

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/341327200>

Analysis, Modeling and Metrics

Book · January 2015

CITATIONS

0

READS

116

1 author:



[Alvaro F. Pinheiro](#)

University of Pernambuco

29 PUBLICATIONS 5 CITATIONS

SEE PROFILE

Fundamentos da Engenharia de Software

Análise, Modelagem e Métricas

PINHEIRO, Álvaro Farias
Autor

Fundamentos da Engenharia de Software

Análise, Modelagem e Métricas

Publicação 2015

O autor acredita que todas as informações aqui apresentadas estão corretas e podem ser utilizadas para qualquer fim legal. Entretanto, não existe qualquer garantia explícita ou implícita, de que o uso de tais informações conduzirá sempre ao resultado desejado. Os nomes de sites e empresas, por ventura, mencionados, foram utilizados apenas para ilustrar os exemplos, não tendo vínculo nenhum com o livro, não garantindo a sua existência nem divulgação. Eventuais erratas estarão disponíveis para download no site de publicação.

Algumas imagens utilizadas neste livro foram obtidas a partir da Internet.

Dados da Publicação

Pinheiro, Álvaro Farias

Fundamentos da Engenharia de Software: Análise, Modelagem e Métrica

1. ed. – Volume III

Recife, 2015

Selo Editorial: Publicação Independente

Bibliografia

ASIN B016G1TFFC

ISBN: 9781520170329

1. Análise de Requisitos
2. Análise e Modelagem
3. Análise de Ponto de Função
4. UML

Fundamentos da Engenharia de Software

Análise, Modelagem e Métricas

Publicação Independente
Livro em português com o título
Fundamentos da Engenharia de Software
Análise, Modelagem e Métricas
Volume 3
1ª Edição
Recife - Pernambuco – Brasil
Primeira revisão em Outubro de 2015
Segunda revisão em Junho de 2016

Fundamentos da Engenharia de Software

Análise, Modelagem e Métricas

À minha formosa esposa, Fabiana Albuquerque de
Freitas Pinheiro que sem a sua constante
colaboração não teria paz para essa empreitada.

Fundamentos da Engenharia de Software

Análise, Modelagem e Métricas

“As pessoas apostam seus empregos, seu conforto, sua segurança, sua diversão, suas decisões e suas próprias vidas nos softwares de computadores. Eles precisam estar certos.”
Roger Pressman

SUMÁRIO

1 Apresentação.....	1
2 Introdução à Processos de Desenvolvimento.....	6
2.1 Evolução.....	7
2.2 Modelo Cascata.....	9
2.3 Modelo Prototipação.....	9
2.4 Modelo Espiral.....	10
2.5 Modelo Iterativo e Incremental	11
2.6 Metodologia Tradicional.....	12
2.7 Metodologia Ágil	12
2.8 Processo RUP	13
2.9 Processo Scrum	14
2.10 Metodologias-Modelos-Processos-Técnicas	15
2.11 Ferramentas	16
2.12 Sugestões de Leitura.....	16
2.13 Referências Web	16
2.14 Referências Bibliográficas	17
3 Introdução à Engenharia de Requisitos	18
3.1 Estudo de Viabilidade	19
3.2 Levantamento de Requisitos	20
3.3 Elicitação de Requisitos.....	21
3.4 Análise de Requisitos	22
Utilizador	23
Sistema	24
Design.....	24
Especificação de Requisitos ou Documento de Requisitos de Sistema (DRS	24
3.5 Atividades.....	32
3.6 Artefatos	32
3.7 Sugestões de Leitura.....	33
3.8 Referências Web	33
3.9 Referências Bibliográficas	33

