo que na maioria das vezes o nível de formalidade adotado é baixo.
- A execução de processos divide-se em:
 - **Procedimental:** é um processo de baixo nível e considera que os processos de *software* devem ser modelados seguindo conceitos da programação convencional, pois considera que um processo de *software* também é *software*.

- o **Baseada em regras:** os processos são modelados como regras que possuem pré e pós-condições, como ocorre no desenvolvimento de sistemas especialistas.
 - o **Evento-condição-ação:** A execução ECA permite identificar eventos antes da regra ser executada, podendo disparar as regras que satisfazem a condição do evento.
 - o **Redes de Petri:** se apoia no formalismo matemático das Redes Petri, que se utilizam de grafos focados em dois nodos específicos: lugares e transições. Lugares representam os objetos ou recursos e as transições, as etapas do processo.
 - o **Redes de tarefas:** é um grafo cuja função é representar a estrutura do processo, sendo que os nodos representam atividades e os arcos apresentam o fluxo de controle/dados entre as diversas atividades.
- As pessoas devem ser sempre orientadas e instruídas durante a execução dos processos de *software*, e isso depende, na maioria das vezes, do tipo de interação escolhida para o ambiente do processo que está em execução. É necessário que ocorra a interação entre os envolvidos; do contrário, coloca-se o processo todo em risco. Um exemplo de interação sempre é percebido entre gerentes e desenvolvedores ou analistas de sistemas e desenvolvedores.
 - Os tipos de orientações de processos para desenvolvedores são:
 - o **Orientação passiva:** entrega aos desenvolvedores informações para que eles possam decidir corretamente em relação às ações a serem tomadas conforme definição dos processos.
 - o **Orientação ativa:** Notifica os desenvolvedores acerca de suas tarefas ou eventos de maior importância.
 - o **Obrigação do processo:** controla os acessos aos dados e ferramentas de forma direta, obrigando os desenvolvedores a seguirem o processo de acordo com a modelagem especificada.
 - o **Automação de processo:** procura executar partes do processo de forma automática, sem intervenção humana.
 - São quatro os paradigmas de interação na execução de processos de *software*:
 - o **Interação orientada a tarefas:** na prática adota-se uma lista de tarefas que deverão ser executadas.
 - o **Interação orientada a documentos:** neste tipo de interação, o desenvolvedor visualiza o processo como um conjunto de documentos a serem criados ou manipulados.
 - o **Interação orientada a metas:** Este paradigma utiliza o conceito de metas a serem atingidas, sem distinguir quais tarefas devem ser realizadas ou quais documentos devem ser manipulados.
 - o **Interação orientada a ferramentas:** este paradigma depende diretamente das ferramentas de desenvolvimento. Porém, não existem ferramentas para controle de trabalho ou *feedback* do desenvolvedor. São consideradas apenas as ferramentas utilizadas para o desenvolvimento dos trabalhos.

- A flexibilidade na execução de processos necessita de:
 - **Modificação dinâmica durante a execução:** deve ser permitido alterar todos os processos, inclusive os que já estão sendo executados, sem que ele apresente inconsistências.
 - **Execução de processos incompletos:** deve ser possível iniciar um processo mesmo que este não esteja completo. Porém, deve-se prever que ele será detalhado posteriormente.
 - **Instanciação das atividades do processo durante a execução:** A alocação de pessoas e recursos pode ser feita após o processo ter sido iniciado.
 - **Escolhas entre caminhos alternativos:** o projetista do processo deve prever caminhos alternativos para que o processo possa mudar o trajeto sem ser necessário que alguém fique monitorando a situação.
 - **Gerência e tratamento de eventos:** Ao invés de monitorar exaustivamente a execução do processo, o gerente pode programar o mecanismo de execução para agir em determinadas situações de forma automática.

AUTOATIVIDADE



1 Defina Processo de *Software*.

2 Um processo de *software* é um conjunto de atividades relacionadas que levam à produção de um produto de *software*. Cite as etapas de um processo de *software*.



3 Através do CMMI, uma organização pode ter sua maturidade medida em cinco níveis. Cite e especifique cada nível.

4 Com relação aos Requisitos de *Software*, avalie se as afirmativas a seguir como falsas (F) ou verdadeiras (V):

- Requisitos funcionais determinam os serviços ofertados pelo sistema e seu comportamento.
- Requisitos não funcionais determinam as restrições que os serviços ou funcionalidades do sistema apresentam.
- Requisitos funcionais são aqueles não diretamente relacionados às funções fornecidas pelo sistema, enquanto que os não funcionais descrevem a função do sistema detalhadamente, incluindo as entradas e saídas.

Marque a opção que indica a resposta CORRETA:

- V – V – F.
- F – V – V.
- V – V – F.
- F – V – F.

5 Imagine a seguinte situação: um analista de sistemas se depara com um problema para o qual precisa encontrar a solução. Isso ocorre em uma etapa específica do desenvolvimento, em que se deve definir o que o sistema vai fazer, sem saber exatamente como isso será feito. A situação descrita é a fase de:

- Projeto do *software*: A validação verifica se os modelos construídos estão em conformidade com os requisitos do cliente.
- Lvantamento de requisitos: A validação executa diversas atividades a fim de se validar o produto de *software*, testando cada funcionalidade de cada módulo.
- Levantamento de Requisitos: A verificação tem por objetivo assegurar que o sistema de *software* está atendendo às reais necessidades do cliente.

- d) Análise de requisitos: A verificação executa diversas atividades a fim de se testar se cada funcionalidade de cada módulo do *software* funcionará adequadamente.
- e) Análise de requisitos: A validação tem por objetivo assegurar que o sistema de *software* está atendendo às reais necessidades do cliente.

EXECUÇÃO DE PROCESSO DE SOFTWARE

1 INTRODUÇÃO

É comum acontecerem mudanças nos processos de *software* adotados nos ambientes de desenvolvimento. Isso ocorre com muita frequência se considerarmos os ambientes altamente instáveis no que se refere às pessoas e tecnologias envolvidas na construção dos aplicativos.

Algumas mudanças são estáticas: acontecem antes da execução dos processos. Geralmente são mudanças organizacionais ou que remetem às experiências de projetos anteriores. Existem também as mudanças dinâmicas: estas podem ocorrer durante a execução de processos e podem ter origem em falhas ou erros detectados durante a execução.

As mudanças dinâmicas são as mais críticas e de difícil tratamento. Estas mudanças afetam diretamente o andamento das execuções, necessitando de um adequado suporte por parte da linguagem de modelagem do processo e do próprio mecanismo de execução (SILVA, 1999).

2 REQUISITOS DE EXECUÇÃO DE PROCESSOS

Os requisitos de processos de *software* são apresentados em três categorias principais:

2.1 REQUISITOS DE PRESCRIÇÃO

O requisito de prescrição existe para garantir que o processo seja executado conforme foi pré-escrito.

Verifique, no quadro a seguir, o resumo dos requisitos de prescrição de processos propostos pelos autores Bandinelli et al. (1992), Dowson e Fernström (1994).

QUADRO 2 – REQUISITOS DE PRESCRIÇÃO DE PROCESSOS

Id	Requisitos de Prescrição	
R1	Fluxo de Controle	A seqüência de atividades no processo modelado deve ser obedecida de acordo com o paradigma de execução adotado (execução ativa, passiva ou obrigação).
R2	Automação de Processo	Ativar automaticamente as atividades que podem ser executadas sem intervenção humana, através de uma integração com as ferramentas do ambiente.
R3	Gerência de Objetos	O armazenamento persistente de todos os dados envolvidos no processo, com controle dos direitos de acesso e gerência de versões.
R4	Registro da história do processo	Coletar dados da evolução do processo para permitir que o processo melhore onde houver necessidade e seja corrigido para atender novos requisitos.
R5	Coleta de Métricas	Prover mecanismos para coleta automática de dados sobre o processo, o produto e os participantes, assim como geração de estimativas quando possível.
R6	Iteração	Algumas seqüências de atividades podem necessitar de repetição. Deste modo, a ativação automática de uma atividade a ser repetida é tarefa do mecanismo de execução.
R7	Restrições e alocação de recursos	Respeito às restrições da execução e gerência da alocação de recursos a fim de saber quem está trabalhando com o que e quais recursos são necessários para quais atividades.

FONTE: Adaptado de Souza, Reis e Reis (2001)

2.2 REQUISITOS DE INTERAÇÃO

Outros requisitos de extrema importância na execução dos processos são os que remetem à interação entre os usuários que necessitam dela para decidir sobre as questões do ambiente e do processo sendo executado. Souza, Reis e Reis (2001) estudaram os requisitos de interação com base na perspectiva de gerentes e desenvolvedores. O quadro a seguir traz o resumo dele, tomando como base o estudo deste autor:

QUADRO 3 – REQUISITOS DE INTERAÇÃO COM DESENVOLVEDORES

R9- Intereração com desenvolvedores	R9.1	Orientar desenvolvedores nas suas tarefas	Informar desenvolvedores sobre tarefas atribuídas a eles, quais documentos devem manipular e quais os resultados esperados.
	R9.2	Fornecer visualização adequada das tarefas do processo	Definir como o desenvolvedor irá visualizar o processo. A visualização pode ser orientada a tarefas, orientada a documentos, tarefas e documentos, metas ou orientada a ferramentas (SOUZA et al., 2001).
	R9.3	Obter <i>feedback</i> do andamento do processo	O <i>feedback</i> pode ser fornecido explicitamente pelo desenvolvedor (formulários, agendas) ou diretamente pelo PSEE (análise de eventos ocorridos).
	R9.4	Fornecer visualização dos estados do processo (atual e anteriores) e mecanismo de <i>undo</i>	Mesma visualização definida para o gerente, mas restrita aos interesses de desenvolvedores.
	R9.5	Flexibilizar a interação	Permitir que a orientação do processo possa ser adaptada durante execução. Esta interação é geralmente determinada pela PML.
R10 - Interação entre desenvolvedores	R10.1	Permitir comunicação informal	Através de mecanismos para comunicação síncrona e assíncrona.
	R10.2	Permitir gerência de reuniões e horários	Fornecer mecanismos para discutir detalhes sobre a definição de prazos e horários de grupo
	R10.3	Permitir monitoração de produtos e processos	Permitir que os desenvolvedores acompanhem o andamento do processo, estando conscientes, por exemplo de quem manipulou produtos
	R10.4	Controlar o acesso aos objetos	Identificar usuários e seus papéis e definir direitos de acesso aos objetos
	R10.5	Múltiplos níveis de compartilhamento de objetos	Permitir compartilhamento de objetos de qualquer tamanho e granulosidade variável.
	R10.6	Registro do histórico dos objetos e mecanismos de <i>undo</i> e <i>redo</i>.	Gerenciar versões dos objetos e permitir desfazer e refazer as últimas alterações.

FONTE: Adaptado de Souza, Reis e Reis (2001)

2.3 REQUISITOS DE FLEXIBILIDADE

Requisitos de flexibilidade são requisitos que passaram a enfatizar princípios como interação com o cliente em todas as fases do projeto, com alterações em todas estas fases e auto-organização da equipe de acordo com o projeto que está sendo desenvolvido.

O quadro a seguir apresenta os requisitos de flexibilidade, já apontados anteriormente. Observe que alguns requisitos de flexibilidade também são requisitos de interação.

QUADRO 4 – REQUISITOS DE INTERAÇÃO COM DESENVOLVEDORES

Id		Requisitos de Flexibilidade
R11	Modificação dinâmica durante a execução	O ambiente deve permitir alteração do processo durante a execução mantendo a sua consistência. A alteração pode atingir o fluxo de controle (dependências entre atividades), o conteúdo das atividades (por exemplo, o script e datas que prescrevem o comportamento em uma atividade), a alocação de desenvolvedores e recursos, dentre outras.
R12	Execução de Processos Incompletos	O processo deve iniciar sua execução mesmo que alguns componentes estejam faltando, como por exemplo, atividades, fluxos de controle e detalhes de alocação.
R13	Instanciação do processo durante execução	As informações sobre pessoas e recursos para atividades poderão ser definidas durante a execução. Esta característica permite que seja iniciada a execução de um processo com partes abstratas, que serão instanciadas somente quando estiverem aptas a executar. Dessa forma, a alocação de pessoas e recursos pode ser adiada.
R14	Escolhas entre caminhos alternativos	No decorrer da execução podem surgir decisões previstas ou não em tempo de modelagem. O mecanismo de execução deve poder tomar decisões automaticamente com base em condições que consultam o estado corrente do processo ou informações do histórico do ambiente. Essa tomada de decisão consiste em uma escolha de caminho alternativo para continuação do processo.
R15	Adaptação ao usuário	O mecanismo de execução deve adaptar a sua interação de acordo com o perfil do usuário. Isso evita que o ambiente tome-se muito intrusivo para desenvolvedores experientes e permite que ele auxilie mais os desenvolvedores novatos.
R16	Gerência e tratamento de eventos	O mecanismo de execução deve manter o registro de todos os eventos ocorridos e ainda deve possibilitar a definição de estratégias para tratamento desses eventos. Assim, é possível que o próprio mecanismo de execução seja capaz de modificar o processo durante a execução em resposta a algum evento. Além disso, esse requisito livra o gerente da tarefa de monitorar todas as ocorrências do processo.

FONTE: Adaptado de Souza, Reis e Reis (2001)

3 MODELOS DE CICLO DE VIDA OU MODELOS DE PROCESSO

Os modelos representam um esboço do processo, incluindo as principais atividades, sua ordem de execução, requisitos e responsabilidades. Porém, o modelo não descreve ações, recursos e restrições de execução. É apenas um apoio, não sendo isoladamente suficiente para controlar todo o processo de desenvolvimento de software (SOMMERVILLE, 2007).

Cada projeto de software é único, mas algumas etapas do ciclo de vida do processo são necessárias, independentemente do projeto que será construído:

- **Planejamento:** o planejamento permite que o gerente faça adequadamente as estimativas de prazos, custos e recursos para o projeto. O planejamento deverá ser atualizado e revisto conforme a evolução do projeto.
- **Análise e especificação de requisitos:** nesta etapa faz-se necessária a compreensão do problema através da elaboração e refinamento dos requisitos, bem como a funcionalidade e os resultados esperados acerca da aplicação que será construída.
- **Projeto:** Nesta fase são definidas a arquitetura do projeto e as especificações do *software*, tomando por base os requisitos e o modelo de processo adotado para desenvolvimento.
- **Implementação:** é a fase de implementação e desenvolvimento da fase de projeto. Consiste em customizar o que foi definido no projeto.
- **Testes:** o objetivo desta etapa é garantir que tudo o que foi especificado, funcione. Para isso é necessário realizar testes integrados no sistema.
- **Operação:** esta é a fase de acompanhamento do uso do *software* junto aos usuários, afim de aferir a estabilidade da aplicação.
- **Manutenção:** é a correção de falhas, erros ou implementação de melhorias solicitadas/apontadas pelos usuários durante a fase de operação.

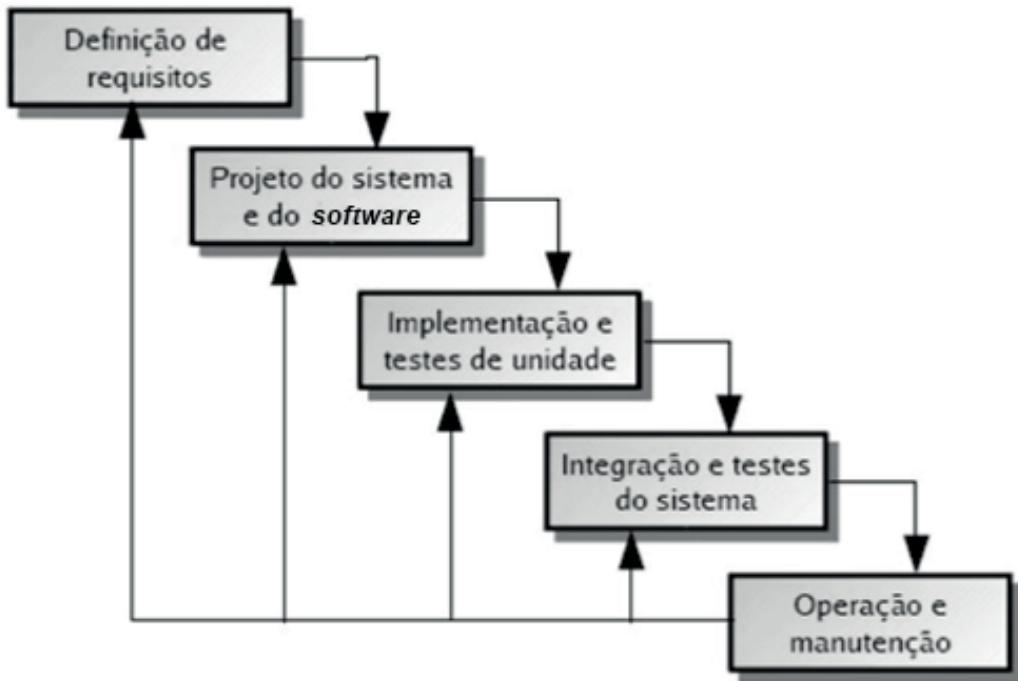
Os principais modelos de ciclo de vida podem ser agrupados em três categorias principais: modelos sequenciais, modelos incrementais e modelos evolutivos.

3.1 MODELOS SEQUENCIAIS

Basicamente, este modelo organiza o processo em uma sequência linear de fases. O modelo em cascata é a principal forma de representação do modelo sequencial, sendo que cada fase necessita da elaboração de um ou mais documentos, bem como sua aprovação, antes de dar início à fase seguinte (PRESSMANN, 2011).

Cada fase finalizada passa por revisão e envolve diretamente o usuário que aprova ou não o trabalho realizado. Isso facilita a gerência das atividades.

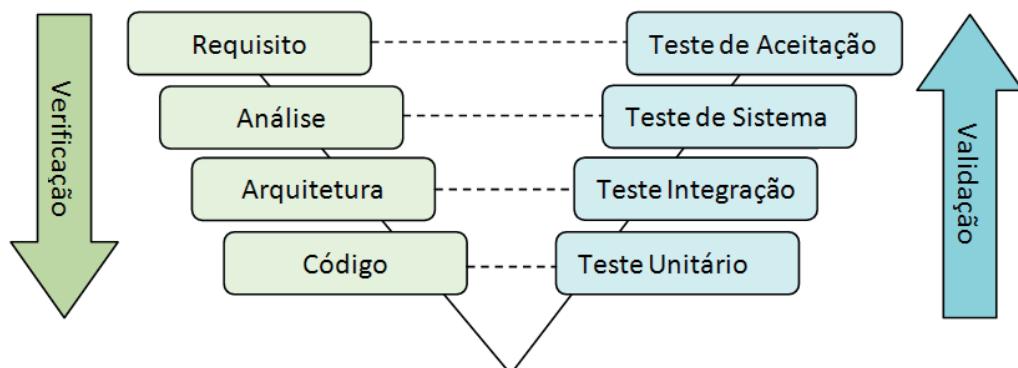
FIGURA 5 – MODELO EM CASCATA



FONTE: Sommerville (2007)

Uma variação do modelo em cascata é o modelo em V, que tem como foco a relação entre as atividades de teste (teste de unidade, teste de integração, teste de sistema e teste de aceitação) e as outras atividades do processo. A figura a seguir exibe o fluxo de processo do modelo em V:

FIGURA 6 – MODELO EM V

FONTE: Disponível em: <<http://www.devmedia.com.br/teste-de-integracao-na-pratica/31877>>. Acesso em: 5 ago. 2016.

O modelo em V verifica através dos testes, a implementação e o projeto detalhado, servindo para a avaliação do projeto.

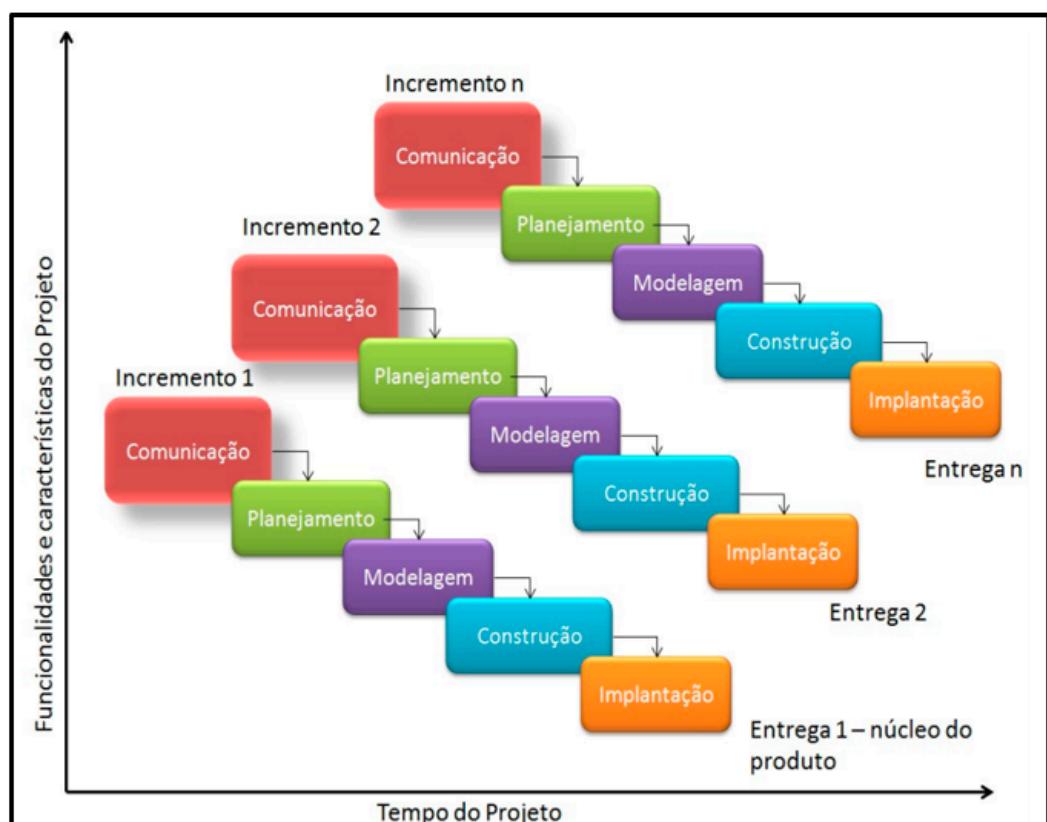
Uma característica dos modelos sequenciais (modelo em cascata e em V), é que o sistema é entregue em sua totalidade, após a realização de todas as atividades previstas.

3.2 MODELOS INCREMENTAIS

Mesmo com um adequado levantamento de requisitos, muitas vezes, o modelo sequencial não atende à necessidade de especificação de aplicações maiores. Neste sentido, pode-se fazer uso do modelo incremental, que possibilita a divisão de um sistema em subsistemas ou módulos.

A figura a seguir mostra as principais possibilidades do modelo incremental:

FIGURA 7 – MODELO INCREMENTAL



FONTE: Sommerville (2007)

Podemos citar como vantagens do modelo incremental:

- Menor custo e menos tempo são necessários para se entregar a primeira versão.
 - Os riscos associados ao desenvolvimento de um incremento são menores, devido ao seu tamanho reduzido.
 - O número de mudanças nos requisitos pode diminuir devido ao curto tempo de desenvolvimento de um incremento.

Como desvantagens, podemos citar:

- Se os requisitos não são tão estáveis ou completos quanto se esperava, alguns incrementos podem ter de ser bastante alterados.

3.3 MODELOS EVOLUTIVOS OU EVOLUCIONÁRIOS

A grande diferença dos modelos evolutivos para os modelos incrementais é que os incrementais entregam as versões operacionais desde a primeira fase ou ciclo, e os evolutivos não se preocupam com isso; nas primeiras fases geralmente são desenvolvidos apenas os protótipos. Porém, este modelo exige ampla gerência de projeto e configuração. Como exemplos de modelos evolutivos podemos citar o modelo Espiral e de Prototipação.

FIGURA 8 – MODELO EVOLUTIVO

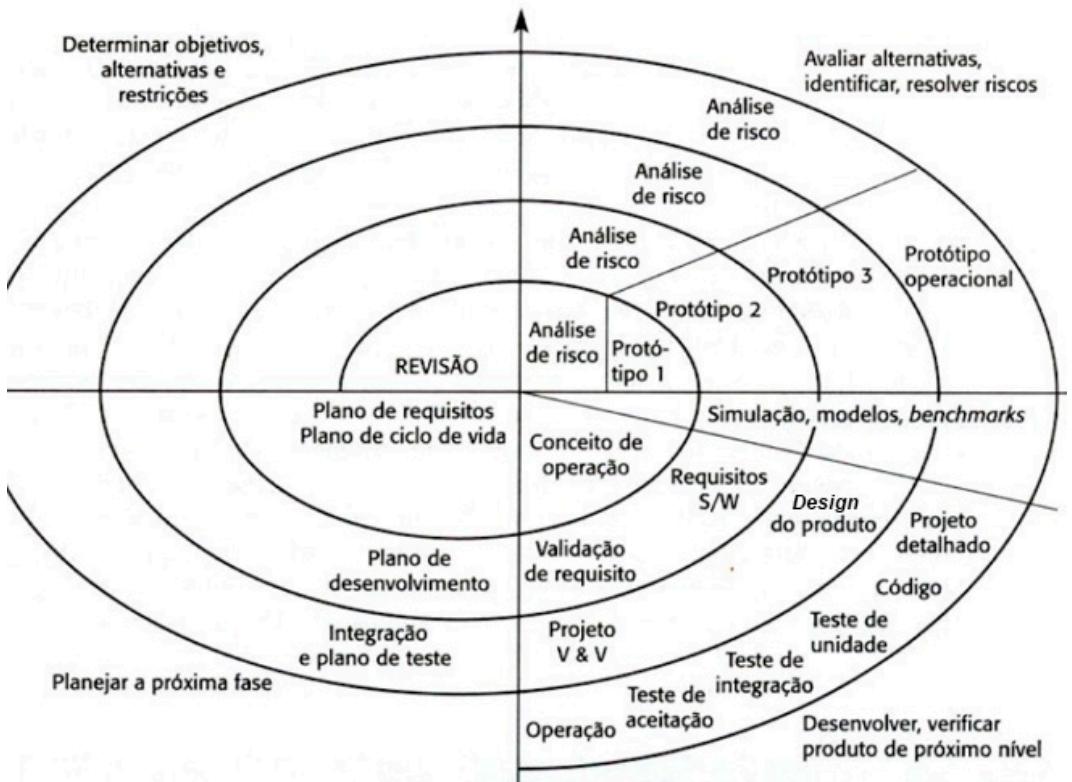


FONTE: Sommerville (2007)

3.3.1 O modelo espiral

O modelo espiral, mostrado na figura a seguir, é um dos modelos evolutivos mais difundidos.

FIGURA 9 - MODELO ESPIRAL

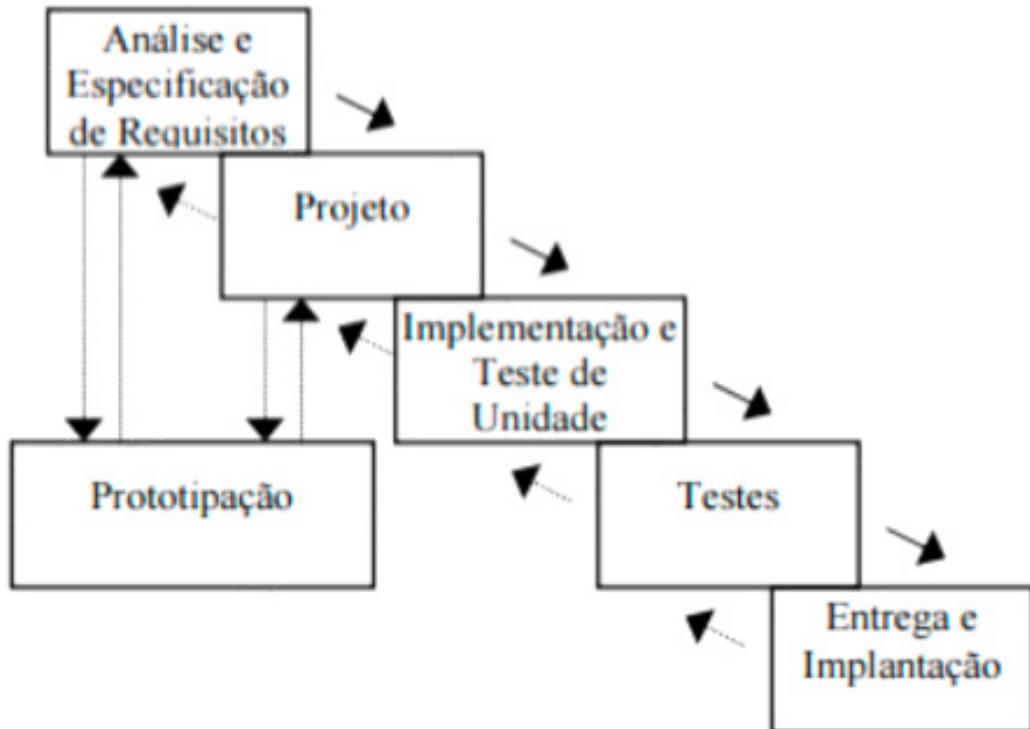


FONTE: Sommerville (2007)

3.3.2 Prototipação

A prototipação auxilia tanto as equipes de desenvolvimento quanto o usuário final no entendimento e identificação das funcionalidades, pois coloca ambos diante do sistema. A prototipação é uma técnica que pode ser usada em qualquer modelo de processo, não sendo restrita apenas ao modelo Espiral. A figura a seguir ilustra um modelo em cascata com prototipação.

FIGURA 10 - MODELO DE PROTOTIPAGEM



FONTE: Sommerville (2007)

Em caso de necessidade de esclarecimentos acerca dos modelos apresentados, você pode consultar o Caderno de Engenharia de *Software*, que trata deste tema de forma aprofundada.

4 NORMAS E MODELOS DE QUALIDADE DE PROCESSO DE SOFTWARE

Você deve ter percebido, pelo estudo feito até aqui, que os modelos se preocupam apenas com as atividades básicas do processo e suas relações, servindo de orientação para o desenvolvimento. Mas, não são suficientes. Neste sentido, vários modelos e normas de qualidade de processo surgiram como suporte para a melhoria contínua na qualidade de processos. Podemos destacar as seguintes normas e modelos: NBR ISO 9000-2000, NBR ISO/IEC 12207, ISO/IEC 15504 e os modelos CMM, CMMI e MPS.BR. As normas da família NBR ISO 9000 foram desenvolvidas para apoiar organizações, de todos os tipos e tamanhos, na implementação e operação de sistemas eficazes de gestão da qualidade (KOSCIANSKI, 2007).

As normas que compõem a série ISO 9000-2000 são:

- NBR ISO 9000: voltada para os fundamentos de sistemas de gestão da qualidade determina a terminologia para esses sistemas.
- NBR ISO 9001: especifica os requisitos de qualidade de um sistema com foco para a satisfação do cliente.
- NBR ISO 9004: estabelece diretrizes que consideram tanto a eficácia como a eficiência do sistema de gestão da qualidade, tendo como objetivo a melhoria do desempenho de todas as partes interessadas no projeto.
- NBR ISO 19011: é uma norma certificadora. A empresa que a conquista tem sua qualidade reconhecida mundialmente. Porém, ela é de caráter geral e quando aplicada ao contexto de *software*, ainda carece de diretrizes específicas para o setor.

Outras normas são adotadas por organizações de desenvolvimento de *software* como suporte para a conquista de uma certificação:

- A norma NBR ISO/IEC 12207 – Tecnologia da Informação – Processos de Ciclo de Vida de *Software*: estabelece uma estrutura (processos, atividades, tarefas) comum e terminologia bem definida e apropriada aos processos de desenvolvimento de *software*.
- A Norma ISO/IEC 15504: Tecnologia da Informação – Avaliação de Processos: é um padrão internacional ISO para avaliação de processos de *software*. Estabelece diretrizes para avaliação de processos de *software* com os seguintes objetivos:
 - o Permitir que a organização entenda os processos visando à melhoria deles.
 - o Determinar a adequação dos processos de uma organização para atender a um requisito particular ou classe de requisitos.
 - o Determinar a adequação de processos da organização para um contrato ou classe de contratos. Essa norma pode ser usada tanto por quem compra o *software* para determinar a capacidade dos processos de *software* de seus fornecedores, quanto por fornecedores para determinar a capacidade de seus próprios processos ou para identificar oportunidades de melhoria.

De acordo com Pessoa (2003), é composta de cinco partes:

- **Conceitos e vocabulário:** introduz os conceitos gerais de avaliação dos processos e propõe um glossário de termos.
- **Avaliação:** é uma norma e estabelece os requisitos mínimos para se realizar uma avaliação. Essa parte é a única que tem caráter normativo.
- **Orientações para se realizar uma avaliação:** orienta na interpretação dos requisitos para a realização da avaliação.

- **Orientações para uso em melhoria de processo e determinação da capacidade de processo:** orienta na interpretação e utilização dos resultados da avaliação na melhoria contínua dos processos.
- **Um exemplo de modelo de avaliação de processo:** contém um exemplo de modelo de avaliação de processo baseado no modelo de referência da ISO/IEC 12207.

Os Modelos CMM e CMMI: O Modelo de Maturidade e Capacidade (*Capability Maturity Model – CMM*) foi desenvolvido pelo Instituto de Engenharia de Software (*Software Engineering Institute – SEI*) da Universidade de Carnegie Mellon. A ideia inicial era de quantificar a capacidade de uma organização produzir produtos de *software* com qualidade (KOSCIANSKI, 2007).

O CMM foi estruturado em cinco níveis: 1 – menos maduro / 5 – mais maduro. A empresa que obtém esta certificação demonstra a sua capacidade de processo de acordo com o nível conquistado no certificado.

O Modelo de Referência Brasileiro – MPS.BR O MPS.BR – Melhoria de Processo do *Software* Brasileiro tem como objetivo definir um modelo de melhoria e avaliação de processo de *software*, adequado, preferencialmente, as micro, pequenas e médias empresas brasileiras, de forma a atender as suas necessidades de negócio e a ser reconhecido nacional e internacionalmente como um modelo aplicável à indústria de *software*. Por este motivo, está adepto a modelos e normas internacionais (PESSOA, 2003). O modelo brasileiro apresenta dois processos que não são encontrados no CMMI: desenvolvimento para reutilização e gerência de portfólio de projetos.

Ainda de acordo com o autor, a base técnica utilizada para a construção do MPS.BR é composta pelas normas NBR ISO/IEC 12207 e suas emendas 1 e 2 e a ISO/IEC 15504, estando totalmente aderente a essas normas. Além disso, o MPS.BR também cobre o conteúdo do CMMI. O MPS.BR está dividido em três componentes:

- **Modelo de Referência (MR-MPS):** contém os requisitos que as organizações deverão atender para estar em conformidade com o MPS.BR. Define, também, os níveis de maturidade e da capacidade de processos e os processos em si.
- **Método de Avaliação (MA-MPS):** contém o processo de avaliação, os requisitos para os avaliadores e os requisitos para averiguação da conformidade ao modelo MR-MPS. Está descrito de forma detalhada no Guia de Avaliação e foi baseado na norma ISO/IEC 15504.
- **Modelo de Negócio (MN-MPS):** contém uma descrição das regras para a implementação do MR-MPS pelas empresas de consultoria, de *software* e de avaliação. O MR-MPS define sete níveis de maturidade: A (Em Otimização),

B (Gerenciado Quantitativamente), C (Definido), D (Largamente Definido), E (Parcialmente Definido), F (Gerenciado) e G (Parcialmente Gerenciado). A escala de maturidade se inicia no nível G e progride até o nível A. Para cada um desses sete níveis de maturidade, foi atribuído um perfil de processos e de capacidade de processos que indicam onde a organização tem que colocar esforço para melhoria de forma a atender os objetivos de negócio.

5 MELHORIA DO PROCESSO DE SOFTWARE BRASILEIRO

A qualidade de *software* depende da capacitação dos processos. Há pouco investimento das empresas em certificações que comprovem a qualidade e a maturidade dos seus processos na fabricação de *software*, impossibilitando a venda deste produto no mercado internacional. Para as pequenas empresas, esse investimento é dificultado devido ao alto valor das certificações. Graças à iniciativa de entidades privadas, centros de estudos e governo brasileiro, existe a possibilidade de melhorarmos os processos de *software* no Brasil, tendo como foco pequenas e médias empresas.

O Projeto mps BR – Melhoria de Processo do *Software* Brasileiro, projeto coordenado pela SOFTEX é apontado como solução para tornar o *software* brasileiro um produto de exportação competitivo geração de negócios no Brasil e no exterior, visando aumentar a competitividade da indústria brasileira de *software*. O Projeto mps BR Iniciado em 2003, a partir de dezembro, conta com a participação de sete instituições brasileiras: a SOFTEX, coordenadora do projeto; três instituições de ensino, pesquisa e centros tecnológicos (COPPE / UFRJ, CESAR, CenPRA); uma sociedade de economia mista, a companhia de Informática do Paraná (CELEPAR), hospedeira do Subcomitê de *Software* da Associação Brasileira de Normas Técnicas (ABNT); e duas organizações não governamentais integrantes do Programa SOFTEX (Sociedade Núcleo de Apoio à Produção e Exportação de *Software* do Rio de Janeiro – RIOSOFT e Sociedade Núcleo SOFTEX 2000 de Campinas).

A Universidade Católica de Brasília também faz arte do projeto. O MPS. Br serve como um selo que indica o nível de maturidade da empresa em relação às práticas relacionadas ao desenvolvimento de *software*. Esse selo possui níveis. Cada nível tem diversas práticas associadas. Uma empresa que possui o "selo" MPS. Br utiliza essas boas práticas e, teoricamente, tem bastantes condições de desenvolver softwares com qualidade e com custos e prazos dentro do estimado. MPS. Br é um movimento importante porque pode ajudar a melhoria dos processos para as organizações.

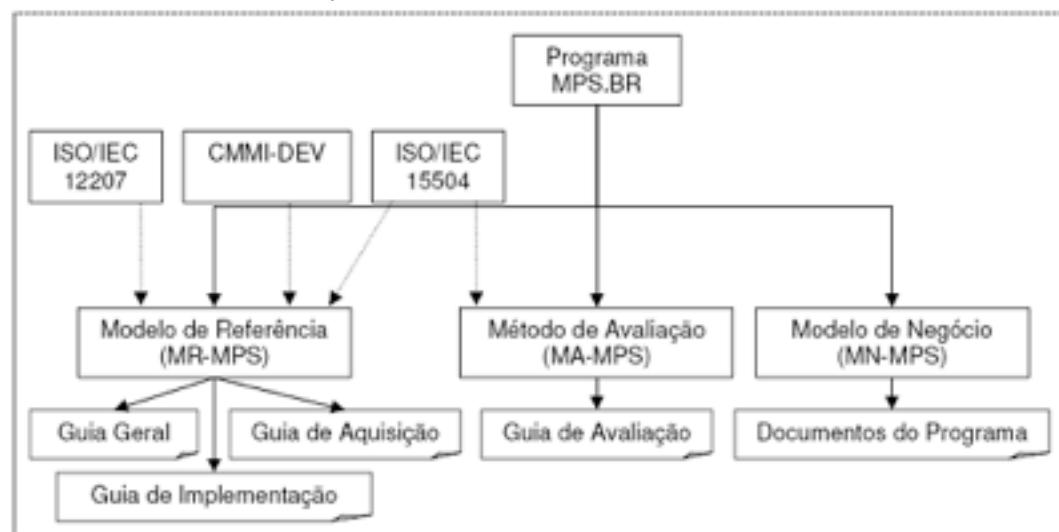
Além disso, ele já está sendo exigido em algumas licitações. Assim, somente empresas certificadas podem participar de alguns processos licitatórios.

5.1 Descrição geral do MPS.BR

Uma das metas do MPS-BR visa definir e aprimorar um modelo de melhoria e avaliação de processo de *software*, visando preferencialmente as micro, pequenas e médias empresas, de forma a atender as suas necessidades de negócio e ser reconhecido nacional e internacionalmente como um modelo aplicável à indústria de *software*. O MPS-BR também estabelece um processo e um método de avaliação, o qual dá sustentação e garante que o MPS-BR está sendo empregado de forma coerente com as suas definições.

A base técnica para a construção e aprimoramento deste modelo de melhoria e avaliação de processo de *software* é composta pelas normas NBR ISO/IEC 12207 – Processo de Ciclo de Vida de *Software*, pelas emendas 1 e 2 da norma internacional ISO/IEC 12207 e pela ISO/IEC 15504 – Avaliação de Processo, portanto, o modelo está em conformidade com essas normas. O programa mobilizador MPS.BR está dividido em três (3) componentes: Modelo de Referência (MR-MPS); Método de Avaliação (MA-MPS); Modelo de Negócio (MN-MPS).

FIGURA 11 – REPRESENTAÇÃO DO MODELO MPS - BR



FONTE: Disponível em: <<https://www.oficinadanet.com.br/artigo/desenvolvimento/melhoria-de-processos-do-software-brasileiro--mpsbr>>. Acesso em: 18 ago. 2016.

5.2 MR-MPS (Modelo de referência para melhoria do processo de software)

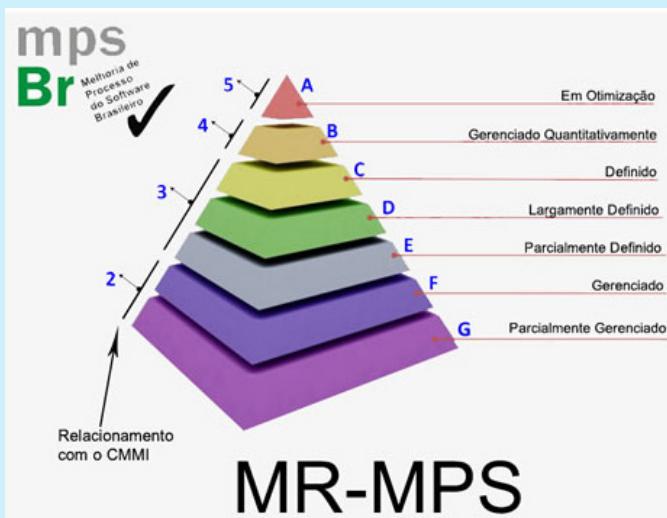
O Modelo de Referência MR-MPS define níveis de maturidade que são uma combinação entre processos e sua capacidade. Isso permite avaliar e atribuir graus de efetividade na execução dos processos. As atividades e tarefas necessárias para atender ao propósito e aos resultados.

5.2.1 Níveis de maturidade

Os níveis de maturidade estabelecem patamares de evolução de processos, caracterizando estágios de melhoria da implementação de processos na organização. 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 (Em Otimização).
- B (Gerenciado Quantitativamente).
- C (Definido).
- D (Largamente Definido).
- E (Parcialmente Definido).
- F (Gerenciado).
- G (Parcialmente Gerenciado).

FIGURA 12 – REPRESENTAÇÃO DOS NÍVEIS DO MPS – BR

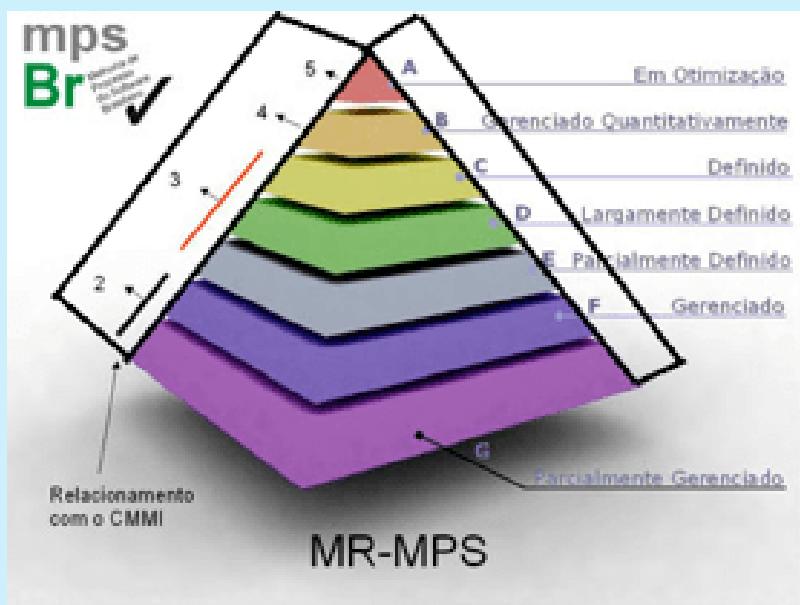


FONTE: Disponível em: <<https://www.oficinadanet.com.br/artigo/desenvolvimento/melhoria-de-processos-do-software-brasileiro--mpsbr>>. Acesso em: 18 ago. 2016.

A escala de maturidade se inicia no nível G e progride até o nível A. Para cada um destes sete níveis de maturidade é atribuído um perfil de processos que indicam onde a organização deve colocar o esforço de melhoria. O progresso e o alcance de um determinado nível de maturidade MPS se obtêm, quando são atendidos os propósitos e todos os resultados esperados dos respectivos processos e dos atributos de processo estabelecidos para aquele nível.

A divisão em estágios, embora baseada nos níveis de maturidade do CMMISE/SWSM tem uma graduação diferente, com o objetivo de possibilitar uma implementação e avaliação mais adequada às micro, pequenas e médias empresas. A possibilidade de se realizar avaliações considerando mais níveis também permite uma visibilidade dos resultados de melhoria de processos em prazos mais curtos.

FIGURA 13 - COMPARAÇÃO ENTRE OS NÍVEIS DO CMMI E OS DO MPS.br



FONTE: Disponível em: <<https://www.oficinadanet.com.br/artigo/desenvolvimento/melhoria-de-processos-do-software-brasileiro--mpsbr>>. Acesso em: 18 ago. 2016.

5.3 Processos

Cada nível de maturidade possui suas áreas de processo, onde são analisados os processos fundamentais (aquisição, gerência de requisitos, desenvolvimento de requisitos, solução técnica, integração do produto, instalação do produto, liberação do produto), processos organizacionais (gerência de projeto, adaptação do processo para gerência de projeto, análise de decisão e resolução, gerência de riscos, avaliação e melhoria do processo organizacional, definição do processo organizacional, desempenho do processo organizacional, gerência quantitativa do projeto, análise e resolução de causas, inovação e implantação na organização) e os processos de apoio (garantia de qualidade, gerência de configuração, validação, medição, verificação, treinamento).

Obs.: O processo de aquisição pode ser excluído sem causar danos, desde que o mesmo não seja utilizado pela organização.

5.3.1 Capacidade do processo

A capacidade do processo é representada por um conjunto de atributos de processo descrito em termos de resultados esperados. A capacidade do processo expressa o grau de refinamento e institucionalização com que o processo é executado na organização. No MPS, à medida que a organização evolui nos níveis de maturidade, um maior nível de capacidade para desempenhar o processo deve ser atingido pela organização. O atendimento aos atributos do processo (AP), através do atendimento aos resultados esperados dos atributos do processo (RAP) é requerido para todos os processos no nível correspondente ao nível de maturidade, embora eles não sejam detalhados dentro de cada processo.

Os níveis são acumulativos, ou seja, se a organização está no nível F, esta possui o nível de capacidade do nível F que inclui os atributos de processo dos níveis G e F para todos os processos relacionados no nível de maturidade F (que também inclui os processos de nível G).

A capacidade do processo no MPS possui cinco (5) atributos de processos (AP) que são:

- AP 1.1 - O processo é executado.
- AP 1.2 - O processo é gerenciado.
- AP 2.2 - Os produtos de trabalho do processo são gerenciados.
- AP 3.1 - O processo é definido.
- AP 3.2 - O processo está implementado.

5.3.2 MA-MPS – Método de avaliação para melhoria do processo de software

É composto basicamente pelo processo de avaliação MPS, método de avaliação MPS e características da qualificação dos avaliadores. Este método permite a realização de avaliações segundo o Modelo MPS. O MA-MPS foi definido em conformidade com os requisitos para modelos de referência de processo e métodos de avaliação de processos estabelecidos na norma ISO/IEC 15504-2 e atende aos requisitos específicos do Programa MPS.BR. Desta forma, o método está em conformidade com a ISO/IEC 15504 e é compatível com o método SCAMPI [1] para avaliação segundo o modelo CMMI, também definido com base na ISO/IEC 15504.

O processo de avaliação é composto por quatro subprocessos:

- Contratar a avaliação.
- Preparar a realização da avaliação.
- Realizar a avaliação.
- Documentar os resultados da avaliação.

5.3.3 MN-MPS – Modelo de negócio para melhoria do processo de software

Instituições que se propõem a implantar os processos MPS.Br (Instituições Implementadoras) podem se credenciar através de um documento onde é apresentada a instituição proponente, contendo seus dados com ênfase na experiência em processos de *software*, estratégia de implementação do modelo, estratégia para seleção e treinamento de consultores para implementação do MR.MPS, estratégia para seleção e treinamento de avaliadores, lista de consultores de implementação treinados no modelo e aprovados em prova específica, lista de avaliadores treinados no modelo e aprovados em prova específica.

5.3.4 Processo de garantia da qualidade - GQA

O propósito do processo de garantia da qualidade é assegurar que os produtos de trabalho e a execução dos processos estejam em conformidade com os planos, procedimentos e padrões estabelecidos.(MPS.BR – Guia Geral, Setembro 2009, p. 31).

Este processo está dividido em outros 4 procedimentos a serem executados, que são descritos abaixo cada, em paralelo foi sugerido através da metodologia ágil *Scrum* como podemos implementar tais processos.

GQA 1. A aderência dos produtos de trabalho aos padrões, procedimentos e requisitos aplicáveis é avaliada objetivamente, antes dos produtos serem entregues e em marcos predefinidos ao longo do ciclo de vida do projeto; Ideias sobre evidências: Relatórios de auditorias de GQA executadas, segundo plano de GQA em pontos do ciclo de vida do projeto. Por exemplo, auditoria de GQA no final do Pregame, durante os *sprints* do *Development* (*presprint*, *sprint* e *postsprint*) e durante o *postgame*. Os relatórios conterão resultados das auditorias de produtos de trabalho, realizados através de *checklists*.

GQA 2. A aderência dos processos executados às descrições de processo, padrões e procedimentos é avaliada objetivamente; Ideias sobre evidências: Relatórios de auditoria de GQA executado, segundo plano de GQA em pontos do ciclo de vida do projeto. Por exemplo, auditoria de GQA no final do Pregame, durante os *sprints* do *Development* (*presprint*, *sprint* e *postsprint*) e durante o *postgame*. Os relatórios conterão resultados de auditoria de processos, realizados através de *checklists*. A mesma instância de auditoria avaliará tanto produto quanto processo.

GQA 3. Os problemas e as não conformidades são identificados, registrados e comunicados; Ideias sobre evidências: Os problemas detectados durante a auditoria (NC Não conformidades), serão registrados numa ferramenta de *issue-tracker*, por exemplo, que deverá permitir o acompanhamento até a sua resolução. Um dos pontos que poderá ser adotado no *Scrum*, o que potencializará a conscientização sobre erros e, principalmente, como evitá-los, será a discussão dessas NC dentro das reuniões da equipe (*daily scrum* ou *sprint review*), por exemplo.

GQA 4. Ações corretivas para as não conformidades são estabelecidas e acompanhadas até as suas efetivas conclusões. Quando necessário, o escalamento das ações corretivas para níveis superiores é realizado, de forma a garantir sua solução; Ideias sobre evidências: O foco aqui é o acompanhamento da NC até a sua conclusão e a possibilidade de escalonamento, em caso de impasses na resolução. Esse ponto tem um componente cultural relativo ao fato dos colaboradores serem "observados" por alguém de fora do "team" e auditado nos seus erros. A filosofia dos métodos ágeis, por conterem traços de comportamento mais libertos e de autogestão, talvez até facilite este aspecto.

FONTE: Disponível em: <<https://www.oficinadanet.com.br/artigo/desenvolvimento/melhoria-de-processos-do-software-brasileiro--mpsbr>>. Acesso em: 18 ago. 2016.

RESUMO DO TÓPICO 2

Neste tópico, você estudou:

- Os requisitos de execução dos processos: requisitos de interação e flexibilidade.
- Os modelos de ciclo de vida dos processos: Modelo sequencial, Modelo Incremental, Modelo Evolutivo, Modelo Espiral, Prototipação.
- As normas e modelos de qualidade dos processos.
- É comum acontecerem mudanças nos processos de *software* adotados nos ambientes de desenvolvimento. Isso ocorre com muita frequência se considerarmos os ambientes altamente instáveis no que se refere às pessoas e tecnologias envolvidas na construção dos aplicativos. As mudanças podem ser estáticas ou dinâmicas.
- Cada projeto de *software* é único, mas algumas etapas do ciclo de vida do processo são necessárias, independentemente do projeto que será construído:
 - **Planejamento:** o planejamento permite que o gerente faça adequadamente as estimativas de prazos, custos e recursos para o projeto. O planejamento deverá ser atualizado e revisto conforme a evolução do projeto..
 - **Análise e especificação de requisitos:** nesta etapa faz-se necessária a compreensão do problema através da elaboração e refinamento dos requisitos, bem como a funcionalidade e os resultados esperados acerca da aplicação que será construída.
 - **Projeto:** Nesta fase são definidas a arquitetura do projeto e as especificações do *software*, tomando por base os requisitos e o modelo de processo adotado para desenvolvimento.
 - **Implementação:** é a fase de implementação e desenvolvimento da fase de projeto. Consiste em customizar o que foi definido no projeto.
 - **Testes:** o objetivo desta etapa é garantir que tudo o que foi especificado, funcione. Para isso é necessário realizar testes integrados no sistema.
 - **Operação:** esta é a fase de acompanhamento do uso do *software* junto aos usuários, afim de aferir a estabilidade da aplicação.
 - **Manutenção:** é a correção de falhas, erros ou implementação de melhorias solicitadas/apontadas pelos usuários durante a fase de operação.
- Os principais modelos de ciclo de vida podem ser agrupados em três categorias principais: modelos sequenciais, modelos incrementais e modelos evolutivos.
- As normas que compõem a série ISO 9000-2000 são:

- o NBR ISO 9000: voltada para os fundamentos de sistemas de gestão da qualidade determina a terminologia para esses sistemas.
- o NBR ISO 9001: especifica os requisitos de qualidade de um sistema com foco para a satisfação do cliente.
- o NBR ISO 9004: estabelece diretrizes que consideram tanto a eficácia como a eficiência do sistema de gestão da qualidade, tendo como objetivo a melhoria do desempenho de todas as partes interessadas no projeto.
- o NBR ISO 19011: é uma norma certificadora. A empresa que a conquista tem sua qualidade reconhecida mundialmente. Porém, ela é de caráter geral e quando aplicada ao contexto de *software*, ainda carece de diretrizes específicas para o setor.

AUTOATIVIDADE



1 Cite as três categorias de requisitos processos de *software*.



2 Para que servem as Normas e Modelos de Qualidade de Processo de *Software*?

3 Um dos modelos utilizados no desenvolvimento de aplicativos é o modelo evolucionário, cujo desenvolvimento evolui com o tempo. Marque a opção que exemplifica de forma correta este modelo:

- a) UML e de qualidade.
- b) Componentes e arquétipo.
- c) Prototipagem e espiral.
- d) Redes de Petri e certificação.

4 O modelo de processo de *software* caracterizado por intercalar as atividades de especificação, desenvolvimento e validação, e que apresenta evolução em um período de tempo, denomina-se:

- a) Modelo de *workflow*.
- b) Modelo de fluxo de dados.
- c) Modelo de desenvolvimento evolucionário.
- d) Modelo em cascata.

5 O modelo em cascata é o mais conhecido e mais utilizado modelo de desenvolvimento de *software*. Considerando o seu conhecimento em relação a este modelo, marque a opção correta:

- a) O primeiro estágio de desenvolvimento de um novo sistema consiste na definição de requisitos.
- b) A divisão dos requisitos para implementação do sistema em *hardware* ou *software* é feita na fase de operação e manutenção.
- c) A especificação do sistema é produzida após o estágio de implementação e teste de unidade.
- d) A integração e o teste dos programas individuais são feitos no estágio de implementação e teste de unidade.

6 As normas propostas pela MPS.BR (Melhoria de Processo do *Software* Brasileiro) apresentam compatibilidade com as normas do CMMI. Isso significa que existem áreas de processos correspondentes em ambas as propostas. No entanto, algumas áreas de processo da norma brasileira não têm correspondente no CMMI. Marque abaixo a opção que indica processos da (MPS.BR) que não possuem correspondentes no CMMI:

- a) Avaliação e melhoria do processo organizacional e planejamento de projeto.
- b) Análise e resolução de causas e gerência do desempenho organizacional.
- c) Desenvolvimento para reutilização e gerência de portfólio de projetos.
- d) Gerência de reutilização e desempenho dos processos da organização.
- e) Projeto e construção do produto e solução técnica.