SUMÁRIO

4	Introdução à Análise e Modelagem.....	34
4.1	Desenvolvimento de <i>Software</i>	34
4.2	<i>Adhoc</i> (Desenvolvimento por Demanda)	35
4.3	Projetar (Desenvolvimento com Planejamento).....	37
4.4	Modelagem.....	39
4.5	Linguagem de Modelagem Unificada (UML)	39
4.5.1	Diagrama de Casos de Uso	41
4.5.2	Diagrama de Classes	46
4.5.3	Diagrama de Objetos	51
4.5.4	Diagrama de Sequência e Comunicação ou Colaboração	51
4.5.5	Diagrama de Atividades	53
4.5.6	Diagrama de Estados	54
4.5.7	Diagrama de Componentes	55
4.5.8	Diagrama de Distribuição	56
4.5.9	Diagrama de Tempo.....	56
4.6	Exemplos.....	56
4.7	Ferramentas	59
4.8	Sugestões de Leitura.....	60
4.9	Referências Web	60
4.10	Referências Bibliográficas	61
5	Introdução à Análise e Projeto Orientado a Objetos	62
5.1	Análise Orientada a Objetos.....	62
5.2	Modelagem Orientada a Objetos	65
5.3	Projeto de <i>Software</i>	71
5.4	Artefatos de um Projeto de <i>Software</i> (<i>Templates</i>).....	74
5.5	Ferramentas	76
5.6	Apresentações (pptx)	76
5.7	Vídeos	76
5.8	Sugestões de Leitura.....	76
5.9	Referências Web	76
5.10	Referências Bibliográficas	78
6	Introdução à Medidas de Esforço	79

SUMÁRIO

6.1 Conceitos.....	81
6.2 Terminologias.....	81
6.3 Análise de Ponto de Função.....	82
6.3.1 Cálculo do Ponto de Função	82
6.3.2 Técnica.....	83
6.4 Ferramentas	89
6.5 Apresentações.....	89
6.6 Sugestões de Leitura.....	89
6.7 Referências Web	90
6.8 Referências Bibliográficas	90
7 Referências Bibliográficas	91
8 Referências Web	93
9 Biografias.....	96
10 Artefatos (Templates) para Projetos de <i>Software</i>	101
11 Ferramentas	103
12 Apresentações e Vídeos.....	104
13 Sugestões de Leitura.....	105

ILUSTRAÇÕES

Figura 1.1 Desenvolvimento Adhoc	5
Figura 2.1 Processo.....	6
Figura 2.2 Evolução dos Processos	7
Figura 2.3 Evolução da UML	8
Figura 2.4 Modelo Cascata.....	9
Figura 2.5 Modelo Prototipação.....	10
Figura 2.6 Modelo Espiral	11
Figura 2.7 Modelo Iterativo e Incremental.....	12
Figura 2.8 Metodologia Tradicional.....	12
Figura 2.9 Metodologia Ágil	13
Figura 2.10 Processo RUP	14
Figura 2.11 Processo Scrum	15
Figura 3.1 Percentual de sucesso	25
Figura 3.2 Custo de correção por tempo de identificação do erro	25
Figura 3.3 Percentual de inserção de erros	27
Figura 4.1 <i>Chaos</i>	35
Figura 4.2 Desenvolvimento <i>Adhoc</i>	36
Figura 4.3 Evolução da UML	40
Figura 4.4 Diagramas da UML.....	41
Figura 4.5 Elementos do Diagrama de Casos de Uso	41
Figura 4.6 Símbolos da UML	42
Figura 4.7 Diagrama de Caso de Uso.....	44
Figura 4.8 Diagrama de Casos de Uso com Inclusão.....	45
Figura 4.9 Diagrama de Casos de Uso com Extensão	45
Figura 4.10 Diagrama de Casos de Uso Realizar Matrícula	45
Figura 4.11 Elementos do Diagrama de Classes.....	47
Figura 4.12 Classe.....	47
Figura 4.13 Diagrama de Classes com Herança	48
Figura 4.14 Diagrama de Classes com Associação.....	49
Figura 4.15 Diagrama de Classes com Agregação.....	49
Figura 4.16 Diagrama de Classes com Composição	49
Figura 4.17 Diagrama de Classes com Realização	49
Figura 4.18 Diagrama de Classes com Dependência.....	50
Figura 4.19 Diagrama de Classes com Relacionamento	50
Figura 4.20 Diagrama de Classes com Comentário	51
Figura 4.21 Elementos do Diagrama de Sequência	51
Figura 4.22 Diagrama de Sequência	52
Figura 4.23 Diagrama de Comunicação	52
Figura 4.24 Elementos do Diagrama de Atividades.....	53
Figura 4.25 Diagrama de Atividades.....	53
Figura 4.26 Diagrama de Estados	54
Figura 4.27 Diagrama de Componentes.....	55
Figura 4.28 Diagrama de Distribuição.....	56
Figura 4.29 Especificação Resumida de Casos de Uso	57
Figura 4.30 Exemplo de Diagrama de Casos de Uso Consultar Cliente	58
Figura 4.31 Exemplo de Diagrama de Classes Consultar Cliente	58

ILUSTRAÇÕES

Figura 4.32 Exemplo de Diagrama de Estados Consultar Cliente	59
Figura 6.1 Análise de Ponto de Função.....	84
Figura 6.2 Tabela Complexidade ALI e AIE.....	84
Figura 6.3 Tabela Complexidade EE	85
Figura 6.4 Tabela Complexidade SE e CE	85
Figura 6.5 Tabela de Contribuição.....	85
Figura 6.6 Exemplo de Planilha de Contagem (PC)	86
Figura 6.7 Passos da Contagem	88

SIGLAS e ACRÔNIMOS

BFPUG - *Brazilian Function Point Users Group*
BPMP - *Business Process Management Initiative*
IRUP - *IBM Rational Unified Process*
ISO - *International Organization for Standardization*
PMBOK - *Project Management Body of Knowledge*
PMI - *Project Management Institute*
QSM - *Quantitative Software Management*
SEI - *Software Engineering Institute*
SWEBOK - *Software Engineering Body of Knowledge*
UML - *Unified Modeling Language*

NOTAÇÕES

Este livre adota as seguintes notações, que devem ser observadas para o perfeito entendimento do material aqui produzido.

Notações Técnicas:

<..> - Menor que / Maior que, nos comandos SQL usado para metadados.

<..> - Menor / Maior, no texto indica que é um *hyperlink* (*link* para um endereço web).

[..] - Abre e fecha parênteses, no texto indica utilização de uma sigla.

(..) - Abre e fecha colchetes, nos comandos SQL indica opcionalidade.

| - Barra, para indicar “ou lógico”.

_ - Sublinhado, nos comandos SQL indica o *default* (padrão).

_ - Sublinhado, no texto indica que é um *hyperlink* (*link* para um endereço web).

Negritos - Títulos de capítulos, seções e subseções.

Itálico - Para palavras de língua estrangeira.

SOBRE O AUTOR



Álvaro Farias Pinheiro tem mais de 30 anos de experiência no campo da computação e nestes, mais de 15 anos na área acadêmica, lecionando disciplinas com foco em engenharia de software.

É mestre em Engenharia de Software pelo Centro de Estudos e Sistemas Avançados do Recife (CESAR) com a dissertação *Uso do Balanced Scorecard na Avaliação de Fábricas de Software* utilizadas como Instrumento de Ensino de Tecnologia da Informação Baseado em Problemas Reais. Especialista em Gestão Pública pela Universidade de Pernambuco (UPE) e em

Metodologias de Desenvolvimento pela União Brasileira de Tecnologia. É Bacharel em Sistemas de Informação pela Faculdade Integrada do Recife (FIR), iniciando sua vida acadêmica no Bacharelado em Ciência da Computação da Universidade Católica de Pernambuco (UNICAP).

Iniciou as atividades profissionais em 1986 como Programador na Elógica Microsistemas desenvolvendo para baixas e altas plataformas nas linguagens dBase, Clipper, FoxPro, Cobol e Natural. Como Analista de Sistemas passou pelas instituições: Centro de Prestação de Serviços Técnicos de Pernambuco (CETEPE), Empresa de Fomento da Informática do Estado de Pernambuco (FISEPE), Instituto de Tecnologia em Informática (ITECI), Apply Solutions e está hoje na Agência Estadual de Tecnologia da Informação do Estado de Pernambuco (ATI-PE) por concurso público, trabalhando como Analista em Gestão da Tecnologia da Informação e Comunicação, em exercício na Procuradoria Geral do Estado de Pernambuco (PGE-PE), atuando como Coordenador da Área de Sistemas.