7 As definições estabelecidas pela Melhoria de Processo do *Software Brasileiro* (MPS.BR), atualizada em 2012, estabelecem que alguns processos podem ser excluídos de forma parcial ou total da avaliação do processo, por não se enquadarem no segmento de negócio. Marque a opção que indica o processo que pode ser excluído da avaliação, desde que ele não seja utilizado:

- a) Aquisição.
- b) Gerência de recursos humanos.
- c) Validação.
- d) Gerência de portfólio de projetos.
- e) Desenvolvimento para reutilização.

TÉCNICAS DE MODELAGEM, MODELO EKD E REUTILIZAÇÃO DE PROCESSOS DE SOFTWARE

1 INTRODUÇÃO

Este tópico tem o objetivo de descrever as técnicas existentes hoje no âmbito científico/acadêmico no que se refere aos processos de negócios.

Processos de negócio apresentam um objetivo claro, têm entrada e saída bem definidos. São formados por um conjunto de atividades que obedecem a uma ordem de execução e dependem de eventos internos e externos ao processo. Afetam de alguma forma as organizações e geram valor agregado aos clientes (PORTER, 2001)

Uma das formas de obter a qualidade do produto de *software* é realizar o seu desenvolvimento buscando entender claramente o domínio do negócio, considerando os processos de negócio e a compreensão do ambiente organizacional como fontes relevantes para a elicitação de requisitos.

O uso de modelos que descrevem os processos de negócios e o contexto de uma organização agrega benefícios para o desenvolvimento de *software*, tais como:

- os requisitos passam a refletir as necessidades do negócio;
- baixo número de redundâncias de requisitos;
- o desenvolvimento do *software* passa a ser guiado pela necessidade do negócio;

Em cada organização, os processos de negócios apresentam características próprias e, por esta razão, é importante dar atenção à modelagem desses processos, explorando as razões e intenções que motivam os diversos componentes do universo organizacional. No entanto, abstrair o contexto de uma organização pode não ser um trabalho trivial. Portanto, antes de se executar este procedimento, é necessário ter a plena certeza de que a organização se conhece, que entende o seu próprio funcionamento, seus recursos e suas limitações.

A Modelagem Organizacional é um processo onde um modelo empresarial integrado é criado, descrevendo uma empresa específica de várias perspectivas diferentes, como: processos, informações, recursos, pessoal, objetivos e restrições de diversos tipos de organização.

Diferentemente da Modelagem de Processos de Negócio que se detém somente aos elementos componentes dos processos que integram a cadeia funcional do negócio. Tanto os modelos criados pela Modelagem Organizacional quanto os modelos criados pela Modelagem de Processos de Negócios são fontes de informação relevantes. Ambos devem ser considerados na elicitação de requisitos durante o desenvolvimento de novas aplicações ou evoluções de sistemas computacionais.

FONTE: Disponível em: <http://www.cin.ufpe.br/~erbr13/arquivos/proceedings/erbr2013_submission_20.pdf>. Acesso em: 9 ago. 2016.

A seguir são relacionadas e descritas as principais técnicas de modelagem de processos de negócios, bem como os padrões de notação.

2 ABORDAGENS DA MODELAGEM ORGANIZACIONAL

Cada organização tem missão, objetivos e processos próprios e é importante dar atenção à modelagem desses itens. Alencar (1999) destaca os seguintes objetivos da Modelagem Organizacional:

- Fornecer um objeto, que seja uma representação compartilhável e reusável da cadeia de fornecimento de informação e conhecimento.
- Suportar tarefas da cadeia de fornecimento, pela habilitação de respostas a questionamentos, que não estão explicitamente representados no modelo.
- Definir os objetos de maneira precisa, de forma que sejam consistentemente aplicados, por meio dos domínios e interpretados pelos usuários.
- Suportar visualização do modelo, de forma intuitiva, simples e consistente.

Muitas técnicas de Modelagem Organizacional são propostas, algumas com o foco principal nos aspectos sociais como em Dobson (1994) que descreve os objetivos, política e estrutura da organização. Na linha de Bubenko (1994), Yu (1993) e Rolland et al. (2000), é realizada a Modelagem Organizacional com múltiplas visões com análise de metas e objetivos da organização. A organização, segundo esses autores, é representada por meio de modelos, que facilitam a realização de especificações de requisitos mais próximas à realidade da organização.

Os modelos de requisitos existentes descrevem o ambiente organizacional em termos de entidades e atividades, sem se importarem com situações em que os usuários poderão tomar diferentes decisões. Esses modelos têm como

objetivo a descrição de sistemas técnicos, em vez de fornecer descrições mais ricas sobre as organizações sócio humanas. As informações capturadas nos modelos existentes, como Diagrama de Fluxo de Dados (DFD) e Diagrama Entidade e Relacionamento (DER), não são suficientes, uma vez que esses modelos descrevem apenas entidades, funções, fluxo de dados e estados do sistema, não expressando as razões envolvidas no processo, ou seja, o porquê de fazer uma determinada ação ou o porquê de tomar uma decisão (alternativas para o “como fazer”).

Assim, faz-se necessário uma ontologia mais rica, que facilite os esforços da Engenharia de Requisitos, para obter uma melhor compreensão sobre os relacionamentos da organização entre os vários atores do sistema, e entender as razões envolvidas nos processos de decisões. Algumas técnicas de Modelagem Organizacional são:

A técnica ORDIT (*Organizational Requirements Definition of Information Technology Systems*), de acordo com Dobson apud Alencar (1999), serve para ajudar os participantes das organizações a definir técnicas alternativas e o futuro organizacional e, consequentemente, o papel da tecnologia da informação, fornecendo um processo sistemático e aproveitável, que seja capaz de suportar gerações de requisitos organizacionais, e fornecer métodos e ferramentas associadas que suportem o processo.

A técnica de modelagem de Furlan (1997) tem como princípio conhecer a missão da organização. A missão é o motivo pelo qual a empresa foi criada, ou seja, a identidade da organização, sendo um texto curto e objetivo. O próximo passo é definir os objetivos executivos ou objetivos da organização, que são os alicerces para a missão, e que devem, portanto, ser totalmente compatíveis com o que estabelece a missão. Os objetivos estarão melhor definidos conforme o desenvolvimento da empresa. Depois, serão definidos os objetivos estratégicos que estão relacionados com as áreas funcionais, com a finalidade de alcançar os objetivos executivos e os fatores chaves de sucesso. Para alcançar os fatores chaves de sucesso são desenvolvidas estratégias, que serão o diferencial da empresa no mercado. Os planos de ação representam a concretização das estratégias.

A técnica F3 de Bubenko (1993) destaca áreas de conhecimento da organização. É constituída por cinco modelos elaborados a partir de objetivos (Modelo de Objetivos – MO), atores (Modelo de Atores – MA), atividades e uso (Modelo de Atividades e Uso – MAU), conceitos (Modelo de Conceitos – MC) e requisitos (Modelo de Requisitos do Sistema de Informação – MRSI).

A técnica i* de Yu apud Alencar (1999) é composta por dois modelos: o Modelo de Dependências Estratégicas (SD) e o Modelo de Razões Estratégicas (SR). O Modelo de Dependências Estratégicas (SD) descreve as relações ou processos. Não trata objetivos organizacionais.

A metodologia EKD (*Enterprise Knowledge Development*), segundo Rolland et al. (2000) e Nurcan (1999), fornece uma forma sistemática de documentar e analisar organização e seus componentes, usando a Modelagem Organizacional. De acordo com Kirikova (2000), a família de modelo do EKD é destinada para responder às questões: o que, como, onde, quem, quando e por que. Essa estrutura serve como um esquema de classificação conveniente ou “tabela periódica” para entidades de informação. Como elementos químicos, essas entidades podem ser combinadas de infinitas formas, para produzir o sistema de informação de interesse da organização. Em outro ponto de vista, é possível ver essa estrutura como uma família de muitos modelos inter-relacionados, em que relacionamentos entre elementos arbitrários pertencentes a submodelos são permitidos. O EKD fornece base para o entendimento e apoio às mudanças organizacionais e ajuda o desenvolvimento de sistemas de informação que apoiará a organização. Para Kirikova (2000), essa talvez seja a teoria mais rica em uso. A proposta de usar o EKD é prover uma descrição clara e não ambígua de:

- Como a organização funciona atualmente.
- Quais são os requisitos e as razões para a mudança.
- Quais alternativas deveriam ser criadas para encontrar esses requisitos.
- Quais são os critérios e argumentos para avaliação dessas alternativas.

FONTE: Disponível em: <<http://www.scielo.br/pdf/gp/v11n2/a06v11n2>>. Acesso em: 8 ago. 2016.

3 MÉTODO ERIKSSON – PENKER

Os sistemas de negócio são sistemas abertos. Não podem ser considerados como sistemas fechados, pois necessitam de informações e recursos de outros negócios, muitas vezes, externos ao processo de negócio em questão. Fornecedores, normas, clientes, tecnologias e leis fazem parte deste círculo.

O método Erikson-Penker criou um método que é o resultado de um conjunto de elementos baseados em modelos já validados da UML para representar os processos de negócio de uma organização. Estes podem ser reutilizados pelo analista de negócio nas fases específicas de arquitetura e desenvolvimento de software (ERIKSSON; PENKER, 2000).

No método proposto por estes autores os sistemas apresentam várias entradas e saídas, em que as partes se comunicam e compartilham informações. Neste caso, o sistema é modularizado e necessita de:

3.1 RECURSOS

Podemos entender como recurso tudo que é usado no processo de negócio: tecnologias, materiais diversos, pessoas, informações etc. Podem ser separados em três categorias distintas: recursos físicos, recursos abstratos e recursos informacionais.

FIGURA 14 – EXEMPLO DO MODELO DE RECURSOS

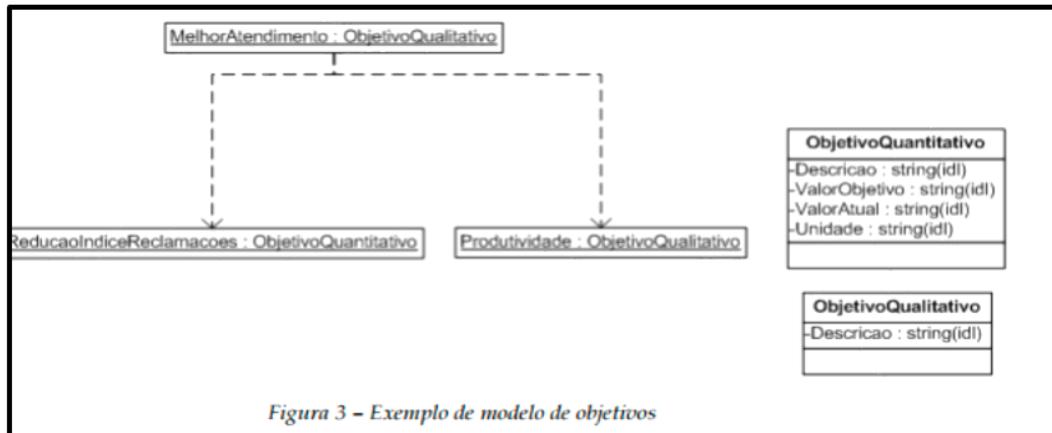


FONTE: Adaptado de Eriksson e Penker (2000)

3.2 OBJETIVOS

É o objetivo do negócio; aquilo que deve ser alcançado e para o qual foi criado o processo de negócio. O método Erikson-Penker representa os objetivos através de objetos, utilizando-se do diagrama de objetos da UML para compor o modelo de objetivos do processo de negócio

FIGURA 15 – MODELO DE OBJETIVOS

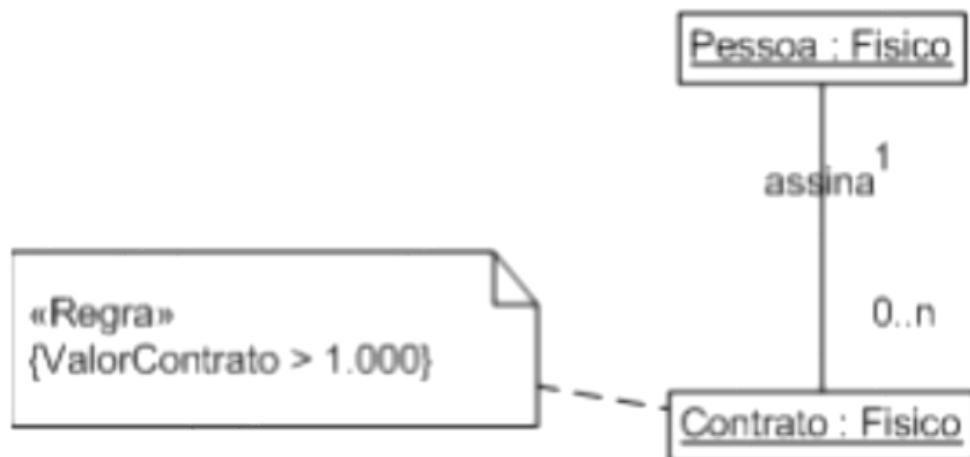


FONTE: Adaptado de Eriksson e Penker (2000)

3.3 REGRAS DE NEGÓCIOS

O modelo de regra de negócio define as restrições e impõe políticas de funcionamento ao processo. Definem como o negócio deve funcionar. As regras podem ser classificadas como: funcionais, comportamentais e estruturais.

FIGURA 16 – MODELO DE REGRAS DE NEGÓCIO



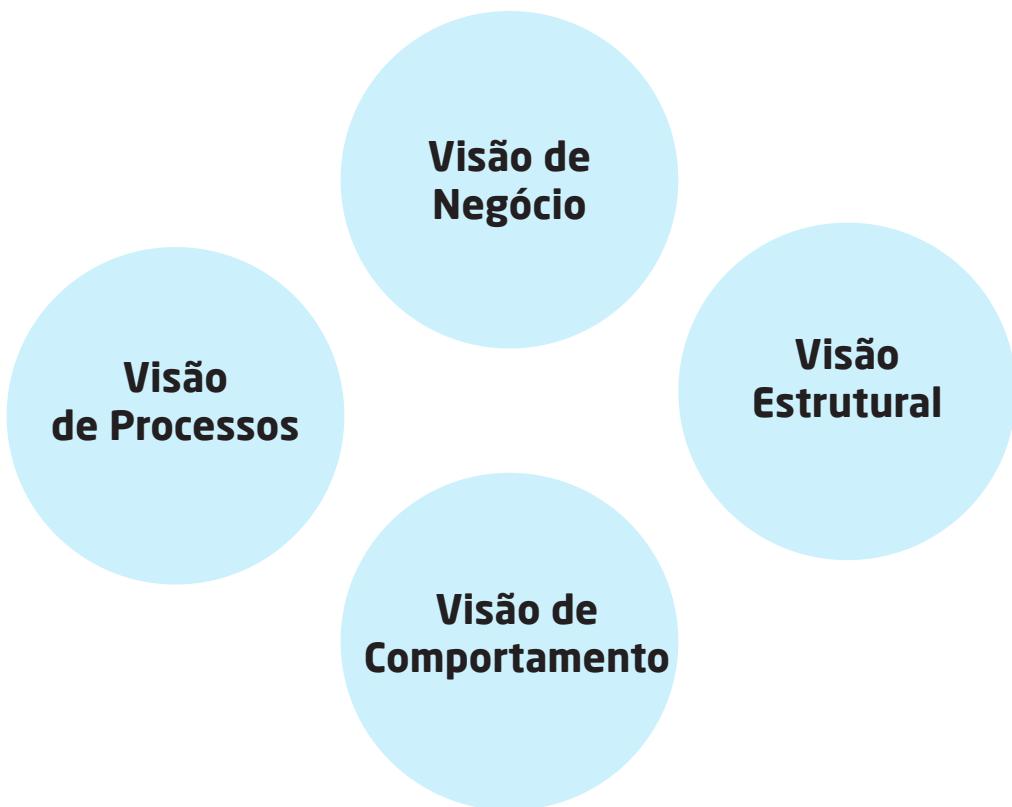
FONTE: Adaptado de Eriksson e Penker (2000)

3.4 PROCESSOS

São as atividades envolvidas no processo de negócio e definem a forma como o trabalho será executado, através de regras pré-definidas. Dentro do processo tudo tem relação. Neste sentido, as regras afetam os recursos e a modelagem de negócio existe justamente para demonstrar as interações entre todos os elementos de um processo.

Neste sentido, os autores propõem um modelo para a visão geral de um negócio, disposto em quatro fases distintas, conforme apresentado na figura a seguir:

FIGURA 17 – VISÕES DE NEGÓCIO DO MODELO ERIKSON – PENKER



FONTE: Adaptado de Eriksson e Penker (2000)

3.5 VISÃO DO NEGÓCIO

Descreve a estrutura geral da empresa no intuito de identificar problemas que devem ser solucionados para viabilizar o alcance dos objetivos (ERIKSSON; PENKER, 2000). A modelagem da visão de negócios deve considerar:

- Missão: é o objetivo macro da organização.
- Fragilidades: fatores que impedem a evolução organizacional.
- Pontos Fortes: diferenciais da organização.
- Pontos fracos: o que precisa ser melhorado internamente.
- Estratégia: ações planejadas para alcance dos objetivos.
- Competências: setores do negócio mais importantes.
- Processos principais: principais atividades.
- Envolvidos: pessoas e funções envolvidas no processo.

3.6 VISÃO DOS PROCESSOS DO NEGÓCIO

É a etapa mais importante, pois define todas as atividades do processo de negócios, além de destacar a interação entre as mesmas (ERIKSSON; PENKER, 2000). Preocupa-se com a atividade em si:

- Quantas atividades estão previstas.
- Qual é a sequência de execução.
- Se existe dependência entre as atividades.
- Como as atividades serão executadas.
- Quem executará cada atividade.
- Como será feito o controle do processo.
- Qual é a relação entre as atividades.

3.7 VISÃO DA ESTRUTURA DO NEGÓCIO

Mostra a estrutura geral do negócio no sentido de sua organização, dos produtos criados e as informações do processo. Utiliza-se do modelo de classes de objetos para a visualização (ERIKSSON; PENKER, 2000). Esta visão é composta pelos modelos de recursos, informações e organização.

3.8 COMPORTAMENTO DO NEGÓCIO

Demonstra o comportamento de cada recurso e processo no modelo do negócio. Utiliza os diagramas de Máquina de Estados, Sequência e Colaboração para a visualização geral.

3.9 INTEGRAÇÃO COM O PROCESSO DE SOFTWARE

Modelo de negócio e modelagem de *software* são desenvolvidos por equipes distintas. O modelo de negócio é que dá suporte para o projeto do *software* em sua totalidade. Importante lembrar que muitos componentes do modelo de negócio não se farão presentes no projeto do *software*. Da mesma forma, muitos detalhes técnicos do modelo de *software* não estarão contemplados ou detalhados no modelo de negócio.

Para auxiliar neste sentido, pode-se utilizar a técnica i, proposta por Eric Yu, a qual fornece apoio organizacional na compreensão do ambiente da institucional e todos os seus relacionamentos.

Esta técnica preocupa-se com os atores que executarão os processos, com as formas alternativas para executá-los e porque as pessoas devem receber as informações. É composta por dois modelos: Modelo de Dependência Estratégica (SD) e Modelo de Razão Estratégica (SR).

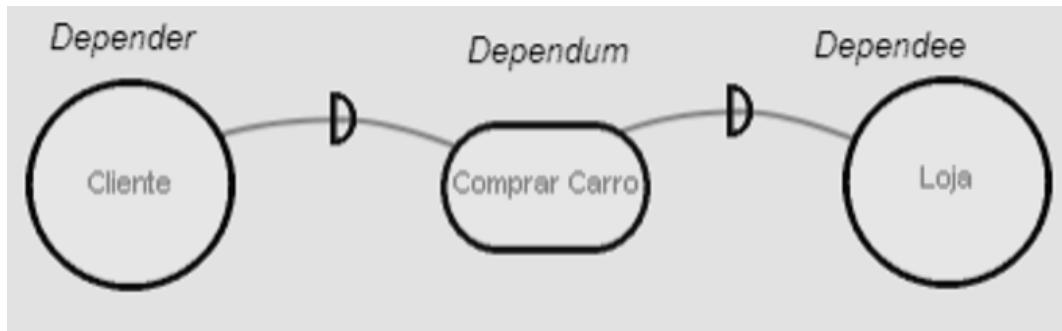
O Modelo de Dependência Estratégica fornece uma descrição intencional de um processo em termos de uma rede de relacionamentos de dependência entre atores.

O Modelo de Razão Estratégica (SR) fornece uma descrição intencional do processo em termos de elementos do processo e as razões que estão por detrás deles.

3.9.1 Modelo de Dependências Estratégicas (SD – Strategic Dependency Model)

Este modelo é representado por um grafo. Cada nó representa um ator. As arestas indicam que um ator depende do outro na execução de atividades. Descreve a relação de dependência dos atores organizacionais. É representado pela nomenclatura a seguir:

FIGURA 18 – EXEMPLO DE MODELO DE DEPENDÊNCIA ESTRATÉGICA

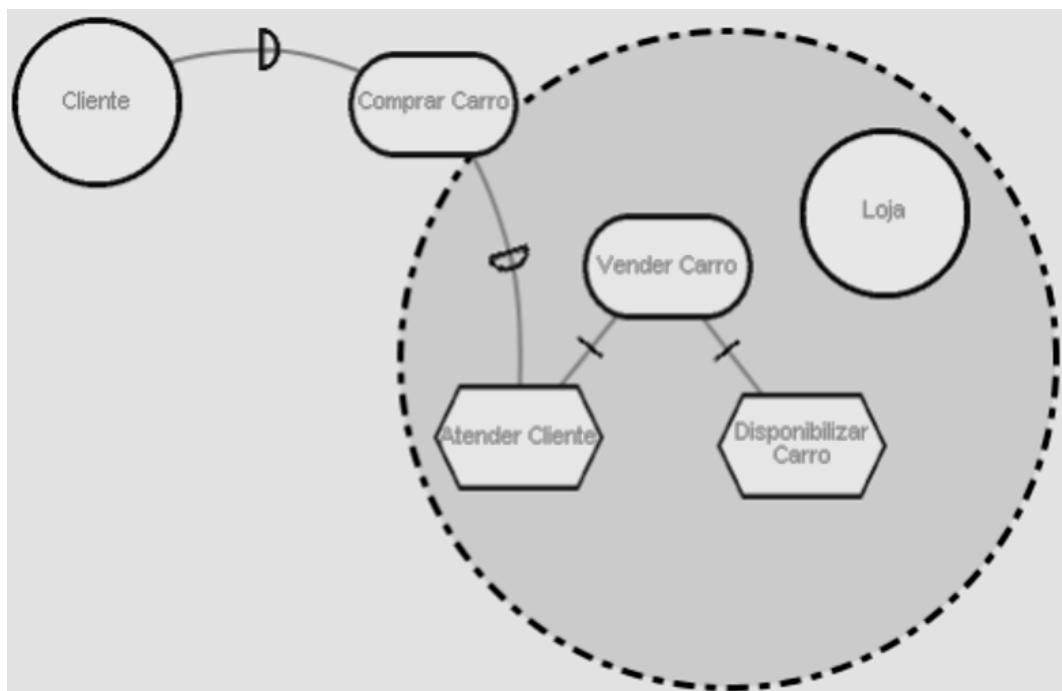


FONTE: Adaptado de Yu (1995)

3.9.2 Modelo das Razões Estratégicas (SR – *Strategic Rationale Model*)

É um modelo mais detalhado. Descreve a responsabilidade de cada ator em relação as suas metas e também sobre os relacionamentos que mantém com outros atores. Tarefas, recursos e objetivos aparecem como fatores internos e diretamente ligados aos atores. A figura a seguir exibe exemplo de modelo SR:

FIGURA 19 – EXEMPLO DE MODELO DE RAZÃO ESTRATÉGICA



FONTE: Adaptado de Yu (1995)

3.9.3 Notação de modelagem da técnica i*

Segue a tabela que descreve os ícones e relacionamentos da notação do método visto nesta seção, i*

FIGURA 20 – NOTAÇÃO DE i*

Figura 9 - Ator	Figura 10 - Agente
Figura 11 - Posição	Figura 12 - Função de atores
Figura 13 - Recurso	Figura 14 - Objetivo Soft
Figura 15 - Objetivo Estratégico	Figura 16 - Dependência
Figura 17 - Término	Figura 18 - Decomposição
Figura 19 - Contribuição AND	Figura 20 - Contribuição HELP
Figura 21 - Instância	Figura 22 - Subclasse
Figura 23 - Parte de um todo	Figura 24 - Executa um papel

FONTE: Adaptado de Yu (1995)

4 MÉTODO EKD – ENTERPRISE KNOWLEDGE DEVELOPMENT

É um método de modelagem organizacional. Ele facilita o conhecimento do ambiente empresarial, facilitando a documentação, entendimento e conhecimento dos componentes empresariais, tendo como propósito descrevê-los de forma clara. Descreve o funcionamento do negócio e seus requisitos principais, além das necessidades de mudança e avaliação de alternativas para alcance dos objetivos organizacionais. É um modelo utilizado para entender, analisar e facilitar as melhorias nos sistemas. É considerado um modelo conceitual, pois examina a organização e seus requisitos. Neste sentido, a organização é percebida sob aspectos distintos, os quais se integram em função de um objetivo único (ALENCAR, 1999).

As pessoas envolvidas no processo de modelagem necessitam passar por um treinamento para conhecer o modelo e seus objetivos. Isso é necessário para que os envolvidos compreendam que os modelos são abstrações do mundo físico, ou seja, uma representação gráfica da realidade.

O EKD viabiliza a construção de outros modelos. Sua grande contribuição é que este modelo é voltado para o trabalho em equipe, pois explora o conhecimento de atores de diferentes áreas. Isso contribui muito para a construção de um modelo mais solidificado e voltado para adequados processos de negócio. Outra característica importante é o fato de facilitar o aprendizado organizacional, fazendo com que os envolvidos tenham total conhecimento e domínio das regras de negócio envolvidas em cada processo.

O *Enterprise knowledge development (EKD)* é uma metodologia que fornece uma forma sistemática e controlada de analisar, entender, desenvolver e documentar um negócio e seus componentes, usando a modelagem organizacional. Este tem como principais objetivos a descrição clara e não ambígua de como o negócio funciona atualmente, quais são os requisitos e razões para que uma determinada mudança ou nova prática seja desenvolvida na empresa, quais são as alternativas que deveriam ser criadas para cumprir esses requisitos e quais são os critérios e argumentos para a avaliação dessas alternativas (BUBENKO et al., 2001).

Entre os principais benefícios, o EKD permite:

- entender melhor o negócio;
- facilitar a aprendizagem e a comunicação organizacional sobre questões essenciais;
- ajudar a entender e a promover as capacidades e processos da organização;
- melhorar a comunicação entre os participantes;
- desenvolver uma descrição estruturada do negócio;
- chegar a uma descrição dos objetivos da organização, entidades, processos e requisitos (BUBENKO et al., 1998).

Além disso, O EKD auxilia nos seguintes propósitos empresariais:

- na engenharia de requisitos, para definição e especificação de requisitos;
- na análise do negócio, para detecção de problemas;
- na reengenharia de processos do negócio, para definição de novos sistemas de negócio;
- no gerenciamento do conhecimento organizacional ou aprendizagem organizacional, para formar a base de programação e ampliação de conhecimento (BÜBENKO et al., 1998).

FONTE: Disponível em: <http://www.abepro.org.br/biblioteca/enegep2009_TN_STO_098_661_12622.pdf>. Acesso em: 9 ago. 2016.

A figura a seguir apresenta a estrutura do modelo EKD.

FIGURA 21 – MODELO EKD



FONTE: Adaptado de Reis (2003)

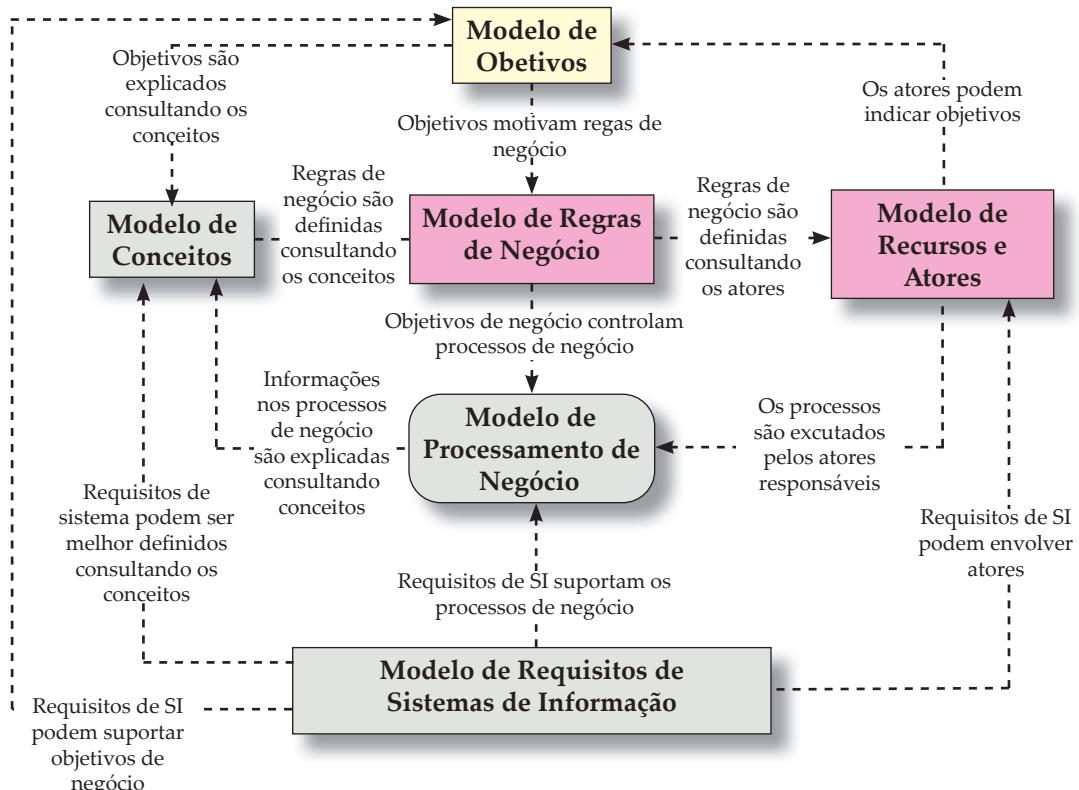
O modelo EKD é estruturado em três etapas (ALENCAR, 1999):

- Participação dos *stakeholders*: pessoas interessadas no processo.
- Descrições técnicas: provê tudo o que é necessário para a elaboração dos modelos.
- Diretrizes de trabalho: são normas, regras e procedimentos que orientam a construção dos modelos, além de fornecer conhecimento para a execução das atividades da modelagem de negócios.

O modelo EKD pode ser percebido e utilizado nas três fases distintas dos processos de negócio: diagnóstico, análise e projeto, respectivamente.

A estrutura do EKD está dividida em submodelos que representam diferentes partes do negócio, conforme mostra a figura a seguir:

FIGURA 22 – EXEMPLO DE MODELO DE RAZÃO ESTRATÉGICA



FONTE: Adaptado de Reis (2003)

Modelo de Objetivos (MO): descreve as metas a serem alcançadas e também como serão alcançadas.

Modelo de Regras do Negócio (MRN): descreve as regras de negócio e as mantém em acordo com o modelo de objetivos.

Modelo de Conceitos (MC): define “coisas, assuntos e questões” de outros modelos.

Modelo de Processos do Negócio (MPN): define processos organizacionais e como eles interagem em relação às informações organizacionais.

Modelo de Atores e Recursos (MAR): descreve como os atores e os recursos organizacionais se relacionam e como interagem com os componentes do modelo de objetivos e do processo de negócio.

Modelo de Requisitos e Componentes Técnicos (MRCT): é utilizado para mapear toda a estrutura de um sistema de informação no intuito de dar suporte para as principais atividades do negócio, de acordo com as definições do modelo de processo de negócio.

Recriar *software* significa construir um sistema a partir de uma programação que já existe. Ou seja, existe a reutilização de conhecimento preexistente para a construção de algo novo.

Reutilizar processos de *software* é uma abordagem importante para:

- Compartilhar experiências de projetos anteriores para melhor adequação de processos posteriores.
- Promover a reutilização dos conhecimentos de pessoas chave dentro do processo.
- Facilitar o aprendizado de pessoas menos experientes.

São três os principais modelos de reutilização de processos (ALENCAR, 1999).

Modelo abstrato: é um *template* genérico para dar suporte na criação de um processo novo.

Modelo instanciado: com base no modelo abstrato, são especificados objetivos e restrições para o cenário de aplicação do processo.

Modelo em execução ou executado: serve de base para registrar o histórico de execuções dos processos.

O ciclo de vida de um processo de reutilização pode ser entendido através do modelo proposto por Costa (2010), conforme figura a seguir.

FIGURA 23 – CICLO DE VIDA PARA REUTILIZAÇÃO DE PROCESSOS DE SOFTWARE



FONTE: Costa (2010)

O modelo proposto pelo autor é composto por quatro fases:

- **Modelagem de Processos Visando a Reutilização:** através da PML (Linguagem de Modelagem de processos) define os processos mais genéricos que especificam o conhecimento e as experiências de uma organização para posterior disseminação. Nesta etapa são definidos os processos padrões através dos quais novos processos serão criados.
- **Reutilização de Processos:** esta fase contempla a utilização de estratégias e práticas para auxiliar os responsáveis pelos modelos na adaptação dos processos genéricos, que por ter sido inicialmente definido como padrão, precisa de ajustes para se adequar ao cenário de execução. Não existem modelos genéricos que possam ser aplicados em todas as situações. Adaptar processos requer mudanças significativas nos modelos genéricos para adaptá-los aos contextos específicos.
- **Execução de Processos:** trata da execução dos processos especificamente no que tange às modificações. Esta fase envolve planejamento e controle para garantir que o processo seja executado conforme a modelagem determinada.
- **Generalização e Avaliação de Processos Encerrados:** extraem e analisam informações de processos já executados em modelos reutilizáveis visando sempre a melhoria contínua.

Na reutilização de processos, a escolha do modelo deve priorizar a captura de elementos e suas variações em projetos específicos para permitir a criação de processos que possam ser aplicados em situações distintas, viabilizando desta forma a sua reutilização, reduzindo o custo e tempo em novos desenvolvimentos.

5 REUTILIZAÇÃO DE PROCESSOS DE SOFTWARE

Processos de *software* podem modelar de várias formas distintas a execução das atividades de um projeto. Os processos facilitam o entendimento dos problemas e auxiliam as equipes de desenvolvimento no trabalho colaborativo e coordenado.

A maioria das empresas possui processos legados que estão desatualizados em relação às melhores práticas. Estes, precisam adaptar-se para tornar possível a reutilização, tornando-se, muitas vezes, um processo padrão, através do qual serão gerados outros processos. Um processo-padrão pode ser entendido como um *template* básico cuja função é guiar todos os processos em vários níveis organizacionais e hierárquicos.

Existem também os subprocessos, que podem ser entendidos como pedaços de um processo maior. Como exemplo, podemos citar: desenvolvimento de requisitos, testes, revisões por pares, entre outros.

Vários autores, entre eles Kellner (1996), afirmam que os processos têm muitas semelhanças com os *softwares*, e que inclusive, pode-se aplicar as técnicas de desenvolvimento de *softwares*, nos processos. O autor defende que uma das principais formas de reutilizar processos de *software* ocorre através do adequado uso dos componentes do processo.

A linha de pesquisa que se apoia na reutilização busca aumentar a produtividade e qualidade na execução de tarefas usando componentes de processos reutilizáveis. Para viabilizar a correta reutilização dos processos torna-se necessária a utilização de uma ou mais bibliotecas de componentes reutilizáveis de processos, para armazenar os componentes e suas linhas de processo.

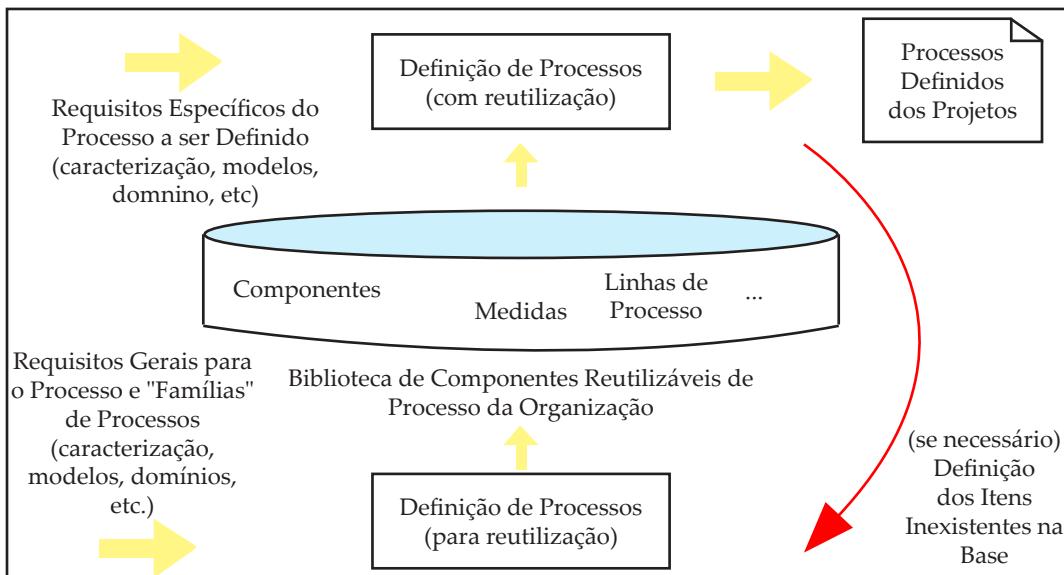
Bibliotecas podem ser usadas na fase de definição de processos e projetos, e tem como objetivo principal auxiliar na definição de processos padrão ou processos oriundos de projetos variados. Os projetos contribuem com a biblioteca de componentes através de suas informações de uso e execução dos componentes.

Principais conceitos envolvidos na reutilização de processos:

- **Elemento de Processo:** São os componentes do processo e as atividades. Somente os componentes podem ser reutilizados.
- **Componente de Processo:** os componentes são importantes, pois podem ter desempenho e apresentar várias versões. Um componente é a unidade básica de um processo.

Através da figura a seguir podemos perceber o ciclo de reutilização de processos em uma organização.

FIGURA 24 – CICLO DE REUTILIZAÇÃO DE PROCESSOS DE SOFTWARE



FONTE: Adaptado de Kellner (1996)

- **Componente de processo concreto:** estes componentes não podem ser alterados. Devem ser executados exatamente como foram especificados. Este tipo de componente pode ser aferido e controlado.
- **Componente de processo abstrato:** Pode sofrer variações, apresentando várias formas de execução. É considerado abstrato, pois pode ser instanciado de formas distintas.
- **Arquitetura de processos:** representa o fluxo de trabalho que é composto por componentes e atividades do processo.
- **Linha de processos:** é a arquitetura de processos que apresenta opçionalidades.
- **Ponto de variação:** é um componente abstrato contido em uma arquitetura de processos.
- **Variante:** é as implementações de um ponto de variação.
- **Característica de processo:** Aplicação do conceito de característica (*feature*) de uma linha de produtos tradicional para o contexto de processos de *software*. Será vista como uma espécie de funcionalidade ou classificação que o processo deve possuir, por exemplo: Apoio ao nível G do MPS.BR, Paradigma Orientado a Objetos, entre outros. Restringe a utilização de componentes, definindo se um conjunto de componentes pode ou não ser utilizado. Podem ser utilizadas para representar diversos tipos de rastreabilidade.

Para possibilitar a reutilização de processos de *software* é necessário, primeiramente, definir como o conhecimento relacionado aos processos de *software* deverá ser estruturado e representado de forma a permitir e facilitar sua posterior reutilização. Nesta abordagem, um componente poderá existir em qualquer nível de detalhamento, ou seja, desde uma única atividade até um processo inteiro. No entanto, um componente sempre deverá ser composto por, no mínimo, uma atividade, pois uma unidade de reutilização inferior a esta possivelmente apenas introduziria complexidade desnecessária e dificultaria a composição de processos.

São requisitos para a estrutura dos componentes de processo utilizados nesta abordagem:

- Possuir uma caracterização geral.
- Permitir a gerência de configuração.
- Possuir definição clara de suas interfaces e arquitetura interna.
- Permitir registro de informações relacionadas a seu uso.
- Permitir a rastreabilidade para modelos de maturidade e normas.
- Permitir a rastreabilidade para domínios de aplicação e tipos de *software*.
- Permitir o registro de conhecimento de apoio.
- Auxiliar na análise de desempenho e capacidade dos processos.
- Permitir a representação de variabilidades, ou seja, um mesmo componente poder ser instanciado ou detalhado de diferentes formas, dependendo da situação.

Nesta abordagem, uma arquitetura de processos representa uma espécie de fluxo de trabalho, podendo ser composta por componentes de processo, atividades ou qualquer combinação entre eles. Um exemplo possível de arquitetura de processos seria um modelo de ciclo de vida. Neste caso, são conhecidas as principais fases do processo, como se relacionam e quais os objetivos de cada uma. Porém, diversas instanciações diferentes são possíveis para essa mesma estrutura, ou seja, diferentes componentes ou atividades que atendam à estrutura podem ser utilizados, dependendo da situação.

Um componente de processo é a unidade básica de composição de processos. Definições de processo são sempre realizadas através a composição de diferentes componentes de processo. Um componente pode possuir uma arquitetura de processos interna. Essa arquitetura permite que um componente

seja mais detalhado e decomposto em outros componentes. No entanto, um componente pode não possuir uma arquitetura interna. Neste caso, o componente define seu propósito, mas permite que sua realização, ou seja, os passos para atingir o propósito definido, sejam realizados de uma forma qualquer, sem seguir uma estrutura definida. Por exemplo, se existisse um componente “Planejar Custos” sem uma arquitetura interna definida, então qualquer abordagem para planejar custos que atingisse o propósito do componente poderia ser usada.

Poderia ser feito com base na experiência, em dados históricos, através de métodos específicos, entre outros. Assim, componentes que não possuem arquitetura interna definida serão chamados pontos de variação, de forma semelhante à nomenclatura dada na área de linhas de produtos de *software*. Um ponto de variação é um componente de processo que admite diversas estruturas internas, que podem ser realizadas por diversas variantes. Uma variante é uma das possíveis implementações de um ponto de variação.

No exemplo do planejamento de custos citado acima, planejamento com base em experiência, planejamento com base em dados históricos e todas as demais alternativas seriam variantes que poderiam ser utilizadas no ponto de variação “Planejar Custos”.

Uma arquitetura de processos que possua pontos de variação também permite variabilidade. No entanto, de maneira diferente, pois pode existir mais de um ponto de variação, e podem também existir componentes “concretos”, ou seja, que devem ser realizados conforme especificado sem variação, além de componentes opcionais que podem ou não estar presentes. Com isso, essa arquitetura é uma composição de pontos de variação, de elementos obrigatórios, e de elementos opcionais. Portanto, uma vez que esse tipo de variabilidade é diferente da variabilidade dos pontos de variação isolados, chamaremos esse tipo de arquitetura de processos de linha de processos.

Supõe-se que um componente de processo seja algo em mais alto nível que uma simples atividade, e que o engenheiro de processos utilize blocos maiores que uma simples atividade para compor seus processos. Isso possivelmente facilitará a reutilização de processos. Mais ainda, um componente será algo suficientemente relevante para ter seu desempenho e sua capacidade medidos e controlados; possuir registros de sua utilização em diferentes contextos; possuir rastreabilidade para modelos de maturidade e domínios de aplicação, entre outros.

A cada componente de processo podem ser associadas medidas que podem ser coletadas ao longo da execução dos componentes de processo. Essas medidas permitem que os componentes de processo (que podem ser considerados subprocessos) tenham seu desempenho e sua capacidade determinados e controlados.

Os dados sobre desempenho e capacidade dos componentes são muito importantes para auxiliar na escolha dos componentes mais adequados. Um exemplo é quando se deseja escolher qual das variantes adotar para um dado ponto de variação. Esses dados devem ser considerados para se verificar qual variante exibiu melhor comportamento em contextos similares, e se sua capacidade em relação aos aspectos relevantes (custo, tempo, qualidade etc.) é adequada para a situação específica.

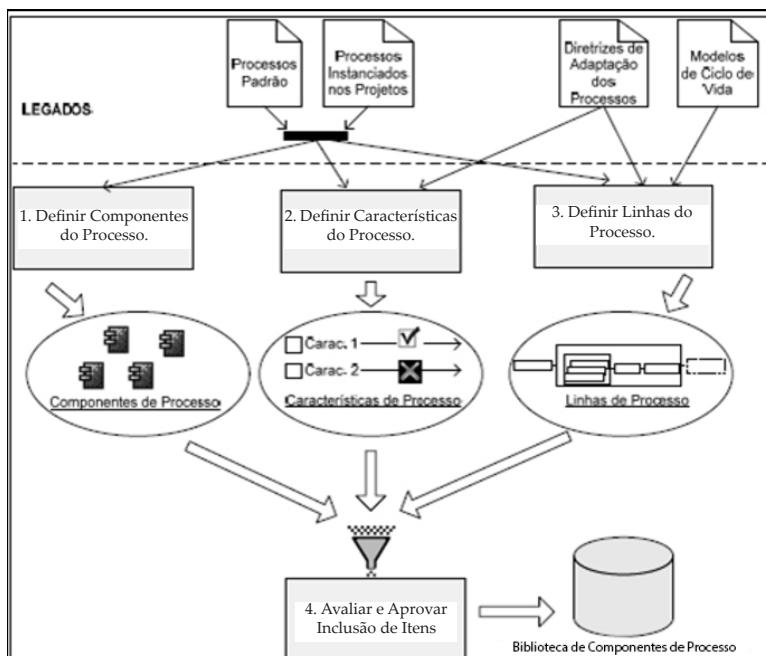
Além dos requisitos citados acima, um componente de processos também precisa possuir informações de versionamento, informações relacionadas à sua utilização e o contexto em que a utilização ocorreu, além de conhecimento de apoio que possa ser útil para auxiliar em sua seleção ou execução. A confiabilidade e certificação dos componentes reusados são problemas enfrentados pela equipe que opta pela prática do reúso.

FONTE: Disponível em: <https://www.researchgate.net/publication/232219509_Uma_Abdoragem_de_Definicao_de_Processos_de_Software_Baseada_em_Reutilizacao>. Acesso em: 28 jun. 2016.

Quatro etapas são necessárias para componentizar um processo, conforme a figura a seguir:

- 1 Definição dos componentes de processos.
- 2 Definição características de processos.
- 3 Definição da linha de processos.
- 4 Aprovação dos elementos reutilizáveis na biblioteca de componentes.

FIGURA 25 – PROCESSOS LEGADOS – COMPONENTIZAÇÃO



FONTE: Adaptado de Kellner (1996)

5.1 DEFINIÇÃO DE COMPONENTES DE PROCESSO

Para definir corretamente os componentes de processo que serão reutilizados devem-se primeiramente analisar os processos utilizados em projetos anteriores para averiguar se existem diferenças significativas em relação aos processos considerados “padrão”. As diferenças em relação ao processo padrão indicam que algumas partes ou fases sofreram variações, destacando que determinadas situações podem ser executadas de maneiras diferentes. As partes que sofrem alterações devem ser consideradas componentes abstratos do processo.

Feito isso, parte-se para a análise das atividades no sentido de identificar:

- Atividades que estejam em um nível de detalhamento tal que possam ser facilmente reutilizadas em outras definições.
- Atividades que sejam potencialmente relevantes para análises de capacidade e estabilidade.
- Atividades que já possuam dados de utilização, de forma a tornar possível a utilização de dados passados nas análises de estabilidade e capacidade futuras. Por exemplo, em processos que são estruturados em macro atividades, essas “macroatividades” são boas candidatas a se tornarem componentes de processo que atendem aos critérios citados neste passo.

5.2 DEFINIÇÃO DE CARACTERÍSTICAS DE PROCESSO

A importância da definição das características do processo consiste no fato de que torna mais fácil rastrear os componentes do processo para que possam ser escolhidos ou não durante a especificação do processo. Logo, esta etapa é importantíssima, pois determina os fatores que influenciam na escolha de um ou outro componente na composição do processo.

Uma das principais análises feitas para escolher componentes é avaliar as diretrizes de adaptação em relação aos processos padrão da organização, no sentido de definir características que representem as diferentes possibilidades de adaptação do processo e associar a elas os componentes de processo que devem ou não ser escolhidos, uma vez que a característica seja selecionada.

Podemos associar os componentes a tipos distintos de características:

- Em disciplinas (projetos, planejamento, requisitos, testes, qualidade).
- Em técnicas e métodos (análise por ponto de função).
- Paradigmas de desenvolvimento (orientação a objetos, programação estruturada).



Lembre-se de que é importante registrar as características já atendidas pelos processos legados.

5.3 DEFINIÇÃO DE LINHAS DE PROCESSO

A linha de processo é a etapa que ordena os componentes para que possam ser utilizados no processo. A linha pode estar associada às características do processo. Então, o primeiro passo para definir a linha de processo é a escolha de quais características a linha deverá atender.

Na verdade, as características escolhidas servem como requisitos que deverão ser analisados e cumpridos ao longo da trajetória da linha, além de permitir a rastreabilidade dos componentes do processo.

Para iniciar a definição da linha, sugere-se a escolha de um modelo de ciclo de vida utilizado na organização. Cada modelo adotado pela organização pode ser considerado uma linha de processos distinta para definir o relacionamento dos componentes e a ordem deles. A linha de processo define quais componentes serão repetidos dentro do modelo de ciclo de vida.

5.4 AVALIAÇÃO E APROVAÇÃO DOS ELEMENTOS REUTILIZÁVEIS

A última etapa consiste na avaliação dos componentes, linhas e características. Isso é feito por todos os principais envolvidos e interessados no processo e define o que será incorporado na base de componentes reutilizáveis da organização. A concordância entre as partes envolvidas é importante para permitir a adequada reutilização no processo, sendo que os itens aprovados serão posteriormente inclusos na biblioteca de componentes da organização.

LEITURA COMPLEMENTAR

LABORATÓRIO DE ENGENHARIA DE SOFTWARE E INTELIGÊNCIA ARTIFICIAL: CONSTRUÇÃO DO AMBIENTE WEBAPSEE

Carla Alessandra Lima Reis
Rodrigo Quites Reis

RESUMO: Este artigo descreve a evolução do WebAPSEE, ambiente de gestão de processo de *software* desenvolvido como *Software Livre* pelo Laboratório de Engenharia de *Software* da Universidade Federal do Pará com participação do SERPRO-Belém, Eletronorte e Universität Stuttgart – Alemanha. O texto enfatiza a descrição dos aspectos metodológicos adotados, os resultados obtidos com o uso do ambiente, e discute as perspectivas para o avanço atual e futuro.

1 INTRODUÇÃO

Ambientes para automação de processos de *software* (GIMENES 1994), aqui genericamente denominados PSEEs – *Process-Centered Software Engineering Environments* possibilitam a coordenação das atividades de equipes de desenvolvimento de *software*, realizando o acompanhamento dos prazos e consumo de recursos, além de facilitar a reutilização de boas práticas gerenciais a partir de diferentes projetos já concluídos. As soluções nesta área devem considerar as especificidades deste contexto, tais como: o caráter criativo do processo, a tendência a mudanças nos produtos e processos, a natureza abstrata do produto resultante (*software*), dentre outros aspectos.

O ambiente WebAPSEE, apresentado neste artigo, é um PSEE com desenvolvimento liderado pelo Laboratório de Engenharia de *Software* da Universidade Federal do Pará (LABES-UFPa) que visa prover simultaneamente automação e flexibilidade em níveis não alcançados por outros PSEEs propostos pela Academia e Indústria. Este ambiente destaca-se ainda pelo uso de ferramentas de *Software Livre* e padrões abertos que foram adotados, variando desde a linguagem de programação (Java), passando pelos *frameworks* para desenvolvimento de componentes, e chegando ao sistema de gerenciamento de banco de dados e ferramentas externas integradas.

Este artigo apresenta os avanços obtidos no ano de 2006 com respeito à evolução da ferramenta WebAPSEE, principalmente no que diz respeito à incorporação de funcionalidades voltadas à gestão organizacional e apoio à tomada de decisão. Além disso, é feita uma avaliação inicial do uso da ferramenta em organizações de desenvolvimento de *software* localizadas no estado do Pará, enfatizando também questões relacionadas com as demandas de melhorias por parte dos usuários.

2 OBJETIVOS E JUSTIFICATIVA

O desenvolvimento do WebAPSEE é baseado em três premissas básicas, descritas a seguir, e expostas em maiores detalhes por Lima Reis (2003):

- O apoio automatizado à execução de processos constitui um elemento fundamental para transformar os modelos de processo em realidade nas organizações de *software* de *software* possuindo, assim, enorme potencial no auxílio à implantação de processos;
- O aumento no nível de flexibilidade fornecido pelas ferramentas no acompanhamento da execução de processos é necessário, pois o processo de *software* é inherentemente dinâmico. Os modelos de processo não evoluem de maneira determinística, o que implica na impossibilidade de antever todas as características dos modelos de processo e produtos de *software* resultantes;
- A elevação do nível de automação fornecido pelas ferramentas (e, por conseguinte, do apoio à tomada de decisão gerencial) é proporcional à riqueza das informações (métricas e decisões) registradas pelo ambiente acerca da realização prática dos processos na organização.

O desenvolvimento e evolução do ambiente WebAPSEE são motivados pela inexistência de produtos que atendam simultaneamente a todas as premissas listadas acima. O ano de 2006 foi decisivo para evolução do ambiente e início do seu uso por parte da indústria. As seções a seguir descrevem a evolução do projeto.

3 METODOLOGIA DE EXECUÇÃO

O início do desenvolvimento do WebAPSEE foi em 2005 com financiamento da FINEP¹ em um projeto em parceria com a regional Belém do SERPRO. Uma primeira versão do ambiente – denominada versão zero – foi desenvolvida e disponibilizada neste período, a qual fornecia funcionalidades para modelagem e execução de processos acessíveis através de serviços *Web*.

Em 2006, o projeto caminhou em duas direções complementares:

- 1) a evolução da ferramenta como produto, visando melhorias na usabilidade e performance geral do sistema;
- 2) a evolução do WebAPSEE como pesquisa, visando a incorporação de melhorias para gestão que contribuam para que o ambiente atinja o estado da arte tecnológico (com apoio do CNPq²).

O desenvolvimento contou ainda com o apoio de um projeto financiado pela Eletronorte 3 com o objetivo de evoluir a ferramenta e utilizá-la como instrumento para a descrição e experimentação de um modelo de processo de *software* para ser implantado pelo Centro de Tecnologia da empresa.

1 Edital MCT/FINEP/*Software Livre*/2004.

2 Projeto LABES-IA apoiado pelo edital PDPG-TI/CNPq.

3 Projeto número 50528 apoiado no ciclo 2004/2005 do Programa de P&D da Eletronorte.

O projeto evoluiu em três versões de distribuição restrita (interna), até que no dia 2 de outubro de 2006 a versão 1.0 do produto foi disponibilizada publicamente para a comunidade de *Software Livre*. Esta versão foi desenvolvida a partir do levantamento das demandas tanto das empresas formalmente vinculadas com o LABES quanto da comunidade externa. Por exemplo, a integração do WebAPSEE com o sistema *Concurrent Version System* (*CVS*) foi baseada nesta estratégia.

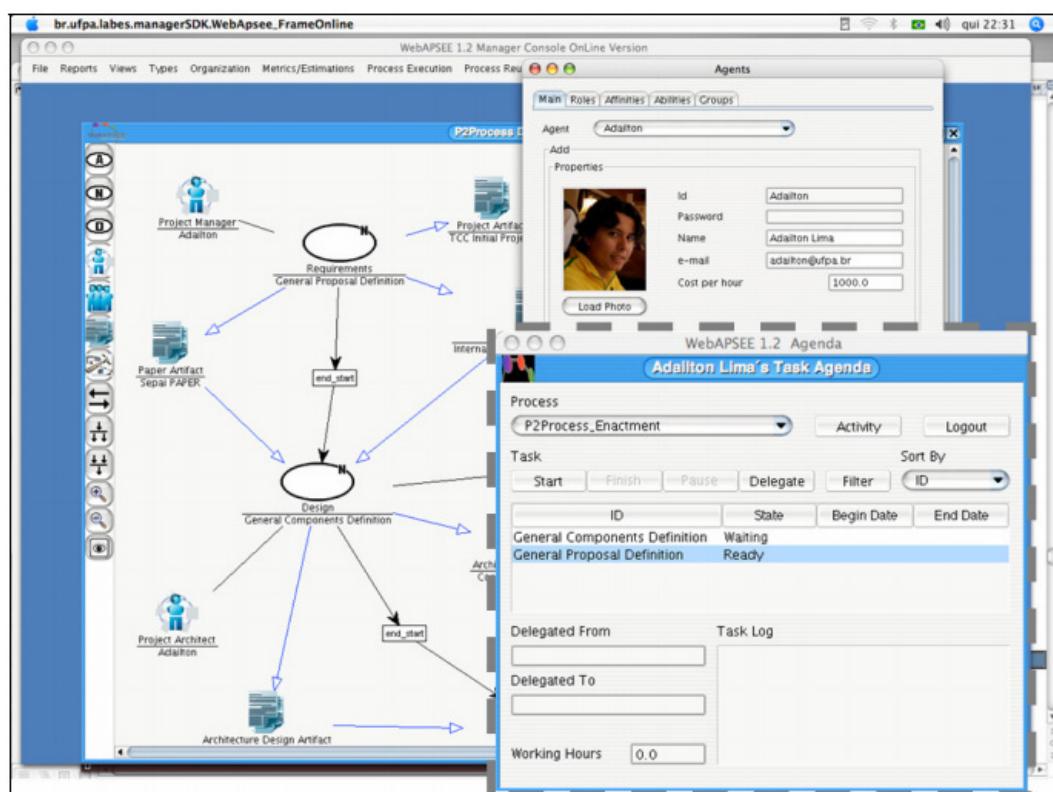
4 RESULTADOS OBTIDOS

As subseções a seguir apresentam os produtos e resultados relevantes obtidos em 2006. 4.1 Produtos de *software* gerados O principal produto gerado por este projeto é a versão 1.2 do sistema WebAPSEE, a qual está disponível para a comunidade de *Software* Livre no endereço do LABESUFPA.

A ferramenta possui duas visões principais: a do gerente e a do desenvolvedor.

A Figura 1 mostra a visão do gerente (*Manager Console*) como parte mais ampla da figura e a visão do desenvolvedor (*Task Agenda*) na parte inferior direita da imagem. Figura 1 Tela principal do ManagerConsole e TaskAgenda O fato de permitir a execução de processos de *software* significa que o ambiente fornece acompanhamento em tempo real do progresso das atividades e da produção dos artefatos do processo. As atividades – representadas por elipses no editor gráfico de processos mostrado na Figura 1 – constituem os elementos principais de modelagem.

Figura 1 Tela principal do ManagerConsole e TaskAgenda



Ao redor das atividades são definidos os artefatos (de entrada/saída), recursos humanos e de apoio e políticas de processo. As atividades têm suas cores (estados) alteradas na tela do gerente em resposta às ações realizadas pelos desenvolvedores em suas agendas. Os elementos de *software* que compõem o WebAPSEE são desenvolvidos para atender prioritariamente as demandas relacionadas com os níveis de maturidade propostos pelo CMMI e MPS-BR.

Como uma discussão detalhada acerca da aderência do WebAPSEE às inúmeras práticas exigidas por estes modelos de referência foge do escopo deste texto⁴, o trecho que segue fornece uma descrição das funcionalidades vinculadas à gestão de configuração de *software* e gerência de projeto exigidas, respectivamente, pelos níveis F e G do MPS-BR.

O WebAPSEE fornece uma maneira inovadora para implantar gestão de configuração de *software* visto que possibilita o controle de todos os artefatos do processo ocultando a complexidade dessa atividade. Tal funcionalidade é visualizada pelo desenvolvedor como operações de upload e download de artefatos, respectivamente associadas ao prévio planejamento da sua produção e consumo por uma atividade. Ressalta-se que os desenvolvedores só manipulam os artefatos definidos nas atividades, obedecendo ao controle de acesso previsto nos projetos da organização.

Assim, toda a interação com o CVS (ou semelhante) é escondida do usuário. Quanto à gestão de projetos, o MPS-BR estabelece como um dos resultados da sua aplicação que “os recursos humanos para o projeto são planejados considerando o perfil e o conhecimento necessários para executá-lo” (SOFTEX, 2007). Tomando o ponto de vista específico da alocação de recursos humanos, o WebAPSEE possui detalhado registro das informações organizacionais. Além disso, é fornecido um mecanismo automático para alocação de pessoas baseado em Políticas (SILVA, 2007), o qual ilustra o nível de automação fornecido pelo ambiente no auxílio à tomada de decisão gerencial.

A linguagem de definição de Políticas permite a definição de estratégias de alocação genéricas as quais podem ser reusadas em diferentes processos. Além disso, as políticas representam uma extensão da linguagem de modelagem do WebAPSEE, fornecendo uma estrutura Orientada a Aspectos, transversal ao modelo de processo.

Funcionalidades adicionais, não detalhadas aqui por limitação de espaço, incluem;

- 1) apoio à recuperação e reutilização de processos (SALES, 2006);
- 2) coordenação de processos de *outsourcing* a partir da execução descentralizada (LIMA, 2007);
- 3) extração de relatórios gerenciais para auxiliar na avaliação da qualidade de *software* (PAXIÚBA, 2007);
- 4) Políticas para alocação de recursos de apoio (análogas às de alocação de pessoas);

- 5) Apoio à Medição/Análise de Projetos (NASCIMENTO, 2006) e
- 6) Modelo de Simulação de Processo de *Software* (FRANÇA, 2007).

4.2 ARTIGOS PUBLICADOS

Este projeto originou seis artigos em conferências nacionais em 2006, listados a seguir. Lima (2006A) e (2006B) são artigos voltados à descrição arquitetural do ambiente. Lima (2006C), premiado como Melhor Artigo Científico da XX Semana Paraense de Informática (SEPAI), fornece uma análise detalhada do ambiente com respeito às práticas de gestão de projetos. Sales (2006) descreve o mecanismo de recuperação e reutilização de processos. Nascimento (2006) fornece uma visão geral de uma metodologia apoiada pelo WebAPSEE para guiar atividades de medição de projetos e sua análise.

Por fim, Paxiúba (2006) descreve o uso de conhecimento organizacional aplicado à simulação e melhoria de processos. Uma avaliação da aderência do WebAPSEE ao MPS-BR é disponível em Covre (2007).

4.3 RECURSOS HUMANOS CAPACITADOS

O WebAPSEE foi assunto principal de cinco trabalhos de conclusão do Curso de Bacharelado em Ciência da Computação da UFPA em 2006. Além disso, o projeto contou com a participação de quinze bolsistas (entre Iniciação Científica e de Mestrado) os quais atuaram em atividades do projeto e implementação de ferramentas. Tais alunos têm tido a oportunidade de participar em um projeto de *software* de alta complexidade, o qual utiliza diversas técnicas de Engenharia de *Software* e paradigmas inovadores de Programação de Computadores no desenvolvimento do núcleo principal e extensões do WebAPSEE.

Além disso, seis destes alunos se especializaram em tarefas diversas envolvendo: o suporte técnico à instalação e uso do produto; a confecção e apresentação de cursos e materiais didáticos relacionados; e atuação no mapeamento e implantação de processos e iniciativas de melhoria organizacional em empresas diversas.

4.4 DISSERTAÇÕES GERADAS

Duas dissertações de mestrado foram geradas em 2006, tendo seus resultados preliminares apresentados no *Workshop* de Teses e Dissertações do Simpósio Brasileiro de Engenharia de *Software* de 2006 estando, no momento desta submissão, aguardando data para realização das defesas na Pós-Graduação em Engenharia Elétrica da UFPA.

4.5 PARCERIAS E PROGRAMAS DE TRANSFERÊNCIA DE TECNOLOGIA EFETUADOS

Antes da liberação da versão 1.0 do sistema para a comunidade de *Software* Livre, versões preliminares da ferramenta estavam disponíveis desde março de 2006 para os parceiros institucionais, a saber: o Centro de Tecnologia

(LACEN) da Eletronorte e SERPRO-Belém. Avaliações de usabilidade do produto se seguiram, além de demandas de melhoria foram registradas e priorizadas para o lançamento oficial do produto. No SERPRO-Belém, a ferramenta foi disponibilizada para acompanhar o andamento de projetos de curta duração. No LACEN, o WebAPSEE vem sendo usado na definição de um modelo de processo para guiar o desenvolvimento de *software* naquela instituição em um projeto de cooperação tecnológica com a Universität Stuttgart (Alemanha) e o Instituto de Estudos Superiores da Amazônia (IESAM). O modelo de processo foi proposto pela equipe do IESAM no primeiro semestre de 2006 e foi mapeado para o WebAPSEE para conduzir pequenos projetos entre agosto e novembro. Um projeto-piloto está sendo acompanhado desde novembro de 2006.

O WebAPSEE é utilizado ainda pela própria equipe do LABES-UFPA para controlar o desenvolvimento de *software* relacionado com demandas diversas. Sabe-se também que a ferramenta (ou suas variações, desenvolvidas a partir do código-fonte disponibilizado) vem sendo utilizada por outras empresas no país.

4.6 OUTROS RESULTADOS

As seções a seguir apresentam resultados que não se enquadram nas categorias acima.

4.6.1 Formação em engenharia e gestão de processos de *software*

A experiência citada anteriormente na implantação da ferramenta na indústria foi determinante para a capacitação de um conjunto de professores e alunos em situações práticas de definição e avaliação de projetos. A partir desta experiência, foi criada a disciplina de “Engenharia de Processos de Software” a qual é fortemente baseada na adoção de tecnologia para apoiar a definição e implantação de processos de *software*, onde o WebAPSEE é usado tanto em exercícios práticos de modelagem quanto no acompanhamento de projetos conduzidos em outras disciplinas. Esta disciplina foi ministrada no primeiro semestre de 2006 para o curso de Bacharelado em Sistemas da Informação da UFPA. No segundo semestre de 2006, uma versão expandida foi ministrada em turmas para a graduação e no Mestrado em Ciência da Computação. A experiência com essa disciplina foi determinante para que os autores propusessem em dezembro de 2006 a criação de um curso de Especialização em Gerência de Projetos de *Software*, com proposta aprovada na UFPA para início em março de 2007.

4.6.2 Repercussão do produto na mídia local e em eventos especializados

O produto WebAPSEE foi oficialmente lançado à comunidade de Tecnologia de Informação em outubro de 2006 no auditório do CAPACIT-UFPA, com a presença de aproximadamente 100 pessoas⁵. Nessa ocasião, foi possível demonstrar a ferramenta e disseminar o trabalho sendo realizado pela equipe do LABES. A repercussão foi muito positiva, com destiques de reportagens na Imprensa local e em sites de notícias relacionados à Computação. Além da premiação de melhor artigo já citada na seção 4.2, outras palestras convidadas foram apresentadas,

onde podemos citar: I *Workshop de Integração de Ferramentas para a Melhoria do Processo de Gestão de Desenvolvimento e Comercialização Colaborativos de Software* (promovido pelo ICMC-USP em maio/2006), apresentações na SEFA (Secretaria de Estado da Fazenda), Banpará (Banco do Estado do Pará) e Banco da Amazônia, além da apresentação no próprio EQPS 2006 em Belém, dentre outras.

Foi bastante positiva também a participação na Sessão de Ferramentas do SBES 2006, onde houve demonstração da ferramenta e suas funcionalidades a pesquisadores da área e elogios foram feitos ao trabalho.

5. APlicabilidade dos resultados

O ano de 2006 representou a evolução de uma proposta acadêmico-científica para um produto de alta tecnologia de propósito geral para atender à forte demanda por ferramentas especializadas na gestão do processo de *software*. A adoção prática de PSEEs na indústria é um desafio relatado pela literatura especializada desde as primeiras propostas da década de 1990. A concepção do WebAPSEE se valeu, então, simultaneamente de tecnologias recentes e formalismos adequados para facilitar o seu uso e evolução. Assim, o fato de ser baseado em tecnologia *Web* (LIMA, 2006a), técnicas de inteligência artificial, e Gramática de Grafos (entre outros métodos formais) na especificação do seu núcleo funcional (LIMA REIS, 2006) constituem exemplos do cuidado tomado desde a concepção do produto.

A adoção satisfatória do WebAPSEE em empresas preocupadas com certificações de qualidade industrial, como o SERPRO-Belém e a Eletronorte – é natural. Nestes casos, as empresas estão plenamente convencidas das vantagens na adoção de modelos de processos rigorosos e a tarefa se resume ao mapeamento destes modelos para a notação fornecida pelo ambiente. Entretanto, deve-se observar o grande interesse que a tecnologia desperta nas micro e pequenas empresas que se sentem 5 Fotografias do lançamento do WebAPSEE 1.0 estão disponíveis no item Eventos no site do LABES, atraídas pela possibilidade de se definir processos e imediatamente obter seus resultados através da monitoração em tempo real fornecida pelo ambiente. A facilidade para lidar com mudanças durante a evolução dos processos – tão comuns em organizações com baixa maturidade e recursos limitados – também tem se demonstrado atrativa.

Outra questão frequentemente elogiada é a facilidade na adoção de práticas de gestão de configuração de *software* fornecida pelo ambiente. Assim, embora o WebAPSEE e todo ferramental tecnológico fornecido esteja alinhado aos níveis mais altos de maturidade de *software*, organizações que buscam certificações iniciais ou mesmo apenas desejam uma melhoria no seu controle têm se valido da ferramenta para auxiliá-los nesta tarefa. 6. Características Inovadoras O ambiente WebAPSEE apresenta uma série de componentes que fornecem funcionalidades inovadoras mesmo quando analisadas em separado. A base formal – que dá sustentação à arquitetura de *software* utilizada – estabelece mecanismos que foram pensados para atender aos requisitos de qualidade de uma ferramenta de grande porte.

Assim, a arquitetura foi desenhada para atender a uma demanda crescente de usuários, permitir a fácil integração de novas ferramentas, e integrar diferentes protocolos de comunicação distribuída. As técnicas de Inteligência Artificial

integraram-se nos componentes que permitem apoio a decisão, como alocação de pessoas (no mecanismo baseado em Políticas de Instanciação), simulação de processos de *software* (baseada em conhecimento), e reutilização de processos (com uso da técnica de raciocínio baseado em casos). Toda a documentação está disponível no *website* do projeto (LABES, 2007), constituindo elemento de distinção perante seus congêneres, cuja documentação normalmente se restringe ao código-fonte. Ressalta-se que, na opinião dos autores, a maior inovação do projeto consiste na integração de todas estas funcionalidades em uma única ferramenta.

A sinergia desta integração permite, por exemplo, que o mecanismo de alocação de pessoas possa definir políticas reutilizáveis em diferentes processos, e estas políticas possam tomar decisão com base no estado corrente (registrado pelo log de execução). Assim, é possível obter mais benefícios que o obtido com soluções isoladas.

6 CONCLUSÃO E PERSPECTIVAS FUTURAS

O uso prático da ferramenta WebAPSEE na definição e implantação de processos de *software* em organizações de pequeno a médio porte constituem a comprovação, com base nas evidências desta realidade, do grande interesse dedicado a este tema. As principais limitações existentes na ferramenta atual dizem respeito à dificuldade em sua instalação e de seus componentes. Por ser baseada em outros produtos (como o MySQL e CVS) a instalação exige a configuração de diversos elementos, os quais são influenciados pela plataforma de execução (e.g., Windows ou Linux), características da rede e outros detalhes técnicos que paulatinamente vão sendo identificados e incorporados à documentação do projeto.

A proposta da ferramenta vem demonstrando o impacto social, técnico e econômico que ela pode causar: sua construção ajuda na formação de recursos humanos altamente capacitados, na contribuição científica à Engenharia de *Software*, enquanto que o seu, livre, ajuda a empresas locais na implantação de melhoria de processos de *software*. Por fim, a visibilidade do projeto também traz bons frutos para a Região Norte, e amplia os horizontes do grupo para mais projetos inovadores.

Fonte: Disponível em: <http://www.mct.gov.br/upd_blob/0014/14602.pdf>. Acesso em: 16 set. 2016.

RESUMO DO TÓPICO 3

Neste tópico vimos que:

- Técnicas de Modelagem de processos.
- Método Eriksson Penker:
 - Modelo de Dependência Estratégica.
 - Modelo de Razão Estratégica.
- Método EKD (Enterprise Knowledge Development).
- Reutilização de processos de *software*.
- Uma das formas de obter a qualidade do produto de *software* é realizar o seu desenvolvimento buscando entender claramente o domínio do negócio, considerando os processos de negócio e a compreensão do ambiente organizacional como fontes relevantes para a elicitação de requisitos.
- O uso de modelos que descrevem os processos de negócios e o contexto de uma organização agrega benefícios para o desenvolvimento de *software*, tais como:
 - os requisitos passam a refletir as necessidades do negócio;
 - baixo número de redundâncias de requisitos;
 - o desenvolvimento do *software* passa a ser guiado pela necessidade do negócio.
- Os sistemas de negócio são sistemas abertos. Não podem ser considerados como sistemas fechados, pois necessitam de informações e recursos de outros negócios, muitas vezes, externos ao processo de negócio em questão. Fornecedores, normas, clientes, tecnologias e leis fazem parte deste círculo.
- O método Erikson-Penker criou um método que é o resultado de um conjunto de elementos baseados em modelos já validados da UML para representar os processos de negócio de uma organização. Estes, podem ser reutilizados pelo analista de negócio nas fases específicas de arquitetura e desenvolvimento de *software* (ERIKSSON; PENKER, 2000).
- O Modelo de Dependência Estratégica fornece uma descrição intencional de um processo em termos de uma rede de relacionamentos de dependência entre atores.
- O Modelo de Razão Estratégica (SR) fornece uma descrição intencional do processo em termos de elementos do processo e as razões que estão por detrás deles.

- O EKD viabiliza a construção de outros modelos. Sua grande contribuição é que este modelo é voltado para o trabalho em equipe, pois explora o conhecimento de atores de diferentes áreas. Isso contribui muito para a construção de um modelo mais solidificado e voltado para adequados processos de negócio. Outra característica importante é o fato de facilitar o aprendizado organizacional, fazendo com que os envolvidos tenham total conhecimento e domínio das regras de negócio envolvidas em cada processo.
- O modelo EKD pode ser percebido e utilizado nas três fases distintas dos processos de negócio: diagnóstico, análise e projeto, respectivamente.
- Reutilizar processos de *software* é uma abordagem importante para:
 - o Compartilhar experiências de projetos anteriores para melhor adequação de processos posteriores.
 - o Promover a reutilização dos conhecimentos de pessoas-chave dentro do processo.
 - o Facilitar o aprendizado de pessoas menos experientes.
- O Ciclo de Vida para Reutilização de Processos de *Software* compreende as etapas de:
 - o Modelagem de Processos Visando a Reutilização.
 - o Reutilização de Processos.
 - o Execução de Processos.
 - o Generalização e Avaliação de Processos Encerrados.
- Os principais conceitos envolvidos na reutilização de processos são:
 - o **Elemento de Processo:** São os componentes do processo e as atividades. Somente os componentes podem ser reutilizados.
 - o **Componente de Processo:** os componentes são importantes, pois podem ter desempenho e apresentar várias versões. Um componente é a unidade básica de um processo.



1 Cite as principais características do modelo EKD.

2 Dentro do ciclo de vida de reutilização de processos existe a etapa de reutilização de processos. Disserte sobre o seu entendimento acerca desta etapa.

3 A respeito dos processos e métodos em engenharia de *software*, assinale a opção correta:

- a) A qualidade de *software* avalia se os métodos e processos empregados devem ser aperfeiçoados, após a entrega final do produto de *software*.
- b) Na engenharia de *software*, o processo define uma metodologia, ao passo que os métodos determinam os procedimentos técnicos.
- c) Os métodos são elementos que proporcionam suporte para o controle do gerenciamento de projeto e estabelecem o marco do projeto.
- d) Em cada processo e método, as ferramentas da engenharia de *software*, denominadas CASE, devem ser utilizadas de forma específica, de maneira que as informações geradas não sejam integradas.

4 A engenharia de *software* baseada em componentes toma como base a reutilização de componentes de *software* padronizados. Em boa parte dos projetos de *software* já existe a prática do reuso de componentes. Isso ocorre pela experiência da equipe que busca em projetos anteriores códigos semelhantes à necessidade atual. Isso agiliza o desenvolvimento do sistema e apesar de uma boa prática, os envolvidos no reuso podem se deparar com problemas como:

- a) Dependência de linguagem de programação dos componentes reusados.
- b) Falta de padronização dos componentes reusados.
- c) Alto custo de desenvolvimento dos componentes reusados em comparação ao custo de integração e de teste dos mesmos.
- d) Confiabilidade e certificação dos componentes reusados.



UNIDADE 2

ENGENHARIA DE SOFTWARE E DE REQUISITOS E GERENCIAMENTO DE PROJETO DE SOFTWARE

OBJETIVOS DE APRENDIZAGEM

A partir desta unidade você será capaz de:

- conhecer a evolução do *software*;
- entender o que foi a crise do *software*;
- conhecer alguns mitos referente ao *software*;
- reconhecer a importância da análise de requisitos no processo de desenvolvimento;
- perceber as diferentes etapas que fazem parte da engenharia de requisitos;
- realizar de forma consistente a atividade de análise de requisitos;
- conhecer como acontece o gerenciamento de projeto de *software*.

PLANO DE ESTUDOS

Esta unidade está dividida em três tópicos. Em cada um deles, você encontrará atividades que contribuirão para a apropriação dos conteúdos apresentados.

TÓPICO 1 – NATUREZA DO SOFTWARE E ENGENHARIA DE SOFTWARE

TÓPICO 2 – REQUISITO E ENGENHARIA DE REQUISITO

TÓPICO 3 – GERENCIAMENTO DE PROJETO DE SOFTWARE

NATUREZA DO SOFTWARE E ENGENHARIA DE SOFTWARE

1 INTRODUÇÃO

Vamos começar essa unidade definindo *software*, segundo Pressman (2011, p. 32):

Software consiste em (1) instruções (programas de computador) que, quando executadas, fornecem características, funções e desempenho desejados; (2) estrutura de dados que possibilitam aos programas manipular informações adequadamente; e (3) informações descritivas, tanto na forma impressa como na virtual, descrevendo a operação e o uso dos programas.

Poderíamos apresentar a você diferentes definições de *software*, algumas mais simples e outras até mais rebuscadas e técnicas. Diferente das coisas que detalhamos com certa facilidade, como é uma casa e até mesmo um computador (*hardware*), um *software* é mais um elemento de sistema lógico do que físico, e dessa forma, tem características que são consideravelmente diferentes daquelas do *hardware* (PRESSMAN, 2011).

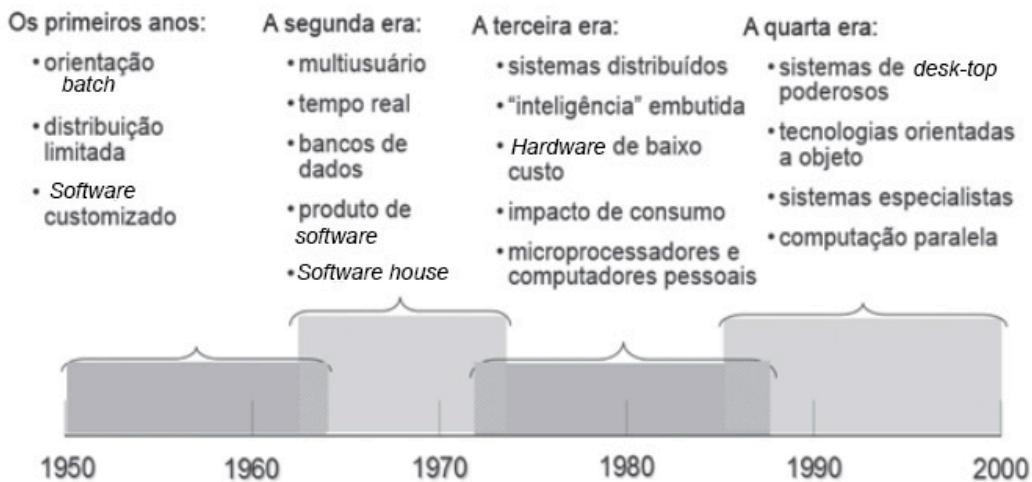
Mas antes de continuarmos, vamos voltar à origem do *software*. Uma época quando o *software* era quase indissociável do *hardware*. Pressman (1995, p. 4), afirma que: “durante as três primeiras décadas da era do computador, o principal desafio era desenvolver um *hardware* que reduzisse o custo de processamento e armazenagem de dados”, problema que hoje já não existe mais.

Pressman (1995, p. 5): “descreve a evolução do *software* dentro do contexto das áreas de aplicação de sistemas baseados em computador” através do tempo, como podemos perceber através da figura a seguir. Percebe-se que foi o *hardware* que nos primeiros anos de desenvolvimento computadorizado sofreu mudanças contínuas, enquanto o *software* era deixado para trás. O *software* era feito virtualmente sem administração, até o momento em que o tempo começasse a se esgotar e os custos subissem vertiginosamente. Em raras exceções o *software* não se dedicava à execução de um único programa, e esse, por sua vez, também se dedicava a uma aplicação exclusiva.



Conheça um pouco mais dos primeiros softwares: SAGE. Acesse: <https://www.ibm.com/developerworks/community/blogs/tlcbr/entry/sage_um_berco_de_inovacao?lang=en>.

FIGURA 26 - EVOLUÇÃO DO SOFTWARE



FONTE: Disponível em: <<http://www.batebyte.pr.gov.br/modules/conteudo/conteudo.php?conteudo=299>>. Acesso em: 29 mar. 2016.

O *software* era projeto feito sob medida para cada aplicação, e sua distribuição era relativamente pequena. Pressman (1995, p. 5), diz que “O produto *software* (isto é, programas desenvolvidos para serem vendidos a um ou mais clientes) estava em sua infância”.

O *software* era desenvolvido e usado pela própria pessoa, ou pela organização em que ela trabalhava. O processo consistia em você mesmo escrever o programa e colocá-lo em funcionamento, e se ele não funcionasse, você mesmo faria as correções. E como o pessoal que fazia esse tipo de serviço tinha pouca rotatividade, os gerentes podiam dormir tranquilos, pois o funcionário que desenvolveu o programa com certeza estaria no dia seguinte para consertá-lo. Devido a esse cenário, dá para imaginar que tanto o projeto como a documentação estavam quase em sua totalidade na cabeça do desenvolvedor.

Considerado como a **Segunda Era da Evolução** dos sistemas computadorizados ela estendeu-se de meados da década de 1960 até o final da década de 1970. Para Azevedo (2009), “A multiprogramação e os sistemas multiusuários introduziram novos conceitos de interação homem-máquina. E as técnicas interativas abriram um novo mundo de aplicações e novos níveis

de sofisticação de *software* e *hardware*". Com o advento dos sistemas de tempo real que podiam coletar analisar e transformar dados de múltiplas fontes, o tempo caiu drasticamente nas aplicações. Os avanços da armazenagem *on-line* levaram à primeira geração de sistemas de gerenciamento de bancos de dados, imprescindíveis até hoje.

Martins Azevedo (2013) afirma que esta segunda também foi caracterizada pelo uso do produto de *software* e pelo advento das "*software houses*", ou fábricas de *software*, onde o *software* era desenvolvido para ampla distribuição num mercado interdisciplinar. Empresários da indústria, governos e universidades puseram-se "desenvolver pacotes de *software*" e a ganhar muito dinheiro, eles desenvolviam programas para *mainframes* e minicomputadores e esses eram distribuídos para centenas e, às vezes, milhares de usuários. Conforme o número de sistemas baseados em computador crescia, começava também a expansão das bibliotecas de *software* de computador. "Projetos de desenvolvimento internos nas empresas produziram dezenas de milhares de instruções de programa" (PRESSMAN, 1995, p. 6).

Ainda segundo o autor, nem tudo eram flores, uma nuvem negra apareceu no horizonte, pois todos esses programas e todas essas instruções precisavam de manutenção, seja por solicitação do usuário ou devido à alteração de um *hardware* que havia sido adquirido.

Essas atividades foram chamadas coletivamente de manutenção de *software*. E ao esforço despendido na manutenção de *software* começou a absorver recursos em índices alarmantes. E, ainda pior, a natureza personalizada de muitos programas tornava-os virtualmente impossíveis de sofrer manutenção. Uma "crise de *software*" agigantou-se no horizonte. (AZEVEDO, 2013, p. 6).



Falaremos logo a seguir sobre a crise do *software* e também sobre os mitos.

A terceira era teve seu início em meados da década de 1970 e continua até hoje em conjunto com a quarta era, segundo Pressman (1995, p. 7).

Os sistemas distribuídos – múltiplos computadores, cada um executando funções concorrentemente e comunicando-se um com o outro aumentaram intensamente a complexidade dos sistemas baseados em computador. As redes globais e locais, as comunicações digitais de largura de banda (*bandwidth*) elevada e a crescente demanda de acesso "instantâneo" a dados exigem muito dos desenvolvedores de *software*. A terceira era também foi caracterizada pelo advento e generalização do uso de microprocessadores, computadores pessoais e poderosas estações de trabalho (*workstations*) de mesa.

"Essa era também foi caracterizada pelo aparecimento e uso generalizado de microprocessadores, computadores pessoais e poderosas estações de trabalho "workstations" de mesa", segundo Martins Azevedo (2013, p. 6). O microprocessador gerou um amplo conjunto de produtos inteligentes. Quase tudo hoje, poucas coisas escapam, desde o automóvel a fornos micro-ondas, de robôs industriais a equipamentos para diagnóstico de soro sanguíneo, todos eles usam algum *software*.

Em muitos casos, a tecnologia de *software* está sendo integrada a produtos, por equipes técnicas que entendem de *hardware*, mas que frequentemente são principiantes em desenvolvimento de *software*. O *hardware* de computador pessoal está se tornando rapidamente um produto primário, enquanto o *software* oferece a característica capaz de diferenciar. (AZEVEDO, 2013, p. 6).

A quarta era do *software* de computador está apenas começando. Para Pressman (1995, p. 7), onde as tecnologias orientadas a objetos estão rapidamente ocupando o lugar das abordagens mais convencionais para o desenvolvimento de *software* em muitas áreas de aplicação. As técnicas de "quarta geração" para o desenvolvimento de *software* já estão mudando a maneira segundo a qual alguns segmentos da comunidade de *software* constroem programas de computador. Os sistemas especialistas e o *software* de inteligência artificial finalmente saíram do laboratório e já são utilizados em aplicações práticas em problemas de amplo espectro do mundo real. O *software* de rede neural artificial abriu excitantes possibilidades para o reconhecimento de padrões e para capacidades de processamento de informações semelhantes as humanas. Quando nos movimentamos para a **quinta era ou era atual**, os problemas associados ao *software* de computador continuam a se intensificar, segundo Pressman (1995, p.8):

- A sofisticação do *software* ultrapassou nossa capacidade de construir um *software* que extraia o potencial do *hardware*.
- Nossa capacidade de construir programas não pode acompanhar o ritmo da demanda de novos programas.
- Nossa capacidade de manter os programas existentes é ameaçada por projetos ruins e recursos inadequados.

2 CRISE DO SOFTWARE

Foi durante os anos 70 que o termo crise do *software* surgiu, uma época em que a engenharia de *software* praticamente inexistia. Esse termo estava relacionado às dificuldades enfrentadas no desenvolvimento de *software*, inerentes ao aumento das demandas e da complexidade delas, aliado à falta de técnicas apropriadas para resolver esses desafios.

O termo crise do *software* expressava as dificuldades do desenvolvimento de *software* devido ao rápido crescimento da demanda por *softwares*, a complexidade dos problemas a serem resolvidos e a inexistência de técnicas de desenvolvimento. Quando o desenvolvimento de *softwares* começou a utilizar os princípios das linguagens estruturadas e modulares, as empresas de *softwares* começaram a falhar constantemente nos prazos de entrega, apresentando

resultados insatisfatórios e além dos orçamentos que ultrapassavam o predeterminado. Um relatório em 1969 apresentou que, de 50% a 80% dos projetos não foram concluídos e outros fracassados por não terem atingindo os objetivos esperados, e dos que foram concluídos, foram entregues acima do prazo estipulado e com orçamento acima daquilo que foi predeterminado. Estes problemas estavam relacionados, principalmente, com a forma de trabalho da equipe, envolvendo também dúvidas em relação aos requisitos do sistema. Todos estes fatores exigiram a existência de novos métodos e a aprimoramento dos métodos já existentes, foi aí que nasceu então a "Engenharia de Software", que amenizou de forma considerável todos estes problemas citados, adotando métodos e principalmente os modelos de processos de software, onde toda tarefa de desenvolvimento eram divididas em etapas, possibilitando a coleta de dados, a interação com o cliente, a apresentação de protótipos do sistema, a estipulação dos prazos de entregas com margem de erros e finalmente, a apresentação de um sistema robusto, eficiente e que viesse satisfazer o cliente apresentando de fato uma solução para o problema existente.

FONTE: Adaptado de <<http://www.clipatecinformatica.com.br/2011/02/crise-do-software.html>>. Acesso em: 4 jun. 2016.

Diante de tantos problemas no desenvolvimento dos softwares, a engenharia de software que até o momento estava começando a aparecer, deparou-se com a tarefa de reduzir os problemas apresentados pela crise do programa. Como saída encontrada, tivemos a padronização no desenvolvimento de software. As primeiras metodologias desenvolvidas são hoje conhecidas como metodologias clássicas, que possuem como principal característica a rigidez e altos níveis de controle, na qual todos os requisitos devem ser conhecidos inicialmente, e têm o seu desenvolvimento em etapas, gerando como ônus uma grande quantidade de documentação, além de que cada fase deve ser finalizada e aprovada para poder seguir para a próxima. Apesar de inúmeras críticas, esse tipo de metodologia ainda é muito utilizado. Outras técnicas foram criadas para amenizar os problemas das metodologias clássicas, mas essa metodologia foi muito importante durante a crise do software.

Para ilustrar, vamos imaginar a complexidade de um software utilizado em aparelhos celulares com reconhecimento de voz que se utiliza da inteligência artificial. Muitos no início acreditavam nisso como ficção científica. Mas com a evolução, softwares de milhões de linhas de instrução apareceram como sistemas operacionais de mercado conhecidos mundialmente. Além dessa imensa complexidade, temos também o fenômeno da urgência em se desenvolver esses softwares, para atender às necessidades de mercado. Devido a todos esses fatos, surgiu o conceito de crise do software, podendo também ser verificado por vários sintomas, como os apresentados por Engholm Jr (2011, p. 33):

- Software de baixa qualidade.
- Projetos com prazos e custos maiores que os planejados.
- Software não atendendo aos requisitos dos stakeholders.
- Custos e dificuldades no processo de manutenção.

Ao analisarmos os sintomas apresentados e compararmos com o que vemos hoje em dia, podemos verificar que a crise ainda está presente. E que mesmo tendo à disposição técnicas apropriadas, ainda nos deparamos com esses problemas. Até mesmo empresas que têm conhecimento das técnicas indicadas, às vezes, não conseguem praticá-las por pressão do próprio cliente, por exemplo, em relação a prazos (ENGHOLM JR, 2011).

3 A NATUREZA DO SOFTWARE

Depois do seu histórico de coadjuvante, o *software* agora assume um duplo papel. Ele é um produto e, ao mesmo tempo, o veículo para distribuir um produto.

Como um produto, fornece o potencial computacional representado pelo *hardware* ou, de forma mais abrangente, por uma rede de computadores que podem ser acessados por *hardware* local. Independentemente de residir em um celular ou operar dentro de um *mainframe*, o *software* é um transformador de informações, produzindo, gerenciando, adquirindo, modificando, exibindo ou transmitindo informações que podem ser tão simples quanto um único *bit* ou tão complexas quanto uma apresentação multimídia derivada de dados obtidos de dezenas de fontes independentes (PRESSMAN, 2011, p. 31).

O *software* se tornou tão importante, que ele é o responsável pela distribuição do produto mais importante de nossa era, a **INFORMAÇÃO**.

As últimas cinco décadas foram de muitas mudanças para o *software*, e atualmente uma enorme indústria de *software* tornou-se fator dominante nas economias do mundo industrializado. Para Pressman (2011), o programador solitário foi substituído por equipes de especialistas, onde cada qual se concentra em uma parte da tecnologia necessária para distribuir uma aplicação complexa. Mas, questões levantadas pelos solitários programadores persistem até hoje (PRESSMAN, 2011, p. 31):

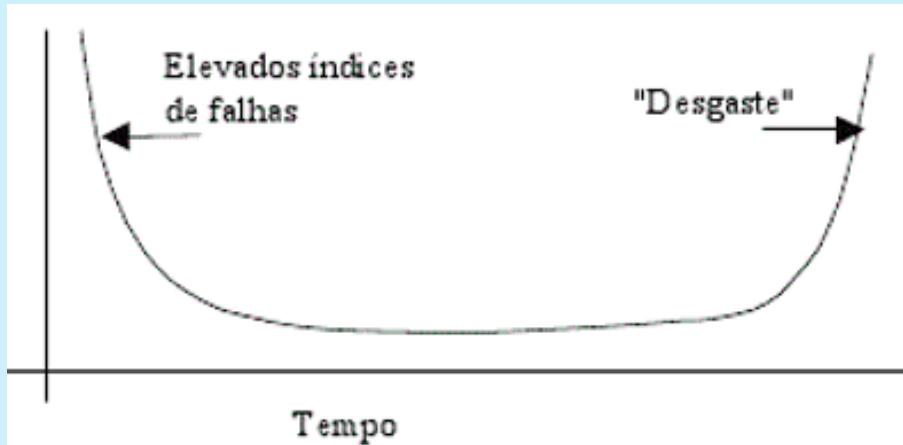
- Por que concluir um *software* leva tanto tempo?
- Por que os custos de desenvolvimento são tão altos?
- Por que não conseguimos encontrar todos os erros antes de entregarmos o *software* ao cliente?
- Por que gastamos tanto tempo e esforço mantendo programas existentes?
- Por que continuamos a ter dificuldades em medir o progresso enquanto o *software* está sendo desenvolvido e mantido? (PRESSMAN, 2011, p. 31)

Para o autor, esses e diversos outros questionamentos evidenciam a preocupação com o *software* e o modo como é desenvolvido, inquietação essa que tem levado à adoção da prática da engenharia de *software*.

Na figura a seguir, temos um exemplo como acontecem os problemas e o desgaste de um *hardware*.

O índice das falhas é relativamente alto no início de seu ciclo de vida devido a erros de fabricação e projeto. Mas, à medida que estes erros são corrigidos, o produto tende a chegar a um nível estável, dentro dos limites aceitáveis. No final do ciclo de vida, as falhas voltam a aumentar novamente por causa do desgaste causado pelo uso, acúmulo de poeira, variações de temperatura, entre outros.

FIGURA 27 - CURVA DE DEFEITO DO HARDWARE

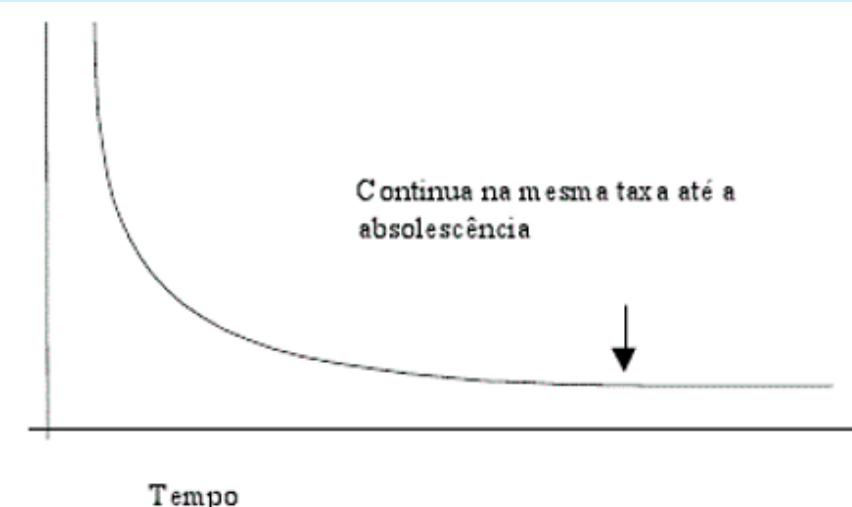


FONTE: Pressman (2010)

Em oposição ao *hardware*, podemos observar as figuras que seguem, onde observamos a diferença das falhas entre *hardware* e *software* durante seu ciclo de vida.

Diferente do *hardware*, um *software* não se desgasta ao longo do tempo por não ser um produto físico. É parte de um sistema lógico. Se um *software* é lançado sem erros, vai permanecer sem erros, desde que sejam mantidas as condições para o seu funcionamento. Então, teoricamente a curva de falhas de um *software* deveria tender ao achatamento, estabilizando e reduzindo ao mínimo as falhas ao longo do ciclo de vida. A figura a seguir, seria o ciclo de vida ideal de um *software*.

FIGURA 28 - CURVA DE FALHAS PARA O SOFTWARE (IDEALIZADA)

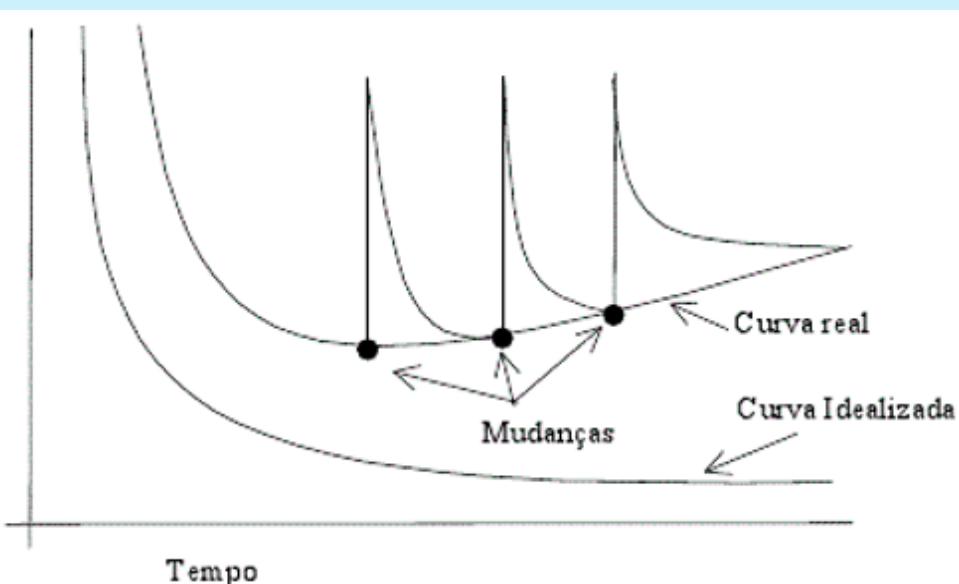


FONTE: Pressman (2010)

Mas na prática o *software* pode se deteriorar em função de erros introduzidos durante a sua manutenção. Alterações num *software* podem gerar novos erros antes mesmo da estabilização dos erros de alterações anteriores. Isto faz com que a curva de falhas de um *software* se pareça com esta apresentada na figura a seguir.

Podemos ver que, a cada mudança, o índice de falhas tende a aumentar até se estabilizar novamente, mas o índice real cresce ao longo do tempo. A curva real está bem acima da curva ideal porque toda vez que se faz uma mudança, mais erros aparecem.

FIGURA 29 - CURVA REAL E CURVA IDEALIZADA DE FALHAS DE UM SOFTWARE



FONTE: Pressman (2010)

Eliminar totalmente não, mas existem técnicas que, se aplicadas corretamente, podem ajudar a entregar um *software* de qualidade. Aliás, a qualidade não está associada somente às falhas, mas a outros fatores como desempenho, usabilidade, confiabilidade, satisfação do cliente, entre outros. Isto é uma preocupação da Engenharia de *Software*, disciplina que lida com todos os aspectos e etapas da produção de *software* e que tem como objetivo fornecer métodos para construir *softwares* de alta qualidade.

FONTE: Disponível em: <<http://blog.tecsystem.com.br/index.php/erros-e-acertos-no-desenvolvimento-de-software/>>. Acesso em: 31 mar. 2016.



No próximo tópico voltaremos a falar da engenharia de *software* e como amenizar esses problemas através do levantamento de requisitos e da gerência de projetos. Mas, antes de continuarmos vamos ver a aplicação dos *softwares* através das suas características.

4 APLICAÇÕES DO SOFTWARE

Já conhecemos um pouco da história evolutiva do *software* e também que o *software* é um conjunto de algoritmos codificados que permite ao computador executar uma operação ou um conjunto de operações culminando em tarefas. Analisaremos, aqui, os tipos de *software* disponíveis, bem como a função e utilidade desses tipos de *software*.



Conheça um pouco mais sobre a internet das coisas: Acesse: <<http://www.timaio.com.br/m/capa-4/artigo-internet-das-coisas/>>.

Segundo Martins Azevedo (2013, p. 7), “desenvolver categorias genéricas para as aplicações de *softwares* é uma tarefa muito difícil. Quanto mais complexo é o sistema, mais difícil é determinar de forma clara os vários componentes do *software*”. Para o Pressman (1995) podem-se dividir as aplicações em:

Software Básico: É um conjunto de programas para dar apoio a outros programas. Eles têm como característica uma forte interação com o *hardware*, operações concorrentes, compartilhamento de recursos, uso por múltiplos usuários e múltiplas interfaces. Como por exemplo: compiladores, editores e gerenciamento de arquivos.

Software de Tempo Real: São programas que monitoram, analisam e controlam eventos do mundo real, devendo responder aos estímulos do mundo externo com restrições de tempo pré-determinadas. Deve-se notar que o termo “*tempo real*” difere de “*interativo*” ou “*time-sharing*” (tempo compartilhado). Um sistema em tempo real deve responder dentro de restrições de tempo estritas.

Software Comercial: O processamento de informações comerciais é a maior área de aplicação de *softwares*, são aplicações que gerenciam as operações comerciais de modo a facilitar o gerenciamento comercial do negócio da empresa, permitindo também a tomada de decisões. Além das aplicações de *software* comerciais (folha de pagamento, contas a pagar e a receber etc.) os *softwares* comerciais também abrangem a computação interativa, como, por exemplo, o processamento de transações em pontos de venda.

Software Científico e de Engenharia: São caracterizados por algoritmos de processamento numérico, dependentes da coleta e processamento de dados para as mais variadas áreas do conhecimento. Suas aplicações variam da astronomia à vulcanologia, por exemplo.

Software Embutido: São *softwares* que são desenvolvidos para executar atividades muito específicas e inseridas em produtos inteligentes, tanto para atividades comerciais como para atividades domésticas. Também conhecidos por *embedded software*, ele reside na memória só de leitura e é usado para controlar produtos e sistemas para os mercados industriais e de consumo. Exemplo desses *softwares* são os teclados de micro-ondas, funções digitais de um automóvel.

Software de Computador Pessoal: São os *softwares* desenvolvidos para o uso pessoal do computador, tais como planilhas eletrônicas, processadores de textos, jogos etc. Eles continuam a representar os mais inovadores projetos de interface com os seres humanos de toda indústria de *software*.

Software de Inteligência Artificial: Também conhecidos por *Artificial Intelligence – AI*, faz uso de algoritmos não numéricos para resolver problemas complexos que não apresentam facilidades computacionais numéricas ou de

análise direta. São exemplos os *softwares* especialistas, os de reconhecimento de padrões como voz e imagem.

Aplicações Web: Também conhecidas como “WebApps”, e, em sua forma mais simples, as WebApps podem ser pouco mais que um conjunto de arquivos de hipertexto interconectados, apresentando informações por meio de texto e informações gráficas limitadas. Mas, com o aparecimento da Web 2.0, essas aplicações evoluíram e se transformaram em sofisticados ambientes computacionais que não apenas fornecedores de recursos especializados, de funções computacionais e de conteúdo para o usuário final, mas também estão integradas a bancos de dados corporativos e aplicações comerciais (PRESSMAN, 2011, p. 35).



Acesse o site: <<http://canaltech.com.br/materia/cinema/filmes-imperdiveis-sobre-inteligencia-artificial-49625/>> e veja uma lista dos 10 melhores filmes sobre IA. Entre eles estão: Matrix (1999), A.I. – Inteligência Artificial (2001), Ex-Machina (2015).

Agora que já conhecemos o histórico e a categorização do *software*, vamos conhecer um pouco da **ENGENHARIA DE SOFTWARE**, e como ela pode nos auxiliar a desenvolver *softwares* que estejam preparados para enfrentar os desafios atuais, segundo Pressman (2011, p. 38):

- O *software* tornou-se profundamente incorporado em praticamente todos os aspectos do nosso dia a dia, e como consequência um crescente número de pessoas interessadas nos recursos e nas funções oferecidas pelos *softwares*. Quando no desenvolvimento de um *software*, muitos usuários devem ser ouvidos, pois cada um deles possui pontos de vista (ideias) diferentes de funções, recursos e dispositivos que o *software* deve incorporar. Assim sendo, desprenda-se do conceito de que a engenharia é “um esforço concentrado para compreender o problema antes de desenvolver uma solução de *software*”. Dessa forma devemos fazer um **esforço concentrado para compreender o problema antes de desenvolver uma solução**.
- Os requisitos de tecnologia de Informação demandados por indivíduos, empresas e órgãos governamentais estão se tornando cada vez mais complexos a cada ano. Atualmente, equipes numericamente grandes desenvolvem programas de computador que antigamente eram desenvolvidos por um único indivíduo. Antes, um *software* sofisticado era implementado de forma independente e previsível em um ambiente computacional; hoje, está incorporado em tudo, desde produtos eletrônicos de consumo a equipamentos médicos e sistemas de armamentos, em que a complexidade desses novos produtos e sistemas exige uma maior atenção para com as interações de todos os elementos do sistema. Então, conclui-se que **projetar tornou-se uma atividade-chave (fundamental)**.

- Indivíduos, negócios e governos dependem, de forma crescente, de um *software* para decisões estratégicas e táticas, assim como para o controle das atividades do dia a dia e, dessa maneira, se o *software* falhar, poderão vivenciar desde pequenos inconvenientes até falhas catastróficas. Assim sendo um *software* deve **apresentar qualidade elevada**.

- À medida que o valor de uma aplicação específica aumenta, a probabilidade é de que sua base de usuários e longevidade também cresçam. Com esse crescimento, tanto de usuários como da longevidade, a demanda por manutenção também crescerá, por isso um *software* deve ser **passível de manutenção**.

Essas simples constatações nos levam a concluir que: “*Software, em todas as suas formas e em todos os seus campos de aplicação, deve passar pelos processos de engenharia*”.

FIGURA 5 - CAMADAS DE ENGENHARIA DE SOFTWARE



FONTE: Adaptado de Pressman (2011, p. 39)

Entre tantas definições, Bauer (apud PRESSMANN, 2011, p. 39) define a engenharia de *software* como “o estabelecimento e o emprego de sólidos princípios de engenharia de modo a obter *software* de maneira econômica, que seja confiável e funcione de forma eficiente em máquinas reais”.

Já o IEEE – Institute of Electrical and Electronic Engineers – (apud PRESSMANN, 2011, p. 39) desenvolveu uma definição mais abrangente: “a aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, na operação e na manutenção de *software*; isto é, a aplicação de engenharia ao *software*”.

Com base nessas definições, Pressman (2011) explica a figura e afirma que a engenharia de *software* é uma tecnologia em camadas, e refere que ela deve estar fundamentada em um comprometimento organizacional com o **foco na qualidade**. A pedra fundamental que sustenta a engenharia de *software* é o foco na qualidade.

Ainda segundo Pressman (2011), a base da engenharia de *software* é a camada de **processos**, ou seja, é a liga que mantém as camadas de tecnologia coesas e possibilita o desenvolvimento de *software* de forma racional e dentro do prazo. O processo define uma metodologia que deve ser estabelecida para a entrega efetiva de tecnologia de engenharia de *software* constituindo a base para o controle do gerenciamento de projetos de *software*, estabelecendo o contexto de

onde métodos técnicos serão aplicados e, quais modelos, documentos, dados, relatórios, formulários serão produzidos e, também com o estabelecimento de marcos, a qualidade é garantida e mudanças são geridas de forma apropriada.

Os **métodos** da engenharia de *software* fornecem as informações técnicas para desenvolver *software*, envolvendo uma ampla gama de tarefas, que incluem a comunicação, análise de requisitos, modelagem de projeto, construção de programa, testes e suporte. Os métodos se baseiam em um conjunto de princípios básicos que governam cada área da tecnologia e inclui atividades de modelagem e outras técnicas descritivas.

Já **ferramentas**, segundo Pressman (2011), da engenharia de *software* fornecem suporte automatizado ou semiautomatizado para o processo e para os métodos e, uma vez integrado, de modo que as informações criadas por uma ferramenta possam ser usadas por outra, é estabelecido um sistema para o suporte ao desenvolvimento de *software*, denominado engenharia de *software* com o auxílio do computador (CASE).

5 PROCESSO DE SOFTWARE

Processo é um conjunto de atividades, segundo Pressman (2011), são ações e tarefas realizadas na criação de algum projeto de trabalho (*work product*). Uma atividade tem por fim atingir um objetivo amplo e é utilizada independentemente da aplicação, do tamanho e da complexidade de esforços ou do grau de rigor com que a engenharia de *software* será aplicada. Uma ação envolve um conjunto de tarefas que resultam num artefato de *software*. Uma tarefa se concentra em um objetivo pequeno, porém bem definido e produz um resultado tangível.

Para o autor, no contexto da engenharia de *software*, um processo não é uma prescrição rígida de como desenvolver um *software*, ao contrário é uma abordagem adaptável que possibilita às pessoas realizar o trabalho de selecionar e escolher o conjunto apropriado de ações e tarefas. A intenção é a de sempre entregar *software* dentro do prazo, a um custo e com qualidade suficiente para satisfazer àqueles que patrocinaram sua criação e àqueles que vão utilizá-lo.

Ainda, segundo Pressman (2011, p. 40), “Uma metodologia (*framework*) de processo estabelece o alicerce para um processo de engenharia de *software* completo, por meio da identificação de um pequeno número de atividades estruturais aplicáveis a todos os projetos, independentemente de tamanho ou complexidade”. Além disso, a metodologia de processo engloba um conjunto de atividades de apoio, também conhecidas como atividades guarda-chuva, aplicáveis em todo o processo de *software*. “Uma metodologia de processo genérico para engenharia de *software* compreende cinco atividades: **Comunicação, Planejamento, Modelagem, Construção e Emprego**” (PRESSMAN, 2011, p. 40).

Comunicação: Antes de iniciar qualquer trabalho técnico, é de vital importância comunicar-se e colaborar com o cliente (e outros interessados). A

intenção é compreender os objetivos das partes interessadas para com o projeto e fazer o levantamento das necessidades que ajudarão a definir as funções e características do *software*.

Planejamento: Qualquer jornada complicada pode ser simplificada caso exista um mapa. Um projeto de *software* é uma jornada complicada, e a atividade de planejamento cria um “mapa” que ajuda a guiar a equipe na sua jornada. O mapa – denominado plano de projeto de *software* – define o trabalho de engenharia de *software*, descrevendo as tarefas técnicas a ser conduzidas, os riscos prováveis, os recursos que serão necessários, os produtos resultantes a serem produzidos e um cronograma de trabalho.

Modelagem: Independentemente de ser um paisagista, um construtor de pontes, um engenheiro aeronáutico, um carpinteiro ou um arquiteto, trabalha-se com modelos todos os dias. Cria-se um “esboço” da coisa, de modo que se possa ter uma ideia do todo – qual será o seu aspecto em termos de arquitetura, como as partes constituintes se encaixarão e várias outras características. Se necessário, refina-se o esboço com mais detalhes, numa tentativa de compreender melhor o problema e como resolvê-lo. Um engenheiro de *software* faz a mesma coisa criando modelos para melhor entender as necessidades do *software* e o projeto que irá atender a essas necessidades.

Construção: Essa atividade combina geração de código (manual ou automatizada) e testes necessários para revelar erros na codificação.

Emprego: O *software* (como uma entidade completa ou como um incremento parcialmente efetivado) é entregue ao cliente, que avalia o produto entregue e fornece *feedback*, baseado na avaliação (PRESSMAN, 2011, p. 40).

Continuando, segundo Pressman (2011), essas atividades metodológicas genéricas podem ser aproveitadas tanto para o desenvolvimento de programas pequenos e simples, como para a criação de grandes aplicações para a internet e para a engenharia de grandes e complexos sistemas baseados em computador. Lembrando que os detalhes do processo de *software* poderão ser bem distintos em cada caso, contudo as atividades continuarão as mesmas.

Pressman (2011) afirma que para muitos projetos de *software*, as atividades metodológicas são aplicadas iterativamente conforme o projeto se desenvolve. Ou seja, comunicação, planejamento, modelagem, construção e emprego são aplicados repetidamente quantas forem as iterações do projeto, sendo que cada iteração produzirá um incremento de *software*. Este disponibilizará uma parte dos recursos e funcionalidades do *software*. A cada incremento, o *software* torna-se mais e mais completo.

As atividades lógicas do processo de engenharia de *software* são complementadas por uma série de atividades de guarda-chuva, geralmente aplicadas ao longo de um projeto, auxiliando a equipe a gerenciar, a controlar o progresso, a qualidade, as mudanças e os riscos (PRESSMAN, 2011, p. 41-42):

- Controle e acompanhamento do projeto: Possibilita que a equipe avalie o progresso em relação ao plano do projeto e tome as medidas necessárias para cumprir o cronograma.
- Administração de riscos: Avalia riscos que possam afetar o resultado ou a qualidade do produto/projeto.
- Garantia da qualidade de *software*: Define e conduz as atividades que garantem a qualidade do *software*.
- Revisões técnicas: Avaliam artefatos da engenharia de *software*, tentando identificar e eliminar erros antes que se propaguem para a atividade seguinte.
- Medição: Define e coleta medidas (do processo, do projeto e do produto). Auxilia na entrega do *software* de acordo com os requisitos; pode ser usada com as demais atividades (metodológicas e de apoio).
- Gerenciamento da configuração de *software*: Gerencia os efeitos das mudanças ao longo do processo.
- Gerenciamento da reusabilidade: Define critérios para o reúso de artefatos (inclusive componentes de *software*) e estabelece mecanismos para a obtenção de componentes reutilizáveis.
- Preparo e produção dos artefatos de *software*: Engloba as atividades necessárias para criar artefatos, por exemplo, modelos, documentos, logs, formulários e listas.