Começou a lecionar em 1992 em cursos técnicos nas empresas ITECI e IBRATEC, e desde 2006 é professor de cursos de graduação e pós-graduação em computação nas instituições de ensino superior da região, a citar, Universidade Federal Rural de Pernambuco (UFRPE), Instituto Federal de Pernambuco (IFPE), Faculdade Estácio do Recife, Faculdade Integrada do Recife (FIR), Faculdade Joaquim Nabuco, Faculdade do Recife, Unibrattec e durante esses anos realizou diversos cursos de extensão como os de Análise de Ponto de Função, Administração de Redes, Linguagens de Programação Natural, Centura, Delphi e CSharp, Banco de Dados Adabas, MySQL, Postgre e Oracle, Análise Estruturada e Orientada a Objetos, entre outros.

Possui experiência na área de Engenharia de *Software* nas disciplinas de Teoria da Computação, Sistemas de Informação e Colaborativos, Desenvolvimento Estrutural e Orientado a Objetos para aplicações Desktop, Web e Mobile, Processos de Desenvolvimento IRUP, XP e Scrum, Banco de Dados SQLServer

SOBRE O AUTOR

e Oracle, e Gestão de Projetos com PMBOK e SWEBOK. Trabalha atualmente na coordenação de projetos e com educação presencial e a distância, ministrando aulas e realizando pesquisas em ambientes virtuais de ensino e linguagens de programação.

<http://www.alvarofpinheiro.eti.br>

<http://br.linkedin.com/in/alvarofpinheiro>

<https://sites.google.com/site/alvarofpinheiro>

<http://www.facebook.com/alvarofpinheiroaulas>

PREFÁCIO

A realização deste livro é motivada pela necessidade de mostrar aos engenheiros de software que independente da tecnologia utilizada para o desenvolvimento de um aplicativo, sem se conhecer as reais necessidades dos demandantes do sistema não há como construir uma efetiva solução que seja útil ao demandante.

Também é evidenciar que sem engenharia de requisitos aplicada à construção, por melhor que o sistema seja nos aspectos de segurança, desempenho, usabilidade, linguagem e banco de dados escolhido para o desenvolvimento, se não atender as premissas solicitadas pelo demandante, os requisitos, o sistema não terá funcionalidade adequada e assim não atenderá ao principal objetivo de uma sistema, que é, resolver o problema do demandante.

Assim este material foi montado através da compilação dos conhecimentos adquiridos nos mais de trinta anos de desenvolvimento para evidenciar a necessidade do uso da análise de requisitos e facilitar seu entendimento.

A integração dos conteúdos foi realizada buscando a lógica para fundamentar os conceitos aqui mostrados, assim, sugiro que a leitura deste livro seja feita de forma sequencial para melhor aproveitamento dos assuntos, porém nada impede que se utilize o capítulo desejado.

O livro aborda os seguintes assuntos:

- **Capítulo 1 Apresentação.** Neste capítulo será apresentado os motivos que fazem a necessidade da existência da engenharia de requisitos, evidenciando a análise como construção do entendimento das causas e efeitos gerados pelos problemas levantados pelos demandantes de software e confirmando a necessidade do uso de processos na construção de projetos de software para chegar ao objetivo, que é o sistema, baseados nos meios e fins da engenharia de software.
- **Capítulo 2 Introdução à Processos de Desenvolvimento.** Inicia-se nesse capítulo os conceitos dos processos de desenvolvimento de *software*, evidenciando a necessidade de qualidade na construção de *software*, conceituando a diferença entre desenvolvimento *Adhoc* e com processo.
- **Capítulo 3 Introdução à Engenharia de Requisitos.** Neste abordamos as técnicas para um levantamento de requisitos que forneça subsídios necessários para uma construção de *software* com maior qualidade, enfatizando a necessidade de se aplicar a engenharia de requisitos na construção de um sistema visando um *software* com menos *bugs*.
- **Capítulo 4 Introdução à Análise e Modelagem.** Neste capítulo aborda-se a linguagem de modelagem unificada para diagramar um projeto de *software*,

PREFÁCIO

explicando a necessidade do uso de modelos na construção de um *software*.