Antes de encerrarmos esse tópico, vamos conhecer alguns mitos relativos aos softwares:

6 MITOS RELATIVOS AO SOFTWARE

Os mitos criados em relação ao *software*, segundo Pressman (2011, p. 45) são: “crenças infundadas sobre o *software* e sobre o processo usado para criá-lo. Elas remontam aos primórdios da computação. Os mitos possuem uma série de atributos que os tornam insidiosos”. Para facilitar o entendimento, vamos exemplificar: “eles parecem ser de fato, afirmações razoáveis (algumas vezes contendo elementos de verdade), têm uma sensação intuitiva e frequentemente são promulgados por praticantes experientes “que entendem do riscado”.

Para entendermos um pouco mais sobre os mitos de *software*, vejamos alguns definidos por Pressman (2011, p. 46-47):

Atualmente, a maioria dos profissionais versados na engenharia de *software* reconhece os mitos por aquilo que eles representam, ou seja, atitudes enganosas que provocaram sérios problemas tanto para gerentes quanto para praticantes da área. Entretanto, antigos hábitos e atitudes são difíceis de ser modificados e resquícios de mitos de *software* permanecem, vamos apresentar alguns para sua melhor compreensão, agrupados por mitos referentes ao gerenciamento,

mitos dos clientes e mitos dos profissionais da área.

Mitos de gerenciamento: Os gerentes que tem responsabilidade sobre *software*, assim como gerentes da maioria das áreas, normalmente estão sob pressão para manter os orçamentos, devem evitar deslizes nos cronogramas além de elevar a qualidade. Como uma pessoa que está se afogando e se agarra a uma tábua, um gerente de *software*, muitas vezes se agarra à crença num mito do *software*, para aliviar a pressão (mesmo que temporariamente).

- **Mito:** Já temos um livro que está repleto de padrões e procedimentos para desenvolver *software*. Ele não supre meu pessoal com tudo que eles precisam saber?

- **Realidade:** O livro com padrões pode muito bem existir, mas ele é usado? Os praticantes da área estão cientes de que ele existe? Esse livro reflete a prática moderna da engenharia de *software*? É completo? É adaptável? Está alinhado para melhorar o tempo de entrega, mantendo ainda o foco na qualidade? Em muitos casos, a resposta para todas essas perguntas é “não”.

- **Mito:** Se o cronograma atrasar, poderemos acrescentar mais programadores e ficarmos em dia (algumas vezes denominado conceito da “horda mongol”).

- **Realidade:** O desenvolvimento de *software* não é um processo mecânico como o de fábrica. Nas palavras de Brooks (95): “acrescentar pessoas num projeto de *software* atrasado só o tornará mais atrasado ainda”. A princípio, essa afirmação pode parecer um contrassenso, no entanto, o que ocorre é que, quando novas pessoas entram, as que já estavam terão de gastar tempo situando os recém-chegados, reduzindo, consequentemente, o tempo destinado ao desenvolvimento produtivo. Pode-se adicionar pessoas, mas somente de forma planejada e bem coordenada.

- **Realidade:** Se uma organização não souber gerenciar e controlar projetos de *software*, ela irá, invariavelmente, enfrentar dificuldades ao terceirizá-los.

- **Mito:** Se eu decidir terceirizar o projeto de *software*, posso simplesmente relaxar e deixar essa empresa realizá-lo.

- **Mitos dos clientes:** O cliente solicitante do *software* computacional pode ser uma pessoa na mesa ao lado, um grupo técnico do andar de baixo, de um departamento de *marketing/vendas*, ou uma empresa externa que encomendou o projeto por contrato. Em muitos casos, o cliente acredita em mitos sobre *software* porque gerentes e profissionais da área pouco fazem para corrigir falsas informações. Mitos conduzem a falsas expectativas (do cliente) e, em última instância, à insatisfação com o desenvolvedor.

- **Mito:** Uma definição geral dos objetivos é suficiente para começar a escrever os programas – podemos preencher detalhes posteriormente.

- **Realidade:** Embora nem sempre seja possível uma definição ampla e estável dos requisitos, uma definição de objetivos ambígua é receita para um desastre. Requisitos não ambíguos (normalmente derivados da iteratividade) são obtidos somente pela comunicação contínua e eficaz entre cliente e desenvolvedor.

- **Mito:** Os requisitos de *software* mudam continuamente, mas as mudanças podem ser facilmente assimiladas, pois o *software* é flexível.

- **Realidade:** É verdade que os requisitos de *software* mudam, mas o impacto da mudança varia dependendo do momento em que ela foi introduzida. Quando as mudanças dos requisitos são solicitadas cedo (antes do projeto ou da codificação terem começado), o impacto sobre os custos é relativamente pequeno. Entretanto, conforme o tempo passa, ele aumenta rapidamente – recursos foram comprometidos,

uma estrutura de projeto foi estabelecida e mudar pode causar uma revolução que exija recursos adicionais e modificações fundamentais no projeto.

• **Mitos dos profissionais da área.** Mitos que ainda sobrevivem nos profissionais da área têm resistido por mais de 50 anos de cultura de programação. Durante seus primórdios, a programação era vista como uma forma de arte. Modos e atitudes antigos dificilmente morrem.

• **Mito:** Uma vez feito um programa e o colocado em uso, nosso trabalho está terminado.

• **Realidade:** Uma vez alguém já disse que “o quanto antes se começar a codificar, mais tempo levará para terminá-lo”. Levantamentos indicam que entre 60 e 80% de todo o esforço será despendido após a entrega do *software* ao cliente pela primeira vez.

• **Mito:** Até que o programa entre em funcionamento, não há maneira de avaliar sua qualidade.

• **Realidade:** Um dos mecanismos de garantia da qualidade de *software* mais eficaz pode ser aplicado desde a concepção de um projeto – a revisão técnica. Revisores de *software* (descritos no Capítulo 15) são um “filtro de qualidade” que mostram ser mais eficientes do que testes para encontrar certas classes de defeitos de *software*.

RESUMO DO TÓPICO 1

Neste tópico, vimos:

- A evolução do *software* e vimos como hoje ele não é mais um mero coadjuvante do *hardware*.
- Como os *softwares* se classificam:
 - **Software Básico:** É um conjunto de programas para dar apoio a outros programas. **Software de Tempo Real:** São programas que monitoram, analisam e controlam eventos do mundo real, devendo responder aos estímulos do mundo externo com restrições de tempo pré-determinadas.
 - **Software Comercial:** O processamento de informações comerciais é a maior área de aplicação de *softwares*, são aplicações que gerenciam as operações comerciais de modo a facilitar o gerenciamento comercial do negócio da empresa, permitindo também a tomada de decisões.
 - **Software Científico e de Engenharia:** São caracterizados por algoritmos de processamento numérico, dependentes da coleta e processamento de dados para as mais variadas áreas do conhecimento.
 - **Software Embutido:** São *software* que são desenvolvidos para executar atividades muito específicas e inseridos em produtos inteligentes tanto para atividades comerciais como para atividades domésticas.
 - **Software de Computador Pessoal:** São *softwares* desenvolvidos para o uso pessoal do computador.
 - **Software de Inteligência Artificial:** Também conhecidos por *Artificial Intelligence- AI*, faz uso de algoritmos não numéricos para resolver problemas complexos que não apresentam facilidades computacionais numéricas ou de análise direta.
- Alguns conceitos da engenharia de *software*.
- Processos de *software*, nesta fase vimos os passos importantes para o processo:
 - **Comunicação:** A importância da comunicação e colaboração dos interessados.
 - **Planejamento:** Criação de um mapa (plano de projeto de *software*) para ajudar a guiar a equipe na sua jornada.
 - **Modelagem:** Cria-se um “esboço” do projeto e se necessário, refina-se o esboço com mais detalhes, numa tentativa de compreender melhor o problema e como resolvê-lo.
 - **Construção:** Essa atividade combina geração de código (manual ou automatizada) e testes necessários para revelar erros na codificação.

- o **Emprego:** O *software* é entregue ao cliente, que avalia o produto entregue e fornece feedback, baseado na avaliação.
- Mitos relativo ao *software*: vimos que muitos mitos são criados e que a realidade é bem diferente, e que estamos sempre em evolução e especialmente derrubando mitos.

AUTOATIVIDADE



- 1 Uma metodologia de processo genérica para engenharia de *software* comprehende cinco atividades, quais são elas?
- 2 O que é um *software* embutido, e onde eles são utilizados?
- 3 Qual era foi caracterizada, pelo aparecimento e o uso generalizado de microprocessadores, computadores pessoais e poderosas estações de trabalho "workstations" de mesa?
- 4 O conceito de crise do *software* pode ser verificado por vários sintomas, quais são?

ENGENHARIA DE REQUISITOS

I INTRODUÇÃO

Nem sempre nós sabemos o que queremos, às vezes, temos dúvidas, de como e o que realmente queremos, ou como faremos para alcançar o desejado, pois estamos sempre em movimento, somos seres em evolução, mudando de opinião, de visual, de estado civil e até mesmo de cidade. Com o desenvolvimento de um *software*, isso também acontece.

Os responsáveis pela especificação das características do novo produto também têm dúvidas se a solução por eles proposta é a mais adequada, ou até mesmo não sabem exatamente como o novo sistema deve se comportar. Por isso, os requisitos de um sistema não permanecem inalterados ao longo do tempo, pois além das dúvidas inerentes aos responsáveis pelo desenvolvimento, o cenário econômico também pode mudar, a realidade e os produtos de uma empresa também mudam. Os motivos são os mais variados, a mudança também pode ser causada por uma alteração de *hardware*.

E para atender com satisfação às necessidades do cliente, a equipe de analistas precisa conhecer bem os objetivos a que se destina o *software*, entender o negócio e conhecer os usuários. Esse conhecimento é de extrema importância para que na coleta de requisitos, a equipe de análise comprehenda os problemas e busque por meio de metodologia própria, solucioná-los.

Mas, o que são requisitos? Compreender os requisitos de um problema está entre as tarefas mais difíceis que um engenheiro de *software* tem pela frente, e, quanto mais pensa nisso, mais difícil parece ser enfrentá-lo, pois o cliente não sabe, geralmente, o que é necessário em um sistema, os usuários finais não têm um bom entendimento das características e funções que o *software* vai oferecer.

Como gerenciar essa fase tão importante do desenvolvimento de um sistema? Nos estudos desta unidade, você vai encontrar apoio para responder a essa e a outras questões relacionadas a esse processo envolvendo os requisitos do sistema.

2 ENGENHARIA DE REQUISITOS

Para Pressman (2006) a atividade de entender os requisitos de um problema está entre as tarefas mais difíceis realizadas por um engenheiro de *software*.

FIGURA 31 - LEVANTAMENTO DE REQUISITOS



FONTE: Disponível em: <<http://www.leanti.com.br/artigos/16/gerenciamento-de-requisitos.aspx>>. Acesso em: 31 mar. 2016.

Podemos dizer, de uma maneira bem simplificada, que os requisitos sugerem o que o futuro sistema deve fazer para que atenda aos usuários, além é claro das restrições e características inerentes ao novo sistema. Podemos dizer também, que o requisito pode ser definido como uma condição ou capacidade de um *software* que deve ser implementada por um sistema ou componente de sistema para que o objetivo seja alcançado no fim, segundo Engholm Jr. (2010). Ainda de acordo com o autor, todo o projeto de *software* tem um conjunto de requisitos, que são determinados pelas necessidades e perspectivas dos usuários que farão uso do *software*, relacionados ao atendimento dos objetivos de negócio da empresa em que trabalham.

O momento em que as necessidades do novo sistema são levantadas ou descobertas por parte do desenvolvedor e em companhia do cliente e sua empresa é a fase de levantamento de requisitos. E para que essa etapa seja realizada com uma proposta de solução adequada, é necessário que tarefas, como o estudo de viabilidade, licitação e análise de requisitos, especificação e gerenciamento de requisitos, sejam realizadas de forma cuidadosa e consistente por todos os envolvidos.

Para ilustrar e entender um pouco mais essa etapa de levantamento dos requisitos para o negócio da empresa, Sommerville (2011) afirma que, os requisitos de um sistema são as descrições do que o sistema deve fazer, os serviços que o sistema oferece bem como as restrições a seu funcionamento. Esses requisitos refletem a necessidade dos clientes para um sistema com finalidade específica, seja ela controlar um dispositivo, colocar ou buscar uma determinada informação. A esse processo de descobrir, analisar, documentar e verificar esses serviços e restrições é chamado de Engenharia de Requisitos (RE, do inglês *Requirements Engineering*). O autor ainda adverte sobre a inconsistência da utilização do termo ‘requisito’ já que por um lado é feita de maneira abstrata e em alto nível dos serviços e restrições que o sistema deve oferecer, no outro extremo é uma definição detalhada e formal de uma função do sistema.

Davis (apud SOMMERVILLE, 2011, p. 57) explica essas diferenças:

Se uma empresa pretende fechar um contrato para um projeto de desenvolvimento de *software* de grande porte, deve definir as necessidades de forma abstrata o suficiente para que a solução para essas necessidades não seja predefinida. Os requisitos precisam ser escritos de modo que vários contratantes possam concorrer pelo contrato e oferecer diferentes maneiras de atender às necessidades da organização do cliente. Uma vez que o contrato tenha sido adjudicado, o contratante deve escrever para o cliente uma definição mais detalhada do sistema, para que esse entenda e possa validar o que o *software* fará. Ambos os documentos podem ser chamados documentos de requisitos para o sistema.

Alguns problemas podem surgir durante o processo de engenharia de requisitos, segundo Sommerville (2011). Falhas em não fazer uma clara separação entre os requisitos dos usuários, que expressam os requisitos abstratos de alto nível, e os requisitos do sistema que expressam detalhadamente o que o sistema deve fazer. Sommerville (2011, p. 58) diz que os requisitos dos usuários e de sistemas podem ser definidos da seguinte maneira:

- Requisitos de usuário são declarações, em uma linguagem natural com diagramas, de quais os serviços que o sistema deverá fornecer a seus usuários e as restrições com as quais deve operar.
- Requisitos de sistema são descrições mais detalhadas das funções, serviços e restrições operacionais do sistema de *software*. O documento de requisitos do sistema (às vezes, chamado especificação funcional) deve definir exatamente o que deve ser implementado. Pode ser parte do contrato entre o comprador do sistema e os desenvolvedores de *software*.

Sommerville (2010, p. 59) apresenta uma separação de quem são os possíveis fornecedores dos requisitos, uma vez que

Os leitores dos requisitos de usuários não costumam se preocupar com a forma como o sistema será implementado, podem ser gerentes que não estão interessados nos recursos detalhados do sistema. Os leitores dos requisitos do sistema, precisam saber mais detalhadamente o que o sistema fará, pois estão mais interessados em como o sistema apoiará os processos dos negócios.

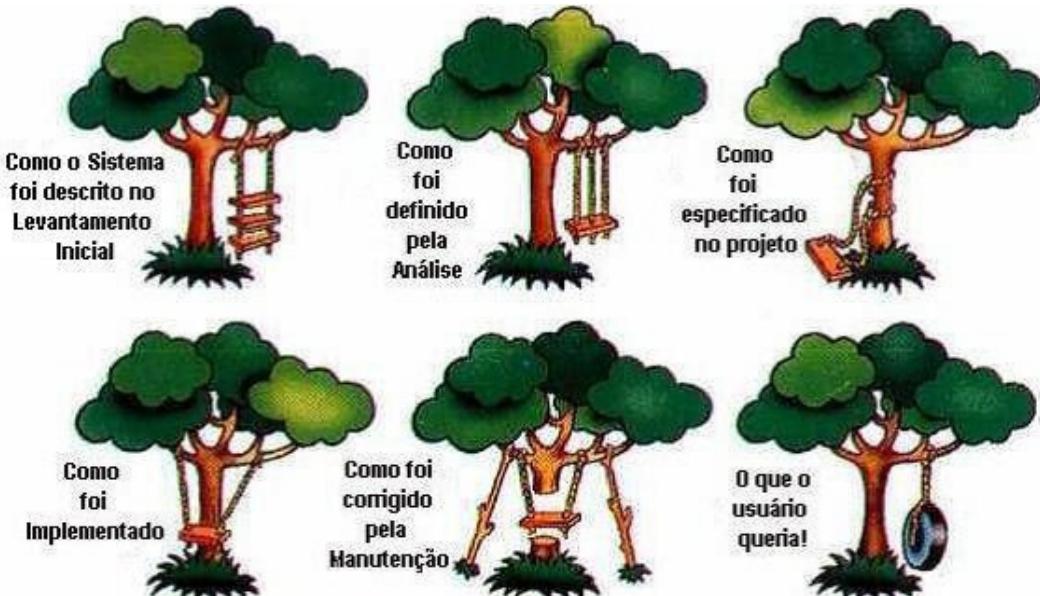
QUADRO - LEITORES PARA OS REQUISITOS DE USUÁRIOS E DE SISTEMAS

Requisitos de usuário	<ul style="list-style-type: none"> • Gerentes clientes. • Usuários finais do sistema. • Engenheiros clientes. • Gerentes contratantes. • Arquitetos de sistema.
Requisitos de sistema	<ul style="list-style-type: none"> • Usuários finais do sistema. • Engenheiros clientes. • Arquitetos de sistema. • Desenvolvedores de <i>software</i>.

FONTE: Adaptado de Sommerville (2010, p. 59)

Uma falha nesse processo de comunicação pode gerar grandes prejuízos e como o resultado pode deixar o cliente insatisfeito, a imagem a seguir demonstra de uma maneira divertida como essa falta de entendimento entre as partes gera confusão.

FIGURA 32 - FALHAS NA COMUNICAÇÃO



FONTE: Disponível em: <<http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035>>. Acessado em: 31 mar. 2016.

E para que esses problemas sejam minimizados é muito importante que os requisitos sejam entendidos pelas partes interessadas. E para ajudar nessa difícil tarefa, os requisitos são divididos em dois grupos: **Requisitos funcionais e Requisitos não funcionais**. De acordo com Sommerville (2011 p. 59):

Requisitos funcionais: São declarações de serviços que o sistema deve fornecer, de como o sistema deve reagir a entradas específicas e de como o sistema deve se comportar em determinadas situações. Em alguns casos, os requisitos funcionais também podem explicitar o que o sistema não deve fazer.

Requisitos não funcionais: São restrições aos serviços ou funções oferecidas pelo sistema. Incluem restrições de timing, restrições no processo de desenvolvimento e restrições impostas pelas normas. Ao contrário das características individuais ou serviços do sistema, os requisitos não funcionais, muitas vezes, aplicam-se ao sistema como um todo.

Vale ressaltar que o próprio autor afirma que essa distinção entre os tipos de requisitos não é tão clara como indicam suas significações, e que os requisitos não são independentes e que muitas vezes provocam ou limitam outros requisitos. Vamos detalhar um pouco mais os dois tipos de requisitos para ajudar no entendimento.

Requisitos Funcionais: São declarações de funções de como o sistema deve reagir a entradas específicas e como deve comportar em determinadas situações. É uma interação entre o sistema e o seu ambiente. Algumas vezes, os requisitos funcionais podem também explicitamente declarar o que o sistema não deve fazer. A especificação deve ser completa e consistente.

Engholm Jr. (2010), afirma que os requisitos funcionais são aqueles que definem as funções ou ações fornecidas pelo sistema e que o usuário pode utilizar, além das regras de negócio e as interfaces internas e externas.

Os requisitos funcionais do sistema variam de requisitos gerais, que abrangem o que o sistema deve fazer, até requisitos bem específicos, que refletem os sistemas e as formas de trabalho de uma organização. Sommerville (2011, p. 60), em relação aos requisitos funcionais afirma que:

Em princípio, a especificação dos requisitos funcionais de um sistema deve ser completa e consistente. Completude significa que todos os serviços requeridos pelo usuário devem ser definidos. Consistência significa que os requisitos não devem ter condições contraditórias. Na prática, para sistemas grandes e complexos, é praticamente impossível alcançar a completude e consistência de requisitos.

Exemplos de requisitos funcionais:

- O sistema deve permitir a inclusão, alteração e remoção de professores com os seguintes atributos: nome, endereço, cidade etc.).
- O sistema fornecerá telas apropriadas para o usuário ler documentos.
- Cada pedido tem um único identificador.

- O sistema deve gerar diariamente para cada unidade de ensino, a lista dos alunos inadimplentes.
- O sistema deve permitir pesquisar a lista dos alunos com nota abaixo da média.

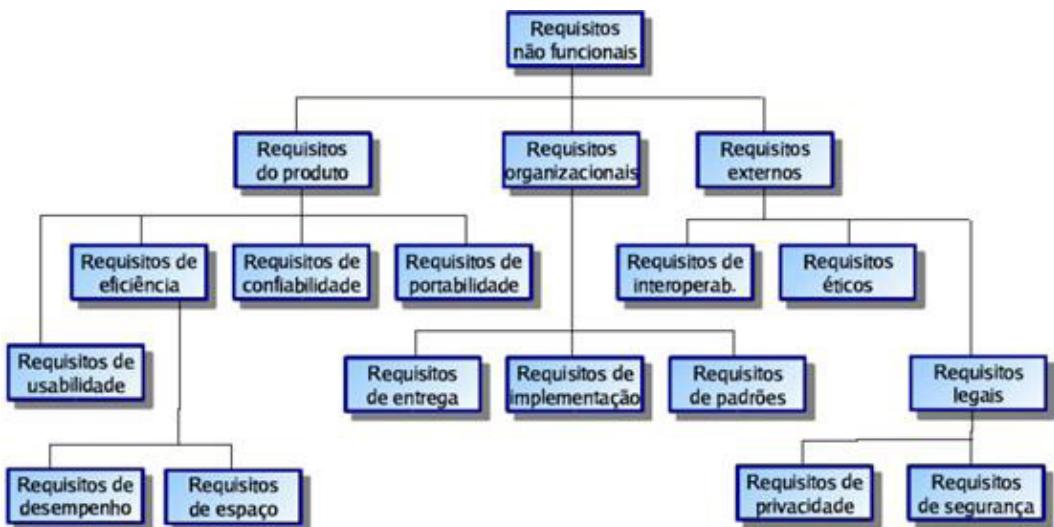
Requisitos Não Funcionais: Eles descrevem os atributos do sistema ou do ambiente do sistema, não se relacionam com a funcionalidade do *software* ou do *designer*, mas definem outras propriedades, que também são muito importantes, como padrões de eficiência, restrições de uso e especificações para alguns requisitos funcionais. Podemos dizer que em linhas gerais, os requisitos não funcionais definem as características qualitativas do sistema, mais especificamente em relação à eficiência dos requisitos funcionais. De acordo com Sommerville (2011), os requisitos não funcionais do sistema, são requisitos que não estão diretamente conexos com os serviços peculiares oferecidos pelo sistema a seus usuários. Esses requisitos podem estar relacionados às propriedades como credibilidade e tempo de resposta. Alguns exemplos de requisitos não funcionais:

- Usabilidade
- Confiabilidade
- Desempenho
- Segurança
- Portabilidade
- Integridade

De acordo com Sommerville (2011), a seguir apresenta uma classificação dos requisitos não funcionais. Podemos ver que os requisitos não funcionais podem ser provenientes das características requeridas para o *software*. De acordo com Sommerville (2011, p. 61):

- Requisitos de produto: Esses requisitos especificam ou restringem o comportamento do software.
- Requisitos organizacionais: Esses são os requisitos gerais de sistemas derivados das políticas e procedimentos da organização do cliente e do desenvolvedor.
- Requisitos externos: Esse tipo abrange todos os requisitos que derivam de fatores externos ao sistema e seu processo de desenvolvimento. Podem incluir requisitos reguladores, que definem o que deve ser feito para que o sistema seja aprovado para uso, tal como um banco central; requisitos legais, que devem ser seguidos para garantir que o sistema opere dentro da lei.

FIGURA 33 - REQUISITOS NÃO FUNCIONAIS



FONTE: Adaptado de Sommerville (2011, p. 61)

Os requisitos não funcionais são de difícil verificação, por isso devem ser escritos quantitativamente, para que possam ser objetivamente testados. Como o exemplo de Sommerville, (2011 p. 63):

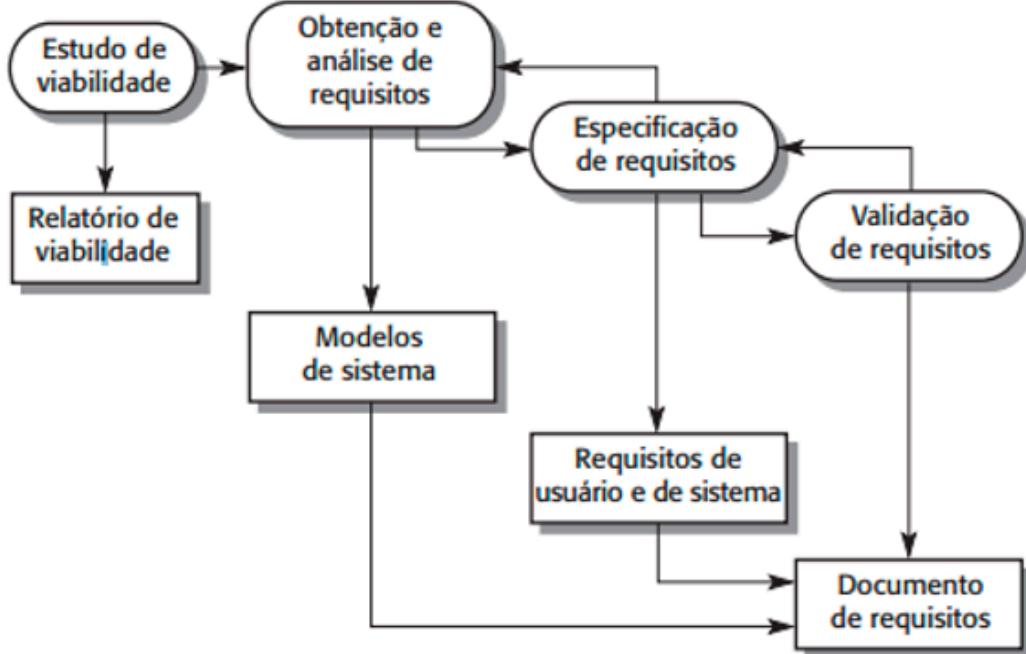
- **Velocidade:** Transações processadas/segundo, tempo de resposta de usuário/evento, tempo de atualização tela.
- **Tamanho:** Megabytes, número de chips de memória ROM.
- **Facilidade de uso:** Tempo de treinamento, número de frames de ajuda.
- **Confiabilidade:** Tempo médio para falha, probabilidade de indisponibilidade, taxa de ocorrência de falhas, disponibilidade.
- **Robustez:** Tempo de reinicio após falha, percentual de eventos que causam falhas, probabilidade de corrupção de dados em caso de falha.
- **Portabilidade:** Percentual de declarações dependentes do sistema-alvo, número de sistemas-alvo.

3 ESTUDO DE VIABILIDADE

De acordo com Sommerville (2011, p. 103): “em todos os sistemas novos, o processo de engenharia de requisitos deve começar com o estudo de viabilidade”. Onde a entrada para esse estudo é uma descrição geral do sistema e de como será sua utilização dentro da organização. Um relatório será o resultados desse estudo, e deve recomendar se vale ou não a pena a realização do processo de engenharia de requisitos e o posterior desenvolvimento do sistema. Vale lembrar, que esse estudo é um estudo breve, e que responder às seguintes perguntas, segundo Sommerville (2011, p. 103):

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

FIGURA 34 – PROCESSO DE ENGENHARIA DE REQUISITOS



FONTE: Adaptado de Sommerville (2003, p. 103)

A primeira pergunta é fundamental, pois se não houver compatibilidade ele não terá valor para a empresa. Mesmo isso sendo óbvio, ainda existem organizações em que isso acontece, uma vez que esses questionamentos foram respondidos, é preciso questionar as fontes das informações, a fim de encontrar as respostas para as próximas perguntas (SOMMERVILLE, 2003, p. 104):

1. Como a organização se comportaria se esse sistema não fosse implementado?
2. Quais são os problemas com os processos atuais e como o novo sistema ajudaria a reduzir esses problemas?
3. Que contribuição direta o sistema trará para os objetivos da empresa?
4. As informações podem ser transferidas para outros sistemas da organização e também podem ser recebidas a partir deles?
5. O sistema requer tecnologia que ainda não foi usada na organização?
6. O que deve ser apoiado pelo sistema e o que não precisa ser apoiado?

Após os estudos iniciais que vão indicar a viabilidade ou não do projeto (SOMMERVILLE, 2003, p. 104) afirma:

[...] o próximo estágio do processo de engenharia de requisitos é a levantamento e análise de requisitos. Nessa atividade, os membros da equipe técnica de desenvolvimento de *software* trabalham com o cliente e os usuários finais do sistema para descobrir mais informações sobre o domínio da aplicação, que serviços o sistema deve fornecer, o desempenho exigido do sistema, as restrições de hardware e assim por diante.

O levantamento e a análise de requisitos podem envolver diferentes tipos de pessoas de uma organização. O termo *stakeholder* é usado para se referir a qualquer pessoa que terá alguma influência direta ou indiretamente sobre os requisitos do sistema. Dentre os *stakeholders* destacam-se os usuários finais que interagirão com o sistema e todo pessoal na organização, que venha a ser por ele afetado.

A elição e a compreensão dos requisitos dos *stakeholders* são difíceis devido a várias razões, de acordo com Sommerville, (2003, p. 105):

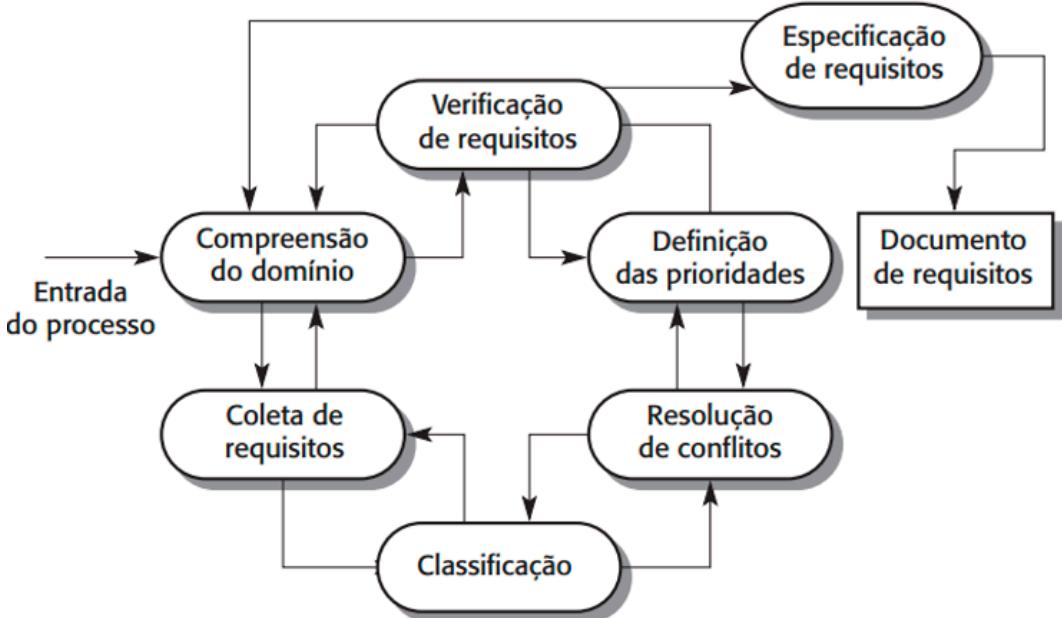
1. Os *stakeholders*, frequentemente, não sabem o que querem do sistema de computador a não ser em termos gerais. Eles podem achar difícil articular o que desejam que o sistema faça ou fazem pedidos não realistas, pois ignoram o custo de seus requisitos.
2. Os *stakeholders* expressam os requisitos naturalmente em seus próprios termos e com o conhecimento implícito de seu trabalho. Os engenheiros de requisitos, sem experiência no domínio do cliente, devem entender esses requisitos.
3. Diferentes *stakeholders* possuem diferentes requisitos, expressos de diferentes formas. Os engenheiros de requisitos precisam considerar todas as fontes potenciais de requisitos e descobrir pontos em comum e conflitos.
4. Fatores políticos podem influenciar os requisitos do sistema. Por exemplo, os gerentes podem solicitar requisitos específicos do sistema que aumentarão a sua influência na organização.
5. O ambiente econômico e de negócios sobre o qual a análise é realizada é dinâmico. Ele muda inevitavelmente durante o processo de análise. Portanto, a importância de determinado requisito pode mudar. Novos requisitos podem surgir de novos *stakeholders* que não haviam sido consultados anteriormente.

Um modelo de processo genérico de levantamento e análise de requisitos é apresentado na figura a seguir. Vale lembrar que esse é um modelo geral, e que cada empresa deverá ter um modelo próprio, variando com fatores locais e características da equipe.



No próximo tópico falaremos um pouco mais a fundo sobre os requisitos.

FIGURA 10 - LEVANTAMENTO DE REQUISITOS



FONTE: Sommerville (2003, p. 106)

4 ELICITAÇÃO DE REQUISITOS

Segundo a IBM (2011), a atividade de elicitação de requisitos é uma das mais importantes práticas da engenharia de software. Através dela, busca-se o entendimento das necessidades do usuário e dos requisitos de negócio, de forma a endereçá-los posteriormente através de uma solução tecnológica. Algumas literaturas adotam o termo elicitação, ao invés de levantamento, pois essa prática não trata simplesmente o levantamento de requisitos, mas também da identificação dos fatos que os compõem e os problemas a serem solucionados. Por ser uma atividade interpessoal, essa prática é muito dependente da capacidade de entendimento do analista e da habilidade do usuário em expressar as suas necessidades.

A descoberta de requisitos ou a elicitação dos requisitos, segundo Sommerville (2011, p. 104), “é o processo de reunir as informações sobre o sistema requerido e os sistemas existentes e separar dessas informações os requisitos de usuários e de sistemas”. Pode listar como fontes de informação durante a fase descoberta de requisitos: documentação, stakeholders do sistema e sistemas e especificações de sistemas similares.

A interação com os stakeholders pode se dar por meio de observação e entrevistas, e para facilitar o entendimento desses profissionais, pode-se usar cenários e protótipos que os ajudam a compreender o funcionamento do futuro sistema.

Sommerville (2003, p. 105) propõe um processo genérico de levantamento e análise que contém as seguintes atividades:

- Compreensão do domínio: Os analistas devem desenvolver sua compreensão do domínio da aplicação.
- Coleta de requisitos: É o processo de interagir com os *stakeholders* do sistema para descobrir seus requisitos. A compreensão do domínio se desenvolve mais 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 apresentarão conflitos. Essa atividade tem por objetivo solucionar esses conflitos.
- Definição das prioridades: Em qualquer conjunto de requisitos, alguns serão mais importantes do que outros. Esse estágio envolve interação com os *stakeholders* para a definição dos requisitos mais importantes.
- Verificação de requisitos: Os requisitos são verificados para descobrir se estão completos e consistentes e se estão em concordância com o que os *stakeholders* desejam do sistema.

Uma das causas do fracasso dos projetos é devido a problemas no levantamento de requisitos, por estarem incompletos, de difícil compreensão ou ambíguos. Para se ter sucesso, é necessário que os requisitos sejam bem levantados, e de maneira que expressem exatamente o que o usuário deseja, da maneira que ele deseja e a forma como poderá ser feito. Para que isso aconteça é importante conhecer técnicas para tornar a construção desses requisitos mais eficaz e evitar o retrabalho.

Considerando-se a complexidade na execução das atividades de elicitação de requisitos e a dependência do relacionamento entre os envolvidos, algumas boas práticas devem ser adotadas pelos analistas de forma a facilitar o processo:

- **Preparação:** Prepare-se previamente e de forma adequada para as atividades planejadas, as quais são geralmente realizadas através de entrevistas, questionários, *brainstorms* e *workshops*.
- **Stakeholders:** Mapeie (com antecedência) quem serão os participantes do processo, quais os seus papéis no projeto e na organização e quais são os seus níveis de conhecimento e influência. É imprescindível que as pessoas corretas sejam envolvidas o quanto antes.
- **Postura:** Busque sempre a efetividade nas comunicações, assim como procure demonstrar ponderação durante as situações de conflito.
- **Entendimento:** Procure focar no entendimento do problema e evitar conclusões precipitadas. Nesse primeiro momento o mais importante é saber escutar.
- **Experiências passadas:** Utilize de forma positiva as experiências vividas anteriormente para ajudar a melhor compreender o problema. Evite considerar que o problema atual é igual a algum outro que tenha sido resolvido em um cliente ou projeto passado.

- **Documentação:** descreva o problema de forma clara e objetiva. Em caso de dúvidas, consulte o cliente e evite inferências. Procure usar exemplos citados pelos *stakeholders*. A adoção de diagramas e figuras sempre ajuda na documentação e entendimento dos requisitos. A criação de protótipos também contribui para o entendimento comum da solução proposta.
- **Validação:** Faça com que os *stakeholders* validem a documentação, verificando o entendimento do problema e as melhorias desejadas e eventualmente façam solicitações de mudanças.

FONTE: Disponível em: <https://www.ibm.com/developerworks/community/blogs/tlcbr/entry/boas_praticas_para_a_elicitacao_de_requisitos?lang=en>. Acesso em: 2 abr. 2016.

5 TÉCNICAS DE LEVANTAMENTO

As técnicas de levantamentos, são ferramentas que tem por objetivo auxiliar a ultrapassar essa fase de grande dificuldade nos projetos. Todas as técnicas apresentadas, possuem características que podem se adequar melhor em uma situação que outra.

Essas técnicas serão detalhadas de maneira resumida aqui, e sugere-se um estudo detalhado para evitar problemas nessa importante fase.

6 ENTREVISTA

As entrevistas sejam elas formais ou informais realizadas com os *stakeholders* são parte da maioria dos processos da engenharia dos requisitos. Nessas entrevistas, os *stakeholders* são questionados pela equipe de engenharia de requisitos sobre os sistemas que usam no momento e também sobre o sistema que será desenvolvido. A partir dessas respostas os requisitos surgem. Sommerville (2011, p. 72) classifica essas entrevistas em dois tipos:

- Entrevistas fechadas: em que o *stakeholder* responde a um conjunto predefinido de perguntas.
- Entrevistas aberta: Neste caso não existe uma agenda predefinida. A equipe de engenharia de requisitos explora uma série de questões com os *stakeholders*, e assim desenvolvem uma melhor concepção das necessidades do novo sistema.

7 ETNOGRAFIA – OBSERVAÇÃO PESSOAL

A etnografia é conhecida como a técnica de observação, onde os analistas passam alguns dias na empresa onde o sistema será utilizado.

Dessa forma, o analista consegue entender a política organizacional da empresa, acompanhar os trabalhos diários, para que assim ele conheça os processos reais que os usuários vão precisar diariamente no sistema, esses processos são todos observados e anotados para a montagem dos principais requisitos do sistema. (SOMMERVILLE, 2011, p. 75).

8 QUESTIONÁRIO

A utilização dos questionários pode ser de grande valia para o levantamento de requisitos, quando a empresa possuir mais de uma unidade, e que a equipe de analistas não consiga visitar a todas as filiais. Lembrando que esses questionários devem ser formulados para as pessoas que serão os usuários finais do sistema a ser desenvolvido. Deve possuir questões objetivas, claras, simples e de fácil compreensão por todos os envolvidos. Gerando, assim, motivação e interesse para responder de forma correta.

A formulação dos questionários que serão utilizados para esse levantamento das informações pode ter questões de múltipla escolha, lista de verificação e questões com espaços em branco. Apenas como sugestão, utilizando alguma dessas formas minimiza-se o tempo gasto em sua resposta.

9 BRAINSTORMING

A técnica para geração de ideias, chamada *Brainstorming*, consiste em uma ou várias reuniões que permitem que as pessoas recomendem e explorem ideias. Para a execução das informações através dessa técnica, são selecionados grupos de setores diferentes e com propostas de ideias que numa visão inicial pode parecer quase impossível de serem executadas, mas no decorrer das reuniões elas são compreendidas e propostas para o sistema. Para a utilização dessa técnica, são necessárias diversas reuniões com os usuários da empresa.

10 JAD (JOINT APPLICATION DESIGN)

Segundo Moraes (2016):

JAD é uma técnica para promover cooperação, entendimento e trabalho em grupo entre os usuários desenvolvedores. A JAD facilita a criação de uma visão compartilhada do que o produto de *software* deve ser. Através da sua utilização os desenvolvedores ajudam os usuários a formular problemas e explorar soluções. Dessa forma, os usuários ganham um sentimento de envolvimento, posse e responsabilidade com o sucesso do produto. A técnica JAD tem quatro princípios básicos:

- Dinâmica de grupo: são realizadas reuniões com um líder experiente, analista, usuários e gerentes, para despertar a força e criatividade dos participantes. O resultado final será a determinação dos objetivos e requisitos do sistema;
- Uso de técnicas visuais: para aumentar a comunicação e o entendimento;
- Manutenção do processo organizado e racional: o JAD emprega a análise top down e atividades bem definidas. Possibilita assim, a garantia de uma análise completa reduzindo as chances de falhas ou lacunas no projeto e cada nível de detalhe recebe a devida atenção;
- Utilização de documentação padrão: preenchida e assinada por todos os participantes. Este documento garante a qualidade esperada do projeto e promove a confiança dos participantes.

11 CRITÉRIOS PARA VALIDAÇÃO E ACEITAÇÃO DE REQUISITOS

A validação de requisitos objetiva mostrar que os requisitos levantados definem o sistema desejado pelos usuários. A importância dessa tarefa está no fato de que erros em um documento de requisitos podem acarretar custos excessivos de trabalho quando são descobertos durante o processo de desenvolvimento, ou mesmo depois, durante a operação do sistema. Sommerville (2011) diz que durante o processo de validação dos requisitos, diferentes tipos de verificações devem ser realizados com os documentos de requisitos. Essas verificações incluem:

QUADRO 5 - TIPOS DE VERIFICAÇÕES

Verificação de validade	Um usuário pode achar que precisa de um software para determinada função. No entanto, uma análise mais detalhada pode identificar outras funções além das solicitadas. Os sistemas têm diversos stakeholders com diferentes necessidades, e qualquer conjunto de requisitos é inevitavelmente um compromisso da comunidade de stakeholders.
-------------------------	---

Verificação de consistência	Os requisitos em um documento devem ser consistentes e não conflitantes. Ou seja, não deve haver restrições contraditórias ou descrições diferentes da mesma função do sistema.
Verificação de completude	O documento de requisitos deve incluir requisitos que definam todas as funções e as restrições pretendidas pelo usuário do sistema.
Verificação de realismo	Ou seja, o que se deseja é possível. Usando o conhecimento das tecnologias existentes, os requisitos devem ser verificados para assegurar que realmente possam ser implementados. Verificando o orçamento e o cronograma de desenvolvimento.
Verificabilidade	Com a intenção de reduzir o potencial de conflito entre o cliente e o contratante, os requisitos do sistema devem ser passíveis de verificação. O que significa que você deve ser capaz de escrever um conjunto de testes que demonstrem que o sistema entregue atende a cada requisito especificado.

FONTE: Adaptado de Sommerville (2011, p. 77)

Sommerville (2011) lista uma série de técnica para a validação dos requisitos, para serem usadas individualmente ou em conjunto.

QUADRO 6 - VALIDAÇÃO DE REQUISITOS

Revisão de requisitos	Os requisitos são sistematicamente analisados por uma equipe de revisores que verifica erros e inconsistências.
Prototipação	Nessa abordagem para validação, um modelo executável do sistema em questão é demonstrado para os usuários finais e clientes. Esses podem experimentar o modelo para verificar se ele atende as suas reais necessidades.
Geração de casos de testes	Os requisitos devem ser testáveis. Se os testes forem concebidos como parte do processo de validação, isso frequentemente revela problemas de requisitos. Se é difícil ou impossível projetar um teste, isso normalmente significa que os requisitos serão difíceis de serem implementados e devem ser reconsiderados.

FONTE: Adaptado de Sommerville (2011, p. 77)

Engholm Jr. (2010, p. 160), afirma que para um requisito possa ser validado e aceito devemos antes verificar se ele é:

- **Claro:** Estar escrito de maneira fácil entendimento, possibilitando a compreensão por todos os integrantes da equipe de projeto.
- **Completo:** Contendo todos os detalhes importantes, como dados de entrada e saída, regras de validação ou interação do usuário com o sistema.
- **Consistente:** Requisitos não podem ser contraditórios.
- **Único:** Requisitos devem ser únicos, sem duplicidade.
- **Viável:** Todos os requisitos devem ser viáveis para serem atendidos pela solução.
- **Testável:** Os requisitos devem poder ser efetivamente testados quando implementados.
- **Rastreável:** Devem poder ser rastreáveis a partir do problema de negócio.

A fase de levantamento de requisitos é imprescindível para o sucesso do projeto e devemos estar atentos e não subestimar os problemas envolvidos na validação dos requisitos. Sommerville (2011, p. 117), afirma que “A validação de requisitos é importante porque a ocorrência de erros em um documento de requisitos pode levar a grandes custos relacionados ao retrabalho, quando esses erros são descobertos durante o desenvolvimento ou depois que o sistema estiver em operação”.

Por isso, devemos atentar e não subestimar os problemas envolvidos na validação dos requisitos. Devido à importância, Sommerville (2011, p. 117) ressalta que:

As revisões de requisitos podem ser informais ou formais. As revisões informais simplesmente envolvem os fornecedores que discutem os requisitos com tantos stakeholders quantos forem possíveis. É surpreendente como muitas vezes a comunicação entre desenvolvedores e stakeholders termina depois da obtenção de requisitos, e não existe nenhuma confirmação de que os requisitos documentados são os que os stakeholders realmente solicitaram. Muitos problemas podem ser detectados simplesmente conversando sobre o sistema com os stakeholders, antes de iniciar uma revisão formal.

Em uma revisão formal de requisitos, a equipe de desenvolvimento deve ‘conduzir’ o cliente pelos requisitos de sistema, explicando as implicações de cada um. A equipe de revisão deve verificar cada requisito, em termos de sua consistência, e checar os requisitos como um todo, sob o ponto de vista de sua completeza. Sommerville (2011, p.117).

12 GERENCIAMENTO DE REQUISITOS

O processo de Gerenciamento de Requisitos corresponde ao conjunto de atividades que auxilia a equipe do projeto a identificar, controlar e rastrear os requisitos, bem como as alterações nos requisitos em muitos momentos do projeto. Em outras palavras, é o processo que gerencia mudanças nos requisitos

de um sistema. Estas mudanças ocorrem conforme os clientes desenvolvem um melhor entendimento de suas reais necessidades.

Pode-se definir que o gerenciamento de requisitos se trata de um modelo sistemático para:

- Identificar, organizar e documentar os requisitos do sistema; e
- Estabelecer e manter acordo entre o cliente e a equipe do projeto nos requisitos variáveis do sistema.

Não importa o quanto cuidadoso você seja sobre a definição dos seus requisitos, sempre haverá mudanças. O que torna complexo o gerenciamento dos requisitos variáveis é que a mudança em um requisito poderá gerar grandes impactos em outros requisitos já implementados ou não no sistema.

Quando não há um controle de alterações bem definido, mudanças de baixa prioridade podem ser implementadas antes daquelas de alta prioridade, e, em muitos casos, não se sabe os reais impactos que essas mudanças podem gerar em relação ao custo e tempo do projeto. Em muitos casos as mudanças são inevitáveis. E quase sempre alguns requisitos acabam mudando enquanto o sistema ainda está sendo desenvolvido. As razões para estas constantes mudanças podem ser originadas de vários fatores tais como:

- Nem sempre os requisitos são óbvios e podem vir de várias fontes.
- Nem sempre é fácil expressar os requisitos claramente em palavras.
- Existem diversos tipos de requisitos em diferentes níveis de detalhe.
- O número de requisitos poderá impossibilitar a gerência se não for controlado.
- Os requisitos estão relacionados uns com os outros, e também com o produto liberado do processo de engenharia do *software*.
- Os requisitos têm propriedades exclusivas ou valores de propriedade. Por exemplo, eles não são igualmente importantes nem igualmente fáceis de cumprir.
- Há várias partes interessadas, o que significa que os requisitos precisam ser gerenciados por grupos de pessoas de diferentes funções.
- Os requisitos são alterados.

Para agravar a situação, os sistemas de modo geral também devem levar em conta que o mundo está em constante mudança, de modo que algumas das hipóteses levantadas nas fases iniciais podem se tornar equivocadas (sem que nos demos conta disso).

Os requisitos não podem ser gerenciados de forma efetiva sem rastreabilidade. Um requisito é rastreável se for possível identificar quem solicitou o requisito, porque o requisito existe, quais os requisitos relacionados e como os requisitos se relacionam às outras informações como *design* de sistemas, implementações e documentos do usuário. Essas informações são utilizadas para identificar todos os requisitos afetados por mudanças propostas.

Boas práticas de gerenciamento de requisitos, como uma manutenção de dependências entre requisitos, têm benefícios em longo prazo, como maior satisfação do cliente e custos de desenvolvimento mais baixos. Uma vez que os retornos não são imediatos, o gerenciamento de requisitos pode parecer uma despesa desnecessária. Entretanto, sem a gerência, a economia de curto prazo será devastada pelos custos em longo prazo.

Todo sistema deve ser desenvolvido de modo que as alterações sofridas ao longo do seu desenvolvimento sejam o menos impactante possível. O processo de mudança dos requisitos precisa ser controlado de modo a garantir a qualidade do sistema. O impacto destas mudanças precisa ser avaliado e compreendido de modo que a sua implementação seja feita de maneira eficiente e a baixo custo.

É de fundamental importância que as alterações dos requisitos sejam:

- Identificadas e avaliadas.
- Avaliadas sob o ponto de vista de risco.
- Documentadas.
- Planejadas.
- Comunicadas aos grupos e indivíduos envolvidos e
- Acompanhadas até a finalização.

Para ter-se uma gerência de requisitos eficaz é necessário, de antemão, possuir um conjunto de políticas. É necessário definir um conjunto de objetivos para o processo de gerência. Esses objetivos devem ser claros e transmitidos para todos os integrantes da equipe. Todos os artefatos (documentos) produzidos durante o desenvolvimento do *software* devem tornar a gerência dos requisitos visível e transparente. Esses documentos devem ser gerados levando-se em conta padrões externos e corporativos, de modo a assegurar consistência e uniformidade das informações. Políticas bem definidas para a gerência de configuração, controle de mudanças, rastreabilidade e garantia da qualidade precisam ser colocadas em prática de modo a viabilizar um processo dinâmico e eficaz de gerência de requisitos.

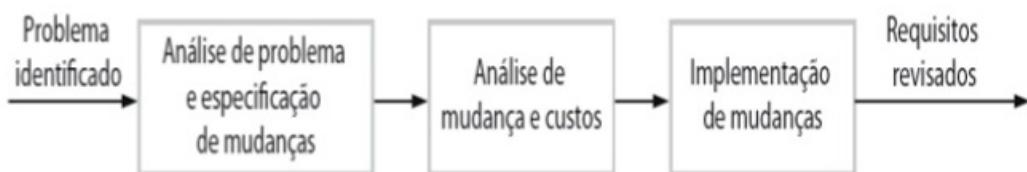
FONTE: Disponível em: <<http://www.leanti.com.br/artigos/16/gerenciamento-de-requisitos.aspx>>. Acesso em: 31 mar. 2016.

13 PLANEJAMENTO DO GERENCIAMENTO DE REQUISITOS

O gerenciamento de uma atividade, sempre exige planejamento. Com o gerenciamento dos requisitos, não é diferente. Por ser uma atividade que dispõe muito tempo, o gerente deve decidir sobre alguns pontos, segundo Sommerville (2011, p. 78):

- 1. Identificação de requisitos:** Cada requisito deve ser identificado de modo único, para que possa ser feita a referência cruzada entre este e outros requisitos para que ele possa ser usado nas avaliações de rastreabilidade.
- 2. Processo de gerenciamento de mudanças:** É o conjunto de atividades que avaliam o impacto e custo das mudanças. Conforme apresentado na figura a seguir.
- 3. Políticas de rastreabilidade:** Essas políticas definem as relações entre os requisitos e entre os requisitos e o projeto de sistema que devem ser registrados e também como esses registros devem ser mantidos.
- 4. Suporte de ferramentas CASE:** O gerenciamento de requisitos envolve o processamento de grandes quantidades de informações sobre os requisitos. As ferramentas que podem ser usadas variam desde sistemas especializados de gerenciamento de requisitos a planilhas e sistemas simples de banco de dados.

FIGURA 36 - GERENCIAMENTO DE REQUISITOS



FONTE: Sommerville (2011, p. 79)

Os requisitos se relacionam com outros requisitos e entre outros projetos do sistema, por isso quando é solicitada uma modificação, deve-se verificar o impacto dessas alterações nos requisitos do projeto em andamento e também sobre outros requisitos de outros projetos do sistema.

14 GERENCIAMENTO DE MUDANÇAS DE REQUISITOS

O gerenciamento de mudanças de requisitos, para Sommerville (2011, p. 79) deve ser aplicado “a todas as mudanças propostas para os requisitos. A vantagem de utilizar um processo formal para o gerenciamento de mudanças é que todas as propostas de mudança são tratadas de modo consistente e que as mudanças no documento de requisitos são feitas de maneira controlada”. Há três estágios principais em um processo de gerenciamento de mudanças, segundo Sommerville (2011, p. 79).

QUADRO 7 - ESTÁGIO DO GERENCIAMENTO DE MUDANÇAS

Estágios	Descrição
Análise do problema e especificação da mudança	O processo começa com a identificação de um problema com os requisitos ou, algumas vezes, com uma proposta específica de mudança. Nesse estágio, é realizada a análise do problema ou da proposta de mudança, a fim de verificar sua validade. Uma proposta mais específica de mudança nos requisitos pode então ser efetuada.
Análise e custo da mudança	O efeito da mudança proposta é avaliado, utilizando-se informações sobre a facilidade de rastreamento e o conhecimento geral dos requisitos do sistema. O custo da mudança é estimado em termos das modificações no documento de requisitos e, se apropriado, no projeto de sistemas e na implementação. Uma vez concluída essa análise, é tomada uma decisão sobre prosseguir com a alteração de requisitos ou não.
Implementação de mudanças	O documento de requisitos e, quando for necessário, o projeto de sistema e a implementação são modificados. O documento de requisitos deve ser organizado de maneira que as mudanças possam ser acomodadas sem muito esforço. Assim como ocorre com os programas, a facilidade de modificações em documentos é alcançada ao se minimizar referências externas e ao tornar as seções do documento tão modulares quanto possível.

FONTE: Adaptado de Sommerville (2011, p. 79)

Sommerville (2011, p. 122) adverte que:

se uma mudança nos requisitos de um sistema for requerida urgentemente, sempre existe a tentação de fazer essa mudança no sistema e, depois, retrospectivamente modificar o documento de requisitos. Isso quase inevitavelmente faz com que a especificação de requisitos e a implementação do sistema se desajustem. Uma vez feitas as mudanças no sistema, as modificações no documento de requisitos podem ser esquecidas ou feitas de uma maneira não consistente com as mudanças no sistema.

RESUMO DO TÓPICO 2

Neste tópico apresentamos:

- Durante o desenvolvimento do sistema é necessário estar sempre atento à qualidade e às necessidades do cliente para que essas sejam atendidas. Para isso é importante saber exatamente o que se espera do sistema.
- As técnicas para o levantamento de requisitos:
 - **Técnicas tradicionais:** São aplicadas em várias áreas do conhecimento. Exemplo: questionários, entrevistas, observação, e análise de documentos.
 - **Técnicas de elicitação de grupo:** Têm por objetivo compreender melhor o pensamento e comportamento dos grupos e as necessidades dos usuários. Exemplo: *brainstorming* e as sessões JAD (*Joint Application Design*).
- **Engenharia de requisitos:** Processo de descobrir, analisar, documentar e verificar esses serviços e restrições é chamado de Engenharia de Requisitos (RE, do inglês *Requirements Engineering*).
- **Definindo os requisitos:** Deve descrever os serviços fornecidos ao usuário. Os requisitos funcionais e não funcionais que o sistema deve ter. Essa descrição pode usar a Linguagem Natural, diagramas ou outras notações comprehensíveis para os clientes. Normas de produtos e processos que devem ser seguidos devem ser especificados.
- **Requisitos funcionais:** São declarações de funções de como o sistema deve reagir a entradas específicas e como deve se comportar em determinadas situações. É uma interação entre o sistema e o seu ambiente. Algumas vezes, os requisitos funcionais podem também explicitamente declarar o que o sistema não deve fazer. A especificação deve ser completa e consistente.
- **Requisitos não funcionais:** São restrições aos serviços ou funções oferecidas pelo sistema. Incluem restrições de *timing*, restrições no processo de desenvolvimento e restrições impostas pelas normas. Ao contrário das características individuais ou serviços do sistema, os requisitos não funcionais, muitas vezes, aplicam-se ao sistema como um todo.
- **Critérios para validação e aceitação de requisitos:** A validação de requisitos tem como objetivo mostrar que os requisitos levantados definem o sistema desejado pelos usuários.

- **Problemas encontrados referentes a requisitos:** É a falta de especificação da real necessidade e expectativas dos usuários. É o maior motivo dessas falhas, seguido por requisitos incompletos, com baixa qualidade e falta de controle de mudanças.
- **Gerenciamento de requisitos:** A tarefa de gerenciar requisitos se preocupa com as mudanças nos requisitos que já haviam sido acertadas entre cliente e desenvolvedor.

AUTOATIVIDADE



- 1 Durante a fase de levantamento de requisitos, podemos utilizar algumas formas para realizar essa tarefa. Comente como pode ser feito o levantamento de requisitos, por meio de entrevistas e quais os tipos de entrevistas que existem.
- 2 Já vimos que os requisitos são de suma importância para o desenvolvimento e sucesso de um projeto, e para um requisito possa ser validado e aceito devemos antes verificar se ele é o quê?
- 3 Quais são algumas das razões para que ocorram problemas nos requisitos de um projeto?

GERENCIAMENTO DE PROJETOS E SOFTWARE

1 INTRODUÇÃO

Antes de iniciarmos nossos estudos, apresentaremos um panorama sobre esse assunto, segundo Pressman (2011).

FIGURA 37 - PANORAMA SEGUNDO PRESSMAN

PANORAMA

O que é? Embora muitos de nós (em momentos mais críticos) adotemos a visão de Dilbert, ainda resta uma atividade bastante útil quando sistemas e projetos computacionais são desenvolvidos. Gerenciamento de projeto envolve planejamento, monitoração e controle de pessoas, processos e eventos que ocorrem à medida que o software evolui desde os conceitos preliminares até sua disponibilização, operacional e completa.

Quem realiza? De certa forma, todas as pessoas gerenciam, mas o escopo das atividades de gerenciamento varia entre os envolvidos de um projeto de software para outro. Um engenheiro de software gerencia suas atividades diárias, planejando, monitorando e controlando as tarefas técnicas. Os gerenciadores de projetos planejam, monitoram e controlam o trabalho de uma equipe de engenheiros de software. Já um gerente sênior coordena a interface entre o "lado comercial" e os profissionais de software.

Por que é importante? Desenvolvimento de software computacional é uma tarefa complexa, principalmente se envolver muitas pessoas trabalhando por um tempo relativamente longo. Por isso os projetos de software precisam ser gerenciados.

Quais são as etapas envolvidas? Entenda os 4 Ps: Pessoas, Produto, Processo e Projeto. As pessoas devem ser organizadas para o trabalho de desenvolvimento de forma efetiva. A comunicação com o cliente e com outros interessados deve ocorrer para que o escopo e os requisitos do produto sejam compreendidos. Deve ser selecionado um projeto adequado para as pessoas e para o produto. O projeto deve ser planejado com base na estimativa do esforço e do prazo para a realização das tarefas: definindo artefatos, estabelecendo pontos de verificação (checagem) de qualidade e identificando mecanismos para monitorar e controlar o trabalho no plano de projeto.

Qual é o artefato? Assim que as atividades de gerenciamento iniciam, faz-se um plano de projeto. Define-se o processo e as tarefas a ser conduzidas, as pessoas que realizarão o trabalho e os mecanismos de avaliação de riscos, do controle das alterações e de avaliação de qualidade.

Como garantir que o trabalho foi realizado corretamente? Nunca se está completamente seguro de que o plano de projeto está correto até que se entregue um produto de alta qualidade, no prazo e dentro do orçamento. Entretanto, um gerente de projeto age corretamente quando encoraja o pessoal de desenvolvimento a trabalhar em conjunto como uma verdadeira equipe, concentrando-se nas necessidades do cliente e na qualidade do produto.

FONTE: Pressman (2011, p. 566)

2 GERENCIAMENTO DE PROJETOS

Desde o início da civilização, a humanidade planeja e gerencia projetos. Através da construção de castelos, estradas ou de megaestruturas como as pirâmides do Egito ou o Coliseu em Roma. E tudo isso sem as ferramentas, técnicas e metodologias que possuímos atualmente. O que se percebeu é que independentemente do tipo de projeto que está sendo realizado, todos possuem itens em comum que precisam ser controlados, como os custos, criação de prazos, aquisição de recursos e gerenciamento de riscos. Embora o gerenciamento de projetos como prática já exista há séculos, só foi reconhecido formalmente como profissão após a 2^a. Guerra Mundial. O gerenciamento de projetos de *software* é uma parte essencial da engenharia de *software*. Um bom gerenciamento não pode garantir o sucesso de um projeto, porém, um mau gerenciamento geralmente resulta em falha do projeto e como consequência o *software* é entregue com atraso, custa mais do que foi originalmente estimado e falha ao atender seus requisitos.

O gerenciamento efetivo de desenvolvimento de *software* tem um foco nos 4 Os: Pessoas, Produto, Processo e Projeto. Essa ordem não é arbitrária. Caso o gerente se esqueça de que o trabalho do engenheiro de *software* consiste em esforço humano nunca terá sucesso no gerenciamento de projeto. E se ele não estimular o encorajamento amplo para a comunicação entre os envolvidos, bem cedo, no início da elaboração de um produto, corre o risco de desenvolver uma solução elegante para o problema errado. Outro problema é um gerente que preste pouca atenção ao processo, arrisca-se a inserir métodos e ferramentas técnicas competentes em um vácuo. E aquele que embarcar sem um plano de projeto sólido compromete o sucesso do projeto. (PRESSMAN, 2011).

A necessidade de gerenciamento de projetos de *software* está no fato de que a engenharia de *software* profissional está sempre sujeita às restrições de orçamento e de cronograma da organização. Com isso o trabalho do gerente de projeto de *software* é assegurar que esse atenda a essas restrições e entregue um *software* que contribua para as metas da empresa que está desenvolvendo o *software*.

O tipo de trabalho dos gerentes de *software* e dos outros gerentes de projetos de outras áreas da engenharia é o mesmo, porém o que difere o engenheiro de *software* é que ele possui algumas distinções que tornam seu gerenciamento mais difícil. O quadro a seguir traz algumas dessas diferenças.

QUADRO 8 - FUNÇÕES DO GERENTES DE PROJETOS

Diferença	Descrição
O produto é intangível	O gerente de um projeto de construção de edifício pode ver o produto que está sendo construído e, se o cronograma atrasar, o efeito sobre o produto é visível. Porém o <i>software</i> é intangível, não pode ser visto ou tocado, os gerentes de projetos de <i>software</i> têm dificuldades de verificar seu progresso. Eles contam com outras pessoas para produzir a documentação necessária para examinar o progresso.
Não existem processos-padrão de <i>software</i>	O processo de engenharia de alguns tipos de sistema, como pontes e prédios, é bem compreendido. Entretanto, os processos de <i>software</i> variam de uma empresa para a outra. Mesmo que nossa compreensão sobre esses processos tenha crescido, ainda não podemos prever, confiavelmente, quando determinado processo de <i>software</i> provavelmente apresentará problemas de desenvolvimento.
Projetos de <i>software</i> de grande porte são, frequentemente, projetos “únicos”.	Projetos de <i>software</i> de grande porte são diferentes de projetos anteriores. Assim sendo, mesmo com a experiência que os gerentes possuem é difícil prever problemas. Além disso, as mudanças tecnológicas podem tornar a experiência dos gerentes obsoleta, e lições anteriores aprendidas não podem ser reutilizadas em novos projetos.

FONTE: Adaptado de Sommerville (2007, p. 62)

Devemos estar atentos que para gerenciar um projeto de *software* com sucesso, deve-se compreender o que pode sair errado, de modo que ações planejadas evitem tais problemas. A seguir apresentamos dez sinais indicadores de que um projeto de sistemas de informações está em perigo, segundo Pressman (2011, p. 577):

1. O pessoal de *software* não comprehende as necessidades de seus clientes.
2. O escopo do produto está parcialmente definido.
3. As alterações são mal gerenciadas/administradas.
4. A tecnologia escolhida muda.
5. As necessidades de negócio mudam (ou são mal definidas).
6. Os prazos estão fora da realidade.
7. Os usuários mostram-se resistentes.
8. O patrocínio é perdido (ou nunca foi propriamente obtido).
9. Faltam profissionais à equipe ou esta não possui pessoal com habilidades adequadas.
10. Gerentes e desenvolvedores evitam práticas e lições aprimoradas e aprendidas.

A gestão de projetos é um conjunto de práticas que serve de guia a um grupo para trabalhar de maneira produtiva. Ela comprehende métodos e ferramentas que organizam as tarefas, identificam sua sequência de execução e dependências existentes, apoia a alocação de recursos e tempo, além de permitir o rastreamento da execução das atividades e medição do progresso relativo ao que foi definido no plano de projeto.

Qualquer que seja o projeto com o qual você esteja envolvido, o plano de projeto é um documento essencial o qual o gerente de projeto “vive e respira” ao longo de todo o projeto. É obrigatório o seu desenvolvimento e manutenção. O plano de projeto define os marcos de projeto (isto é, os ‘milestones’) e as principais atividades necessárias à execução do projeto. Esse documento define a data de cada marco de projeto, as atividades associadas, os artefatos gerados e respectivos responsáveis.

Perceba que é, até certo ponto, natural àqueles que desenvolvem novos produtos e sistemas de *software* iniciarem as atividades de desenvolvimento antes mesmo que eles entendam o que tem de ser feito, ou seja, antes mesmo de saberem qual é o problema a ser tratado. Esse tipo de atitude, comumente, resulta no insucesso de projetos. Estudos apontam que cerca de 40% de insucesso de projetos são devidos ao mau planejamento ou, até mesmo, a inexistência de plano de projeto, enquanto que quase 20% se deve a uma gestão inadequada do projeto. Interessante destacar que aproximadamente 50% dos problemas e insucesso de projeto se devem a uma ‘pobre’ comunicação entre os principais envolvidos no projeto (ou seja, os *stakeholders*). Num projeto, as questões essenciais que um gerente deve se fazer são:

- Que problema precisamos solucionar?
- Quais recursos necessito para resolver?
- Quanto tempo disponho para o projeto e implementação da solução?

A lição que fica é: sem o entendimento completo do problema a ser tratado e um bem elaborado plano de projeto em mãos, você (gerente) e sua equipe não saberão onde querem e precisam chegar. A consequência é você (juntamente com sua equipe) deparar-se com a inserção de defeitos logo cedo no desenvolvimento, os quais virão, apenas bem mais tarde, a serem descobertos, resultando em atraso no projeto e/ou comprometendo a qualidade do produto final.

Essas situações apontadas ocorrem, com frequência, quando não há qualquer ‘preocupação’ com a gestão de projeto do *software*. Essas, dentre outras, são razões pelas quais muitos projetos se transformam em casos de insucesso.

3 POR QUE A GESTÃO DE PROJETOS É ESSENCIAL?

A gestão de projetos é uma atividade ortogonal às demais atividades de projeto e atua como guia para a boa execução do projeto. Todas as pessoas envolvidas com um projeto têm a necessidade de acesso às suas informações. Quando lidamos com projeto de médio a grande porte e de natureza complexa, uma atividade chave é a coordenação. Um gerente de projeto precisa coordenar:

- Múltiplas pessoas de formação diversa.
- Múltiplas tarefas onde ocorre relação de dependência.

- Uso de múltiplos recursos (como equipamentos, ferramentas, laboratórios).
- Decisão e aprovação em múltiplos pontos num projeto.
- Alocação adequada de recursos humanos e financeiros a tarefas.

Portanto, é preciso dar visibilidade e compartilhar informações de projeto, pois as decisões precisam ser tomadas com base de informações bem entendidas e explicitadas. A qualidade resultante de um produto ou sistema é determinada a partir do início de seu desenvolvimento. Uma criteriosa análise, feita logo cedo no projeto, visa encontrar erros, identificar inconsistências e averiguar quão correto e completo é o entendimento do problema e adequada é a solução trabalhada. Isso torna a gestão de projeto uma atividade essencial à execução de projetos e sucesso de produtos. A gestão de projetos de *software* pode ser vista sob duas perspectivas: técnica e pessoal onde a ênfase se dá sobre atividades de planejamento e execução, conforme ilustrado na figura a seguir.

FIGURA 38 - GESTÃO DE PROJETOS



FONTE: Perspectivas da gestão de projetos

Dominar as habilidades necessárias a uma boa gestão de projetos requer tempo, experiência e reciclagem. Embora algumas pessoas sejam levadas a acreditar que a capacidade para fazer a gestão de projetos seja um mito, isso não passa de uma falácia. Não se trata de característica inata que o indivíduo traz consigo, mas de um conjunto de habilidades que podem ser reconhecidas, classificadas e desenvolvidas pelas pessoas.

O reconhecimento desses fatos tem motivado os profissionais a buscarem atualizar-se, bem como às empresas a considerarem a gestão de projetos no plano estratégico da instituição. Hoje em dia, profissionais e empresas buscam a capacitação em gestão de projetos que seguem as orientações da principal referência mundial no assunto: o PMI (Project Management Institute) (www.pmi.org) o qual tem tido um crescimento quase exponencial no número de afiliações, que hoje conta com mais de 240 mil profissionais membros.

4 GESTÃO DE PROJETOS E PMI

O PMBOK comprehende um guia contendo todo corpo de conhecimento de práticas tradicionais, avançadas e inovadoras em gestão de projetos, ou seja, o PMBOK serve como guia que contém um conjunto de diretrizes para gestão de projetos e uma estrutura de como ela é decomposta em áreas de conhecimento. Segundo as diretrizes do PMBOK, a gestão de projetos comprehende um conjunto de processos que contém áreas que constituem o corpo do conhecimento da gestão de projetos.



Para maiores informações VEJA: <<http://escritoriodeprojetos.com.br/processos-do-guia-pmbok.aspx>>.

Cinco fases compõem a gestão de projetos: **Inicialização, Planejamento, Execução, Controle e Encerramento.**

Inicialização: Como o próprio nome já diz trata de iniciar um novo projeto onde um gerente de projeto é designado, são identificados os principais envolvidos e interessados (isto é, os *stakeholders*), além de finalizar o documento de caso de negócio que trata da solução a ser implementada.

Planejamento: O gerente de projeto define o escopo do projeto, a equipe envolvida, a WBS (*Work Breakdown Structure*), quando se define o escopo e organização de todo o projeto numa estrutura de árvore), um conjunto das atividades mais prioritárias e um cronograma de projeto.

Execução: É dedicada à implementação da solução definida para o projeto, no uso adequado dos recursos e liderança junto à equipe.

Controle: É uma importante fase já que o gerente monitora o andamento do projeto, os resultados alcançados, os artefatos gerados, todo e qualquer desvio que possa ocorrer em relação ao que foi definido no plano de projeto e, essas informações, são usadas para produzir relatório de status e progresso.

Encerramento: Aqui o gerente é encarregado de obter a aceitação final do cliente, avaliar e relatar as lições aprendidas na execução do projeto.

QUADRO 5 - CONHECIMENTO DA GESTÃO DE PROJETOS BASEADO NO PMI

	Fases		
Áreas de Conhecimento	Planejamento	Execução	Controle
Integração	Desenvolvimento de plano de projeto	Execução de plano de projeto	Controle integrado de mudanças
Escopo	Planejamento do escopo		Verificação do escopo
	Detalhamento do escopo		Controle de mudança do escopo
Tempo	Definição das atividades		Controle do cronograma
	Sequenciamento das atividades		
	Estimativa de duração das atividades		
	Desenvolvimento do cronograma		
Custo	Planejamento dos recursos		Controle de custo
	Estimativa dos custos		
	Projeção de orçamento		
Qualidade	Planejamento da qualidade	Garantia da qualidade	Controle da qualidade
Recursos humanos	Planejamento organizacional	Desenvolvimento da equipe	
	Montagem da equipe		
Comunicações	Planejamento das comunicações	Distribuição das informações	Relato de desempenho
Risco	Planejamento dos riscos		Controle e Monitoração de riscos
	Identificação dos riscos		
	Análise qualitativa dos riscos		
	Análise quantitativa dos riscos		
	Planejamento de respostas a riscos		
Aquisições	Planejamento das aquisições	Requisição de propostas	
	Preparação das aquisições	Seleção de fornecedores	
		Administração de contratos	

FONTE: Conhecimento da gestão de projetos baseado no PMI

Uma das atividades que requer devida atenção do gerente de projetos é a gerência de riscos que impacta diretamente sobre as demais atividades. A ocorrência e a natureza dos riscos podem acarretar em dificuldades sérias no projeto envolvendo elevação de custos projeto e atraso, dentre outros. Tais fatores são indesejáveis ao cliente e podem comprometer significativamente o sucesso do projeto. Essa atividade, especificamente, será tratada na seção seguinte.

Cabe aqui destacar que um dos documentos mais importantes para a gestão de projetos é o plano de projeto pertinente à gerência de integração. O plano de projeto é um guia essencial à condução do projeto sem o qual o gerente fica perdido. Com o plano, há um entendimento dos riscos e compromissos inerentes. O plano constitui uma base para execução sistemática do projeto além de servir como mecanismo eficiente de comunicação entre membros da equipe e entre empresa desenvolvedora e (empresa) cliente.

Um plano de projeto, geralmente, contém um conjunto de informações que permitirá ao gerente não apenas executar o projeto, mas também monitorar seu progresso. A lista a seguir apresenta uma relação dos itens que são imprescindíveis de compor um plano de projeto. Note que não há aqui a intenção de ser completo. Entretanto, os itens relacionados são considerados como obrigatórios num plano de projeto de empresa.

- 1 Introdução:** Contém uma descrição dos objetivos do documento, o público ao qual ele se destina e em linhas gerais o propósito do projeto a ser desenvolvido. Pode adicionalmente conter termos e abreviações usadas, além de informar como o plano deve evoluir.
- 2 Escopo do projeto:** Esta seção descreve em linhas gerais o projeto a ser desenvolvido, comunicando o propósito do mesmo, e a importância do projeto para todas as partes envolvidas. O escopo do projeto que será executado é apresentado com uma descrição dos requisitos técnicos (isto é, os requisitos do produto a ser desenvolvido) que podem ser funcionais, não funcionais (desempenho, usabilidade, portabilidade, confiabilidade etc.) e tecnológicos (Tecnologia a ser utilizada). Também, apresentam-se requisitos não técnicos (como, por exemplo, treinamento) e o escopo não contemplado.
- 3 Organização do projeto:** Apresenta-se uma descrição da estrutura organizacional do projeto, incluindo organograma e a definição de papéis e responsabilidades.
- 4 Equipe e infraestrutura:** Contém descrição da equipe e da infraestrutura utilizada para o desenvolvimento do projeto, incluindo: pessoal, equipamentos, ferramentas, *software* de apoio, materiais, dentre outros. Isto visa garantir uma estrutura adequada para a execução das atividades previstas no plano. Nesta seção também é apresentada o planejamento da alocação de pessoal no projeto.
- 5 Acompanhamento do projeto:** Esta seção do plano de projeto relaciona os momentos para realização das atividades de verificação do projeto, as quais poderão ser feitas pela equipe técnica das instituições envolvidas (desenvolvedora e cliente), e também a forma como essas atividades serão

realizadas. Essas atividades incluem a realização de reuniões e geração de relatórios descrevendo informações sobre o progresso do projeto.

- 6 Marcos do projeto:** Contém uma descrição de marcos importantes do projeto (incluindo as datas de início e fim do projeto), bem como os artefatos que serão entregues pela empresa desenvolvedora nestes marcos, quando aplicável. Apenas marcos relevantes devem ser listados, ou seja, aqueles que contribuirão para a medição do desempenho do projeto. Por exemplo: reuniões de revisão, apresentação de protótipos ou realização de testes de aceitação. Note que é possível inserir uma visão do cronograma do projeto neste item, destacando apenas os marcos importantes e suas datas alvo.
- 7 Gerência de riscos:** Os riscos identificados para o projeto estão detalhados e monitorados nos relatórios de progresso. Exemplos de riscos compreendem: risco de pessoal, risco tecnológico e de escopo, dentre outros. Um caso de risco de escopo é a falta de clareza na definição do escopo de projeto que pode resultar em inúmeras solicitações de mudança de escopo.
- 8 Qualidade do produto (ou sistema):** Informa-se a metodologia de desenvolvimento adotada no projeto. Caso, por exemplo, alguma ferramenta específica de desenvolvimento venha a ser utilizada no projeto, isso deve ser descrito neste item. Adicionalmente, informam-se como os artefatos serão gerados por este projeto, os padrões adotados, formatos dos arquivos e *templates* a serem empregados. Também, neste item, costuma-se informar os critérios de aceitação do projeto.
- 9 Testes do produto (ou sistema):** Este item apresenta uma descrição do projeto de testes do projeto, incluindo detalhamento da estratégia de implementação dos testes, com estágios e tipos de testes a serem realizados para garantir a conformidade do produto com as especificações de requisitos funcionais, não funcionais e requisitos de aceitação do projeto.
- 10 Referências:** Apresenta-se uma relação dos documentos pertinentes ao projeto.

5 GERÊNCIA DE RISCOS

Risco é entendido como a probabilidade de que uma situação indesejável irá acontecer. Nesse sentido, a gerência de riscos está interessada em identificar riscos e buscar mecanismos que possam atenuar ou, até mesmo, eliminar os riscos. Os riscos podem ser de três naturezas: riscos de projeto, de produto ou de negócios. O foco desta seção recai, especificamente, no primeiro. Questões que motivam decisões e são consideradas na gerência de riscos compreende:

- Como mudanças de requisitos ou tecnologia podem afetar o cronograma e o sucesso de um projeto?
- Quais métodos ou ferramentas devem ser empregados num projeto?
- Quais atributos da qualidade devem ser priorizados?
- Quantas pessoas devem ser alocadas no projeto e quais suas habilidades mínimas?

Em muitas empresas, é comum haver a gestão de projetos sem a devida atenção à gerência de riscos. Como não há a prática da gerência de riscos, em tais casos, os riscos costumam ser identificados e analisados de forma aleatória, sem quaisquer disciplinas. Geralmente, o que é feito é apenas um *brainstorming* que pode resultar na não antecipação de futuras ocorrências que podem impactar o produto ou sistema que está sendo desenvolvido. A boa prática da gestão de projetos recomenda a inclusão no processo de engenharia de sistemas de *software* da gerência de riscos que compreende quatro atividades:

- 1 Identificação de riscos onde se busca identificar riscos de projeto ou de negócios. Para cada risco identificado, associa-se uma magnitude de risco que serve para indicar o grau de severidade e, portanto, de prioridade de tratamento do risco.
- 2 Análise de risco que visa obter a probabilidade de ocorrência desse e correspondente impacto sobre o projeto. Note que o impacto pode ser o atraso no projeto, que é tanto indesejável ao cliente quanto implica em custo adicional.
- 3 Administração de risco que tem duas metas: (a) desenvolver uma estratégia de controle que serve para mitigar ou reduzir o impacto de um risco, e (b) elaborar um plano de contingência o qual recomenda as decisões (alternativas) a serem tomadas caso o risco aconteça.
- 4 Monitoração de risco que faz uso de indicadores com o objetivo de monitorar e detectar a ocorrência ou probabilidade de ocorrência de um risco.

Diversas fontes de riscos são consideradas num projeto de *software*. Dentre elas podemos destacar o conjunto abaixo:

- **Equipe:** Membro chave da equipe não estará disponível quando necessário.
- **Escopo:** Falta de clareza no escopo resultará em várias mudanças de escopo.
- **Gerência:** Falta de experiência do gerente de projeto resultará em atraso no cronograma.
- **Tecnológico:** A tecnologia empregada é relativamente nova e pouco conhecida dos membros da equipe que pode resultar em atraso.
- **Equipamento:** Dispositivos necessários não serão entregues no prazo programado.
- **Cliente:** Recursos do cliente não estarão disponíveis como planejado.
- **Físico:** Um vírus de computador infectará o ambiente de desenvolvimento do projeto.
- **Entrega:** Os requisitos da capacidade do sistema excederão a capacidade disponível.

Para os riscos apresentados, é importante identificar a probabilidade de sua ocorrência, o impacto que eles podem trazer (ao projeto), o grau de severidade e como eles podem ser administrados. Por exemplo, se considerarmos o risco da perda de um membro importante da equipe, isto pode ser diagnosticado quando ocorrem reclamações por parte do membro da equipe, bem como quando ele

solicita ser realocado para outra atividade. Em outras situações, outros gerentes na mesma empresa consultam sobre a disponibilidade de membros de sua equipe. Em tais circunstâncias, o gerente de projeto deve fomentar junto ao grupo o forte senso de trabalho em equipe e também destacar a importância do projeto e a participação dos membros naquela equipe.

6 GERENTE DE PROJETOS – TAMBÉM UM LÍDER

Um conjunto de itens que auxilia o sucesso de um projeto compreende: objetivos ‘claros’ do produto ou sistema a ser desenvolvido, escopo bem delimitado, uso de infraestrutura de *software* padrão, uso de estimativas confiáveis, apoio da alta direção, envolvimento de usuários e experiência do gerente.

Aqui, vale ressaltar que o gerente tem um papel de suma importância para o sucesso do projeto que envolve a aplicação de conhecimentos, habilidades e técnicas para projetar atividades que visem atingir os objetivos do projeto. O gerente acompanha o projeto fazendo uso de processos tais como: iniciação, planejamento, execução, controle e encerramento. Além disso, ele tem de conceber e manter um ‘esquema’ de trabalho que lhe permita alcançar as metas de negócio e executar efetivamente as atividades que se encontram estabelecidas no plano de projeto (escopo, riscos, cronograma, qualidade, dentre outras).

Todavia, um excelente profissional não tem apenas o papel de gerente. Ele é bem mais que isso. Ele é também um líder de uma equipe e hábil negociador. Portanto, a gestão de projetos deve ser orientada para o cliente (como apontado no início desse artigo) e considerar:

- Pensamento estratégico corporativo.
- Valor do cliente e estratégia de relacionamento.
- Táticas e estratégias de negociação.
- Análise e planejamento estratégico.
- Indicadores de desempenho e mercadológico.

O gerente de projeto é uma peça fundamental para coordenação da execução de um projeto. Ele precisa não apenas conhecer, mas também externar o pensamento da corporação de modo a ‘atrair’ e reter o cliente. Precisa ainda saber como se relacionar com o cliente (do projeto). Nesse sentido, o gerente precisa dispor de táticas de negociação. Adicionalmente, indicadores de desempenho e mercadológicos podem ser considerados.

O gerente de projeto é considerado um líder, isto é, uma pessoa visionária, quando o rumo (do projeto) não é conhecido. Também é um colaborador

quando o consenso é necessário e ele é capaz de motivar aqueles que estão ao seu redor de forma a canalizar os esforços da equipe para realização e sucesso do projeto. Liderança é, portanto, uma característica essencial para os gerentes de projeto. Para tanto, é necessário conhecer a cultura organizacional e o desejo de mudanças. Nesse sentido, vale lembrar um pensamento do lendário Peter Drucker que dizia: “a coisa mais importante na comunicação é ouvir o que não é dito”. Pense nisso.

FONTE: Disponível em: <<http://www.devmedia.com.br/gestao-de-projetos-de-software/9143>>. Acesso em: 31 mar. 2016.

LEITURA COMPLEMENTAR

OS 7 PASSOS DO GERENCIAMENTO DE PROJETOS

O enxugamento dos quadros de pessoal e o aumento da necessidade de especialização técnica têm levado muitas empresas a recrutar no mercado profissionais por período determinado apenas para a execução de projetos específicos. Neste contexto, entender o processo de gerenciamento de projeto tem se tornado vital para organizações à medida em que mais e mais novos negócios vão se revestindo da aura de projeto e passam a exigir um cabedal de técnicas gerenciais que nem sempre estão disponíveis nas empresas.

Um projeto é um empreendimento temporário, com data de início e fim, cujo objetivo é criar ou aperfeiçoar um produto ou serviço. Gerenciar um projeto é atuar de forma a atingir os objetivos propostos dentro de parâmetros de qualidade determinados, obedecendo a um planejamento prévio de prazos (cronograma) e custos (orçamento). Ou seja, dadas as metas e as restrições de recursos e tempo, cabe ao gerente de projetos garantir que ele atinja os objetivos propostos.

Muitas empresas estão adotando a estrutura de projetos no seu dia a dia. Desde a concepção de um novo *software* até a implantação dos procedimentos de atendimento a clientes, desde a construção de uma ponte até a revisão dos processos de venda com vistas a aumentar a taxa de fechamento de negócios, muitos empreendimentos no seio das organizações se enquadram na classe de projetos. Nos mais diversos setores, a abordagem de gerenciamento de projetos está ganhando terreno por permitir um melhor uso dos recursos para se atingir objetivos bem definidos pela organização. Sabendo da importância de se gerenciar bem um projeto, vamos ver os passos que nos levam a melhorar nossas habilidades de gerenciamento de projeto.

Tudo começa com a contratação de uma empresa para tocar o projeto ou a definição dos colaboradores internos que integrarão a equipe desse. Num dia determinado, inicia-se o projeto. Este momento deve ser formalizado com um documento que se chama de “termo de início do projeto”. Em projetos maiores,

deve ser um documento assinado pelos patrocinadores e pelo gerente do projeto, para menores, pode ser um e-mail que o gerente envia aos patrocinadores, copiando os demais envolvidos, para notificar que naquele momento se inicia o projeto e todos estão envolvidos com a sua execução.

1 Escolha e adote uma metodologia

Uma metodologia é um processo a seguir que dá maior controle sobre os recursos que serão utilizados no projeto. Controlando melhor o processo a equipe será mais eficiente pois entregará o projeto com maior grau de acerto em termos de prazos e custos. O bom uso de uma metodologia é importante porque permite evitar práticas que levam ao insucesso e com isso reproduzir o sucesso.

A Microsoft usa o MSF (*Microsoft Solutions Framework*) no desenvolvimento de seus produtos. Muitas empresas na área de *software* optam pelo CMM (*Capability Maturity Model*). A opção pela metodologia deve ser tomada a partir de alguns fatores: as exigências de cada mercado em que a empresa atua, a disponibilidade de mão de obra e a cultura organizacional necessária para adotá-la. Para exportar *software*, muitas empresas nacionais têm se alinhado com o padrão CMM para dar credibilidade a sua iniciativa em mercados dominados por indianos e chineses, que já possuem capacitação neste padrão.

Em última instância, uma metodologia é um conjunto de regras de como conduzir um projeto com sucesso. Pode até não ter siglas bonitas, mas é importante que já tenha se mostrado eficiente dentro da sua empresa, de preferência em situação similar à que você está vivendo no seu projeto atual. Para quem gosta de siglas, há uma que está bem na moda: a UML (*Unified Modeling Language*) que, como já diz o nome, não é uma metodologia, mas uma linguagem, uma forma de se documentar um projeto. Uma linguagem de modelagem é uma notação, em geral feita com símbolos gráficos, que se usa para traduzir processos abstratos. A empresa que criou a UML desenvolveu uma metodologia conhecida como RUP, “*Rational Unified Process*”.

2 Comunique-se: não é só o peixe que morre pela boca!

Quando falta comunicação, os boatos e outras formas de ruídos tomam seu lugar. Na falta de versão oficial, ficam circulando informações que podem minar a moral da equipe e levantar suspeitas sem fundamento. O gerente de projeto deve evitar esse tipo de prática, conhecida por “rádio-peão”, dando informações claras e confiáveis sobre o *status* do projeto. Certamente esta é uma área em que a diplomacia é essencial. Se há um problema, o gerente de projetos pode e deve não só falar sobre ele, mas também informar que está trabalhando na solução, e não apenas comunicar que o problema existe. Problemas sem uma perspectiva de solução são angustiantes e causam um desconforto na equipe que muitas vezes é desnecessário.

A criação de relatórios de progresso do projeto ajuda no processo de comunicação, sobretudo por que torna o processo impessoal e mais objetivo. Imagine o efeito de um *e-mail* onde se critica um membro da equipe pelo atraso do projeto. Imagine a mesma informação vinda de um relatório em que a data de término real de uma tarefa está em branco: objetivamente a situação é a mesma, o indivíduo não fez a sua parte, mas no caso do *e-mail* a pessoa envolvida pode se melindrar. No relatório, temos um dado objetivo, que salta aos olhos, mas que não gera ressentimentos.

3. Defina o escopo do projeto e detalhe as atividades

O “escopo do projeto” é o trabalho que deve ser realizado para se obter um produto ou serviço com determinadas características e recursos. Comece por definir o que deve ser feito e o que não deve. Esse processo nos permite entender os contornos do projeto e traçar uma linha divisória entre o que deve ser feito e o que não deve ser, pelo menos neste momento. Muitos novatos se perdem em discussões intermináveis sobre recursos do produto final que o tornariam “perfeito”. Sempre me lembro de um amigo muito experiente que, ante a minha ânsia em acertar todos os detalhes logo de cara, me dizia que “o ótimo é inimigo do bom”, ou seja: enquanto perseguimos o “ótimo” nos distanciamos de algo que está bem mais próximo, o “bom”, e que temos mais chance de conseguir atingir. Com o tempo achei uma forma elegante de contornar as exigências de projeto sem decepcionar os clientes: não é que não faremos o que está sendo pedido, mas devemos ver que este recurso cabe na versão 2, 3 etc., mas não cabe na versão 1, que é o que estamos tentando desenvolver neste momento! Afaste o fantasma da perfeição.

Para você não se perder numa lista interminável de características da versão 1, uma boa ideia é pedir ao cliente que liste só o que é “absolutamente essencial”. Claro que se você der a ele 30 minutos para responder, tudo será “absolutamente essencial”. Não adianta, temos de ser realistas, o tempo é curto e temos de escolher só o que realmente é importante. Se “escrever é cortar” como dizem os grandes escritores, a arte de se definir o escopo do projeto passa por saber o que abandonar e o que reter do universo de necessidades do cliente.

Definido o escopo do projeto, podemos passar para a fase de detalhamento das tarefas. O objetivo é chegar ao WBS (*Work Breakdown Structure*), onde temos as “unidades de trabalho” com tempo medido em dias ou horas de trabalho. Como regra, uma atividade deve ocupar entre 4 e 80 horas, nem mais, nem menos.

Em paralelo, deve ser elaborado um orçamento levando em conta quantas horas de cada profissional serão necessárias. Veja um modelo simples:

Profissional	Tarefa 1	Tarefa 2	Tarefa 3	Total (h)	Custo/h	Custo
Gerente de projeto	20	0	3	23	150,00	3.450,00
Líder de projeto	10	3	2	15	80,00	1.200,00
Analista Sênior	20	0	0	20	50,00	1.000,00
Programador	0	40	20	60	30,00	1.800,00
Testador/ Documentador	0	20	30	50	15,00	750,00
Total	-	-	-	168	-	8.200,00

Para montar este modelo, você precisa saber o custo-hora de cada profissional e estimar o tempo que cada um gastará no projeto. Os profissionais podem estar envolvidos em outros projetos e quando o programador está cuidando de uma fase do projeto A, o gerente de projeto já pode estar planejando o projeto B, só voltando ao projeto A quando for para entregar ao cliente e obter a sua aprovação, sobre o que falaremos mais adiante.

Essas estimativas são mais precisas à medida em que se avança no detalhamento do projeto. Para estimativas iniciais, admite-se uma variação de -25% a +75%. Na fase de planejamento, o orçamento deve ter uma variação de -10% a +25%. Lembre-se que nesta fase, o gerente de projeto já envolveu quem realizará a tarefa. Na estimativa final, a margem de erro é menor: de -5% a +10%. Aqui, o conhecimento do gerente de projeto de situações anteriores fará diferença. Eu, por exemplo, sei que quando lido com determinados clientes, haverá tanto “overhead” administrativo, como dezenas de *reports* para cima e para baixo antes que cada passo importante seja dado, que eu já estimo 50% a mais do tempo nas tarefas em que o cliente está diretamente envolvido. Vai da experiência do gerente, mas nessa hora, se a empresa tem um histórico de projetos semelhantes, vale a pena se basear neste *background*, mesmo que tenha sido com outras equipes e outros gerentes de projeto.

Um dos grandes segredos do gerenciamento de projetos é proteger o seu escopo. Projetos que ficam mudando o escopo durante sua execução têm sérias dificuldades em cumprir o cronograma e estouram o orçamento. O risco mais comum é o que se chama de “*scope creep*”, quando o escopo vai crescendo à medida que o cliente vai entendendo suas necessidades e reformulando seus objetivos. Há quem chame este problema de “Jacques”. Seria uma homenagem a um francês ilustre? Não, trata-se apenas da forma como o cliente costuma abordar o assunto: “já que o sistema faz isso, ele pode então fazer aquilo. Agora eu quero aquilo também incorporado ao projeto”. O gerente do projeto deve ter calma e analisar com cuidado cada demanda: ao rejeitar um pedido, ele pode se indispor com o cliente, mas se aceitar ele pode estar dando um tiro no próprio pé, já que o prazo e orçamento não serão tão “elásticos” quanto as exigências. Devemos sempre contar com uma certa “margem de manobra”, mas nos tempos atuais, em que eficiência

é a palavra que está na ordem do dia, não há muita “gordura para queimar” e os compromissos assumidos pelo gerente podem se transformar num sacrifício, muitas vezes desnecessário, para toda a equipe.

Em projetos de *software*, o “*scope creep*” é uma situação tão comum que não dá para começá-los sem tomar algumas precauções. O primeiro cuidado é negociar a forma de remuneração: fixa ou variável. Se for fixa, o risco das mudanças está todo com o gerente do projeto, se for variável, o cliente assume os custos extras. Mesmo neste caso, o gerente do projeto deve cuidar para que o cliente seja informado *a priori* dos novos custos. Por precaução, eu sempre redijo um adendo ao escopo colocando o que será feito, em quanto tempo e a que custo. Colho a assinatura do cliente e só depois autorizo a execução da tarefa. Gerentes financeiros não participam destas reuniões e podem alegar que não há previsão de recursos para os extras, então mantenha-os informados das novas condições para evitar dissabores na hora do recebimento.

O segundo cuidado é documentarmeticulosamente o escopo do projeto. Este documento resume o que será feito, com que características e com que recursos. Ele é um “quase-contrato”, mas não traz as cláusulas de rescisão e as penalidades. Neste momento, tudo está bem e todos concordam. Só que, na cabeça de cada um, há uma imagem diferente do que será o produto final. A medida que este produto vai tomando forma e sendo entregue, o cliente vai vendo que o que ele imaginou “não é bem aquilo” e podem começar as decepções.

A satisfação do cliente depende em muito do que será dito e prometido no que se chama de “pré-venda”. É neste momento que o gerente de projetos deve entrar em cena para meticolosa, cuidadosa e disciplinadamente escrever tudo o que o sistema deve ter e fazer. Este processo é o “planejamento de escopo” e num *software* dele abrange das telas até os relatórios. Esta tarefa pode ser delegada para um analista, mas a responsabilidade não sai nunca das mãos do gerente. Eu costumo especificar toda a interface dos usuários com o sistema: telas e relatórios. Depois de “colocar tudo no papel”, o gerente deve obter do cliente um “de acordo”, de preferência assinado no final do documento em que todas as páginas serão rubricadas com um “visto” para que ele tome ciência do que será feito. Não há palavras para expressar a importância deste planejamento em que as expectativas serão levantadas e moldadas, de forma que, diante do produto final, o cliente não possa se dizer decepcionado.

O terceiro cuidado é definir prioridades. O gerente deve ter a sensibilidade para identificar quais são os requisitos obrigatórios e quais os desejáveis, marcando cada um segundo com a sua prioridade. Isso evita que alguém arbitre o que é importante no lugar do cliente. Há gerentes de projeto que vão mais longe e pedem ao cliente para definir o que ele considera “sucesso” do projeto. Por exemplo, num sistema em que havia desperdício de 30% da matéria-prima, foi considerado sucesso reduzir esta taxa para 15%. Mas este número ainda é alto, diria você. Sim, mas o cliente considerou que uma redução de 50% dos desperdícios já representaria

benefícios suficientes que compensariam os investimentos no projeto. Além do mais, lembre-se de que: “o ótimo é inimigo do bom”.

Em suma: definir o escopo, no fundo, é saber o que deve ser feito para atender a necessidade do cliente.

4 Conheça os envolvidos e monte seu time

Todos os envolvidos no projeto são os "*stakeholders*". Nesse grupo estão não apenas os membros da equipe, mas também os clientes e fornecedores envolvidos. Dentro da empresa do cliente, há uma pessoa que se destaca por ser a patrocinadora ("*sponsor*") do projeto. Ela é que cria as condições para a contratação do projeto, mesmo que não seja ela que vá usar o produto final.

É importante que o gerente do projeto conheça os interesses de todos os envolvidos. Imagine como é arriscado contar com um membro da equipe que não está disposto a colaborar. Ele pode ser um problema maior do que uma solução dentro do grupo: sabendo disso, melhor pensar em chamar outra pessoa. Eu passei por uma situação dessas quando fui destacado para gerenciar um projeto onde havia um colaborador mais antigo e que entendia que ele é quem deveria estar gerenciando. Eu não percebi seu ressentimento a princípio e à medida que o projeto avançava essa pessoa se tornava um problema cada vez maior, na medida em que, não só ele não fazia a sua parte, como minava os demais membros da equipe contra minhas decisões. Um dia, eu o chamei e abri o jogo. Ele então me explicou o que estava sentindo e fizemos um acordo: ele se enquadraria para completar o projeto, que graças a ele já estava atrasado, e eu o apoiaria junto à direção para que recebesse seu próprio projeto para gerenciar. É claro que manter um “profissional” com este tipo de atitude não é bom negócio para a empresa no longo prazo, porque cedo ou tarde ele vai acabar atirando contra a própria equipe novamente, só para mostrar que as “coisas têm de ser feitas do jeito dele”.

No processo de definição do escopo, as habilidades necessárias vão ficando mais claras. Nesse momento, é importante formar uma equipe com competência diversificada e com experiência nas áreas de atuação do projeto. Em projetos em que há muito conhecimento técnico envolvido, surge a figura do “líder de projeto”, um profissional com grande conhecimento técnico e com capacidade de liderança entre os técnicos. Em geral é um profissional sênior, com credibilidade junto aos demais técnicos e com muita bagagem. A experiência desse especialista pode economizar muito tempo e dinheiro no projeto. Dê-lhe voz ativa, sobre dele *insights* que você não tem e respeite a sua opinião. Só assim ele estará sempre do seu lado, mesmo quando você errar.

5 Desenvolva o cronograma junto com quem põe a mão na massa

Uma vez que temos as tarefas definidas a partir do escopo, temos de estimar a duração de cada uma. Procure fazer esta estimativa de tempo de execução com a ajuda de quem está escalado para executar o trabalho. Ao mesmo tempo

em que essa pessoa é quem melhor sabe quanto tempo precisará, ela estará se comprometendo com um prazo para a sua execução. Por outro lado, quando se trabalha com consultores externos, o custo será função direta do tempo estimado para a execução do projeto. Ao fixar o cronograma, o profissional está dando por tabela um orçamento da sua parte.

Veja estas atividades que representam as linhas gerais de um projeto de sistema:

	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1	Levantamento das necessidades	7 days	Mon 5/7/04	Tue 13/7/04		
2	desenho das telas	3 days	Mon 5/7/04	Wed 7/7/04		Analista
3	layout dos relatórios	2 days	Thu 6/7/04	Fri 9/7/04	2	Analista
4	regras de negócio	2 days	Mon 12/7/04	Tue 13/7/04	3	Analista
5	Programação	8 days	Thu 8/7/04	Mon 19/7/04		
6	telas	3 days	Thu 8/7/04	Mon 12/7/04	2	Programador
7	relatórios	2 days	Tue 13/7/04	Wed 14/7/04	6	Programador
8	funções de cálculo	3 days	Thu 15/7/04	Mon 19/7/04	7	Programador
9	Testes	9 days	Tue 20/7/04	Fri 30/7/04		
10	Teste interno - alfa	2 days	Tue 20/7/04	Wed 21/7/04	8	Programador
11	Teste com cliente - beta	2 days	Thu 22/7/04	Fri 23/7/04	10	Analista, Programador
12	homologação com cliente	5 days	Mon 26/7/04	Fri 30/7/04	11	Analista, Programador
13	documentação	10 days	Wed 14/7/04	Tue 27/7/04		
14	diagramas dos processos	2 days	Wed 14/7/04	Thu 15/7/04	4	Analista
15	manual do usuário	1 day	Mon 26/7/04	Mon 26/7/04	11	Analista, Programador
16	informações para manutenção	1 day	Tue 27/7/04	Tue 27/7/04	15	Analista, Programador

Note que além de saber o que deve ser feito, as tarefas têm três propriedades importantes: duração, interdependência e responsável. A duração é importante, mas se as tarefas podem ser realizadas em paralelo, como é ilustrado neste caso onde há duas figuras: o analista e o programador, a duração total do projeto encurta. Dessa possibilidade de *trade-off* entre tempo e recursos alocados, alguns gerentes acreditam que se o projeto está atrasado, então “basta colocar mais gente” que o problema se resolve. Isso raramente ajuda uma vez que com mais gente, os problemas de comunicação aumentam e o projeto que já está atrasado atrasa mais ainda. Trazer mais gente pode ser útil quando se precisa de especialistas em temas que os membros não dominem. A rigor, se o planejamento foi bem-feito, já se sabe que esta mão de obra será recrutada em algum momento do projeto. A atitude de simplesmente aumentar a equipe para acelerar a produção é que está errada e deve ser combatida. Só que alguns gerentes de projeto medem seu poder pelo tamanho da equipe que gerenciam. Você pode imaginar como isso acaba: contratamos mais pessoas, eu fico mais “poderoso” e temos todas as explicações para os atrasos, afinal o projeto era mesmo “muito grande”.

O gerente de projetos deve trazer sua experiência para corrigir as expectativas muito otimistas de algum colaborador mais afoito. Sim, há quem estime 50 horas e depois, com a maior tranquilidade, cobre pelas 120 horas que foram necessárias para realizar a tarefa. Ele só errou em 140%! Se o preço é fechado,

o risco fica todo com o consultor, mas a sua boa-vontade e a qualidade do produto final podem sofrer em decorrência da pressa. Se a remuneração ficar vinculada ao tempo de prestação de serviço, o contratante precisa de um mecanismo de controle minimamente confiável. Eu não uso uma fórmula geral, prefiro trabalhar segundo as características do profissional, mas de todos exijo um relatório de horas que contém o dia, data de hora e início, tempo de trabalho e a(s) tarefa(s) realizadas no dia.

Se no planejamento da semana há tarefas que não foram realizadas, na reunião de avaliação, eu pergunto porque a coisa não seguiu o ritmo programado e quanto isso impacta na data final de entrega. Procure estabelecer pontos de controle, "*check-points*", que são datas onde se medirá o andamento do projeto em face do cronograma que havia sido programado. Nestas datas, pode-se estar apenas executando-se uma verificação do progresso das atividades (*milestones*) ou pode haver entrega de produtos ou subprodutos (*deliverables*) tais como desenhos, especificações, protótipos, modelos etc.

Quem já reformou ou construiu uma casa sabe que esta tão trivial experiência de gerenciamento de projeto pode acabar mal. Quantas histórias existem de gente que foi pagando o pedreiro sem atrelar os pagamentos a entregas de tarefas determinadas. Nestas histórias tristes, o dinheiro acaba antes da obra, e o pedreiro some, deixando o cliente sem dinheiro e sem a sua casa. Tudo porque ele não cuidou de atrelar entregas de tarefas a pagamentos, não criou pontos de controle que lhe dariam visibilidade do atraso. Sabendo antes que a "vaca está indo para o brejo" o cliente pode optar por "apertar" o pedreiro ou suspender os trabalhos enquanto ainda tem dinheiro, que poderá ser usado para pagar uma equipe mais eficiente.

É verdade que em projetos de TI nem sempre dá para "trocar o pedreiro" porque há muito conhecimento e estudo envolvidos. Mas por isso mesmo, temos de ser muito mais cuidadosos na monitoração para saber em que momento o projeto começa a atrasar e como fazer para recuperar o ritmo no futuro próximo.

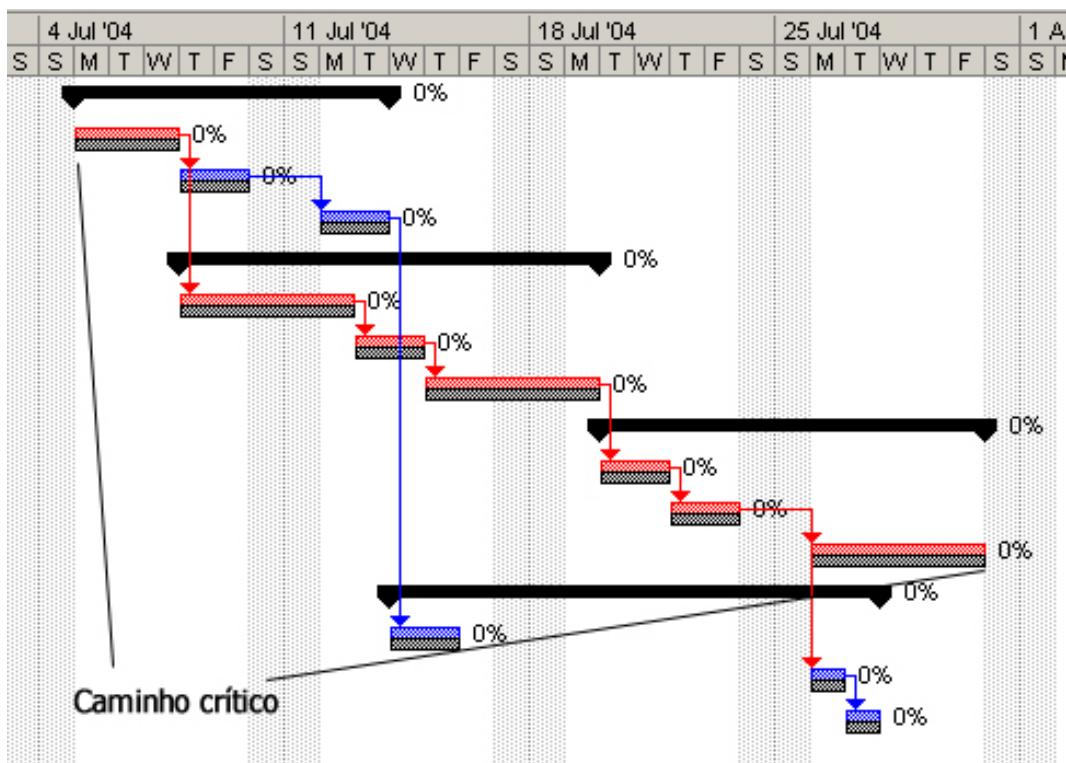
6 Monitore os riscos e seja proativo

Agora que todos sabem o que devem fazer, é importante mitigar os riscos que podem impedir o bom desenvolvimento do projeto. Desenvolva uma lista de fatores de risco e um plano para lidar com eles. Mas lembre-se de que são duas coisas diferentes: a monitoração do risco e controle do risco.

A monitoração dos riscos envolve acompanhar o *status* de cada risco e as opções de ações definidas para enfrentá-los, caso eles venham a se tornar problemas reais. A monitoração também se preocupa em avaliar a probabilidade de ocorrência de um risco, qual o seu impacto no andamento do projeto e como contorná-lo. Por exemplo, numa determinada tarefa crítica a contratação de dois profissionais pode parecer um exagero, mas o gerente do projeto sabe que se algo

acontecer nesta área do projeto o impacto será grande no restante. Os profissionais passam a ser um *backup* do outro dentro da linha de que “quem tem um, não tem nenhum”.

Voltando ao nosso projeto de exemplo, chamo a atenção para um recurso que o MS Project tem e que deve ser usado para se identificar riscos. Veja a tela do diagrama de Gantt que obtivemos a partir da lista de tarefas que elaboramos acima:



Note que há uma sequência de tarefas que quando alinhadas compõem o prazo de duração do projeto todo. Destaquei o início e o final só para que você perceba que se trata de uma série de processos que devem ser gerenciados mais de perto uma vez que o atraso em algum deles acarretará o atraso do projeto todo. Por isso é que se chama este de “caminho crítico”. Os riscos que estão embutidos nestas tarefas são os que se deve gerenciar mais de perto, de forma mais proativa.

O controle dos riscos é o processo de executar o plano de ações e divulgar seus relatórios de *status*. Inclui também possíveis mudanças no plano de riscos, e eventualmente até nos planos do projeto. Essas mudanças são referentes a recursos, funcionalidades ou cronograma.

7 Formalize o início e o encerramento do projeto

O início do projeto é um momento solene. O patrocinador deve formalizar a todos os envolvidos que o projeto está iniciado e o cronômetro está correndo. Muita gente não gosta de se preocupar com isso, mas imagine que haja resistência de setores da empresa que se opõem ao projeto. Sem um documento que atesta que o projeto começou, o gerente pode não conseguir apoio algum. Além disso, este documento funciona como um “cumpra-se” de uma autoridade da empresa: não cabe discutir a ordem, o projeto começou e todos os “arrolados” devem participar.

Outro momento importante é o do encerramento do projeto. É preciso formalizar o final para que fique claro para todos os envolvidos, especialmente para o cliente, que o projeto está concluído e que novas necessidades serão atendidas em um novo projeto. Qualquer extensão ou alteração deverá ser orçada e todo o ciclo se inicia novamente. Com relação à manutenção do sistema entregue, não se pode considerá-lo um projeto na medida em que, a princípio, trata-se de um processo contínuo. O que pode ocorrer é definir-se projetos ao longo da vida útil do sistema com o objetivo de melhorá-lo. Por exemplo, a atualização dos equipamentos eletrônicos (aviônicos) de um avião para auxílio ao voo é um projeto que se distingue da sua manutenção rotineira.

Ao final faz-se também uma reunião de avaliação dos erros e acertos da equipe. Chamadas de reuniões "*post-mortem*", elas servem para se gerar uma lista de "melhores práticas" contribuindo para a formação de uma base de conhecimento que poderá ser muito útil em projetos futuros. Da minha experiência pessoal, posso dizer que tirei grandes lições quanto às "piores práticas", atitudes e decisões que se mostraram ruins e que devem ser evitadas em projetos futuros.

CONCLUSÃO

Acima de tudo, gerenciar projetos é planejar e acompanhar a execução com "um olho no peixe e outro gato". O gerente do projeto deve se manter alerta e flexível com os acontecimentos do dia a dia, mas deve estar sempre se reportando ao plano inicial para não perder o controle. A principal qualidade do gerente de projeto é saber se comunicar bem com todos. Ele é o ponto focal das informações, nele convergem as informações que ele depois deverá processar e divulgar para todo o restante da equipe.

O segredo é envolver a equipe, cliente e fornecedores de tal forma que todos se sintam diretamente responsáveis pelo sucesso do projeto. Como diz aquele velho ditado caipira, "quando todos empurram na mesma direção, não há carroça que não saia do atoleiro".

FONTE: Disponível em: <<https://www.microsoft.com/brasil/msdn/tecnologias/carreira/gerencprojitos.mspx>>. Acesso em: 31 mar. 2016.

RESUMO DO TÓPICO 3

Neste tópico, vimos que:

- O gerenciamento de *software* é diferente em alguns pontos do gerenciamento em outras áreas da engenharia.
- Um bom gerenciamento de projeto de *software* é essencial para que os projetos de engenharia do *software* sejam desenvolvidos dentro do cronograma e do orçamento.
- Cinco fases compõem a gestão de projetos: **Inicialização, Planejamento, Execução, Controle e Encerramento.**
- Conhecemos algumas fontes de riscos em um projeto de *software*.
 - Equipe: Membro-chave da equipe não estará disponível quando necessário.
 - Escopo: Falta de clareza no escopo resultará em várias mudanças de escopo.
 - Gerência: Falta de experiência do gerente de projeto resultará em atraso no cronograma.
 - Tecnológico: A tecnologia empregada é relativamente nova e pouco conhecida dos membros da equipe que pode resultar em atraso.
 - Equipamento: Dispositivos necessários não serão entregues no prazo programado.
 - Cliente: Recursos do cliente não estarão disponíveis como planejado.
 - Físico: Um vírus de computador infectará o ambiente de desenvolvimento do projeto.
 - Entrega: Os requisitos da capacidade do sistema excederão a capacidade disponível.

AUTOATIVIDADE



- 1 Quais são as atividades que um gerente de projeto precisa coordenar?
- 2 Os riscos são parte importante e precisam ser gerenciados, liste alguns riscos de um projeto.
- 3 Explique quantas e quais são as fases que compõem a gestão de projetos.