- **Capítulo 5** *Introdução à Análise e Projeto Orientado a Objetos*. Aqui o foco está no paradigma orientado a objetos com os conceitos de requisitos e modelagem integrados objetivando a construção de um *software* com qualidade.
- **Capítulo 6** *Introdução à Medidas de Esforço de Desenvolvimento*. Todo projeto precisa informar seu prazo e custo de desenvolvimento e a resposta para essas indagações devem ser dadas através de uma métrica, assim aqui é abordado à análise de ponto de função como medida de esforço de desenvolvimento de *software* em um projeto que visa cumprir o planejamento, assim criando parâmetros factíveis de serem atendidos.

Finalizando, este livro possui várias subseções que disponibilizam *hyperlink's* para *wiki's* sobre os assuntos abordados, como também *link's* para outros *ebook's* utilizados como referencial bibliográfico para o conteúdo produzido. Também *Links* para os sites oficiais dos temas estudados, alguns com vídeos, além de atividades propostas, exemplos e biografia dos principais estudiosos do tema.

Álvaro Farias Pinheiro

1 Apresentação

A Engenharia de Software é um segmento da Computação que existe decorrente da necessidade de se construir *software* com qualidade, buscando planejar e organizar o desenvolvimento e dentro deste campo o uso de processos de software nessa construção mostra-se ao longo dos anos uma necessidade que foi evidenciada com a crise do software ocorrida na década de sessenta.

No uso de processos de software a aplicação de fases e disciplinas nas etapas de desenvolvimento vem se mostrando uma necessidade na construção organizada de projetos de software, o que é inverso do desenvolvimento Adhoc que traduzo desenvolvimento por demanda, isto é, o usuário pede e o desenvolvedor faz, sem planejamento e assim tudo ocorre com base na experiência do desenvolvedor e consequentemente os prazos e custos são palpites e não dados estimados.

O texto a seguir demonstra a consequência do uso do desenvolvimento Adhoc, com o intuito de fazer o melhor entendimento da grande necessidade de se analisar e modelar corretamente um projeto de software vamos inserir neste livro uma parte do texto retirado do artigo Um Mundo Feito (quase completamente) de *Software* de Silvio Lemos Meira que é Professor Titular de Engenharia de Software da Universidade Federal de Pernambuco e Cientista-Chefe do Centro de Estudos e Sistemas Avançados do Recife (CESAR). Segue a citação:

[...] Ao entrar no restaurante, a hostess lhe informa, gentil e sorridente, dos resultados de uma pesquisa sobre os trinta mil últimos pedidos realizados na casa. Em números redondos, 28% dos pratos servidos eram o que os fregueses haviam pedido, chegaram no tempo previsto e pelo preço do menu. Em 49% dos casos, alguma coisa deu errado no prato, no preço, no prazo ou numa combinação de todos. Teve gente que pediu bacalhau e recebeu tapioca, pelo mesmo preço, tempos depois. Em muitos casos, as discussões quase chegaram às vias de fato; em pelo menos um, o cliente sacou uma 9000S e queria mandar o chefe direto para o banquete celestial. Enfim, nos outros 23% dos pedidos, os pratos e os clientes não se encontraram nunca; às vezes porque os primeiros demoraram tanto que os clientes desistiram; noutras, porque o chefe veio à mesa, mais de uma vez, para aumentar o preço; noutras, porque o cliente descobriu que o prato ia chegar, mas seria muito diferente do pedido. Em