UNIDADE 3

MODELAGEM ESTRUTURADA DE SISTEMAS, MÉTRICAS DE PROCESSO DE SOFTWARE, FUNDAMENTOS DE MELHORIA DE PROCESSO DE SOFTWARE, FERRAMENTAS DE GESTÃO DE PROCESSOS DE SOFTWARE BPMN E EPF.

OBJETIVOS DE APRENDIZAGEM

A partir desta unidade você será capaz de:

- entender quais artefatos fazem parte do conceito e da modelagem estruturada de sistemas;
- verificar a importância de validar via métricas os processos de *software*;
- identificar quais são as metodologias e a importância de se obter melhoria de processos de *software*;
- identificar as principais ferramentas no mercado para confecção de processos de *softwares*;
- compreender sobre o BPMN funciona, pois serve de base à gestão de processos;
- compreender um case de processo de *software* utilizando a ferramenta EPF Composer.

PLANO DE ESTUDOS

Esta unidade contém três tópicos. No final de cada um deles, você encontrará atividades que contribuirão para a apropriação dos conteúdos.

**TÓPICO 1 – INTRODUÇÃO À MODELAGEM ESTRUTURADA E
MÉTRICAS DE PROCESSO DE SOFTWARE**

**TÓPICO 2 – FUNDAMENTOS DE MELHORIA DE PROCESSO DE
SOFTWARE E FERRAMENTA DE GESTÃO DE PROCESSOS
DE SOFTWARE**

**TÓPICO 3 – BUSINESS PROCESS MANAGEMENT NOTATION (BPMN) E
ESTUDO DE CASO DE UM PROCESSO SOFTWARE COM
O AUXÍLIO DO EPF**

INTRODUÇÃO À MODELAGEM ESTRUTURADA DE SISTEMAS E MÉTRICAS DE PROCESSO DE *SOFTWARE*

1 INTRODUÇÃO

Esta unidade explicará como a análise estruturada facilita a comunicação e a especificação de sistemas, retratando seu fluxo e conteúdo das informações utilizados ou construídos, bem como por que as métricas de *software* servem para apresentar medidas que refletem características específicas de processos e de produtos.

2 MODELAGEM ESTRUTURADA DE SISTEMAS

A Engenharia de *Software* define Processos de *Software* como um conjunto de atividades relacionadas que levam à produção de um produto de *software*. Seu ciclo de vida de desenvolvimento é descrito através de modelos e metodologias definindo como ocorre a construção e manutenção de sistemas envolvendo atividades que vão desde um simples relacionamento entre equipes até a mais complexa tecnologia para atingir suas soluções.

Sabe-se que os processos de *software* são complexos e, como todos os processos intelectuais e criativos, passam primeiro por fatores cognitivos de seus profissionais (analistas, programadores, testadores etc), um processo de trabalho que precisa ser claramente documentado através de boas especificações técnicas via métodos eficazes.

FIGURA 39 - ETAPAS DO PROCESSO DE DESENVOLVIMENTO



FONTE: Disponível em: <<http://www.wso.com.br/Images/fabrica-de-software-etapas.jpg>>. Acesso em: 22 out. 2016.

Os objetivos são fundamentais para que o sistema possa ser criado, analisado e construído. Na verdade, se os objetivos não forem muito bem definidos, dificilmente o sistema atingirá seu sucesso. Muitas vezes ocorre que os objetivos declarados não correspondem aos objetivos reais. Durante a análise é fundamental procurar ouvir o máximo de pessoas envolvidas e assegurar de que os objetivos estão realmente estabelecidos para o desenvolvimento. Para isto, devem tentar entender não somente os fatos relativos ao sistema, mas também a lógica que está por trás dos fatos, avaliando a compreensão dos diferentes níveis e categorias de usuários sobre os objetivos, visto que os diferentes agentes envolvidos no sistema geralmente têm visões diferentes dos objetivos do sistema.

Segundo Teixeira (2016), a análise estruturada é um conjunto de técnicas e ferramentas cujo objetivo é auxiliar na análise e definição de sistemas, que serve para construção de um modelo do sistema utilizando técnicas gráficas envolvendo a construção top-down do sistema por refinamentos sucessivos. Na abordagem top-down é formulada uma visão geral do sistema, partindo de uma instância final para a inicial, como uma engenharia reversa. Cada nível vai sendo detalhado, do mais alto ao mais baixo, de forma a se chegar nas especificações dos níveis mais básicos do elemento abordado.

Ainda nos informa Teixeira (2016, p. 7-8) que:

A análise estruturada nada mais é como um método de análise de requisitos de software, é uma atividade de construção de modelos. O sistema é dividido em partições funcionais e comportamentais e assim descrever a essência daquilo que será construído. Esse método é um

dos processos de engenharia de *software* que mais se destaca entre os profissionais de TI. Na análise estruturada, o analista de sistemas utiliza uma representação gráfica formada por símbolos que permitem criar modelos de fluxo de informação.

Para Bonfim (2013, p. 9) a Análise estruturada consiste na construção de um modelo lógico de sistema, “utilizando técnicas gráficas capazes de levar usuários, analistas e projetistas a formarem um quadro claro e geral do sistema e de como suas partes se encaixam para atender às necessidades daqueles que dele precisam”.

A análise estruturada objetiva facilitar a comunicação entre o usuário, analistas e projetistas produzindo uma especificação de sistema rotativa e melhorada. Sua finalidade é de retratar o fluxo e o conteúdo das informações utilizadas pelo sistema, dividindo-o em partições funcionais e comportamentais e ainda descrever a essência daquilo que será construído (TEIXEIRA, 2016).

Segundo Anjos (2016, s.p.):

O uso de codificação estruturada torna possível quantificarmos alguns benefícios resultantes: melhor produtividade em linhas de codificação por dia uso mais apropriado do tempo de teste e assim por diante. Com projeto estruturado, os benefícios também são reais, porém mais difíceis de quantificá-los. Um estudo não publicado sugere que a modificação de um sistema que utilize projeto estruturado chega a ser sete vezes mais fácil e barata do que sistemas tradicionais. Realmente, sob certos aspectos, se o trabalho de análise fosse realizado de forma perfeita, o único resultado seria ausência de problemas.

A análise estruturada deve ser usada apenas para problemas pequenos e simples. Embora seja informal e não validado por computação, o diagrama de fluxo de dados é a parte mais importante da análise estruturada. É de uso bem fácil. Pode ser utilizado para a determinação dos componentes básicos de processamento e dos fluxos de dados de um sistema. Pode ser acompanhado por uma modelagem de dados mais formal.

As principais ferramentas da modelagem da análise estruturada são o Diagrama de Fluxo de Dados (DFD), que ilustra as funções que o sistema deve desempenhar, o Dicionário de Dados (DD), que descreve com precisão o significado dos componentes e das ligações existentes nos modelos de representação gráficas, e o Diagrama de Entidade e Relacionamento (DER), que dá ênfase a dados e o relacionamento entre eles. As sessões a seguir nos irão apresentar maiores detalhes destes artefatos da análise estruturada.

2.1 DIAGRAMA DE FLUXO DE DADOS (DFD)

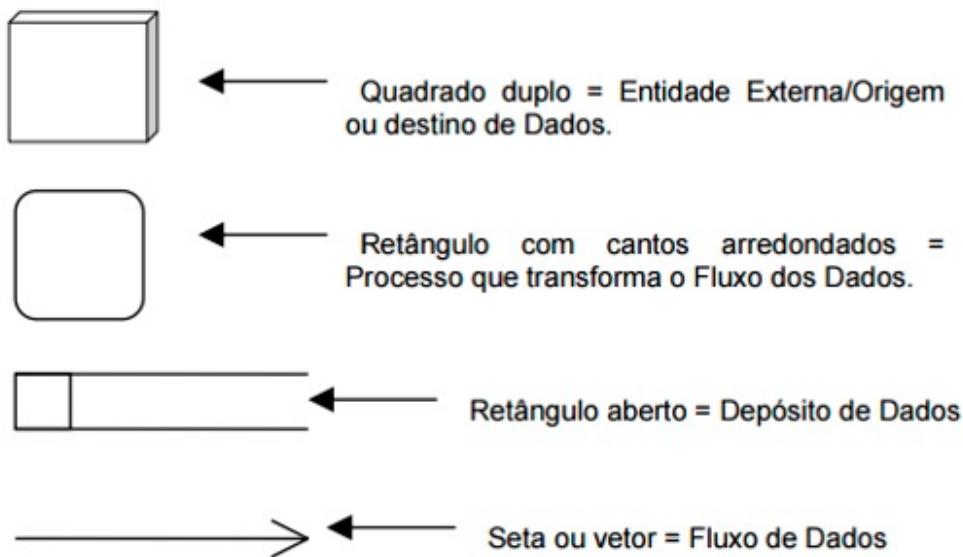
Entre as técnicas estruturadas para análise a processos, o Diagrama de Fluxo de Dados (DFD) é, com certeza, a mais clássica e a mais utilizada. A história da análise estruturada, como Metodologia de Desenvolvimento de Sistemas, se confunde com a do Diagrama de Fluxo de Dados, pois esta técnica caracteriza todo o desenvolvimento estruturado de sistemas.



Os conceitos e notações relativos ao Diagrama de Fluxo de Dados (DFD) a serem apresentadas aqui são principalmente da visão dos grandes autores relacionados à análise estruturada: Tom DeMarco; Chris Gane & Trish Sarson; e Eduard Yourdon.

Segundo Filho (2016), o DFD é uma das principais ferramentas para modelagem de fluxo de dados através da representação de processos que usam e geram dados. Retrata o sistema em termos de suas partes componentes, com todas as interfaces entre os componentes indicados. “É uma representação gráfica baseado apenas em quatro símbolos, que mostra a estrutura do sistema e sua fronteira, onde todas as relações entre os dados, os processos que transformam esses dados é o limite entre o que pertence ao sistema e o que está fora dele” (FILHO, 2016, s.p.).

FIGURA 40 - COMPONENTES DE UM DFD



FONTE: Disponível em: <<http://www.lyfreitas.com.br/ant/pdf/DFD.pdf>>. Disponível em: 22 out. 2016.

O DFD possui apenas quatro elementos que são necessários e suficientes para modelar o sistema em termos conceituais e funcionais. Estes elementos são:

- **Entidade externa:** de onde partem ou para onde chegam os dados (origem e destino).
- **Fluxo de dados:** indicam os dados e o caminho por onde percorrem no sistema.
- **Processo ou função:** ponto de processamento dos dados (tratamento, alteração ou manuseio de dados).
- **Depósito de dados:** indica qualquer tipo de armazenamento de dados.

A representação é feita em níveis ou camadas, partindo-se sempre de uma visão mais global para a visão mais detalhada ou específica. As vantagens de utilizar o DFD para modelagem de sistemas são várias, dentre elas podemos citar:

- Representar o sistema através de um DFD é uma maneira sistemática de definição e especificação.
- DFD é uma especificação que possui algum formalismo, tal como proposto nos princípios da Engenharia de *Software*.
- DFD permite registrar as necessidades e preferências do usuário.

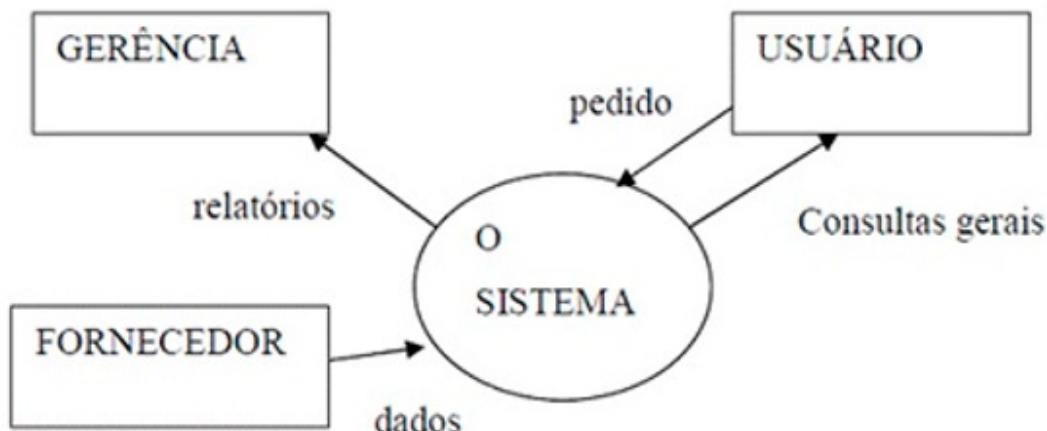
- DFD permite uma visão gráfica do problema, evitando as ambiguidades, redundâncias e omissões próprias de uma especificação narrativa.
- A utilização do DFD permite que o mesmo instrumento de análise sirva também de documentação.
- Uso de DFD facilita o controle de projeto.

FONTE: Disponível em: <<http://docslide.com.br/documents/apostila-de-analise-sistemas.html>>. Acesso em: 22 out. 2016.

Como o DFD é modelado em uma série de níveis, para que possa expor suas funções mais detalhadas em um sistema complexo, cada nível oferece mais detalhes que seu nível superior. Portanto, o DFD Nível 0 é um detalhamento do Diagrama de Contexto, incluindo as mesmas informações daquele diagrama, acrescentando-se o detalhamento de processos, que operam sobre os fluxos de dados, e os depósitos de dados (MACORATTI, 2016, s.p.).

Segundo Macoratti (2016), o detalhamento do Diagrama de Contexto nível mais alto do DFD consiste em uma única bolha, representando o sistema inteiro; os fluxos de dados mostram as interfaces entre o sistema e as entidades externas, esse diagrama chama-se Diagrama de contexto.

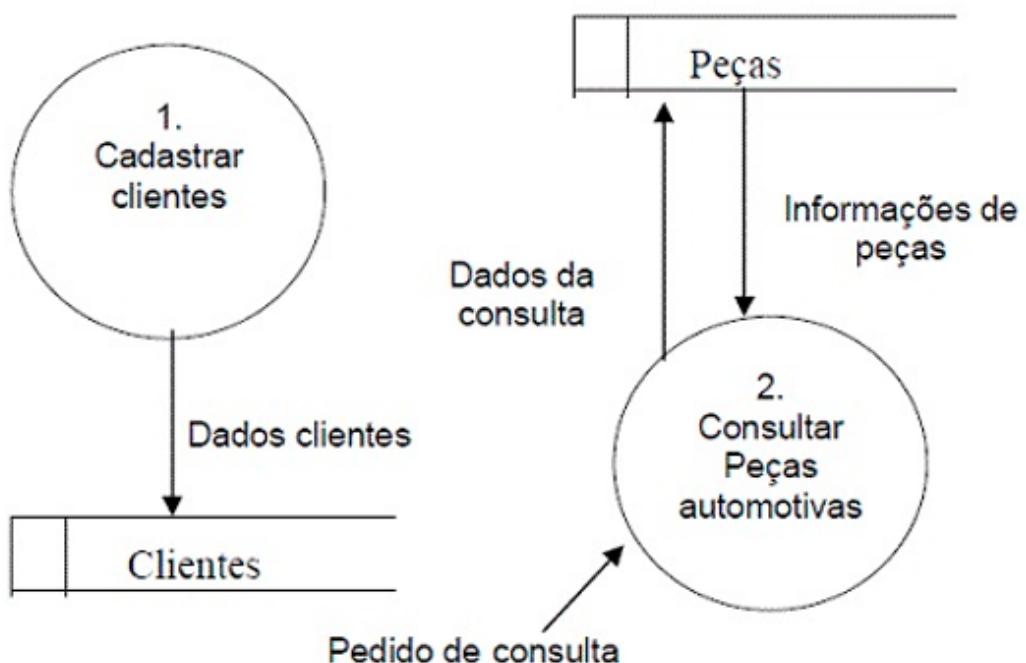
FIGURA 41 - DIAGRAMA DE CONTEXTO



FONTE: MACORATTI. Diagramação de software – D.F.D – II. Disponível em: <http://www.macoratti.net/13/08/net_dfd2.htm>. Acesso em: 22 out. 2016.

"Já após o Diagrama de Contexto é criado o DFD nível 0, acrescentando-se o detalhamento de processos, que operam sobre os fluxos de dados e os depósitos de dados" (MACORATTI, 2016, s.p.). Trazem a visão mais alta das funcionalidades do sistema e suas interfaces entre essas funções.

FIGURA 42 - DFD NÍVEL 0

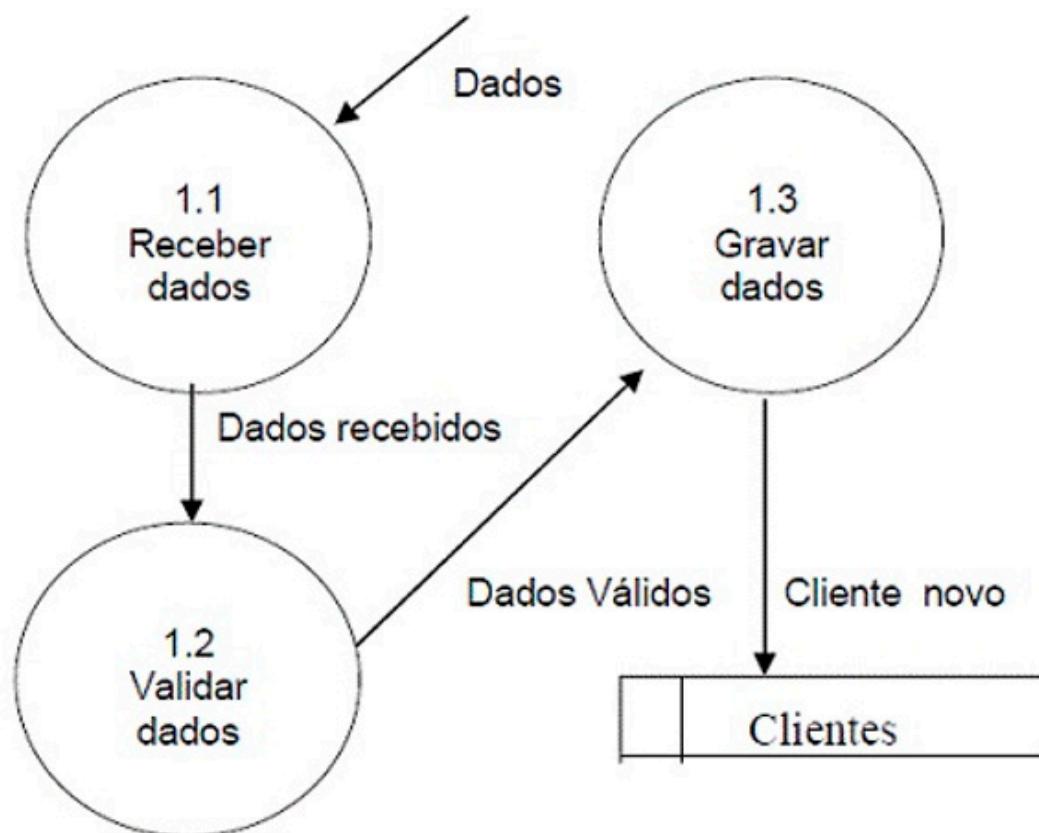


FONTE: MACORATTI. Diagramação de software – D.F.D – II. Disponível em: <http://www.macoratti.net/13/08/net_dfd2.htm>. Acesso em: 22 out. 2016.

Dando continuidade ao assunto, Macoratti (2016, s.p.) informa que:

O próximo nível de detalhamento do DFD nível 0 chama-se DFD nível 1, que apresenta detalhes das funções ainda maior que seu nível superior. A numeração das bolhas depende da numeração da bolha do DFD de nível imediatamente superior, que descreve o relacionamento com tal bolha, esse tipo de prática melhora a identificação e organização das funções do sistema.

FIGURA 43 - DFD NÍVEL 1 – EXEMPLO DO CADASTRAR CLIENTES – DO DFD NÍVEL 0



FONTE: MACORATTI. Diagramação de software - D.F.D – II. Disponível em: <http://www.macoratti.net/13/08/net_dfd2.htm>. Acesso em: 22 out. 2016.

2.2 DICIONÁRIO DE DADOS (DD)

A segunda técnica da Análise Estruturada é o Dicionário de Dados (DD), que descreve com precisão uma lista organizada de todos os elementos de dados que são pertinentes a um sistema, descrevendo as entradas, saídas, composição de depósitos de dados e alguns cálculos intermediários.

No que se refere às características de um DD, Oliveira (2002) contextualiza que o conteúdo de um DD é composto pela especificação dos fluxos de dados, especificação de arquivos e especificação de processos. O seu conteúdo deve ser preciso, conciso e não redundante e cada ocorrência deve contemplar, pelo menos, os seguintes aspectos:

1. Nome, e.g., detalhes_de_quarto;
2. Tipo, e.g., fluxo;
3. Descrição, e.g., conjunto de dados que caracterizam um quarto;
4. Pseudónimos (outros nomes possíveis)
5. Especificação, e.g., detalhes_de_quarto = no + tipo + estado + extensão_telefonica + ([no_de_camas | lotação]);
6. Comentários significativos, que poderão incluir volume, frequência, política de partilha, segurança dos dados (OLIVEIRA, 2002, p. 2)

A seguir Macoratti (2016) nos apresenta alguns símbolos utilizados no Dicionário de dados e seus respectivos significados.

QUADRO 10 - SÍMBOLOS DO DICIONÁRIO DE DADOS

Símbolo	Significado
=	É composto de
+	E
[]	Escolha uma das opções alternativas
{}	Interação de
0	Opcional (pode estar presente ou ausente)
/	Separa opções alternativas na construção[]
* *	Comentário
@	Identificador (campo chave) de um deposito

FONTE: Disponível em: <http://www.macoratti.net/13/08/net_dfd2.htm>. Acesso em: 22 out. 2016.

Para entendermos melhor como elaborar um DD você poderá utilizar o Microsoft Excel para criar as tabelas do DD para concentrar as informações.

QUADRO 11 - DICIONÁRIO DE DADOS DE UM SISTEMA DE INFORMÁTICA

Entidade: Cliente				
Atributo	Classe	Domínio	Tamanho	Descrição
Codigo_cliente	Determinante	Numérico		
Nome	Simples	Texto	50	
Telefone	Multivalorado	Texto	50	Valores sem as máscaras de entrada
Cidade	Simples	Texto	50	
data_nascimento	Simples	Data		Formato dd/mm/aaaa

FONTE: LUIS.BLOG. Disponível em: <<http://www.luis.blog.br/dicionario-de-dados.aspx>>. Acesso em: 22 out. 2016.

As informações que fazem parte do DD apresentadas no Quadro 2 são:

- **Entidade:** é o nome da entidade que foi definida no DER. A entidade é uma pessoa, objeto ou lugar que será considerada como objeto pelo qual temos interesse em guardar informações a seu respeito.
- **Atributo:** os atributos são as características da entidade cliente que desejamos guardar.
- **Classe:** a classe pode ser: simples, composta, multivalorada e determinante. Simples indica um atributo normal. Composta indica que ela poderá ser dividida em outros atributos, como por exemplo, o endereço. Multivalora é quando o valor do atributo poderá não ser único e determinante. É um atributo que será usado como chave, como CPF, Código do cliente etc.
- **Domínio:** pode ser numérico, texto, data e booleano. Podemos chamar também de tipo do valor que o atributo irá receber. A definição desses tipos deve seguir um processo lógico, exemplo: nome é texto, salário é numérico, data de nascimento é data e assim por diante.
- **Tamanho:** define a quantidade de caracteres que serão necessários para armazenar o seu conteúdo. Geralmente, o tamanho é definido apenas para atributos de domínio texto.
- **Descrição:** é opcional e pode ser usado para descrever o que é aquele atributo ou dar informações adicionais que possam ser usadas futuramente pelo analista ou programador do sistema.

FONTE: Disponível em: <<http://www.luis.blog.br/dicionario-de-dados.aspx>>. Acesso em: 22 out. 2016.

2.3 DIAGRAMA DE ENTIDADE E RELACIONAMENTO (DER)

O Diagrama de Entidade e Relacionamento (DER) tem o objetivo de mostrar a relação entre os dados expressando de forma global a estrutura de um banco de dados (BD). “Desenvolvido a fim de facilitar o projeto de bancos de dados, permitindo a especificação de um esquema da estrutura lógica global do BD” (RODRIGUES, 2016, s.p.).

Segundo Asselvi (2004), o Diagrama Entidade Relacionamento (DER) representa o modelo de dados de um sistema. Ele serve para demonstrar como os

dados de um sistema se relacionam entre si. As notações criadas para este diagrama consistem basicamente dos conceitos de entidade, relacionamento, cardinalidade e atributos.

ENTIDADE:

É todo objeto sobre o qual serão armazenadas informações. Uma entidade possui existência própria e dados a seu respeito. Para identificar-se uma entidade, três perguntas básicas devem ser feitas sobre o objeto em análise:

- Existe mais de uma unidade do objeto?
- Objeto possui dados?
- Objeto é identificado por um ou mais dos seus próprios dados?

Se a primeira pergunta for respondida com um "não", com certeza o objeto não é uma entidade, visto que para caracterizar-se uma entidade é preciso haver mais de uma ocorrência do mesmo objeto. Por exemplo, dificilmente cada setor ou departamento de uma empresa é considerado como uma entidade, uma vez que normalmente não existe mais de um setor ou departamento com o mesmo nome dentro da empresa. Assim, Setor de Compras ou Contabilidade não são entidades no modelo de dados do sistema.

Se a segunda pergunta for respondida com um "não", com certeza o objeto não é entidade, visto que não possui dados a serem armazenados sobre ele. Quando isto acontecer, deve ser verificado se o objeto é um relacionamento (já que relacionamentos podem não ter dados), ou um atributo, ou classe de dados.

Se a terceira pergunta for respondida com um "não", com certeza o objeto não é uma entidade, uma vez que a entidade possui existência própria, o que significa que possui identificador próprio. Quando isto acontecer, deve ser verificado se o objeto é um relacionamento, ou um atributo, ou classe de dados.

Existem quatro tipos de entidades, de acordo com seus relacionamentos e ligações com os negócios da empresa. Estes quatro tipos de entidades são: entidade fundamental; entidade essencial; entidade associativa; e entidade atributiva.

Entidade fundamental é aquela que contém dados fundamentais, ou seja, alimentadores ou resultantes das transações de negócios da empresa. Exemplos: nota fiscal, pedido de venda, cliente, fornecedor etc.

Entidade essencial é uma entidade fundamental que tem existência efetiva antes mesmo que o negócio da empresa comece a funcionar, gerando transações. Exemplos: unidade da federação, moeda, cliente, fornecedor etc.

Entidade associativa é a entidade que é gerada através do relacionamento entre outras duas entidades. Este tipo de entidade será útil durante a fase de projeto de banco de dados. Exemplos de entidades associativas: subscrição de disciplina do aluno, disciplinas do curso, recebimento de material do fornecedor etc.

Entidade atributiva é uma entidade associativa que possui dados qualificadores, ou seja, dados que não são identificadores da entidade. Exemplos: item da nota fiscal, item do pedido de compra etc.

RELACIONAMENTO:

É a associação entre duas ou mais entidades. No caso do Modelo Entidade-Relacionamento Estendido Binário, o relacionamento ocorre necessariamente entre apenas duas entidades. Os relacionamentos podem ser de três tipos: relacionamento de um para um; relacionamento de um para muitos; e relacionamento de muitos para muitos.

- O relacionamento de um para um é aquele onde uma ocorrência de uma entidade relaciona-se com, no máximo, uma ocorrência da outra entidade e vice-versa. Exemplo: Departamento-Chefe, onde cada departamento possui um único chefe e cada chefe chefia um único departamento.
- O relacionamento de um para muitos é aquele onde uma ocorrência de uma entidade relaciona-se com, no máximo, uma ocorrência da outra entidade e esta pode relacionar-se com mais de uma (muitas) ocorrências da primeira entidade. Exemplo: Aluno-Curso, onde um aluno pode cursar um único curso, que por sua vez pode ter muitos alunos.
- O relacionamento de muitos para muitos é aquele onde uma ocorrência de uma entidade pode relacionar-se com muitas ocorrências da outra entidade e vice-versa. Exemplo: Disciplina-Aluno, onde uma disciplina pode ser cursada por muitos alunos, que podem cursar muitas disciplinas.

O tipo do relacionamento é definido em função de sua cardinalidade.

CARDINALIDADE:

É o grau do relacionamento. No Modelo Entidade Relacionamento é representado pelo grau mínimo e máximo.

- O grau mínimo significa se o relacionamento é obrigatório ou não. O relacionamento obrigatório possui cardinalidade mínima 1. O relacionamento não obrigatório possui cardinalidade mínima 0.
- O grau máximo identifica o tipo do relacionamento, mostrando quantas ocorrências de uma entidade estão relacionadas à outra entidade. A cardinalidade máxima 1 identifica que uma ocorrência de uma entidade se relaciona com, no máximo, uma ocorrência da outra entidade. A cardinalidade

máxima N (ou, muitos) identifica que uma ocorrência de uma entidade pode relacionar-se com muitas ocorrências de outra entidade.

A cardinalidade é medida nos dois sentidos de um relacionamento. Exemplo: no relacionamento Aluno-Curso, existe uma cardinalidade mínima e máxima para a relação entre aluno e curso (aluno faz curso) e outra cardinalidade mínima e máxima para a relação entre curso e aluno (curso é cursado pelo aluno). Neste exemplo, um aluno faz no mínimo um e no máximo um curso, e um curso é cursado por no mínimo zero e no máximo muitos alunos.

ATRIBUTO:

É a decomposição da entidade em seus dados. A associação de dados de uma mesma entidade pode ser chamada de classe de dados. Existem vários tipos de atributos, em função de sua existência dentro da entidade. Estes tipos são:

- **Chave:** é o atributo que identifica uma entidade.
- **Obrigatório:** é o atributo que deve existir para todos as ocorrências (registros ou tuplas) de uma entidade.
- **Facultativo:** é o atributo que pode ser omitido.
- **Classificador:** é o atributo obrigatório, que distingue a entidade em dois ou mais tipos. Exemplo: Sexo (M masculino, F – feminino).
- Derivado:** é o atributo que é definido como o resultado de uma operação matemática sobre outros atributos da entidade.

Os atributos identificadores, ou chaves, também podem ser de vários tipos:

- **Chave primária:** é o principal mecanismo de acesso a uma entidade.
- **Chave candidata:** é todo atributo, ou conjunto de atributos, capaz de identificar uma única ocorrência em um conjunto de entidades do mesmo tipo.
- **Chave secundária:** é toda chave candidata que não é chave primária.
- **Chave estrangeira ou transposta:** é a chave primária de uma entidade que passa para outra para fazer a relação entre duas entidades.

Os símbolos que podem ser utilizados para demonstrar os elementos do Diagrama Entidade-Relacionamento, segundo as notações citadas no início desta seção, estão representados na Figura a seguir.

FIGURA 44 - NOTAÇÃO DO DIAGRAMA ENTIDADE-RELACIONAMENTO

	James Martin	Peter Chen	Merise	MEREB
Entidade				
Relacionamento				
Cardinalidade	Minimo 0, máximo 1 Minimo 0, máximo N Minimo 1, máximo 1 Minimo 1, máximo N 	Minimo 0, máximo 1 Minimo 0, máximo N Minimo 1, máximo 1 Minimo 1, máximo N 	Minimo 0, máximo 1 Minimo 0, máximo N Minimo 1, máximo 1 Minimo 1, máximo N 	Minimo 0, máximo 1 Minimo 0, máximo N Minimo 1, máximo 1 Minimo 1, máximo N

FONTE: ASSELVI. **Apostila análise de sistemas**. 2004. Disponível em: <<http://docsslide.com.br/documents/apostila-de-analise-sistemas.html>>. Acesso em: 22 out. 2016.

Uma extensão possível de ser feita ao Diagrama Entidade Relacionamento é a representação da dependência de existência. Uma entidade é existencialmente dependente de outra quando ela necessitar da chave primária da outra entidade para identificar uma única ocorrência sua. Neste caso, a entidade existencialmente dependente é chamada de entidade fraca e aquela de quem ela depende, de entidade forte.

O(s) atributo(s) da entidade fraca que servem para identificar uma única ocorrência da entidade para uma mesma chave primária da entidade forte é(são) chamado(s) de discriminador(es).

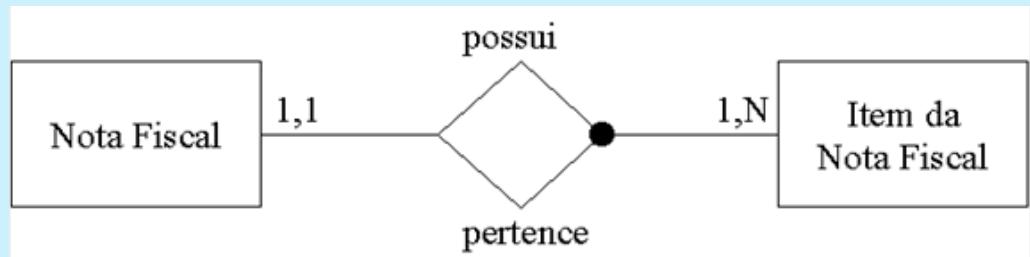
O relacionamento onde há dependência de existência no Diagrama Entidade-Relacionamento é representado com um círculo cheio na extremidade do losango do relacionamento, voltado para o lado da entidade fraca.

Como exemplo, podemos citar o relacionamento entre as entidades Nota Fiscal e Item da Nota Fiscal. O Item da Nota Fiscal é existencialmente dependente da Nota Fiscal, uma vez que não existe item sem que a nota exista.

Em termos de chaves, não é possível identificar um único item no conjunto de itens de nota fiscal apenas usando-se o número do item (por exemplo, haverá vários itens de nota fiscal numerados como item 1). Porém, é possível identificar

um único item de nota fiscal, utilizando-se o número do item, dentro de uma única nota fiscal (por exemplo, não há mais de um item, dentro de uma mesma nota fiscal, cujo número seja 1). Desta forma, o número do item é o discriminador e a chave primária é composta do número da nota fiscal (chave primária da entidade forte) e o número do item (discriminador da entidade fraca). A figura a seguir mostra a representação deste exemplo.

FIGURA 45 - EXEMPLO DE REPRESENTAÇÃO DE DEPENDÊNCIA DE EXISTÊNCIA NO DER

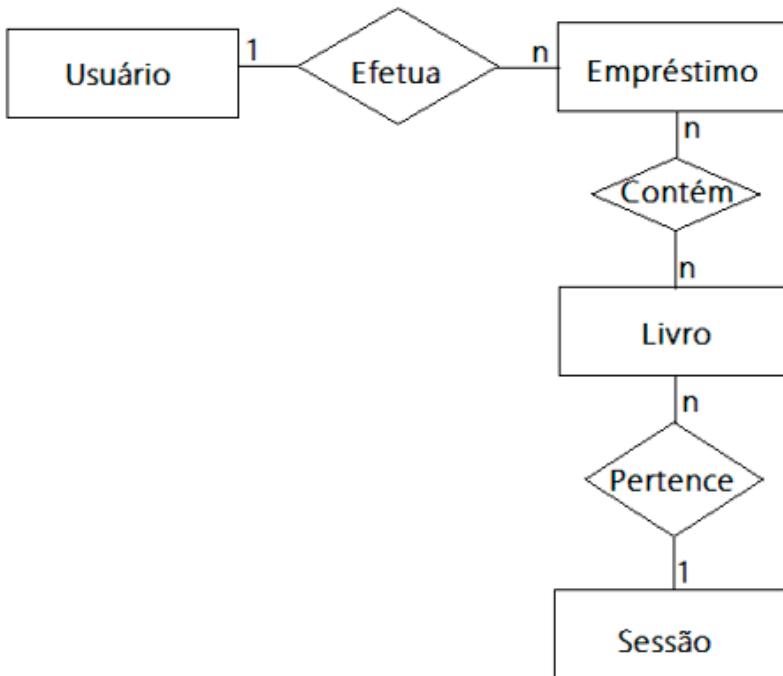


FONTE: ASSELVI. *Apostila análise de sistemas*. 2004. Disponível em: <<http://docslide.com.br/documents/apostila-de-analise-sistemas.html>>. Acesso em: 22 out. 2016.

A seguir, um exemplo de um DER para modelar um sistema de bibliotecas, focando especificamente no empréstimo de livros apresentado por Joel Rodrigues (2016):

Sabe-se que as entidades físicas existentes no sistema são o Usuário da biblioteca e o Livro que será emprestado. Se considerar que o livro pertence a uma sessão para a organização das obras do acervo e mais a entidade lógica Empréstimo com relacionamento com o usuário e com o livro. Então o esboço do DER de forma simples com as suas principais entidades e relacionamento do sistema de biblioteca fica conforme a figura a seguir.

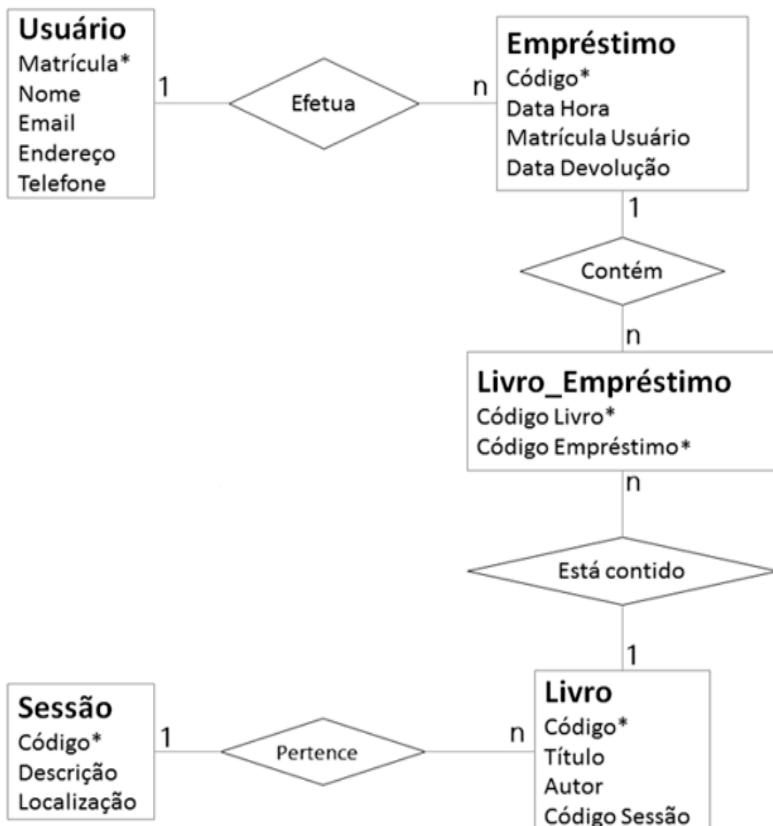
FIGURA 46 - PRIMEIRO DER DE UM SISTEMA PARA BIBLIOTECA



FONTE: Disponível em: <<http://www.devmedia.com.br/modelo-entidade-relacionamento-mer-e-diagrama-entidade-relacionamento-der/14332>>. Acesso em: 22 out. 2016.

Para Rodrigues (2016), neste primeiro diagrama as entidades fortes são o Usuário, Livro e Sessão e a entidade fraca é o empréstimo. Os relacionamentos são um Usuário efetua vários Empréstimos, vários Empréstimos contêm vários Livros, vários Livros pertencem a uma Sessão.

FIGURA 47 - PRIMEIRO DER DE UM SISTEMA PARA BIBLIOTECA



FONTE: Disponível em: <<http://www.devmedia.com.br/modelo-entidade-relacionamento-mer-e-diagrama-entidade-relacionamento-der/14332>>. Acesso em: 22 out. 2016.

Segundo Rodrigues (2016, s.p.):

Especificamos os atributos de cada entidade e marcamos algumas com um asterisco, indicando que aquela é a chave primária da tabela, ou seja, um atributo único, que nunca poderá se repetir entre as entidades do mesmo tipo. Note que neste momento ainda não é necessário especificar o tipo de cada atributo (texto, número, data etc.) [...]. Surgiu a entidade associativa **Livro_Empréstimo**, que representa os livros contidos em um empréstimo (considerando um empréstimo contém vários livros e um livro pode estar contido em vários empréstimos). Esta entidade é composta pelas chaves das duas entidades principais. Se fosse necessário, nesta entidade também poderíamos adicionar informações complementares como quantidade (não se aplica neste caso, mas caberia em um sistema de vendas, por exemplo) e observações sobre o item. Na entidade associativa, o relacionamento n..n foi dividido em dois relacionamentos do tipo 1..n, agora lidos da seguinte forma: um empréstimo contém vários itens, mas um item só pode estar contido em um único empréstimo (restrito pelas chaves primárias); um livro pode estar contido em vários itens de empréstimo (ser emprestado várias vezes), mas cada item refere-se a um único livro.

3 MÉTRICAS DE PROCESSO DE SOFTWARE

Métricas de *software* servem para apresentar medidas, preferencialmente quantitativas, que refletem características específicas de processos e de produtos em construção, podendo ser utilizadas em diferentes dimensões, como esforço, tamanho, complexidade, entre outras. A coleta adequada de métricas, com suas respectivas análises, pode auxiliar o engenheiro de *software* na tomada de decisões ao longo do desenvolvimento de um projeto, visando à melhoria da qualidade do processo e do produto em construção (CHAVES, 2016, s.p.).

Segundo Macêdo (2016, s. p.),

métricas do processo e do projeto de *software* são medidas quantitativas que permitem ao pessoal de *software* ter ideia da eficácia do processo de *software*. Indicadores de projeto permitem à organização de engenharia de *software* ter ideia da eficácia de um determinado processo existente, enquanto os indicadores de processo tentam identificar problemas que atingem a produção de todos os projetos na empresa.

FIGURA 48 - MÉTRICA DE PROCESSO DE SOFTWARE



FONTE: MÉTRICAS. Disponível em: <<http://startupsorocaba.com/wp-content/uploads/2014/08/m%C3%A9tricas-digitais.jpg>>. Acessado em: 23 out. 2016.

Segundo Roger Pressman e Maxim (2016, s.p.):

Medições permitem que gerentes e profissionais melhorem o processo de *software*: ajudam no planejamento, acompanhamento e controle dos projetos de *software*; e avaliam a qualidade do artefato de *software* produzido. Medições de atributos específicos de processo, projeto e produto são usadas para calcular as métricas de *software*. Essas métricas podem ser analisadas para fornecer indicadores que guiam a gerência e as ações técnicas.

No que se refere a métricas para qualidade de *software*, Perboni (2013, s.p.), nos diz que podemos utilizar métricas para estimativa, previsão e avaliação.

A estimativa normalmente está relacionada ao cálculo dos recursos necessários com base em outros fatores – normalmente tamanho e produtividade – para fins de planejamento. A previsão é um pouco diferente da estimativa e normalmente indica o cálculo do valor futuro de algum fator com base em seu valor atual, e outros fatores de influência. Por exemplo, com uma amostra dos dados do desempenho, é possível saber (prever) como o sistema funcionará com carga total, ou em uma configuração de recursos restritos. Já avaliação é usada para estabelecer a posição atual para comparar com um limite ou identificação de tendências, ou para comparação entre alternativas, ou como base da estimativa ou previsão.

Já para Pressman e Maxim (2016, p 586):

As métricas de processo permitem que a organização tenha uma visão estratégica, fornecendo informações sobre a eficiência de um processo de *software*. Métricas de projeto são táticas. Elas permitem que o gerente de projeto adapte o fluxo de trabalho do projeto e a abordagem técnica em tempo real.

Métrica é um conjunto de medidas. Medição existe em qualquer processo de construção de qualquer coisa. A medição é realizada não apenas na engenharia de *software*. É fundamental para qualquer atividade, principalmente de engenharia. Seu propósito é avaliar alguma coisa. A partir dela, podemos ter o entendimento da eficácia de algumas situações, como do processo de *software* (MACÊDO, 2016, s.p.).

Segue explicando Macêdo (2016, s.p.), que:

A medição além de ajudar na avaliação do processo de *software*, ajuda ainda nas estimativas, por exemplo, para estimar quanto tempo é necessário para a produção de um sistema. Atualmente, erra-se muito nessas estimativas por não se ter muito conhecimento ou medição do processo. Com a medição, aperfeiçoamentos reais podem ser conseguidos ao longo do tempo. Então, as razões para medir processos, produtos e recursos de *software* podem ser: para caracterizar, para avaliar, para prever; e para aperfeiçoar.

“Um engenheiro de *software* realiza medidas e desenvolve métricas de modo a obter indicadores. Medida é um valor real, quantidade, dimensão, capacidade ou tamanho de algum atributo” (MACÊDO, 2016, s.p.).

Podemos considerar como medidas diretas do processo de engenharia de *software* o custo e o esforço aplicados no desenvolvimento e manutenção do *software* e do produto, a quantidade de linhas de código produzidas e o total de defeitos registrados durante um determinado período de tempo. Porém, a qualidade e a funcionalidade do *software* ou a sua capacidade de manutenção são mais difíceis de ser avaliadas e só podem ser medidas de forma indireta (CORDEIRO, 2016, s.p.).

FIGURA 49 - MÉTRICAS NA ENGENHARIA DE SOFTWARE



FONTE: METRICS. Disponível em: <<https://i.ytimg.com/vi/RxZaZW2UgxA/hqdefault.jpg>>. Acessível em: 23 out. 2016.

Para Macêdo (2016), na engenharia de *software* existem dois tipos de medidas, as indiretas e as diretas, sendo elas: (1) medidas diretas: custo, esforço, linhas de código, velocidade de execução, memória, número de erros. (2) medidas indiretas: funcionalidade, qualidade, complexidade, eficiência e confiabilidade.

Uma métrica pode ser entendida como “a relação entre duas medidas de grandezas iguais ou diferentes. Um exemplo seria o número de defeitos identificados em um lote de produtos finalizados (defeitos [número] / total do lote [número])” (GOMES, 2016, s.p.).

Para Chaves (2016, s.p.):

Existem dois tipos de métricas no contexto de desenvolvimento de produtos de *software*: as métricas diretas, que são realizadas em termos de atributos observáveis, como por exemplo, esforço, tamanho e custo, e as métricas indiretas ou derivadas, que podem ser obtidas através de outras métricas, como por exemplo, complexidade, confiabilidade, e facilidade de manutenção. Quanto ao contexto, podem ser aplicadas em produtos ou em processos. Quando as métricas incidem diretamente no produto de *software*, são chamadas de métricas de predição, quando em processos de *software*, são comumente chamadas de métricas de controle e sua aplicação normalmente é realizada em processos já maduros e controlados.

Já Perboni (2013, s.p.) classifica métricas de *software* em três categorias: métricas de produto, métricas de processos e métricas do projeto. Métricas de produto descrevem as características do produto, tais como tamanho, complexidade, desempenho e qualidade. Métricas de processo podem ser utilizadas para melhorar o desenvolvimento e manutenção de *software*.

Exemplos incluem a eficácia da correção de *bugs* durante o desenvolvimento, o padrão de submissão de erros e o tempo de resposta para corrigi-los. Métricas de projeto descrevem as características do projeto e sua execução. Exemplos incluem o número de desenvolvedores, a variação da equipe ao longo do ciclo de vida do projeto, custos, cronograma e produtividade.

A medição pode ser aplicada ao processo de *software*, com o objetivo de melhorá-lo continuamente.

Métricas do processo e do projeto de *software* são medidas quantitativas que permite ao pessoal de *software* ter ideia da eficácia do processo de *software*. Com a medição, aperfeiçoamentos reais podem ser conseguidos ao longo do tempo. Então, as razões para medir processos, produtos e recursos de *software* podem ser para caracterizar; para avaliar; para prever; para aperfeiçoar (MACÊDO, 2016, s.p.).

RESUMO DO TÓPICO 1

Neste tópico, você aprendeu que:

- A análise estruturada é um conjunto de técnicas e ferramentas utilizadas para análise e definição de sistemas, serve para construção de um modelo utilizando técnicas de representação gráfica, formada por símbolos que permitem criar modelos de fluxo de informação.
- A análise estruturada objetiva facilitar a comunicação entre o usuário, analistas e projetistas produzindo uma especificação de sistema rotativa e melhorada.
- As principais ferramentas da modelagem da análise Estruturada são o Diagrama de Fluxo de Dados (DFD), que ilustra as funções que o sistema deve desempenhar, o Dicionário de Dados (DD), que descreve com precisão o significado dos componentes e das ligações existentes nos modelos de representação gráficas, e o Diagrama de Entidade e Relacionamento (DER).
- A história da Análise Estruturada, como Metodologia de Desenvolvimento de Sistemas, se confunde com a do Diagrama de Fluxo de Dados, pois esta técnica caracteriza todo o desenvolvimento estruturado de sistemas.
- O DFD é uma das principais ferramentas para modelagem de fluxo de dados através da representação de processos que usam e geram dados. Retrata o sistema em termos de suas partes componentes, com todas as interfaces entre os componentes indicados. É uma representação gráfica baseado apenas em quatro símbolos/elementos:
 - Entidade externa: de onde partem ou para onde chegam os dados (origem e destino).
 - Fluxo de dados: indicam os dados e o caminho por onde percorrem no sistema.
 - Processo ou função: ponto de processamento dos dados (tratamento, alteração ou manuseio de dados).
 - Depósito de dados: indica qualquer tipo de armazenamento de dados.
- Dicionário de Dados (DD) que descreve com precisão uma lista organizada de todos os elementos de dados que são pertinentes a um sistema, descrevendo as entradas, saídas, composição de depósitos de dados e alguns cálculos intermediários.

- O Diagrama de Entidade e Relacionamento (DER) tem o objetivo de mostrar a relação entre os dados expressando de forma global a estrutura de um banco de dados (BD). Desenvolvido a fim de facilitar o projeto de bancos de dados permitindo a especificação de um esquema da estrutura lógica global do BD e as notações criadas para este diagrama consistem basicamente dos conceitos de entidade, relacionamento, cardinalidade e atributos.
- Métricas de *software* servem para apresentar medidas, preferencialmente quantitativas, que refletem características específicas de processos e de produtos em construção, podendo ser utilizadas em diferentes dimensões, como esforço, tamanho, complexidade, entre outras (CHAVES, 2016, s.p.)
- Métricas do processo e do projeto de *software* são medidas quantitativas que permitem ao pessoal de *software* ter ideia da eficácia do processo de *software*. Indicadores de projeto permitem à organização de engenharia de *software* ter ideia da eficácia de um determinado processo existente, enquanto os indicadores de processo tentam identificar problemas que atingem a produção de todos os projetos na empresa (MACÊDO, 2016, s.p.).
- Segundo Macêdo (2016), na engenharia de *software* existem dois tipos de medidas, as indiretas e as diretas, sendo elas:
 - o Medidas Diretas: Custo, Esforço, Linhas de Código, Velocidade de Execução, Memória, Número de Erros.
 - o Medidas Indiretas: Funcionalidade, Qualidade, Complexidade, Eficiência e Confiabilidade.
- Perboni (2013) classifica Métricas de *software* em três categorias:
 - o Métricas de produto: descrevem as características do produto, tais como tamanho, complexidade, desempenho e qualidade.
 - o Métricas de processo: podem ser utilizadas para melhorar o desenvolvimento e manutenção de *software*.
 - o Métricas de projeto: descrevem as características do projeto e sua execução.

AUTOATIVIDADE



1. Defina o que é a análise estruturada.
2. Quais são os benefícios da análise estruturada?
3. Quais são as principais ferramentas para a modelagem de Análise Estruturada e qual o objetivo destas ferramentas?
4. Defina o que são entidade, relacionamento, cardinalidade e atributos notações do Diagrama Entidade Relacionamento (DER).
5. Defina o que é e para que serve Métrica de Processo de *Software*.
6. Explique que são métricas diretas e indiretas no contexto de desenvolvimento de produtos de *software*.



FUNDAMENTOS DE MELHORIA DE PROCESSO DE SOFTWARE E FERRAMENTA DE GESTÃO DE PROCESSOS DE SOFTWARE

1 INTRODUÇÃO

Para que um processo atenda de forma correta e eficiente à proposta de desenvolvimento de projetos de *software* é inevitável haver em paralelo um trabalho de gestão de processo que busque acompanhar seu desempenho a fim de obter sua melhoria contínua.

2 MELHORIA DE PROCESSO DE SOFTWARE

Segundo Moreira (2016,):

Atualmente, no mundo dos negócios, praticamente tudo é dirigido por processos. As pessoas, em sua vida cotidiana, estão acostumadas a lidar com processos, mesmo sem serem organizados. Elas se dirigem para o trabalho e fazem suas atividades diárias seguindo às vezes seus próprios procedimentos.

A melhoria de processo de *software* é uma ação executada para avaliar e melhorar as operações e atividades internas ou externas durante o desenvolvimento de *software*. Nota-se que é comum as organizações não aproveitarem seus recursos e competências de forma correta para aumentar continuamente sua produtividade atendendo da melhor forma possível as necessidades de negócio e alcançar de modo efetivo, suas metas de negócio. O objetivo principal de se melhorar os processos de *software* é basicamente aumentar a capacidade desses processos de maneira contínua e incremental.

Moro (2008, s.p.) enfatiza que “um dos principais motivos para que organizações de *software* adotem uma visão de **melhoria contínua de seus processos de software**, é o fato da qualidade do produto final depender diretamente da qualidade do processo de desenvolvimento adotado”. Assim, quanto maior a preocupação e o esforço para que seus processos estejam alinhados aos objetivos de negócio, maiores serão os benefícios alcançados pela organização. Além disso, a melhoria contínua dos processos faz com que a organização se posicione de uma forma bastante competitiva no mercado em que atua.

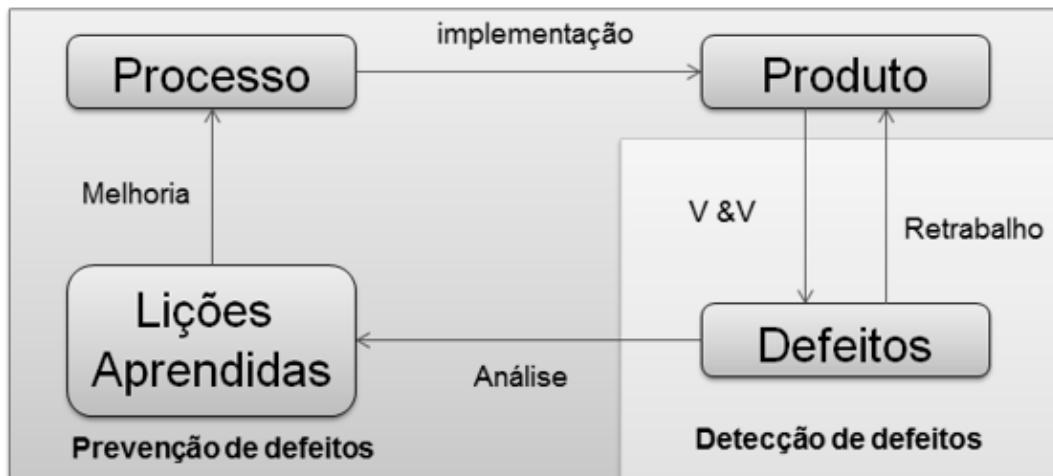
Processo de *software* é importante porque fornece estabilidade, controle e organização para uma atividade que pode, se deixada sem controle, tornar-se bastante caótica.

Segundo Paula Filho (2000, p. 20):

Em organizações com baixa maturidade de capacitação em *software*, os processos geralmente são informais. Processos informais existem apenas na cabeça de seus praticantes. Geralmente são processos individuais. Podem ser parcialmente transferidos para outras pessoas, por transmissão oral e por imitação. Por outro lado, um processo definido tem documentação que detalha todos os seus aspectos importantes: o que é feito, quando, por quem, as coisas que usa e as coisas que produz.

Para produzir um produto de *software* com qualidade deve-se possuir processos formais que visem à prevenção e detecção de defeitos durante o desenvolvimento de *software*. A origem do produto se dá pela implementação de um processo consistente e em constante melhoria contínua.

FIGURA 50 – VALIDAÇÃO DO DESENVOLVIMENTO DE SOFTWARE



FONTE: O autor

Várias técnicas são utilizadas para identificar defeitos nos produtos de trabalho. Esses defeitos são eliminados através de retrabalho, que têm efeito imediato na produtividade do projeto.

Defeitos também são encontrados em atividades de teste e podem ser analisados, a fim de se identificar suas causas. A partir dessa análise, lições aprendidas podem ser usadas para criar futuros produtos e prevenir futuros defeitos e, dessa forma, ter impacto positivo na qualidade do produto e na produtividade do projeto. (VASCONCELOS, 2006, p. 81).

Todo processo de *software* deve possuir junto ao plano de projeto uma documentação específica da qualidade, denominada como plano de qualidade que deve compreender informações sobre como a equipe de qualidade irá garantir o cumprimento da política de qualidade, no contexto do programa ou projeto a serem desenvolvidos, quais métodos, técnicas, métricas, treinamentos e padrões devam ser utilizados ao longo de todo o ciclo de vida do projeto. O plano deve oferecer a base do gerenciamento dos riscos, dos testes, das inspeções, das auditorias e como deverão ocorrer os reportes de problemas e ações corretivas.

Pode-se citar entre tantos outros exemplos, que a técnica de prevenção de defeitos em um processo de desenvolvimento de *software* se dá pelo uso de instruções de procedimentos (padrões formais), treinamentos, documentação, modelagem e reengenharia, já as técnicas de detecção de defeitos podem ser pela análise de código; revisão por pares; testes, auditorias, verificações e validações.