suma, sua chance de sair dali realmente satisfeito é de 28%; a pergunta que a hostess lhe faz é... "Posso levar-lhe à mesa, senhor?" Você iria ou não? Nunca vi nenhuma recepção assim, em restaurante nenhum. Até porque a vasta maioria deles tem um recorde melhor do que este. Mas não acho que entraria num buraco, por mais chique que fosse, onde minha chance de ser bem servido, comer bem e pagar o combinado fosse uma em quatro. E não acho que haja tantos gastro-masoquistas por aí, a ponto de manter restaurantes do tipo acima abertos. Só que os números dizem respeito a projetos de software e são do Standish Group, empresa que realiza, desde 1994, um levantamento bastante sofisticado sobre projetos de software realizados em empresas norte-americanas. O campo é fértil, pois o gasto anual americano em projetos de aplicações de software chega a US\$275 bilhões, com mais de 200 mil projetos em andamento. O Standish Group levantou a história de 30 mil desses projetos desde 1994 e, para alguns deles, realizou estudos detalhados. Os dados e as conclusões revelam o tamanho dos problemas que ainda existem em desenvolvimento de software, bem como o tamanho da oportunidade para quem tentar e conseguir resolvê-los.

A situação vem melhorando, e de forma muito significativa em alguns aspectos. Em 1994, o aumento médio de preço nos projetos chegava a 189%, tendo caído para 45% em 2000. Uma senhora queda, diga-se não tão de passagem assim, que reflete o aumento da preocupação, atenção e da competência mundial em relação ao desenvolvimento de software, que está passando a ser a ferramenta mais básica e fundamental dos processos econômicos e sociais. É quase impossível se passar um dia sem sofrer o efeito de algum tipo de sistema de informação; para nosso bem, portanto, é bom que eles funcionem, a contento, no prazo e no preço que, por exemplo, o supermercado combinou. Senão, sempre pagaremos uma parte do prejuízo, seja em aumento de preços e/ou piores serviços, ou em diminuição de competição, pois software de má qualidade, fora do prazo ou muito caro (ou os três, vez por outra) pode ser falência certa. Um dos casos mais rumorosos pode ter destruído, em 1996, um dos maiores distribuidores de medicamentos dos EUA, a FoxMeyer Drug Co., que faturava US\$5 bilhões por ano.

Os liquidantes processaram a SAP e a Andersen Consulting (hoje Accenture), pedindo uma compensação de US\$ 1 bilhão, processo que deverá ser julgado este ano, correndo o risco de criar jurisprudência nos EUA.

Alguns críticos acham que as companhias “não aprendem com fracassos anteriores”, mas o fato é que o estado da prática de construção de software vem mudando, e para muito melhor, nos últimos 15 anos. Segundo dados do mesmo estudo do Standish Group, o atraso médio dos projetos de software caiu de 222% do tempo inicialmente previsto, em 1994, para 63% em 2000; em 1994, apenas 61% da funcionalidade prevista era entregue, em média, número que subiu para 67%. Pode não ser essas coisas todas saber que sua encomenda de software, se estiver na média, custará 45% mais caro do que o contratado, terá um atraso de 63% sobre o prazo previsto para entrega e não terá 33% das funcionalidades encomendadas. Mas há de se convir (compare acima) que houve um progresso gigantesco.

Construir software não é uma atividade simples; o problema que se trata é normalmente o da replicação virtual de processos, sistemas e, na maioria das vezes, instituições e redes delas. Fazer com que o item cujo estoque caiu abaixo do mínimo seja encomendado e reposto na prateleira do supermercado é muito fácil... desde que todos os processos, sistemas e instituições envolvidos sejam individualmente entendidos e se entendam entre si. Na maioria dos casos, é muito difícil, em projetos complexos, de porte e de longo prazo, manter a instituição estática, esperando que o projeto se complete. Na prática, os requisitos que um grande sistema tem que atender vão mudando à medida em que ele vai sendo construído, como se seu pedido para a cozinha do restaurante fosse sendo mudado enquanto está sendo preparado: o chefe começa fazendo seu sirigado com arroz de frutos do mar e, na hora em que o prato chega, seu pedido já era javali ao barolo. Não dá.