Segundo Moro (2008, s.p.), os programas de melhoria de processos devem ser justificáveis através de análises de retorno do investimento.

Estas análises procuram medir, para cada unidade monetária investida, quantas unidades monetárias retornam em determinado prazo, através da redução de custos ou do aumento da renda. Existem muitas práticas de Engenharia de *Software*; os programas de melhoria devem dar prioridade às práticas com melhor retorno de investimento. Por exemplo, os dados seguintes, sustentam algumas das práticas mais prioritárias para melhoria dos processos:

- Captar um requisito correto é 50 a 200 vezes mais barato que corrigi-lo durante a implementação ou em operação. Portanto, a engenharia e a gestão dos requisitos estão entre as práticas de maior retorno de investimento.
- Fazer um desenho correto é 10 vezes mais barato que corrigi-lo durante os testes de aceitação. Portanto, o desenho tem forte impacto nos custos dos projetos, embora menos que a engenharia de requisitos.
- Refazer defeitos de requisitos, desenho e código consome 40% a 50% do custo total dos projetos. Portanto, a garantia da qualidade se paga rapidamente, na medida em que diminui a necessidade de refazer.
- Cada hora gasta em prevenção de defeitos representa de 3 a 10 horas menos de correção de defeitos. Dentre as atividades de qualidade, as atividades ligadas à prevenção de defeitos são mais eficazes que aquelas que focalizam a correção.

Para Boria (2013), o principal inimigo de uma empresa desenvolvedora de *software* é a baixa qualidade. Até hoje, ninguém desenvolveu uma proposta válida para melhorar a qualidade que não fosse relacionada à melhoria de processos, que passa a ser então a questão principal. É possível argumentar que as pessoas e as ferramentas são importantes em seu impulso para a melhoria da produtividade. Isto é verdade, mas só quando os processos estão no lugar onde se consiga o aproveitamento das condições dos indivíduos e das ferramentas de *software*.

Durante a implementação da melhoria de processos de *software*, é comum que as organizações adotem normas de qualidade e modelos de referência para apoiar suas iniciativas, utilizando-as como guias

para melhorar seus processos. Com base nos modelos, são estabelecidas melhores práticas que oferecem um caminho efetivo no alcance de uma maior maturidade. E muitas vezes, as organizações necessitam utilizar mais de uma norma ou modelo para alcançar seus objetivos de negócio. Nesse cenário, conhecer as similaridades e diferenças entre as normas e modelos pode ser relevante (MELLO, 2016, s.p.).

As organizações precisam entender a situação atual de seus processos de *software* e realizando continuamente sua avaliação, com base nos pontos fracos identificados agir e identificar melhoria. Estas ações são inicialmente implantadas em projetos selecionados, sendo acompanhadas para verificar os resultados. “Aquelas ações que forem consolidadas podem então ser institucionalizadas dentro da organização. Para guiar estas ações de melhoria e avaliações de processo de *software*, existem vários modelos, dentre os principais, o CMMI, a norma ISO/IEC 15504 e o modelo brasileiro MPS.BR” (THIRY, 2016, s.p.).

FIGURA 51 - CONTÍNUA AVAILAÇÃO



FONTE: Disponível em: <<https://3.bp.blogspot.com/-ja6qX9hmflU/VvKYGH0Odul/AAAAAAAAMw/hr6lxPdX26Q2kLs5ofxFErnu9p7Jxojg/s1600/avalia.jpg>>. Acesso em: 26 out. 2016.

A implementação de melhorias em processos de *software* é uma atividade complexa e intensa de conhecimento. Os modelos de qualidade mais difundidos nas indústrias de *software* no Brasil são o CMMI e MPS.BR.

O principal propósito do CMMI (*Capability Maturity Model Integration* ou Integração dos Modelos de Capacitação e Maturidade de Sistemas) “é fornecer diretrizes baseadas em melhores práticas para a melhoria dos processos e habilidades organizacionais, cobrindo o ciclo de vida de produtos e serviços completos, nas fases de concepção, desenvolvimento, aquisição, entrega e manutenção” (FERNANDES, 2014, p. 301).

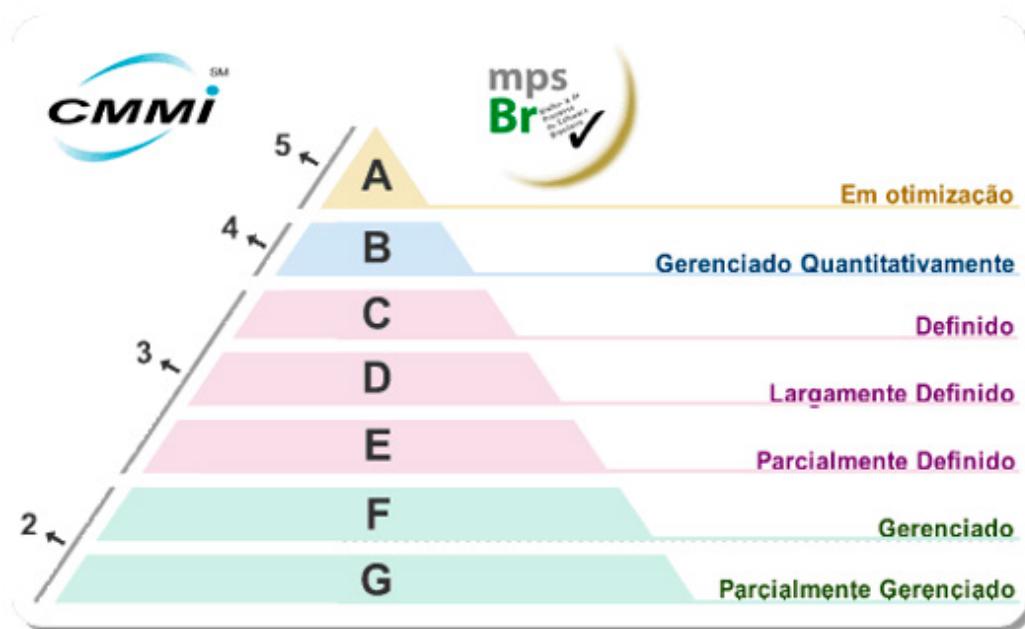
O CMMI é um dos modelos mais aceitos para a melhoria da qualidade e do processo de *software* em todo o mundo e define os princípios e práticas que devem ser aplicados a uma organização para atingir estágios evolutivos de maturidade em seu processo de *software*. Os cinco níveis de maturidade são: (1) inicial: Processo imprevisível e sem controle. (2) Repetível: processo disciplinado. (3) Definido: processo consistente e padronizado. (4) Gerenciado: processo previsível e controlado e (5) Otimização: processo aperfeiçoado continuamente.

O MPS. BR (Melhoria de Processo de *Software* Brasileiro) é um programa que foi criado para melhorar a capacidade de desenvolvimento de *software* nas empresas brasileiras voltados para médias e pequenas empresas e com baixo custo de implantação.

MPS.BR possui as seguintes metas: (1) definir e implementar o Modelo de Referência para Melhoria de Processos de *Software* (MR mps). (2) criar cursos em diversos locais do país para capacitar e formar consultores do modelo. (3) credenciar instituições e centros tecnológicos capacitados a implementar e avaliar o modelo com foco em grupo de empresas.

Os sete níveis de maturidade do MPS.Br são: (G) Parcialmente gerenciado: inicia o gerenciamento de requisitos e projetos; (F) Gerenciado: inclui medições, gerência de configurações e garantia de qualidade; (E) Parcialmente definido: inclui treinamento, adaptação de processos para gerência de projetos; (D) Largamente definido: envolve teses e integração de produto; (C) Definido: gerência de riscos; (B) Gerenciado quantitativamente: avalia o desempenho dos processos e a gerência quantitativa dos mesmos; e (A) em otimização: preocupação com a inovação e análise de causas (SOFTEX, 2012).

FIGURA 52 - NÍVEIS DE MATURIDADE DO CMMI E MPS.BR



FONTE: Disponível em: <<http://www.micreiros.com/wp-content/uploads/MPS-BR.gif>>. Acesso em: 23 out. 2016.

Segundo Franciscani (2016 apud OLIVEIRA, 2008) existem medições entre os modelos e as comparações entre eles podem ser visualizadas no quadro a seguir.

QUADRO 12 – COMPARATIVO ENTRE OS MODELOS CMMI E MPS.BR

CMMI	MPS.BR
O modelo de Qualidade CMMI é reconhecido internacionalmente.	O MPS.BR é mais conhecido nacionalmente e na América Latina.
O modelo CMMI envolve um grande custo na avaliação e certificação do Modelo.	No MPS.BR o custo da certificação é mais acessível.
No CMMI é necessário investir tempo, geralmente para se chegar aos níveis de maturidade mais altos.	No MPS.BR as avaliações são bienais.
O CMMI tem foco global voltado para empresas de maior porte.	MPS.BR é um modelo criado em função das médias e pequenas empresas.
O CMMI possui cinco níveis de maturidade por estágio e seis na contínua.	MPS.BR possui sete níveis de maturidade, onde a implementação é mais gradual.
O CMMI é aceito como maturidade para licitações.	O MPS.BR é aceito como maturidade para licitações.
O CMMI torna as empresas competitivas internacionalmente.	O MPS.BR não torna as empresas competitivas internacionalmente.
O CMMI não utiliza contrato conjunto de empresas.	No MPS.BR pode acontecer contrato cooperado em grupo de empresas que queiram a certificação.
Implementação mais complexa.	Implementação mais simples.
Desenvolvido pelo <i>Software Engineering Institute</i> – SEI em 1992.	Desenvolvido por algumas instituições Brasileiras em 2003.



Caro Acadêmico! Os estudos referentes a estas principais normas são abordados nas disciplinas de Engenharia e Projeto de Software que ocorre no primeiro ano do curso em que apresenta de forma clara e detalhada seus objetivos, etapas de execução e processo de certificação existentes hoje no mercado de trabalho.

No que diz respeito à capacitação do processo, Sodré (2016, s.p.) explica que apenas uma pequena parte possui certificação de elevada maturidade no seu processo de desenvolvimento de *software*. “Dentre as empresas com certificação CMMI, MPS.BR ou ISO, a maioria está associada a produtos, enquanto as empresas de serviços empregam métodos próprios, mas sem certificação, criando uma lacuna importante do ponto de vista de afirmação internacional”.

Para conduzir programas de melhoria de Processo de *Software*, as organizações normalmente se baseiam em modelos de referência como MPS.BR e o CMMI que são fundamentados na tríade processo-tecnologia-pessoas. O componente “tecnologia” é apoiado por ferramentas e ambientes de desenvolvimento de *software*, em relação ao elemento pessoas, o número de pesquisas ainda está aquém do esperado, pois é necessária uma melhor compreensão do elemento humano que influencia os programas de melhoria, portanto, nota-se que pesquisas adicionais sobre a influência do elemento humano em programa de melhoria de Processo de *Software* são necessárias (SANTOS, 2015).

A melhoria do processo de *software* pode ser considerada hoje uma das grandes prioridades para as organizações que trabalham com *software*. Isto se deve à exigência do mercado por produtos com maior qualidade, que sejam entregues mais rapidamente e com menor custo de desenvolvimento. Estudos apontam que ao tentarem melhorar seus processos, as empresas estão em busca de: entender as características dos processos existentes e os fatores que afetam a sua capacidade; planejar, justificar e implementar ações que modificarão os processos, tornando-os mais coerentes com as necessidades de negócios e; avaliar os impactos e benefícios ganhos, comparando-os com os custos advindos das mudanças realizadas (OLIVEIRA, 2016, s.p.).

3 FERRAMENTAS DE GESTÃO DE PROCESSOS DE SOFTWARE

Este tópico abordará os conceitos de ferramentas de gestão de processo de *software* utilizadas para dar suporte ao processo de desenvolvimento, conforme abordado até aqui, neste caderno de estudos. Estas ferramentas servem para oferecer um conjunto de serviços e atividades a fim de minimizar o tempo de execução mantendo alto nível de qualidade em projetos de *software*.



São muitas ferramentas CASE com objetivos e definições amplas no mercado de desenvolvimento de software. Ainda nesta Unidade 2 queremos apresentar somente dez (3.1 até 3.10). Porém no próximo e último tópico desta Unidade, falaremos com detalhes sobre o processo de negócio utilizando a notação BPMN, pois as dez ferramentas utilizam seus princípio.

Estas ferramentas podem ser utilizadas para a modelagem ao desenvolver diversos diagramas do projeto, para geração de scripts do banco de dados compatíveis ao projeto, para os recursos para gerenciar fontes do projeto, realizar sua documentação, testes e programação. Enfim, nesta ferramenta estão disponíveis para as organizações implementarem gestão de processos ágil freeware e de fácil utilização para desenhar, diagramar, documentar e publicar os processos utilizando o padrão BPMN (Business Process Management Notation), que fornece uma linguagem comum que permite todas as partes envolvidas para comunicar os processos de forma clara, completa e eficiente. Para aumentar produtividade pela notação BPMN devemos utilizar uma boa ferramenta.

3.1 FERRAMENTA CASE: BIZAGI MODELER

Segundo Segplan (2014, p. 6), o Bizagi Modeler permite a simulação dos fluxos de trabalho a fim de facilitar a análise de melhorias tanto em relação ao tempo quanto em relação ao custo das atividades desenvolvidas. Foi desenvolvido para modelagem descritiva, analítica e de execução, de processo de negócio utilizando a notação BPMN elaborando uma documentação rica em formatos “diferentes de arquivos, inclusive no formato *web*, visando dar maior publicidade às atividades praticadas pelas organizações que prezam pela gestão do conhecimento e pela transparência dos serviços prestados”.

O Bizagi possui um conjunto de relatórios e indicadores que permite a análise dos processos que seguem em operação em tempo real e informações para análise histórica dos processos. Estes indicadores possibilitam aos gestores do negócio e aos donos dos processos terem visibilidade sobre suas operações e identificar gargalos, medir a performance dos recursos e atender aos níveis de acordo de serviço (FERMINO, 2016, s.p.).

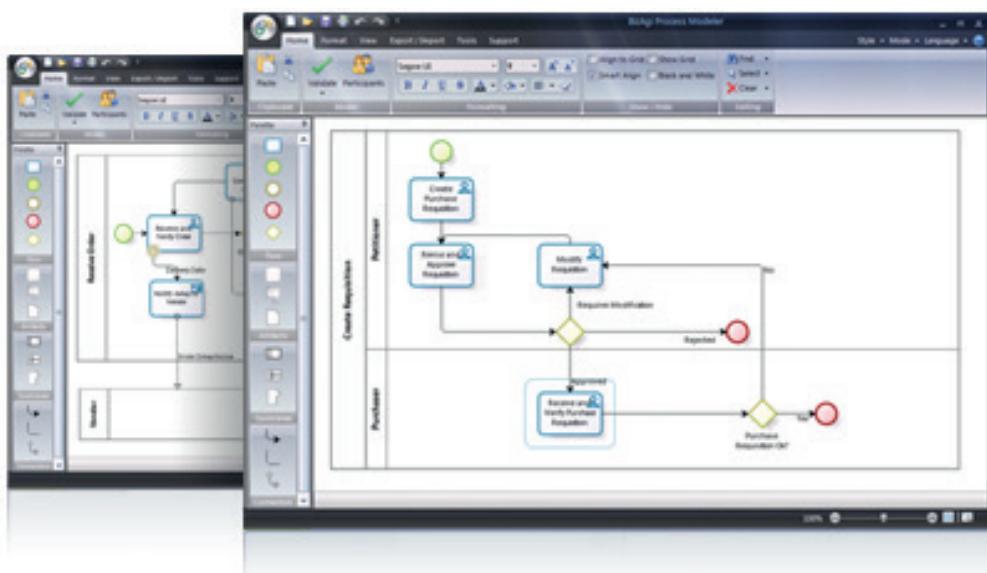


O Bizagi Modeler é um padrão mundial de mapear processos, dos mais simples aos complexos, tornando-os inteligíveis para equipes multidisciplinares. É fácil de usar e usa cores nos elementos para facilitar a identificação. Para baixar, experimentar e descobrir mais sobre esta ferramenta: <<http://www.bizagi.com/pt/produtos/bpm-suite/modeler>>. Para aprender mais sobre esta ferramenta acessar este endereço do youtube: <<https://www.youtube.com/watch?v=Bs156QQVr2I>>.

Seus processos “são retratados como um gráfico de fluxo de elementos, os quais são atividades, eventos, gateways e fluxos de sequência (conectores), que definem uma semântica finita de execução” (SEGPLAN, 2014, p. 8).

Bizagi é o *software* líder de mercado. A imagem a seguir mostra um exemplo de processo e da tela do *software*.

FIGURA 53 - FERRAMENTA BIZAGI

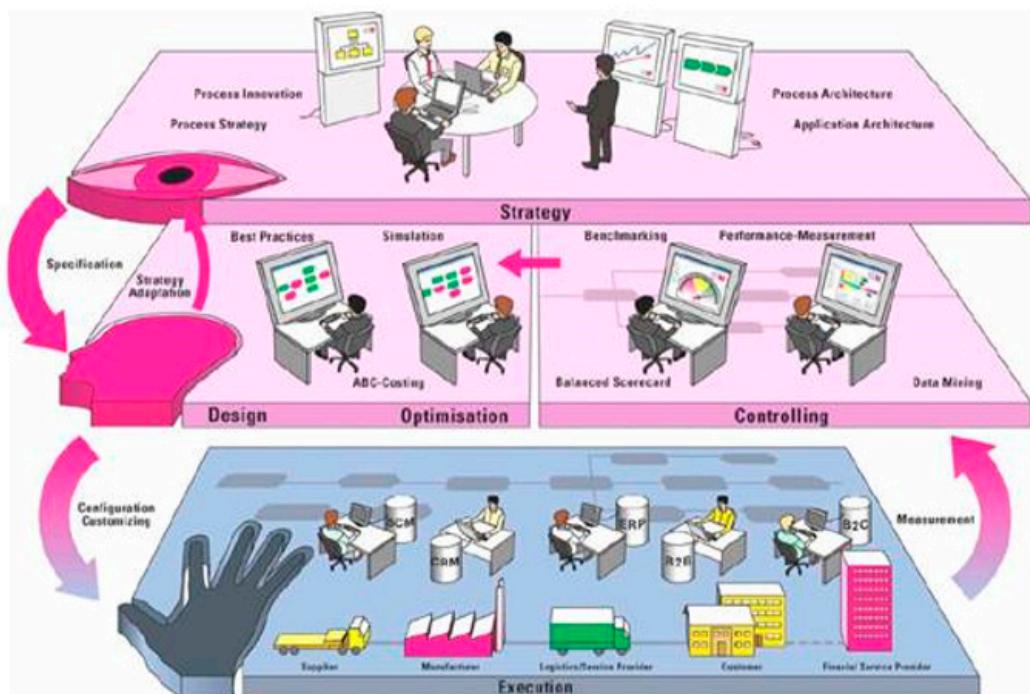


FONTE: Disponível em: <<http://ingeti.blogspot.com.br/2011/07/poderosa-ferramenta-para-modelagem-de.html>>. Acesso em: 5 nov. 2016.

3.2 FERRAMENTA CASE: ARIS EXPRESS

Segundo Castro, Oliveira e Braga (2016), o Aris Express é uma arquitetura, criada em 1984, alinhada ao conceito de BPMN, adequado, principalmente, para usuários ocasionais e novatos construída em três níveis: estratégia, especificação (composta de design, otimização e controle) e execução, como pode ser visto na figura a seguir as principais funcionalidades e benefícios da modelagem e gerenciamento de processos utilizando o ARIS Express.

FIGURA 54 - MODELAGEM ARIS EXPRESS EM TRÊS CAMADAS



FONTE: Disponível em: <<http://www.devmedia.com.br/arist-express-exemplo-pratico-de-modelagem-de-processos-revista-engenharia-de-software-magazine-39/21960>>. Acessado em: 5 nov. 2016.

O ARIS Express foi desenvolvido baseado no ARIS Method (Método de modelagem ARIS ou a forma de modelar usando ARIS) e em padrões da indústria. Possui uma interface intuitiva, é de fácil aprendizagem e pode gerar resultados instantâneos. Pode ser utilizado com a finalidade de aprendizado, em universidades. A versão disponibilizada consiste em uma versão *freeware* e não demo, o que possibilita seu uso como uma substituição razoável de outras ferramentas utilizadas para a mesma finalidade: “Desenhar” os processos das organizações (CASTRO; OLIVEIRA; BRAGA, 2016, s.p.).



Atualmente está disponível somente nos idiomas inglês e alemão.

Para baixar, experimentar e descobrir mais sobre esta ferramenta: <<http://www.ariscommunity.com/arist-express>> para fazer o download, é preciso criar uma conta na Aris Community. Para aprender mais sobre esta ferramenta acesse este endereço do youtube: <https://www.youtube.com/watch?v=dU2SKewQ_Vg>.

As características do Aris Express são:

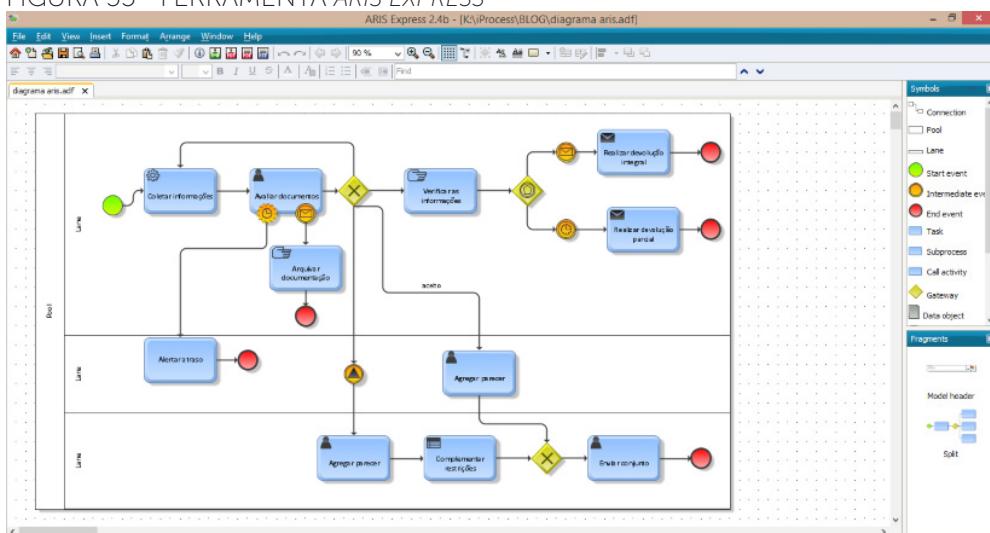
Smart Design (Design inteligente): permite aos usuários da ferramenta capturar de forma rápida e fácil informações sobre a empresa, não se preocupando com padrões de modelagem ou a colocação correta dos objetos. O modelo é gerado instantaneamente e pode ser alterado a qualquer momento.

Model Fragments (Fragmentos do modelo): os usuários devem ter cuidado com as convenções da modelagem ou fazem a mesma combinação de objetos sempre. O problema pode ser tratado com o uso de “fragmentos”, ou seja, uma combinação de objetos que podem ser reutilizados.

Context Sensitive Modeling (Modelagem sensível ao contexto): O ARIS Express possui uma espécie de ajuda em cada um dos componentes inseridos no modelo, mostrando ao usuário quais os tipos de conexões (interligações) que podem ser efetuadas usando aquele objeto (CASTRO; OLIVEIRA; BRAGA, 2016, s.p.).

Segundo Sganderla (2016, s.p.), “a versão Express desta ferramenta permite criar diagramas com notações como cadeia de valor, organograma, modelo de dados, EPC (Event-driven Process Chain) e é claro, BPMN. Porém, diferentemente da plataforma, os arquivos são gravados no próprio computador do usuário”.

FIGURA 55 - FERRAMENTA ARIS EXPRESS

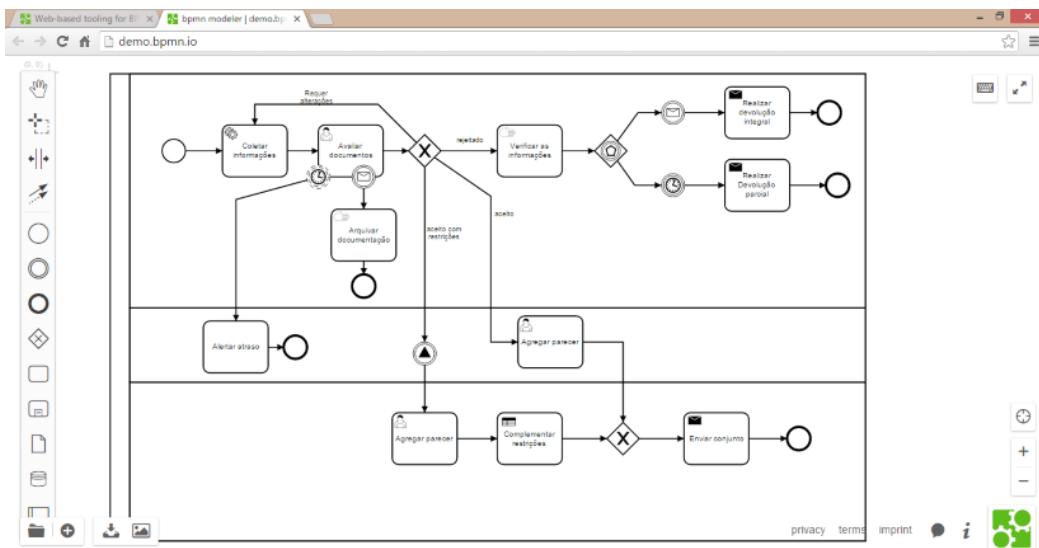


FONTE: Disponível em: <http://blog.iprocess.com.br/wp-content/uploads/2016/08/editores_bpmn_exemplo_aris_express.png>. Acesso em: 5 nov. 2016.

3.3 FERRAMENTA CASE: BPMN.io.

Segundo Fermino (2016, s.p.), o BPMN.io é uma ferramenta leve e desenvolvida pela alemã Camunda, que permite criar diagramas de processo sem precisar instalar nenhum aplicativo. “O editor é totalmente web e funciona diretamente no browser do computador. É simples e muito fácil de sair criando diagramas e possui grande aderência à notação BPMN para diagrama de processo (orquestração)”.

FIGURA 56 - FERRAMENTA BPMN.IO



FONTE: Disponível em: <<http://blog.iprocess.com.br/2016/09/7-ferramentas-gratuitas-para-criar-diagramas-de-processos-com-bpmn/>>. Acesso em: 3 nov. 2016.

Permite apenas criar diagramas gráficos, sem muita informação adicional. Não possui recursos complementares.



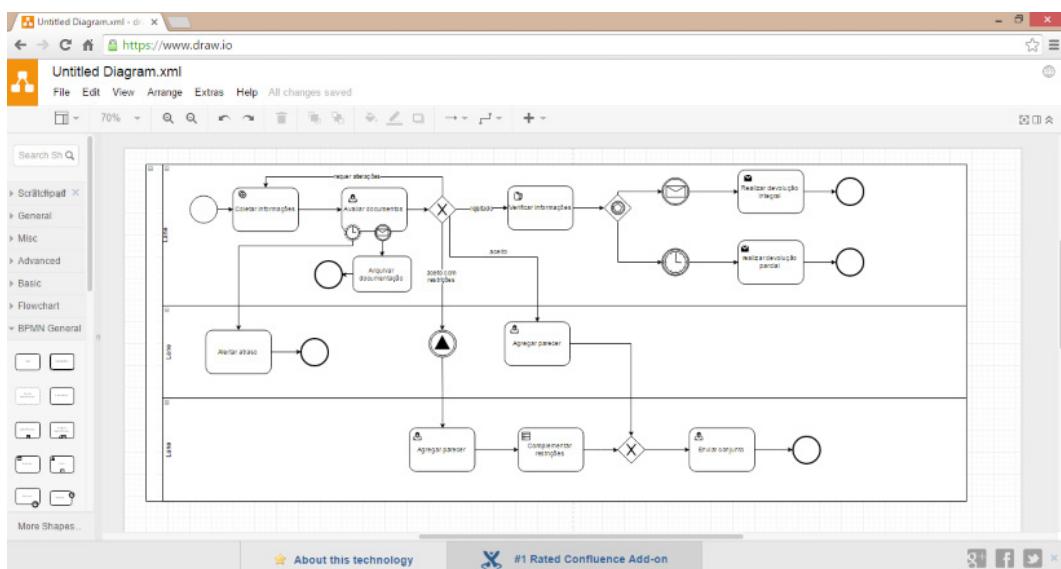
Os diagramas criados podem ser baixados para o seu computador no formato padrão .bpmn (que pode ser recarregado e editado posteriormente) e em formato de imagem PNG. Para acessar, experimentar e descobrir mais sobre esta ferramenta: <<http://bpmn.io>>. Para aprender mais sobre esta ferramenta acessar este endereço do YouTube: <<https://www.youtube.com/watch?v=PXOjuei5jt4>>.

3.4 FERRAMENTA CASE: DRAW.io.

Segundo Sganderla (2016, s.p.):

Originalmente criado como uma ferramenta online de criação de diagramas tipo flowchart, o Draw.io incorporou como uma das notações visuais para desenho a palheta de elementos de BPMN. Com isso, este editor vai na mesma linha do BPMN.io, um pouco mais rico nas funcionalidades visuais (é possível usar cores) mas com menos recursos específicos para a modelagem, como a ausência de validação. Como permite criar vários tipos de diagramas diferentes e misturar as palhetas, pode ser um pouco confuso para iniciantes discernir se estão criando um diagrama corretamente, pois é possível usar em um mesmo diagrama os tipos de elementos visuais de notações diferentes.

FIGURA 57 - FERRAMENTA DRAW.IO.



FONTE: Disponível em: <<http://blog.iprocess.com.br/2016/09/7-ferramentas-gratuitas-para-criar-diagramas-de-processos-com-bpmn/>>. Acessado em: 03 nov. 2016.

Segundo Fermino (2016, s.p.)

Os diagramas são salvos em qualquer um dos principais serviços de armazenamento na nuvem, como Google Drive, Dropbox ou OneDrive (basta ter uma conta) ou então pode ser gravado no seu próprio computador (em formato XML, que pode ser recarregado e editado posteriormente). A principal vantagem de usar um serviço de armazenamento na nuvem é que outros usuários poderão acessar o mesmo diagrama e fazer contribuições. A palheta de BPMN está espalhada em grupos como “General”, “BPMN General”, “BPMN Gateways”, “BPMN Events”. Além disso, a ferramenta possui muitos elementos “inventados” sobre a notação BPMN que não existem na especificação formal, o que pode tornar seu uso ainda mais confuso (até mesmo para analistas experientes).



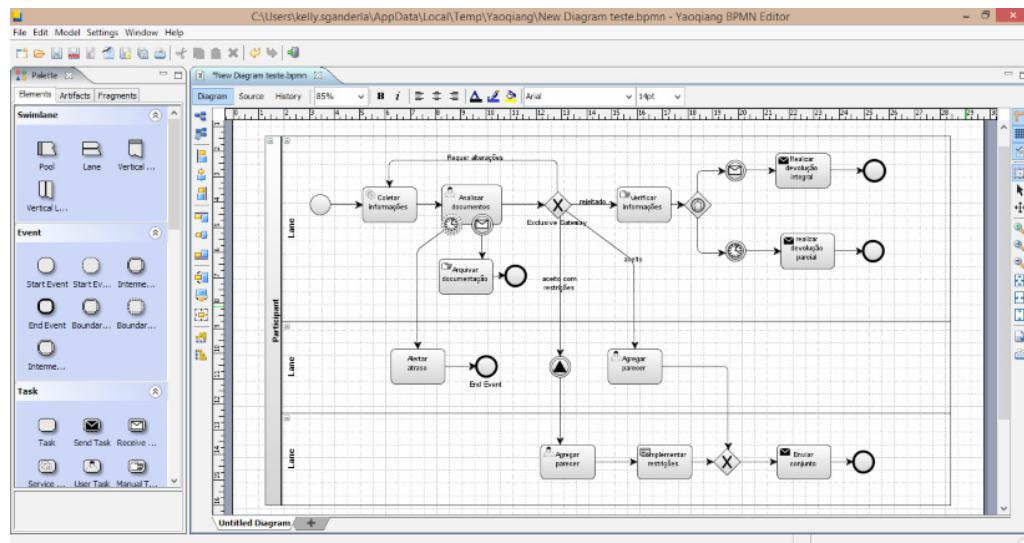
Para acessar, experimentar e descobrir mais sobre esta ferramenta: <<http://draw.io>>. Para aprender mais sobre esta ferramenta acessar este endereço do YouTube: <<https://www.youtube.com/watch?v=MXITjOpesq4>>.

3.5 FERRAMENTA CASE: YAOQIANG BPMN EDITOR

Para Fermino (2016)

Esta ferramenta de modelagem de processos desenvolvida pela Blenta Software é uma das mais completas e aderentes à notação e é *software* livre. Além de modelagem de diagrama de Processo (Orquestração), também possui todos os elementos para a criação dos dois outros tipos de diagramas da notação: conversação e coreografia (o único editor gratuito que identificamos até agora que comporta os três tipos de diagramas nativamente).

FIGURA 58 - FERRAMENTA YAOQIANG BPMN EDITOR



FONTE: SGANDERLA, Kelly. Disponível em: <<http://blog.iprocess.com.br/2016/09/7-ferramentas-gratuitas-para-criar-diagramas-de-processos-com-bpmn/>>. Acessado em: 3 nov. 2016.

Segundo Sganderla (2016, s.p.):

Um ponto forte deste editor é que a ferramenta possui um compromisso estabelecido em seguir rigidamente as definições da especificação formal da OMG para a representação gráfica dos processos, o que garante um elevado nível de aderência dos elementos e validações de regras de utilização. Embora não possua um apelo visual muito grande, em termos de funcionalidades para a criação de diagramas, é bastante completa.



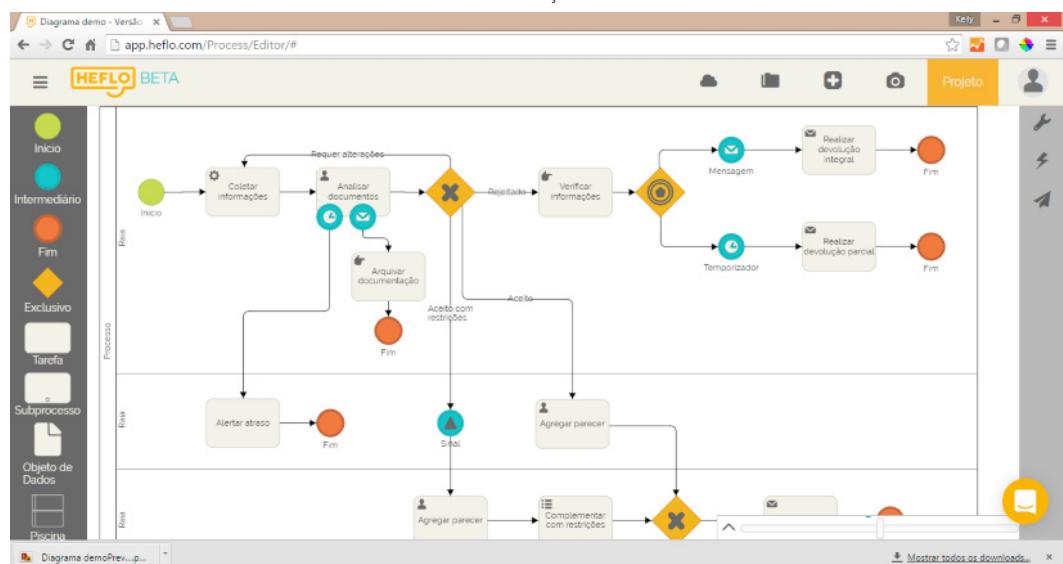
Os processos mapeados podem ser baixados para o seu computador no formato-padrão .bpmn (que pode ser recarregado e editado posteriormente) e em formato de imagem PNG. A ferramenta é em inglês (e possui outros idiomas como alemão, francês e chinês – sim, porque a empresa fabricante é chinesa!). Os diagramas criados podem ser gravados no formato padrão .bpmn e em formatos variados de imagem, como PNG, BMP e JPG. Para baixar, rodar e usar o endereço é <<https://sourceforge.net/projects/bpmn/>>. Como pré-requisito: Java Runtime Environment. Para aprender mais sobre esta ferramenta acessar este endereço do YouTube: <<https://www.youtube.com/watch?v=M3QSxZVDVfY>>.

3.6 FERRAMENTA CASE: HEFLO! DOCUMENTAÇÃO

Segundo Fermino (2016, s.p.):

Com uma interface leve e agradável, o módulo de documentação BPMN da HEFLO é on-line e permite criar diagramas de processo sem precisar instalar nenhum aplicativo. Embora o produto esteja na versão beta, a empresa se comprometeu (através de seu site) que este módulo será gratuito para sempre.

FIGURA 59 - FERRAMENTA HEFLO! DOCUMENTAÇÃO



FONTE: Disponível em: <<http://blog.iprocess.com.br/2016/09/7-ferramentas-gratuitas-para-criar-diagramas-de-processos-com-bpmn/>>. Acesso em: 3 nov. 2016.

Segundo Fermino (2016, s.p.), durante a modelagem, as raias se autoajustam conforme os elementos são adicionados.

Além da criação do diagrama, é possível gerar documentações ricas de cada elemento através de um editor de texto bastante rico, com formatação de texto, criação de tabelas, entre outros. Os diagramas criados são gravados em nuvem própria do produto. Além disso, é possível exportá-lo para o formato padrão .bpmn, imagem PNG ou de documentação, como PDF, DOC e HTML (estático).

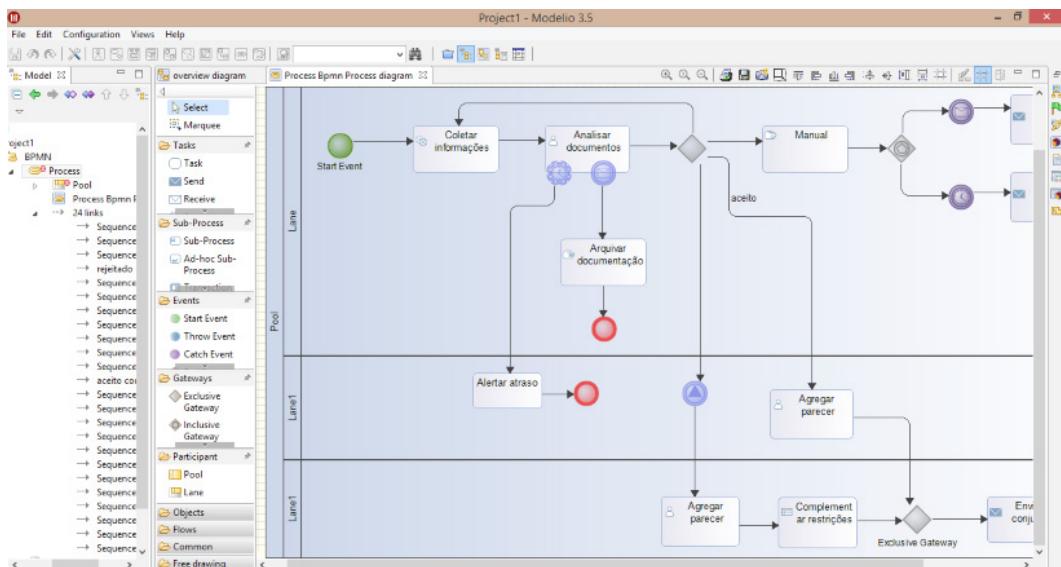


A ferramenta é toda em português! Para acessar, experimentar e descobrir mais sobre esta ferramenta acesse:
<https://app.heflo.com/>.

3.7 FERRAMENTA CASE: MODELIO

Segundo Sganderla (2016, s.p.), esta ferramenta de modelagem de processos *software* livre cujo “objetivo original é a modelagem de diagramas UML, mas que foi estendida para criar diagramas de processos em BPMN. Quem utiliza tem a sensação de trabalhar em uma ferramenta de desenvolvedor, pois ela é baseada no eclipse, famosa interface para desenvolvimento de *software*”.

FIGURA 60 - FERRAMENTA MODELIO



FONTE: Disponível em: <<http://blog.iprocess.com.br/2016/09/7-ferramentas-gratuitas-para-criar-diagramas-de-processos-com-bpmn/>>. Acesso em: 3 nov. 2016.

Segundo Fermino (2016)

Os facilitadores de modelagem encontrados em outras ferramentas não ocorrem neste editor. Para começar a modelar é preciso criar um “Projeto” e então dentro dele criar um diagrama BPMN. Os elementos, inclusive conectores, precisam ser adicionados um a um no diagrama. A definição dos tipos de tarefas e eventos também não é muito simples – é preciso acessar a janela de propriedades do elemento no diagrama para então escolher entre os tipos. Em termos de notação BPMN a ferramenta é bastante aderente, pois contém praticamente todos os elementos e usando a funcionalidade de Audit pode-se fazer validação das regras da notação conforme a especificação da OMG.



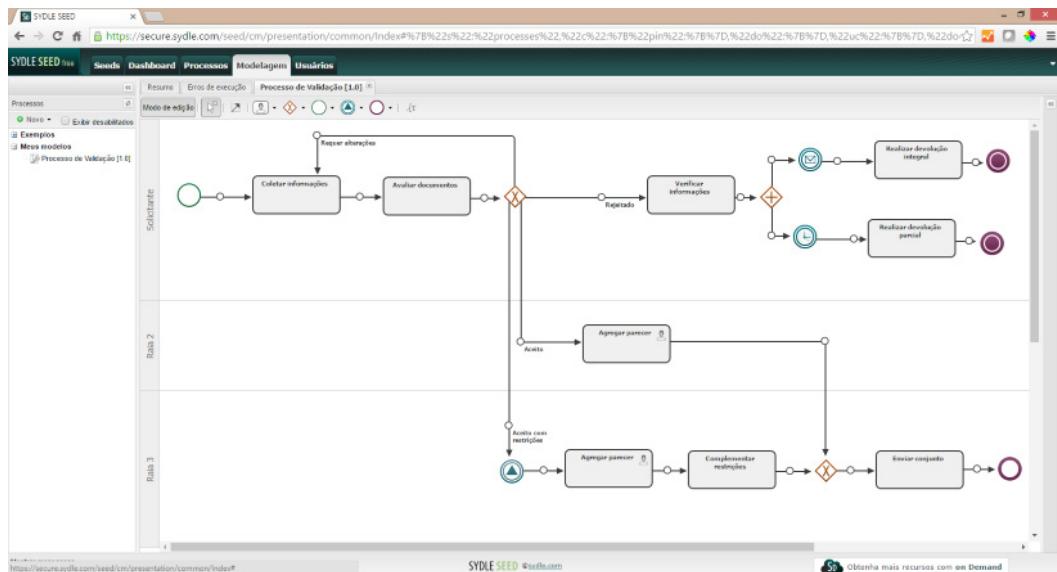
Os diagramas não são salvos avulsos, eles ficam dentro do arquivo de projeto. É possível exportar para formato de imagem como PNG ou JPG. A interface é somente em inglês. Para baixar, rodar e descobrir mais sobre esta ferramenta: <<https://www.modelio.org/>>. Para aprender mais sobre esta ferramenta acessar este endereço do YouTube: <https://www.youtube.com/watch?v=AJ33XXT-_Pc>.

3.8 FERRAMENTA CASE: SYDLE

Segundo Fermino (2016, s.p.)

Esta ferramenta brasileira é mais do que apenas um editor de diagramas em BPMN, é uma suíte para gerenciamento de processos. Por este motivo, a solução possui muitas outras funcionalidades que vão além da modelagem do processo. O modelador é web, portanto, não é necessário baixar ou instalar nenhum programa. Para criar modelos no Sydle é preciso criar uma conta, mas pode ser na versão gratuita (Community) da ferramenta. A utilização de outras funcionalidades, além da modelagem, pode exigir a escolha por uma licença. Como o foco principal da ferramenta é a modelagem de processos para serem automatizados através do próprio produto, estão disponíveis para a modelagem apenas os elementos de BPMN implementados na automação. Mesmo assim, atende a quase toda a especificação na subclasse descritiva (à exceção dos elementos de pool e fluxo de mensagem e os acessórios data object e data store).

FIGURA 61 - FERRAMENTA SYDLE



FONTE: SGANDERLA, Kelly. Disponível em: <<http://blog.iprocess.com.br/2016/09/7-ferramentas-gratuitas-para-criar-diagramas-de-processos-com-bpmn/>>. Acesso em: 3 nov. 2016.

Segundo Sganderla (2016, s.p.)

Para a documentação, a ferramenta disponibiliza um minieditor de texto. O salvamento dos diagramas é realizado no servidor da Sydle, e é possível extrair uma versão impressa de documentação tipo manual de processo, com a imagem do diagrama e o detalhamento da documentação com recursos ricos de formatação de texto, inclusive uso de tabelas.

Como outras ferramentas, possui o recurso de menu de contexto, que permite criar o fluxo a partir do conector no elemento anterior. A ferramenta faz autossalvamento a cada elemento adicionado no processo. “Não há funcionalidade específica para fazer a validação do modelo no editor, mas algumas regras básicas são verificadas automaticamente (por exemplo, não permite conectar o evento final a outro elemento de fluxo)” (SGANDERLA, 2016, s.p.). Não há recursos padrão de exportação.



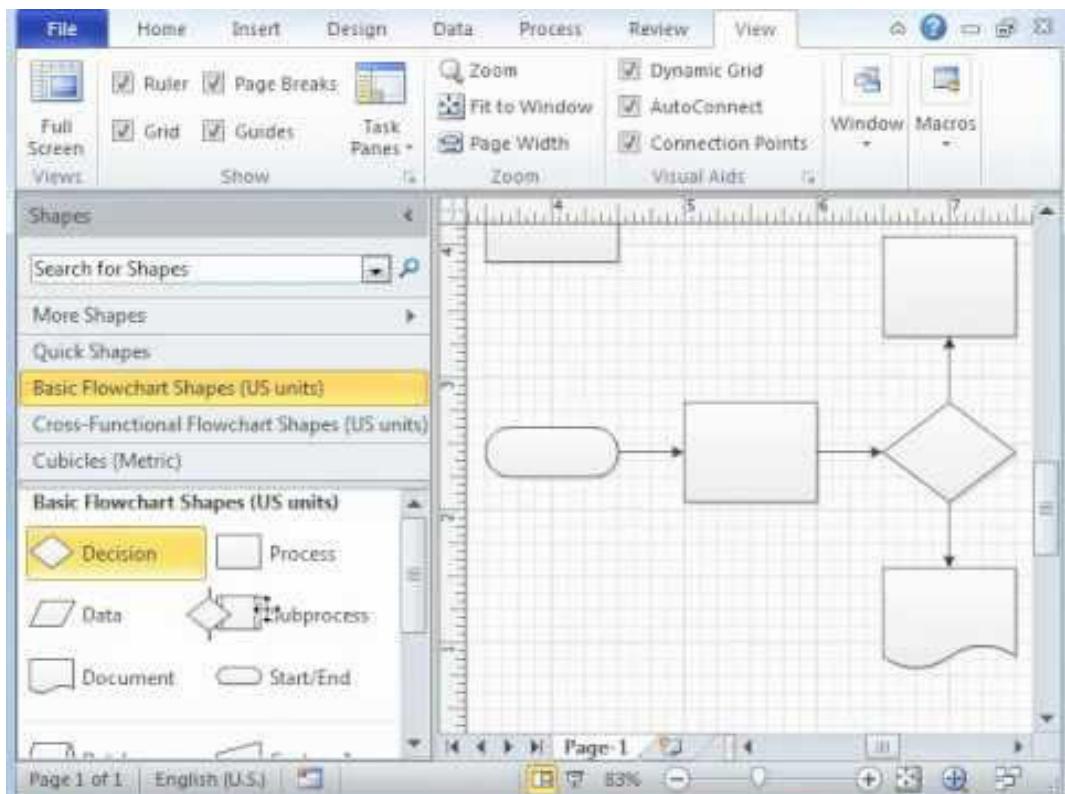
Esta solução é brasileira e a interface é totalmente em português. Para criar uma conta e experimentar esta ferramenta: <<http://www.sydle.com.br/bpm/>>. Para aprender mais sobre esta ferramenta acessar este endereço do YouTube: <<https://www.youtube.com/watch?v=b1lkC-o2Ah8>>.

3.9 FERRAMENTA CASE: MICROSOFT VISIO®

Segundo Santos (2006, p. 10)

Microsoft Visio® é um aplicativo para criação de diagramas para o ambiente Windows. O ponto forte do Visio são os diagramas técnicos e profissionais. O ponto forte do Visio são os diagramas técnicos e profissionais, com imagens vetoriais, que podem ser ampliados e manipulados com facilidade. O Visio pode ser utilizado para gerar diagramas de diversos tipos, como organogramas, fluxogramas, modelagem de dados (usando UML ou outra notação gráfica qualquer), diagramas de redes, plantas baixas, cartazes etc.

FIGURA 62 - FERRAMENTA MS VISIO®



FONTE: Disponível em <<https://i.ytimg.com/vi/eyM5WoaYoUw/hqdefault.jpg>>. Acesso em: 5 nov. 2016.

Segundo Techsoup Brasil (2016, s.p.):

Microsoft Visio disponibiliza diversas ferramentas para auxiliar no gerenciamento de projetos e milhares de figuras, formas e desenhos diferentes, permitindo que você possa desenhar desde o planejamento de uma estrutura de rede para aplicação em uma empresa até um organograma empresarial, além de WBS, fluxograma, diagramas de modelagem para desenvolvimento, entre outros.



Para criar uma conta e experimentar esta ferramenta: <<http://www.baixaki.com.br/download/microsoft-visio-premium.htm>>. Para aprender mais sobre esta ferramenta acessar este endereço do YouTube: <https://www.youtube.com/watch?v=L_NajdWjYLA>.

3.10 FERRAMENTA CASE: EPF-COMPOSER

Segundo Pagliares (2016, s.p.):

O EPF Composer (*Eclipse Process Framework – Composer*) é uma iniciativa *open-source* do Eclipse em parceria com organizações com grande experiência na área de elaboração de processo de desenvolvimento de *software*. Este processo, assim como o UP (*Unified Process*), fornece um conjunto de melhores práticas em desenvolvimento de *software* considerando as diferentes fases do desenvolvimento (desde requisitos até manutenção).

A ferramenta *open source EPF composer* possibilita a definição de processos, reutilização de conteúdo, gerenciamento de bibliotecas e configuração e publicação de processos.

Segundo Aleixo (2016), o objetivo do EPF Composer é prover um *framework* customizável para engenharia de processos de *software*. É uma ferramenta que proporciona uma estrutura para gestão de processo de *software* através de um desenvolvimento iterativo, ágil e incremental, otimizado para pequenos projetos.

O metamodelo utilizado pela EPF baseia-se no SPEM – *Software Process Engineering Metamodel* – definido pela OMG [OMG,2005]. O SPEM tem como objetivo descrever os elementos que compõem processos de desenvolvimento de *software* concretos. Este é representado por um modelo orientado a objetos, utiliza UML como notação e é estruturado como um UML Profile. UML Profiles são pacotes dedicados a agrupar extensões UML e são definidos a partir de mecanismos de extensão, cujo o objetivo é permitir adaptar a UML para domínios específicos, no

caso processo de desenvolvimento de *software* (DAFLON, 2004).

Segundo Carneiro (2016), o Eclipse Process Framework (EPF) visa produzir um *software* com características próprias, fornecendo ferramentas e conteúdos que podem ser usados como base para uma grande variedade de processos de TI a fim de resolver necessidades. O EPF ajuda na autoria de métodos, ferramentas, processos, gestão de configuração de bibliotecas de métodos e publicação de processos.



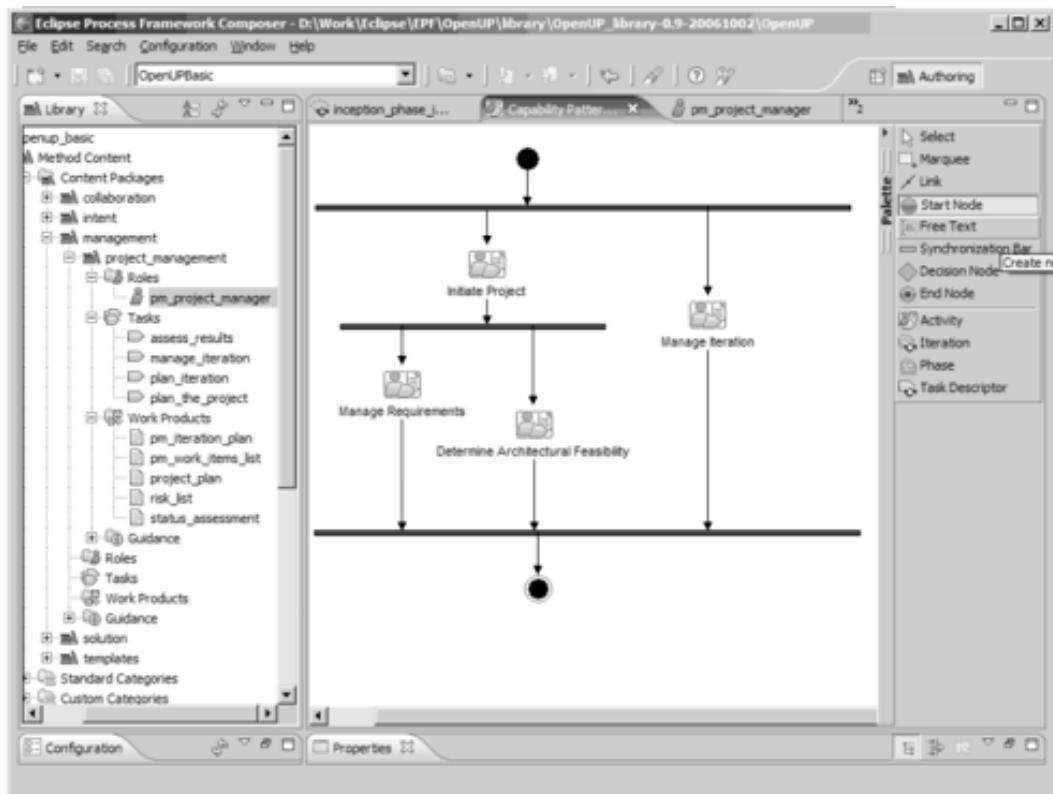
No final desta Unidade 3 será apresentado um exemplo prático de processo de *software* através do EPF Composer.

O *Framework* conta com exemplos de processos prontos para serem adaptados e ferramentas que são suportadas por vários tipos de projetos e formas de desenvolvimento. Ao usar EPF Compositor pode-se criar o seu próprio processo de desenvolvimento de *software* e estruturá-lo em sentido específico, usando um esquema predefinido (CARNEIRO et al., 2008, s.p.).

Para Carneiro et. al. (2008, s.p.) este esquema é uma evolução da *Software Process Engineering Meta-Model* (SPEM) de especificação referida, como o da Arquitetura do método unificado (UAM). “Grandes partes da UAM foi recentemente adaptado para a revisão da SPEM. A UAM e a SPEM são projetos de apoio à organização, com grande quantidade de descrições de métodos e processos de desenvolvimento”. Tais métodos possuem conteúdos e processos que não precisam ser limitados à engenharia de *software*, mas também podem abranger outras concepções de engenharia, tais como a engenharia mecânica, transformação das empresas e assim por diante.

Os projetos individuais de diversas organizações de desenvolvimento de *software* podem facilmente transferir e implantar os processos capturados no EPF. Eles podem personalizar processos existentes através da mistura de correspondência de conteúdo de vários processos, por remoção, adição ou personalização de conteúdo através da aplicação de conteúdos fornecidos por plug-ins. Os processos resultantes podem facilmente ser implantados, e continuamente evoluir, acomodando as lições aprendidas à medida que o projeto avança (CARNEIRO et al., 2008, s.p.).

FIGURA 63 - FERRAMENTA EPF-COMPOSER



FONTE: Disponível em: <http://www.aprocesssgroup.com/wp-content/uploads/epf_composer_iteration.bmp>. Acesso em: 5 nov. 2016.



Para fazer download desta ferramenta favor acessar o site do eclipse: <<https://eclipse.org/epf/downloads/downloads.php>>. Para ilustrar o uso do EPF foi desenvolvido um vídeo, utilizando o Camtasia Studio 5, que é um programa que nos possibilita captar movimentos do cursor, seleções de menus, janelas pop-up, janelas em camadas, digitações, e outros itens que você vê na tela. O vídeo demonstra passo a passo como aplicar o processo no EPF (Criação de plugin, pacotes, papéis, atividades, produtos de trabalho, guias e configuração), bem como a publicação do processo. Este vídeo está disponível no site <<http://www.cipedya.com/doc/176182>>.

LEITURA COMPLEMENTAR

AVALIAÇÃO E MELHORIA DE PROCESSOS DE SOFTWARE

A melhoria de um processo é, sobretudo, um processo de mudança (ZAHRAN, 1998) e, sendo assim, necessita ser analisada sob o ponto de vista de todos os envolvidos nessa mudança. A tendência natural das pessoas é ser resistente às mudanças. Então, para se conseguir êxito em uma melhoria de processos, é preciso observar os pontos de vista dos diferentes envolvidos. A alta gerência está normalmente preocupada com resultados e custos. Por outro lado, desenvolvedores estão mais preocupados com o trabalho realizado em suas atividades cotidianas (CONRADI et al., 2002). Assim, é muito importante que se consiga uma visão de melhoria de processos em que ambas as partes entendam a importância da melhoria.

A melhoria de processos envolve, em muitos casos, o estudo e a utilização de novas ferramentas e técnicas de coleta e análise de dados. Essas iniciativas podem causar desconforto e desconfiança nas pessoas, caso a organização e sua equipe de processos não deixem claros os objetivos das melhorias (UMARGI et al., 2005).

Tendo em vista o exposto, definir e divulgar aos envolvidos os objetivos que devem ser atingidos com uma melhoria de processo são ações essenciais para o sucesso dessa empreitada. Mais ainda, para que os objetivos determinados para uma melhoria sejam atingidos, é fundamental que seja estabelecida uma gestão adequada dos processos envolvidos na melhoria contínua (CAMPOS, 2005). Neste contexto, uma abordagem interessante é considerar uma melhoria de processo como um projeto organizacional, de modo que as práticas de planejamento, execução, monitoramento e controle de um projeto sejam aplicadas, aumentando, com isso, a probabilidade de sucesso do projeto de melhoria.

Processos não podem ser definidos e mantidos “congelados” para sempre. Processos precisam continuamente passar por mudanças e refinamentos para aumentar a sua habilidade de lidar com requisitos e expectativas da organização e do mercado no qual ela atua. Assim, processos precisam ser continuamente avaliados e melhorados (FUGGETTA, 2000).

A avaliação sistemática da qualidade de um processo, de seus ativos e de seus produtos resultantes é essencial para apoiar a implementação de estratégias de melhoria dentro de uma organização (FUGGETTA, 2000). Contudo, uma avaliação sistemática não pode ser conduzida em bases meramente subjetivas. Assim, é necessário identificar características capazes de indicar a qualidade de um processo, medir essas características, analisar os resultados dessas medições e

concluir sobre as necessidades de melhoria. Assim, a melhoria contínua passa pela medição e avaliação de processos para, aí sim, mediante informações objetivas, a organização poder tomar a decisão sobre a necessidade e a viabilidade de uma determinada melhoria. Portanto, para se falar de melhoria é necessário falar também de medição e avaliação.

FONTE: MORO, Rodrigo Dal. **Avaliação e melhoria de processos de software**: conceituação e definição de um processo para apoiar a sua automatização. 2008. Disponível em: <http://inf.ufes.br/~falbo/files/Avaliacao_e_Melhoria_de_Processos_de_Software.pdf>. Acesso em: 25 out. 2016.

RESUMO DO TÓPICO 2

Neste tópico, você aprendeu que:

- O objetivo principal de se melhorar os processos de *software* é basicamente aumentar a capacidade desses processos de maneira contínua e incremental. É uma ação executada para avaliar e melhorar as operações e atividades internas ou externas durante o desenvolvimento de *software*.
- A melhoria de processos envolve, em muitos casos, o estudo e a utilização de novas ferramentas e técnicas de coleta e análise de dados. Processo de *software* é importante porque fornece estabilidade, controle e organização para uma atividade que pode, se deixada sem controle, tornar-se bastante caótica.
- Para produzir um produto de *software* com qualidade deve-se possuir processos formais que visem à prevenção e detecção de defeitos durante o desenvolvimento de *software*.
- A técnica de prevenção de defeitos em um processo de desenvolvimento de *software* se dá pelo uso de instruções de procedimentos (padrões formais), treinamentos, documentação, modelagem e reengenharia.
- As técnicas de detecção de defeitos podem ser pela análise de código; revisão por pares; testes, auditorias, verificações e validações.
- As organizações precisam entender a situação atual de seus processos de *software* e realizando continuamente sua avaliação, com base nos pontos fracos identificados agir e identificar melhoria.
- Para conduzir programas de melhoria de Processo de *Software*, as organizações normalmente se baseiam em modelos de referência como MPS.BR e o CMMI que são fundamentados na tríade processo-tecnologia-pessoas.
- O CMMI (*Capability Maturity Model Integration* ou Integração dos Modelos de Capacitação e Maturidade de Sistemas) fornece diretrizes baseadas em melhores práticas para a melhoria dos processos e habilidades organizacionais, cobrindo o ciclo de vida de produtos e serviços completos, nas fases de concepção, desenvolvimento, aquisição, entrega e manutenção.
- O MPS. BR (Melhoria de Processo de *Software* Brasileiro) é um programa que foi criado para melhorar a capacidade de desenvolvimento de *software* nas empresas brasileiras voltados para médias e pequenas empresas e com baixo custo de implantação.

- A melhoria do processo de *software* pode ser considerada hoje uma das grandes prioridades ao trabalhar com *software*. Isto se deve à exigência do mercado por produtos com maior qualidade, que sejam entregues mais rapidamente e com menor custo de desenvolvimento.
- Estudos apontam que ao tentarem melhorar seus processos, as empresas estão em busca de: entender as características dos processos existentes e os fatores que afetam a sua capacidade; planejar, justificar e implementar ações que modificarão os processos, tornando-os mais coerentes com as necessidades de negócios e; avaliar os impactos e benefícios ganhos, comparando-os com os custos advindos das mudanças realizadas (OLIVEIRA, 2016).
- A melhoria contínua passa pela medição e avaliação de processos para, aí sim, mediante informações objetivas, a organização poder tomar a decisão sobre a necessidade e a viabilidade de uma determinada melhoria.

As mais importantes ferramentas CASE de gestão de processo de *software* utilizadas para dar suporte ao processo de desenvolvimento são:

- Bizagi Modeler: permite a simulação dos fluxos de trabalho a fim de facilitar a análise de melhorias tanto em relação ao tempo quanto em relação ao custo das atividades desenvolvidas.
- Aris Express: é uma arquitetura alinhada ao conceito de BPMN adequado principalmente para usuários ocasionais e novatos construída em três níveis: estratégia, especificação (composta de Design, Otimização e Controle) e execução.
- BPMN.io: ferramenta leve com editor *web* e que permite criar diagramas de processo sem precisar instalar nenhum aplicativo.
- Draw.io: uma ferramenta *on-line* de criação de diagramas tipo *flowchart*, incorporou como uma das notações visuais para desenho a palheta de elementos de BPMN.
- Yaoqiang BPMN Editor: além de modelagem de diagrama de Processo (Orquestração), também possui todos os elementos para a criação dos dois outros tipos de diagramas da notação: conversão e coreografia (o único editor gratuito que identificamos até agora que comporta os três tipos de diagramas nativamente).
- Heflo: O editor é totalmente *web*, interface fácil de usar, muito fácil de sair criando diagramas e possui grande aderência à notação BPMN para diagrama de processo (orquestração).
- Modelio: ferramenta cujo objetivo original é a modelagem de diagramas UML, mas que foi estendida para criar diagramas de processos em BPMN. Quem utiliza tem a sensação de trabalhar em uma ferramenta de desenvolvedor, pois ela é baseada no Eclipse, famosa interface para desenvolvimento de *software*.

- o Sydle: ferramenta brasileira além de possuir editor de diagramas em BPMN, é uma suíte para gerenciamento de processos. Por este motivo, a solução possui muitas outras funcionalidades que vão além da modelagem do processo.
- o Microsoft Visio®: é um aplicativo para criação de diagramas para o ambiente Windows. O ponto forte do Visio são os diagramas técnicos e profissionais, com imagens vetoriais, que podem ser ampliados e manipulados com facilidade.
- o EPF Composer (Eclipse Process Framework – Composer) é uma iniciativa open-souce do Eclipse. Fornece um conjunto de melhores práticas em desenvolvimento de *software* considerando as diferentes fases do desenvolvimento (desde requisitos até manutenção) e possibilita a definição de processos, reutilização de conteúdo, gerenciamento de bibliotecas e configuração e publicação de processos.

AUTOATIVIDADE



1. Quais são os principais motivos para que organizações de *software* adotem melhoria contínua de seus processos de *software*?
2. Explique as técnicas de prevenção e detecção de defeitos em um processo de desenvolvimento de *software*.
3. Quais são os níveis de maturidade do CMMI e do MPS.Br.



BUSINESS PROCESS MANAGEMENT NOTATION (BPMN) E UM ESTUDO DE CASO DE UM PROCESSO SOFTWARE COM O AUXILIO DO EPF

1 INTRODUÇÃO

Dada a importância que as áreas de notação de processo de negócios têm para as organizações, o Tópico 3 abordará o BPMN, o modelo mais popular e utilizado pelos analistas de negócios e analistas de sistemas em empresas de TI.

2 BUSINESS PROCESS MANAGEMENT NOTATION (BPMN)

Antes da definição do *Business Process Management Notation* (BPMN) gostaríamos de reforçar estas duas definições de Ian Sommerville (2011), que nos diz que existem vários processos de desenvolvimento de *software* diferentes, mas que todos envolvem (1) especificação – definição do que o sistema deve fazer; (2) projeto e implementação – definição da organização do sistema e implementação do sistema; (3) validação – checagem de que o sistema faz o que o cliente deseja; e (4) evolução – evolução em resposta a mudanças nas necessidades do cliente.

Lembrando que em todo este ciclo de vida informado por Sommerville (2011, p. 20), se incluem as seguintes descrições:

- Produtos, que são os resultados de uma atividade do processo.
- Papéis, que refletem as responsabilidades das pessoas envolvidas no processo.
- Pré e pós-condições, que são declarações que são verdadeiras antes e depois de uma atividade do processo ser executada, ou um produto produzido.

Enfim, agora vamos entender o BPMN – *Business Process Model and Notation* e perceber que ele segue estes princípios apontados por Ian Sommerville, o BPMN é a notação gráfica lançado a partir de 2004 mais aceita para modelar processos. Ilustra o processo de uma maneira simples e clara utilizando uma gramática de símbolos para mapear, de maneira padrão, todos os processos de negócio de uma organização.

No BPMN, um processo de negócio é representado através do encadeamento de eventos e atividades, ligados através de conectores que demonstram a sequência em que os mesmos são realizados. Além de eventos e atividades, outros elementos de controle de fluxo podem ser utilizados na modelagem para permitir a criação ou unificação de fluxos paralelos que ocorram no decorrer de um mesmo processo de negócio (SGANDERLA, 2012, s.p.).

Na modelagem BPMN existem quatro grupos de elementos:

- Objetos de fluxo
- Objetos de conexão
- Raia de piscina
- Artefatos.

Os objetos de fluxos são os principais elementos gráficos para definir o comportamento do processo de negócio. Eles podem ser de três tipos:

1 Eventos: algo que acontece ou pode acontecer em um processo. Estes eventos afetam o fluxo do processo e têm geralmente uma causa (*trigger*) ou um impacto (*result*).

Eles são definidos como:

- Início: simples, tempo, condicional, sinal, múltiplo.
- Intermediário: simples, condicional, ligação e múltiplo. Associados a estas atividades: cancelamento, compensação, condicional, sinal e múltiplo.
- Fim: simples e término.

2 Atividades: Passos lógicos que ocorrem dentro do processo; é um termo genérico para um trabalho executado. Os tipos de atividades são: tarefas e subprocessos. O subprocesso é distinguido por uma pequena cruz no centro inferior da figura.

Essas tarefas podem ser dos tipos:

- Human Task.
- Service Task.
- Send Task/Receive Task.
- Manual Task.
- Script Task.
- Business Rule Task.

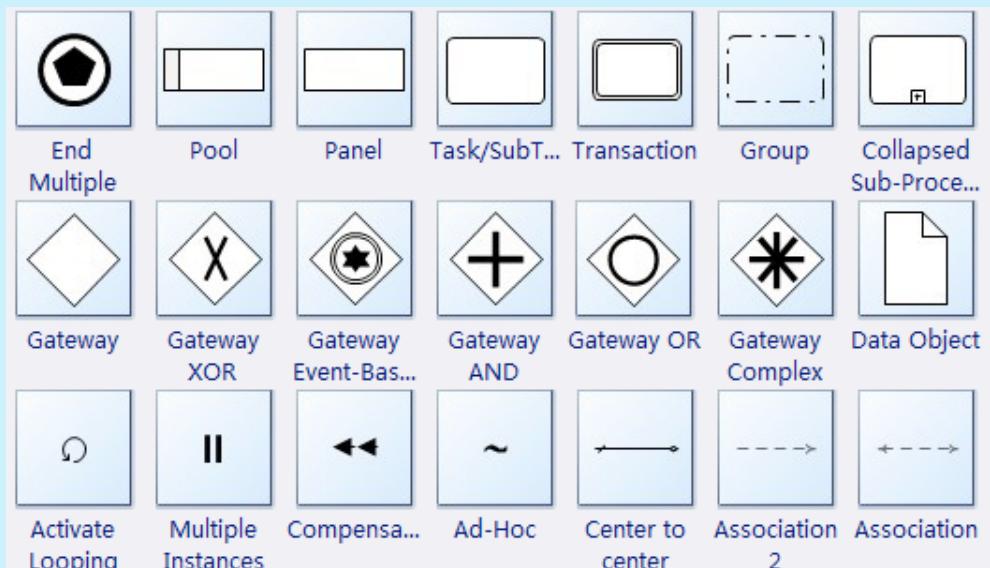
3 Decisões: Chamado de *gateways*, que controlam o fluxo de sequência. São usados para controlar a divergência e a convergência da sequência de um fluxo. Assim, determinará decisões tradicionais, como juntar ou dividir trajetos.

Os *gateways* podem ser:

- Gateway exclusivo baseado em dados.
- Gateway exclusivo baseado em dados com marcador.
- Gateway exclusivo baseado em eventos.
- Gateway inclusivo.
- Gateway Completo.
- Gateway Paralelo.

Na figura a seguir, você tem uma visão geral dos elementos da notação.

FIGURA 64 - ELEMENTOS DA NOTAÇÃO BPMN



FONTE: Disponível em: <<http://www.devmedia.com.br/introducao-ao-business-process-modeling-notation-bpmn/29892>>. Acessado em: 6 nov. 2016.

Os objetos de conexão representam a forma como os objetos de fluxo se conectam. Eles se dividem em três tipos também:

- Fluxo de sequência: representa a ordem do fluxo, sua sequência.
- Fluxo de mensagem: representa o fluxo das mensagens entre o emissor e o receptor.
- Associação: usada para associar dados, textos e outros artefatos aos objetos do fluxo.

Na figura a seguir você acompanha a representação gráfica de cada um dos fluxos.

FIGURA 65 - OBJETOS DE CONEXÃO

OBJETO	FIGURA
Fluxo de Sequencia	→
Fluxo de Mensagens	↔
Associação	→

FONTE: Disponível em: <<http://www.devmedia.com.br/introducao-ao-business-process-modeling-notation-bpmn/29892>>. Acesso em: 6 nov. 2016.

Os *swinlanes* representam uma forma de organização das atividades em categorias visuais separadas e agem como um contêiner para os objetos de fluxos. São elas:

- **Pool:** representa a organização em si, é onde são desenhados os elementos representativos do processo. Ele atua como um container para dividir um conjunto de atividades de outras piscinas. São utilizados quando o diagrama envolve duas entidades de negócio (ou participantes) que estão separados fisicamente no diagrama e especifica o "que faz o que" colocando os eventos e os processos em áreas protegidas, chamados de *pools*.
- **Lane:** são as subdivisões de um *pool*. É usada para organizar as atividades do processo. Nessas subdivisões podemos separar as atividades de acordo com suas associações (função ou papel). O *lane*, por exemplo, representa um departamento dentro dessa organização que é representada pelo *pool*.

Na figura a seguir, você observa a representação gráfica dos objetos descritos acima.

FIGURA 66 - SWINLANES

OBJETO	FIGURA
Pool	
Lane	

FONTE: Disponível em: <<http://www.devmedia.com.br/introducao-ao-business-process-modeling-notation-bpmn/29892>>. Acesso em: 6 nov. 2016.

Os **artefatos** são usados mais para colocar informações adicionais no processo. Eles podem também ser usados para representar as entradas ou saídas de uma atividade. Ao todo temos três tipos de artefatos:

- Objetos de dados: elementos produzidos ou requeridos por uma atividade, conectados por meio de associações.
- Grupo: possui finalidade de documentação ou análise.
- Anotações: usado para passar ao leitor informações adicionais de uma atividade.

FONTE: NOGUEIRA, Rhaíssa. Introdução ao Business Process Modeling Notation (BPMN). 2015. Disponível em: <<http://www.devmedia.com.br/introducao-ao-business-process-modeling-notation-bpmn/29892>>. Acesso em: 6 nov. 2016.



Leonardo Marques consultor BPM disponibiliza treinamentos sobre BPMN. Para maiores conhecimentos favor acessar o endereço: <<http://www.fazenda.gov.br/pmimf/frentes-de-atuacao/inovacao-projetos-e-processos/download-de-arquivos/treinamento-bpmn.pdf>>.

E também um manual de um curso sobre mapeamento de processo de trabalho com BPMN e Bizagi elaborado pelo tribunal de contas da união – Instituto Serzedello Corrêa disponível em: <http://portal3.tcu.gov.br/portal/page/portal/TCU/comunidades/gestao_processos_trab/curs...aula%202_v%202013.pdf>.

Segundo Campos (2013, s.p.):

A maioria das organizações já está trabalhando suas iniciativas de modelagem de processos utilizando notação BPMN. E aquelas que ainda não estão, buscam capacitação e consultoria nesta área. O governo brasileiro, inclusive, estabeleceu a notação BPMN como obrigatória para todas as suas iniciativas em modelagem de processos, tanto as conduzidas com os profissionais da própria instituição quanto as conduzidas por consultorias. Exatamente por se tratar de um padrão aberto.

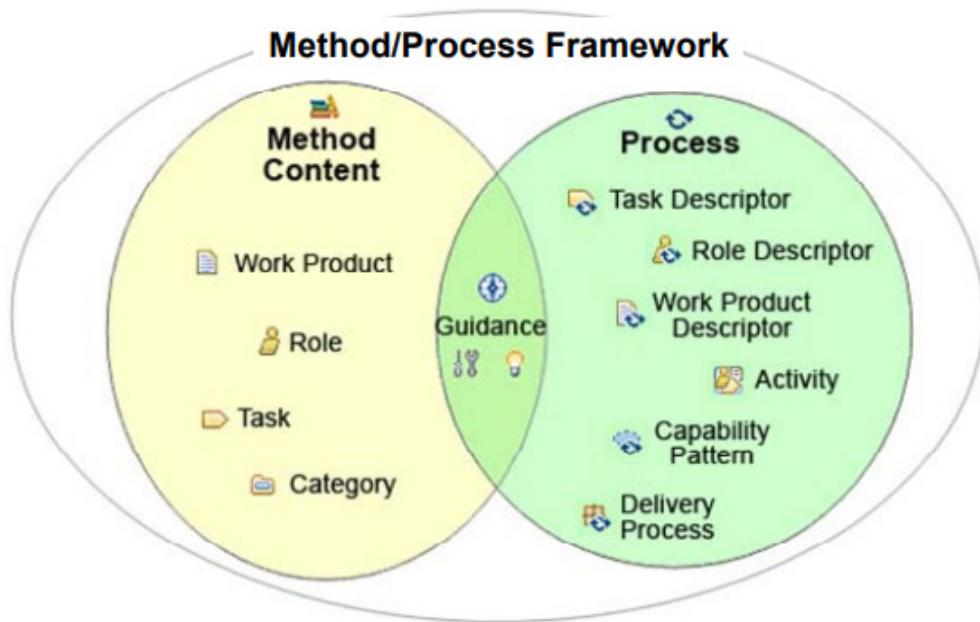
Fortalecendo a linguagem única para a modelagem e gestão de processos de negócio, contribuindo para o ganho de produtividade e clareza na distribuição de informações.

3 ESTUDO DE CASO DE UM PROCESSO SOFTWARE COM O AUXILIO DO EPF

Para finalizar esta unidade de estudos será apresentado o Estudo de Caso de um sistema utilizado pelo Núcleo de Tecnologia da Informação e Comunicação (NTI) da Universidade Federal Fluminense (UFF) voltado ao desenvolvimento e manutenção de sistemas computacionais na plataforma *web* extraído da dissertação de mestrado em Informática de Glória Maria de Paula Oliveira pela Programa de Pós-Graduação em Informática da PUC-Rio. O modelo de maturidade que serviu como guia para fazer o processo do NIT foi o CMMI (*Capability Maturity Model Integration* ou Modelo de Maturidade em Capacitação e Integração).

A seguir será demonstrado a base conceitual da ferramenta do EPF, ou seja, os recursos utilizados para definir o processo de *software*.

FIGURA 67 - UM RESUMO DA TERMINOLOGIA PROPOSTA PELO ECLIPSE PROCESS FRAMEWORK



FONTE: Disponível em: <<http://www.eclipse.org/epf/general/EPFComposerOverviewPart1.pdf>>.

Observa-se pela figura que o Conteúdo de Método (*Method Content*) engloba os produtos de trabalho, papéis, tarefas e guias. Os produtos de trabalho podem ser diagramas, projetos arquiteturais, programas ou módulos do sistema a ser desenvolvido. Os papéis são conjuntos de funções atribuídos a um ou mais integrantes da equipe tais como analista de sistemas, analistas de testes, gerente. As tarefas serão executadas pelos papéis definidos e podem ser categorizadas de acordo com sua natureza, boas práticas e padrões. Uma atividade do processo é formada por um conjunto de tarefas.

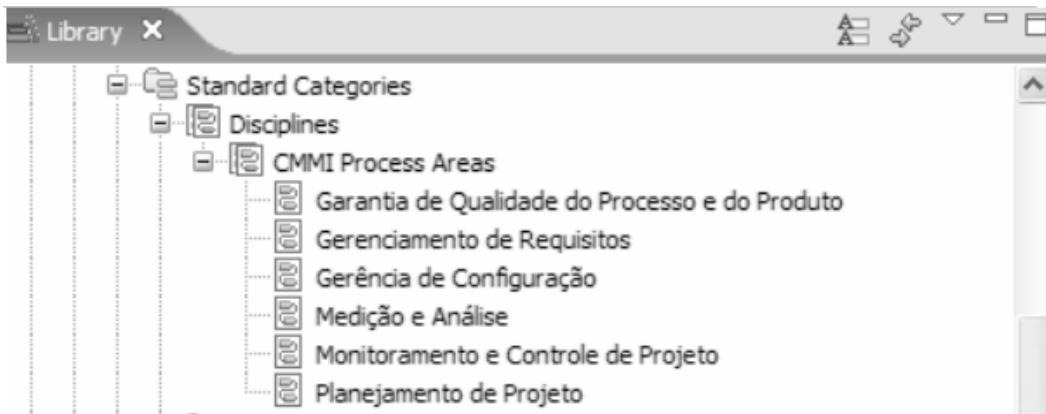
Guias – tais como linhas de direção, conceitos, *templates*, *checklists*, exemplos e planos de caminho – são posicionados na interseção de Conteúdo de Método e Processos. Em outras palavras, guias podem estar relacionadas e expressar informações relevantes para Conteúdo de Método bem como Processos (OLIVEIRA apud HAUMER, 2007, tradução nossa).

Já do lado direito da figura estão os conceitos utilizados para representar processos onde principal conceito é a atividade que pode ser aninhada para

definir estruturas hierárquicas. Atividades relacionadas com outras define o fluxo de trabalho. Elas contêm referências para o conteúdo de método. Atividades são usadas para definir dois principais tipos de processo que o EPF sugere: *Delivery Processes* e *Capability Processes*. *Delivery Processes* representam um completo e integrado *template* para executar em um específico tipo de projeto. Eles descrevem um ciclo de vida completo de um projeto e pode ser usado como uma referência para executar projetos com características similares. *Capability patterns* são processos que expressam conhecimento de processo para uma área chave de interesse como uma disciplina ou boas práticas (OLIVEIRA apud HAUMER, 2007).

Para a elaboração deste processo via EPF Composer do NIT foi criada uma disciplina com as áreas de processo do nível 2 do CMMI, onde “as disciplinas são um tipo de categoria padrão oferecida pela EPF e podem ser utilizadas para categorizar as tarefas do processo”.

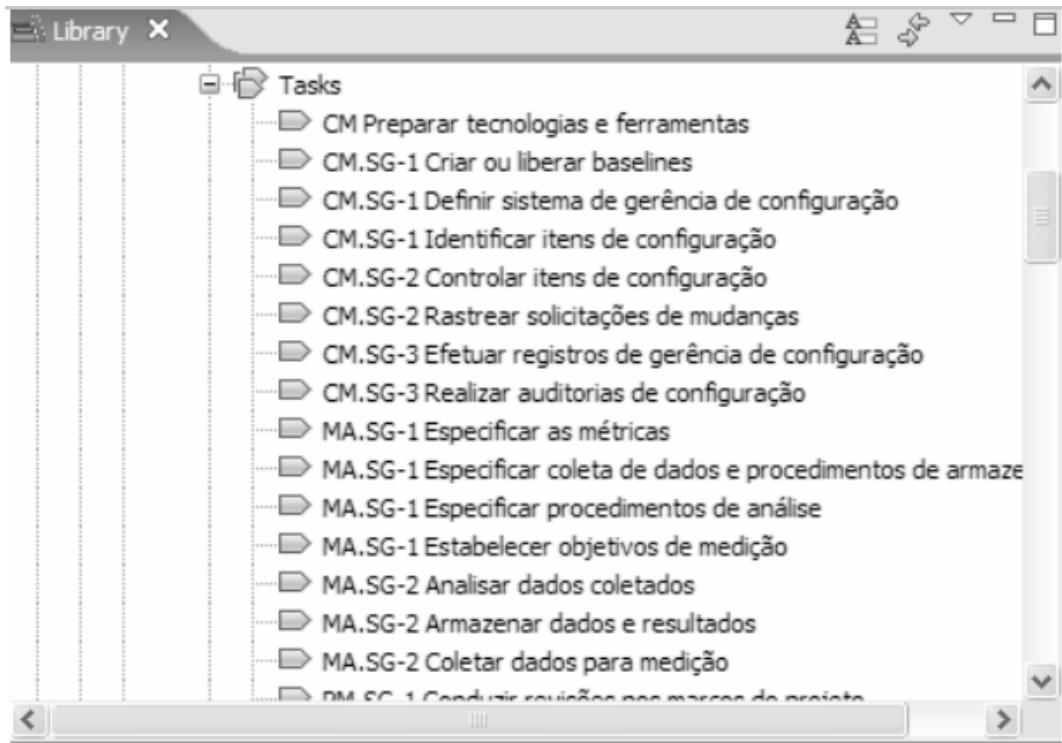
FIGURA 68 - ÁREAS DE PROCESSO DO CMMI CONFIGURADAS COMO DISCIPLINAS



Fonte: Oliveira (2007)

Com base nestas áreas de processo, foram criadas tarefas que estão relacionadas a cada uma delas. Cada tarefa representa um objetivo da área de processo do CMMI (SEI, 2005). No exemplo abaixo demonstra a lista de tarefas da área de processo Gerência de Configuração.

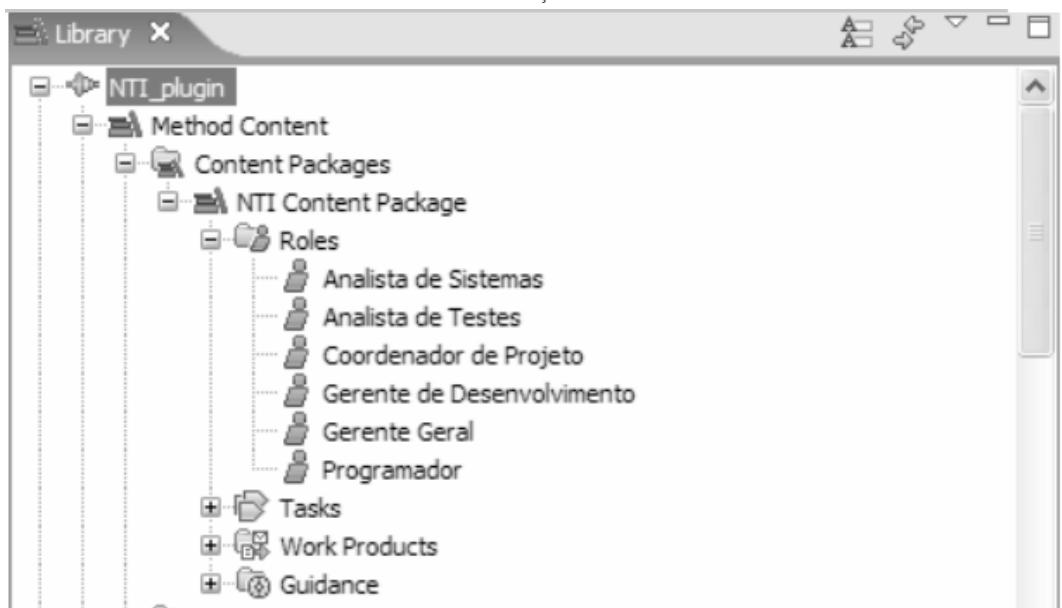
FIGURA 69 - TAREFAS DISPONÍVEIS PARA SEREM UTILIZADAS NOS PROCESSOS



FONTE: Oliveira (2007)

Para finalizar esta etapa de preparação foram definidos no EPF Composer os papéis para as diversas funções existentes no NTI. Cada papel poderá ser responsável por executar ou assessorar uma ou mais tarefas do processo.

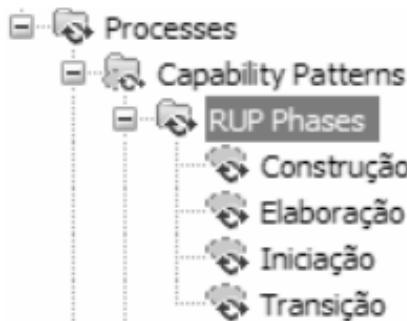
FIGURA 70 - PAPÉIS DISPONÍVEIS PARA A DEFINIÇÃO DO PROCESSO



FONTE: Oliveira (2007)

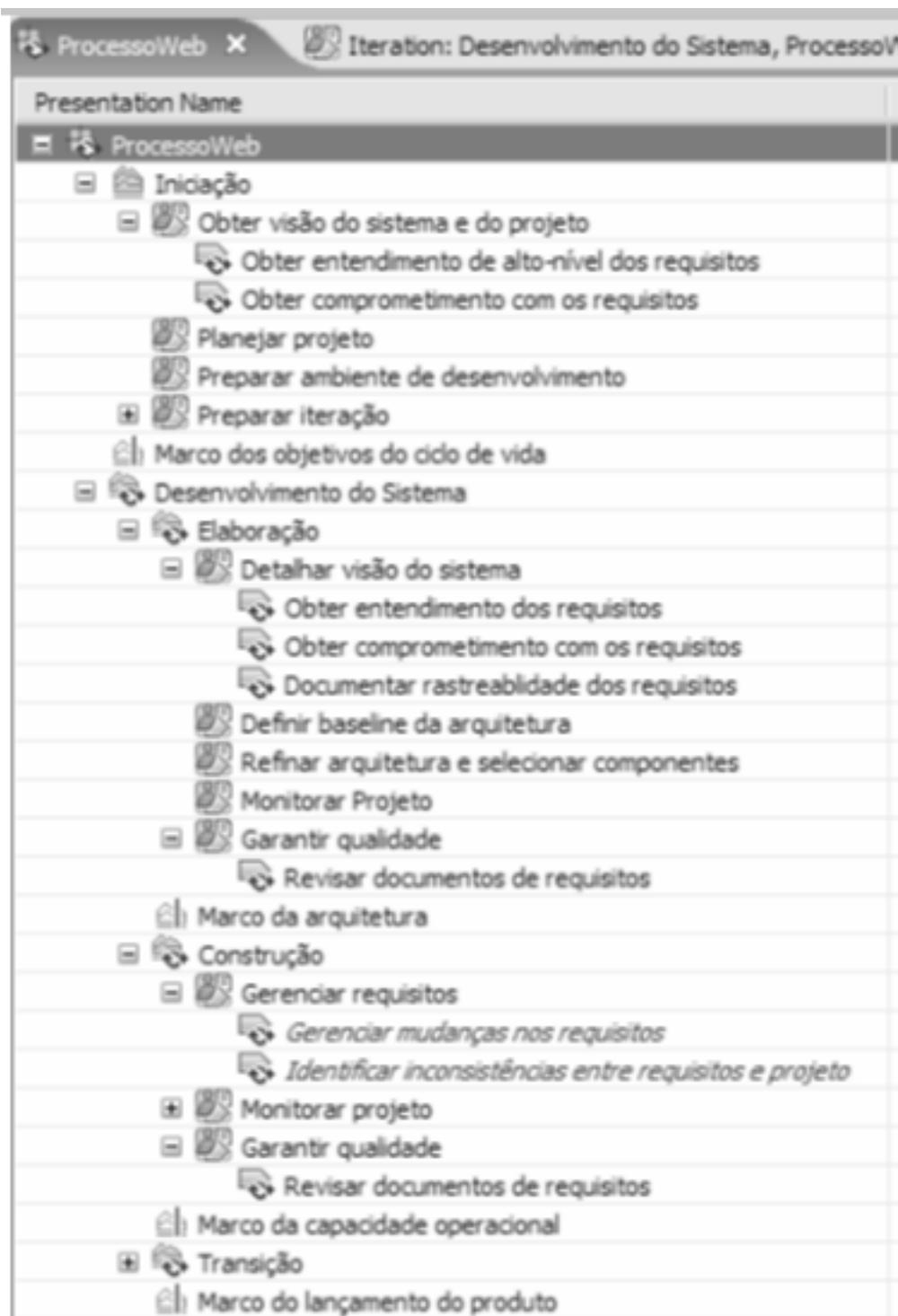
No exemplo de Oliveira (2008), os produtos de trabalho, ferramentas e guias foram sendo criados simultaneamente à elaboração do processo. Para criar a estrutura do processo e padronizar as fases de desenvolvimento, optou-se por utilizar as denominações existentes no RUP: Iniciação, Elaboração, Construção e Transição conforme demonstra a figura a seguir. Sendo que para cada uma das fases se definem as atividades e, neste caso, conforme figura a seguir, as atividades da fase de iniciação são comuns no NIT para todos os projetos e as demais fases passam a sofrer adaptações conforme tipo e estratégia do projeto a ser desenvolvido. Vale destacar que as atividades descritas definem o mínimo que deve ser realizado nesta fase.

FIGURA 71 - CAPABILITY PATTERN PARA REUTILIZAÇÃO DAS FASES DO RUP



FONTE: Oliveira (2007)

FIGURA 72 – PROCESSO DE DESENVOLVIMENTO DE SISTEMAS WEB NIT



FONTE: Oliveira (2007)

No EPF é possível criar um diagrama de atividade conforme figura a seguir para representar a sequência das tarefas. Como o processo herda as características do processo Padrão, o diagrama de atividades se apresenta bem semelhante. Como forma de melhor visualizar o processo *Web*, foi criado um diagrama de atividades para a fase de desenvolvimento do sistema.

FIGURA 73 - DIAGRAMA DE ATIVIDADE PARA AS FASES DE DESENVOLVIMENTO WEB



FONTE: Oliveira (2007)

Lembrando que cada atividade possui tarefas relacionadas, os papéis que executam.



Todas as informações para a definição das tarefas com seus papéis podem ser obtidas acessando o Apêndice D até o Apêndice G da monografia de Glória Maria de Paula Oliveira disponível no link: <http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0511001_07_postextual.pdf>.

RESUMO DO TÓPICO 3

Neste tópico, você aprendeu que:

- BPMN – *Business Process Model and Notation* é a notação gráfica lançado a partir de 2004 mais aceita para modelar processos. Ilustra o processo de uma maneira simples e clara utilizando uma gramática de símbolos para mapear, de maneira padrão, todos os processos de negócio de uma organização.
- Na modelagem BPMN existem quatro grupos de elementos: objetos de fluxo; objetos de conexão; raia de piscina; artefatos.
- Os objetos de fluxos são os principais elementos gráficos para definir o comportamento do processo de negócio com os tipos: eventos, atividades, decisões.
 - Eventos: algo que acontece ou pode acontecer em um processo. Estes eventos afetam o fluxo do processo e têm geralmente uma causa (*trigger*) ou um impacto (*result*).
 - Atividades: passos lógicos que ocorrem dentro do processo. É um termo genérico para um trabalho executado. Os tipos de atividades são: tarefas e subprocessos.
 - Decisões: chamados de *gateways*, que controlam o fluxo de sequência. São usados para controlar a divergência e a convergência da sequência de um fluxo. Assim, determinará decisões tradicionais, como juntar ou dividir trajetos.
- Os objetos de conexão representam a forma como os objetos de fluxo se conectam. Eles se dividem em três tipos também:
 - Fluxo de sequência: representa a ordem do fluxo, sua sequência.
 - Fluxo de mensagem: representa o fluxo das mensagens entre o emissor e o receptor.
 - Associação: usada para associar dados, textos e outros artefatos aos objetos do fluxo.
- O governo brasileiro, inclusive, estabeleceu a notação BPMN como obrigatória para todas as suas iniciativas em modelagem de processos, tanto as conduzidas com os profissionais da própria instituição quanto as conduzidas por consultorias. Exatamente por se tratar de um padrão aberto fortalecendo a linguagem única para a modelagem e gestão de processos de negócio, contribuindo para o ganho de produtividade e clareza na distribuição de informações.

AUTOATIVIDADE



- 1 Defina o que é o *Business Process Management Notation* (BPMN).
- 2 Definir produtos de trabalho, papéis, tarefas e guias referentes aos conteúdos de métodos do PF Composer.

REFERÊNCIAS

AGUIAR, Heron Vieira. **SPEM – Software Process Engineering Metamodel**. 2006. Disponível em: <<http://slideplayer.com.br/slide/2674010>>. Acessado em: 14 dez. 2016.

ALEIXO, F. **Eclipse Process Framework**. Disponível em: <http://docente.ifrn.edu.br/fellipealeixo/disciplinas/tads-2012/processo-de-desenvolvimento-de-software/material/07_epfcomposer.pdf>. Acesso em: 5 nov. 2016.

ALENCAR, F. M. R. **Mapeando a modelagem organizacional em especificações precisas**. 1999. p. 304 Tese (Doutorado) – Centro de Informática, Universidade Federal de Pernambuco, Recife, 1999.

ANJOS, Mateus Macedo dos. **Análise estruturada**: diagrama de fluxo de dados. Disponível em: <<https://pt.scribd.com/doc/313408122/Analise-Estruturada-pdf>>. Acesso em: 20 out. 2016.

ASSELVI. **Apostila análise de sistemas**. 2004. Disponível em: <<http://docsslide.com.br/documents/apostila-de-analise-sistemas.html>>. Acesso em: 22 out. 2016.

BANDINELLI, S. et al. Process Enactment in Spade. In: **European Workshop on Software Process Technology**, 2. ed. Norway. 1992.

BONFIM, Wagner. **Análise essencial e análise estruturada**. 2013. Disponível em: <<http://pt.slideshare.net/wagnerbonfim/anlise-essencial-e-anlise-estruturada>>. Acesso em: 22 out. 2016.

BORIA, J. L. et al. **A história da Tahini-Tahini**: melhoria de processos de software com métodos ágeis e modelo MPS. 2013. Disponível em: <<http://www.softex.br/wp-content/uploads/2015/11/Livro-PBQP-SW-Tahini-Tahini-PT-vFinal.pdf>>. Acesso em: 15 out. 2016.

BORIA, Jorge Luis et al. **A história da Tahini-Tahini**: melhoria de processos de software com métodos ágeis e modelo MPS. 2013. Disponível em: <<http://www.softex.br/wp-content/uploads/2015/11/Livro-PBQP-SW-Tahini-Tahini-PT-vFinal.pdf>>. Acesso em: 15 out. 2016.

CAMPOS, A. **Modelagem de processo**: notações. 2013. Disponível em: <<http://www.tiespecialistas.com.br/2013/01/modelagem-de-processos-notacoes/>>. Acesso em: 6 nov. 2016.

CARNEIRO, G. et al. **Usando o eclipse process framework na definição de um processo de software ágil**. 2008. Disponível em: <<https://www.researchgate.net/publication/256088891>>. Acesso em: 5 nov. 2016.

CASTRO, R. M. de; OLIVEIRA, W. M.; BRAGA, J. L. ARIS Express: exemplo prático de modelagem de processos. **Revista Engenharia de Software Magazine** 39. Disponível em: <<http://www.devmedia.com.br/aris-express-exemplo-pratico-de-modelagem-de-processos-revista-engenharia-de-software-magazine-39/21960>>. Acesso em: 5 nov. 2016.

CHAVES, T. **Artigo da Revista Engenharia de Software edição 21**. Disponível em: <<http://www.devmedia.com.br/artigo-engenharia-de-software-21-metricas-de-software/15776>>. Acesso em: 13 out. 2016.

CORDEIRO, M. A. **Métricas de software**. Disponível em: <<http://www.batebyte.pr.gov.br/modules/conteudo/conteudo.php?conteudo=88>>. Acesso em: 12 out. 2016.

COSTA, A. J. S. **Um mecanismo de adaptação de processos de software**. Dissertação de Mestrado. PPGCC/UFPA, Belém, Brasil. 2010.

CURTIS, B.; KELLNER, M. I.; OVER, J. **Process modeling**. Communications of the ACM. New Yourk, v. 35 n. 9, 1992, p. 75-90. 1992.

DOWSON M.; FERNSTRÖM, C. **Towards Requirements for Enactment Mechanisms**. In: B. Warboys (Ed.), *Software Process Technology*. Third European Workshop, EWSPT. 1994.

ENGHOLM JÚNIOR, Hélio. **Engenharia de software na prática**. São Paulo: Novatec Editora, 2011.

ENGHOLM Júnior, Hélio. **Engenharia de software na prática**. São Paulo: Novatec Editora, 2010.

ERIKSSON, H.E.; PENKER, M. *Objektorientering: Handbok och lexikon*. Lund, Sweden: Studentlitteratur, 2000.

FERMINO, Uderson. **6 ferramentas gratuitas para criar diagramas de processos com BPMN**. Disponível em: <<https://www.linkedin.com/pulse/6-ferramentas-gratuitas-para-criar-diagramas-de-com-bpmn-fermino?articleId=8208305741074523908>>. Acesso em: 2 nov. 2016.

FERNANDES, Aguinaldo Arugon; ABREU, Vladímir Ferraz de. **Implantando a governança de TI**: da estratégia à gestão dos processos e serviços. 4. ed. Rio de Janeiro: Brasport. 2014.

FILHO, Ly Freitas. **Diagrama de fluxo de dados**. Disponível em: <<http://www.lyfreitas.com.br/ant/pdf>>. Acesso em: 22 out. 2016.

FRANCISCANI, J.; PESTILI, L. C. **CMMI e MPS.BR**: um estudo comparativo. Disponível em: <<http://www.unicerp.edu.br/images/revistascientificas/3%20-%20>>

CMMI%20e%20MPS.BR%20Um%20Estudo%20Comparativo1.pdf>. Acesso em: 27 out. 2016.

GAMMA, E. et al. **Design patterns**: elements of reusable object-oriented *software*. Addison-Wesley Publishing Co., 1995.

GIMENES, I. M. S. et al. **FORMLAB**: um ambiente integrado de apoio aos métodos formais para o desenvolvimento de *software*. IDEAS'98 Workshop Ibero-americano de Engenharia de Requisitos e Ambientes de *Software*, Torres, Brasil, 1998.

GOMES, Luciano. **Medidas, métricas e indicadores na gestão de processos**. Disponível em: <<http://blog.iprocess.com.br/2014/05/medidas-metricas-e-indicadores/>>. Acesso em: 13 out. 2016.

GRUHN, V. **Process-Centered Software Engineering Environments**: a brief history and future Challenges. Annals of *Software Engineering*, v 14, p. 363-382. Kluwer, 2002.

HAUMER, P. **Eclipse Process Framework Composer – Part 1**: Key Concepts. 2007. Disponível em: <<http://www.eclipse.org/epf/general/EPFComposerOverviewPart1.pdf>>. Acesso em: 8 nov. 2016.

Humphrey, W. S. Managing the *software* process. Addison-Wesley Publishing, Company, Massachussets, 1990.

IBM. **Boas práticas para a elicitação de requisitos**. Disponível em: <https://www.ibm.com/developerworks/community/blogs/tlcbr/entry/boas_praticas_para_a_elicitacao_de_requisitos?lang=en>. Acesso em: 2 abr. 2016.

KELLNER, M. I. **Connecting Reusable Software Process Elements and Components**. In: Proceedings of the 10th International Software Process Workshop (ISPW '96), Dijon, France. 1996.

KOSCIANSKI, A.; Soares, M. S. **Qualidade de software – aprenda as metodologias e técnicas mais modernas para desenvolvimento de software**. 2. Ed. São Paulo: Novatec Editora, 2007.

LERNER, B. S. et al. **Modeling and managing resource utilizations in process, workflow and activity coordination**. New England. Departament of Computer Science, University of Massachusetts, 2000.

Lima, A. et al. **WebAPSEE**: Um Ambiente Livre e Flexível para Gerência de Processos de *Software*. VII Workshop de *Software* Livre. Porto Alegre. 2006.

LUIS BLOG. **Dicionário de dados – Modelo de entidade e relacionamento**. Disponível em: <<http://www.luis.blog.br/dicionario-de-dados.aspx>>. Acesso em: 22 out. 2016.

MACÊDO, Diego. **Métricas de processo e projeto de software**. Disponível em: <<http://www.diegomacedo.com.br/metricas-de-processo-e-projeto-de-software/>>. Acesso em: 13 out. 2016.

MACORATTI.NET. **Diagramação de software – D.F.D – II**. Disponível em: <http://www.macoratti.net/13/08/net_dfd2.htm>. Acessado em: 13 dez. 2016.

MELLO, Marcelo Santos de. **Melhoria de processos de software multimodelos baseada nos modelos MPS e CMMI-DEV**. Disponível em: <<http://www.cos.ufrj.br/uploadfile/1303916283.pdf>>. Acesso em: 13 out. 2016.

MÉTRICAS. Disponível em: <<http://startupsorocaba.com/wp-content/uploads/2014/08/m%C3%A9tricas-digitais.jpg>>. Acesso em: 23 out. 2016.

METRICS. Disponível em: <<https://i.ytimg.com/vi/RxZaZW2UgxA/hqdefault.jpg>>. Acessível em: 23 out. 2016.

MORAES, JANAINA B. D. **Engenharia de software 2: técnicas para levantamento de requisitos**. Disponível em: <<http://www.devmmedia.com.br/articles/viewcomp.asp?comp=9151>>. Acesso em: 22 mar. 2016.

MOREIRA, Ronney. ARIS Express: **Exemplo prático de modelagem de processos – Revista Engenharia de Software Magazine 39**. Disponível em: <<http://www.devmmedia.com.br/aris-express-exemplo-pratico-de-modelagem-de-processos-revista-engenharia-de-software-magazine-39/21960>>. Acessado: 14 dez. 2016.

MORO, Rodrigo Dal. **Avaliação e melhoria de processos de software: conceituação e definição de um processo para apoiar a sua automatização**. 2008. Disponível em: <http://inf.ufes.br/~falbo/files/Avaliacao_e_Melhoria_de_Processos_de_Software.pdf>. Acesso em: 25 out. 2016.

NOGUEIRA, Rhaíssa. **Introdução ao Business Process Modeling Notation (BPMN)**. 2015. Disponível em: <<http://www.devmmedia.com.br/introducao-ao-business-process-modeling-notation-bpmn/29892>>. Acesso em: 6 nov. 2016.

NUNES, D. J. **Estratégia data driven no desenvolvimento de software**. SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES. Instituto de Informática da UFGRS, v.1, p. 81-95. 1992.

OLIVEIRA, C. 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>>. Acessado em: 14 dez. 2016.

OLIVEIRA, Glória Maria de Paula; STAA, Arndt von. **Using software engineering concepts to define software development processes**. Rio de Janeiro, 2007. 92 p. Master Dissertation – Computer Science Department, Pontifical Catholic University of Rio de Janeiro.

OLIVEIRA, J. V. **Dicionário de Dados**. 2002. Disponível em: <<http://w3.ualg.pt/~jvo/ep/dd.pdf>>. Acesso em: 22 out. 2016.

OLIVEIRA, José Valente. Dicionário de dados. 2002. Disponível em: <<http://w3.ualg.pt/~jvo/ep/dd.pdf>>. Acesso em: 22 out. 2016.

OLIVEIRA, Rodrigo. **Conceitos introdutórios sobre melhoria e avaliação de processos de software**. Disponível em: <<http://www.devmedia.com.br/conceitos-introdutorios-sobre-melhoria-e-avaliacao-de-processos-de-software/10577>>. Acesso em: 26 out. 2016.

PAGLIARES, Rodrigo; GHIDELI, Arthur. Modelagem de processos na prática – Engenharia de Software Magazine 58. Disponível em: <<http://www.devmedia.com.br/modelagem-de-processos-na-pratica-engenharia-de-software-magazine-58/28048>>. Acessado em: 5 nov. 2016.

PAULA FILHO, Wilson de Pádua. **Engenharia de software: fundamentos, métodos e padrões**. 2000. Disponível em: <http://aulasprof.6te.net/Arquivos_Aulas/07-Proces_Desen_Soft/Livro_Eng_Soft_Fund_Met_Padroes.pdf>. Acesso em: 25 out. 2016.

PERBONI, Marcos. **Processo, qualidade e métricas de desenvolvimento de software**. Disponível em: <<https://marcosvperboni.wordpress.com/2013/02/15/processo-qualidade-e-metricas-de-desenvolvimento-de-software/>>. Acesso em: 13 out. 2016.

PESSOA, M. S. de P. **Introdução ao CMM – Modelo de Maturidade de Capacidade de Processo de Software**. Lavras: UFLA/FAEPE, 2003.

PMI (PROJECT MANAGEMENT INSTITUTE). **Um guia do conhecimento em gerenciamento de projetos** (Guia PMBOK®) – Quinta Edição. Newtown Square: Project Management Institute, 2013.

PORTER, M. E.: **Strategy and the internet**. Harvard Business Review, v. 79, n. 1, p. 63-78, March, 2001.

PRESMANN, R. **Engenharia de software: uma abordagem profissional**. 7. ed. Rio de Janeiro: Mc Graw Hill, 2011. Cap. 2.

PRESSMAN, R; MAXIM, Bruce. **Engenharia de software: uma abordagem profissional**. Editora Bookman. 2016. 8. ed. p. 968. São Paulo.

PRESSMAN, Roger S. **Engenharia de software: uma abordagem profissional**. São Paulo: Pearson Makron, 1995.

PRESSMAN, Roger S. **Engenharia de software: uma abordagem profissional**. 6. ed. São Paulo: Pearson Makron, 2006.

PRESSMAN, Roger. MAXIM, Bruce. **Engenharia de software**: uma abordagem profissional. Editora Bookman. 2016. 8^a edição, p. 968. São Paulo.

PRESSMANN, Roger S. **Engenharia de software**: uma abordagem profissional. 7. ed. Porto Alegre: AMGH, 2011.

REIS, C. A. **Uma abordagem flexível para execução de processos de software evolutivos**. 267 p. Tese (Doutorado) – Curso de Ciência da Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2003.

RODRIGUES, Joel. **Modelo Entidade Relacionamento (MER) e Diagrama Entidade-Relacionamento (DER)**. Disponível em: <<http://www.devmedia.com.br/modelo-entidade-relacionamento-mer-e-diagrama-entidade-relacionamento-der/14332>>. Acessado em: 22 out. 2016.

SANTOS, D. V. dos; VILELA, D.; SOUZA, C. de; CONTE, T. **Programas de melhoria de processo de software**: uma pesquisa sobre a influência dos aspectos humanos. 2015. Disponível em: <https://www.researchgate.net/profile/Tayana_Conte/publication/266067131_Programas_de_Melhoria_de_Processo_de_Software_Uma_pesquisa_sobre_a_influncia_dos_aspectos_humanos/links/54ad6b860cf2213c5fe3f205.pdf>. Acesso em: 25 out. 2016.

SANTOS, R. F. **Tutorial Visio®**. 2006. Disponível em: <http://suporte.buicklogistica.com.br/menu/manual/visio/manual_visio_completo.pdf>. Acesso em: 5 nov. 2016.

SEGPLAN. **Modelagem de processos com Bizagi Modeler**. 2014. Disponível em: <<http://www.sgc.goiás.gov.br/upload/arquivos/2014-10/manual-de-padronizacao-de-modelagem-de-processos-usando-bizagi---v3-1.pdf>>. Acesso em: 29 out. 2016.

SEI – SOFTWARE ENGINEERING INSTITUTE. CMMI Acquisition Module (CMMIAM) Version 1.1 (CMU/SEI-2005-TR-011). Pittsburgh, PA: *Software Engineering Institute, Carnegie Mellon University, May 2005*. Disponível em: <<http://www.sei.cmu.edu/library/abstracts/reports/05tr011.cfm>>. Acessado em: 14 dez. 2016.

SGANDERLA, K. **7 Ferramentas gratuitas para criar diagramas de processos com BPMN**. Disponível em: <<http://blog.iprocess.com.br/2016/09/7-ferramentas-gratuitas-para-criar-diagramas-de-processos-com-bpmn/>>. Acesso em: 3 nov. 2016.

SGANDERLA, Kelly. **Um guia para iniciar estudos em BPMN (I): atividades e sequência**. 2012. Disponível em: <<http://blog.iprocess.com.br/2012/11/um-guia-para-iniciar-estudos-em-bpmn-i-atividades-e-sequencia/>>. Acesso em: 6 nov. 2016.

SILVA, Elvis Ferreira da; SOUZA, Marta Alves de; COSTA, Helder Rodrigues da. **Vantagens da aplicação do programa de melhoria de processo de software brasileiro mps.br nos ambientes de desenvolvimento de software**. 2011. Disponível em: <http://revistapensar.com.br/tecnologia/pasta_upload/artigos/a14.pdf>. Acessado em: 14 dez. 2016.

SILVA, F. A. D. **Estudo sobre execução, validação e simulação de processos de software**. Trabalho Individual. PPGC-UFRGS. 1999.

SODRÉ, E. B. **Mps Br – melhoria do processo de software brasileiro**. Disponível em: <http://www.techoje.com.br/site/techoje/categoria/detalhe_artigo/245>. Acesso em: 13 out. 2016.

SOFTEX.BR. **MPS.BR - Melhoria de Processo do Software Brasileiro: Guia Geral MPS de Software**. 2012. Disponível em: <http://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_Geral_Software_2012-c-ISBN-1.pdf>. Acesso em: 14 dez. 2016.

SOMMERVILLE, I. **Engenharia de software**. 8. ed. Rio de Janeiro: Pearson, 2007. Cap. 4.

SOMMERVILLE, I. **Engenharia de software**. 9. ed. Pearson Education, 2011.

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

SOMMERVILLE, Ian. **Engenharia de software**. São Paulo: Addison Wesley, 2003.

SOUZA, A., REIS, C. A. L., e REIS, R. Q. Interação humana durante execução de TECHSOUPE BRASIL. **Microsoft Visio**: a melhor ferramenta para criação de diagramas. Disponível em: <<https://www.techsoupbrasil.org.br/node/11863>>. Acesso em: 5 nov. 2016.

TECHSOUPE BRASIL. **Microsoft Visio**: a melhor ferramenta para criação de diagramas. Disponível em: <<https://www.techsoupbrasil.org.br/node/11863>>. Acessado em: 5 nov. 2016.

TEIXEIRA, Marcel Neves. **Análise estruturada de sistemas**. Disponível em: <http://www.3msolucoes.com.br/adm/downloads/AE_Aulas_final.pdf>. Acesso em: 22 out. 2016.

THIRY, Marcello. **Avaliação e melhoria de processos de software**. Disponível em: <<http://www.univali.br/ensino/pos-graduacao/mestrado/mestrado-em-computacao-aplicada/temas-de-pesquisa/avaliacao-e-melhoria-de-processos-de-software/Paginas/default.aspx>>. Acessado em: 25 out. 2016.

VASCONCELOS, Alexandre Marcos Lins de; ROUILLER, Ana Cristina; MACHADO, Cristina Ângela Filipak; MEDEIROS, Teresa Maria Maciel de.

Introdução À Engenharia De Software e à Qualidade de Software. Disponível em: <<http://docslide.com.br/documents/mod01mps-engenhariaqualidadesoftware-v280906.html#>>. Acesso em: 14 dez. 2016.

VILELA, Vagner. Ferramentas Case – Parte I. Disponível em: <<http://www.devmedia.com.br/ferramentas-case-parte-i/1505>>. Acesso em: 20 out. 2016.

YOUNG, P. customizable process specification and enactment for technical and non-technical users. Ph.D. Thesis, University of California, Irvine, USA, 1998.

Yu, E. SK. Modeling strategic relationships for process reengineering (Phd Thesis). University of Toronto, 1995.

ANOTAÇÕES