Grandes sistemas são como o que talvez tenha falido a FoxMeyer: a encomenda custou US\$35 milhões. Segundo o Standish Group, a taxa de sucesso para projetos acima de US\$10 milhões, que tipicamente envolvem quinhentas ou mais pessoas por períodos de

três anos ou mais, é estatisticamente zero! Para projetos de até US\$750 mil, que tipicamente envolvem seis pessoas por seis meses, a taxa de sucesso é 55%. Para projetos pequenos ("micro-projetos"), que custam até US\$250 mil e duram de três a quatro semanas, taxas de sucesso de 70% podem ser esperadas depois de alguma prática e há razões para se pensar que, havendo um processo tão afinado quanto a cozinha dos melhores restaurantes, as taxas de sucesso possam chegar perto dos 100%.

A receita do Standish Group - Chaos: a recipe for success – (e de outros, como Donald Reifer) para o sucesso inclui envolvimento e apoio dos executivos e usuários, gerentes de projeto experientes, objetivos claros e escopo reduzido. Estes poucos itens, bem cuidados, seriam responsáveis por 70% das chances de sucesso de projetos de software. A receita é mais simples do que parece: gasta muito esforço, envolvimento de todas as pessoas-chave das instituições, educação contínua e implantação de métodos e processos claros e efetivos. A mudança é de longo prazo, mas é para sempre e começa por quem contrata. Se for você, comece por perguntar, da próxima vez, pelos métodos e processos de quem você vai contratar.[...]

Fonte: Um Mundo Feito (quase completamente) de Software.

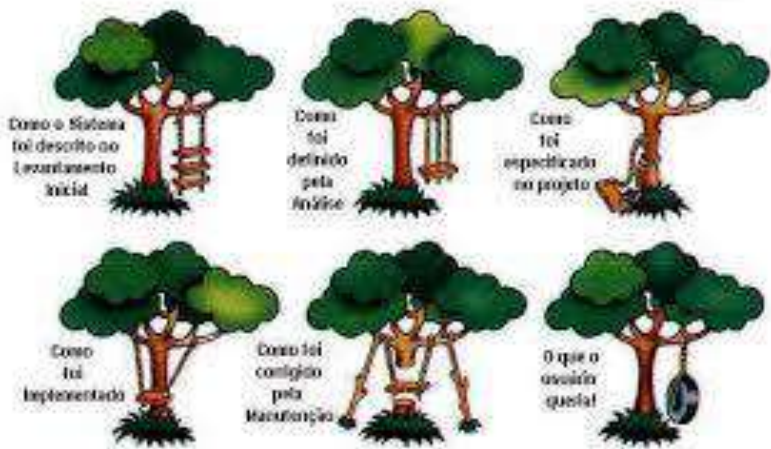
Disponível em: < <http://goo.gl/YE26CS> >

Agora vamos nos perguntar. Porque só 28% dos projetos de software terminam com sucesso? Será que os programadores são inexperientes ou as tecnologias são ineficientes ou será que os analistas não usam a análise de requisitos da forma correta? Não é o caso desta produção investigar os fatores que levam a esse indicador e sim subsidiar a análise com os conceitos que podem minimizar as falhas nos projetos de software em relação ao seu entendimento e por consequência sua construção.

Caso o desenvolvimento não dê a atenção devida à etapa de análise na construção do software e priorize outras etapas como a codificação e o banco de dados, não adiantará está usando a linguagem mais moderna do paradigma de desenvolvimento mais utilizado com o banco de dados com maior potencialidade, se o produto do projeto do software, que é o sistema, não atende adequadamente as necessidades do demandante ou usuário.

O gráfico a seguir demonstra bem o que está dito acima.

Figura 1.1 Desenvolvimento Adhoc
AS FASES DO DESENVOLVIMENTO DE SISTEMAS

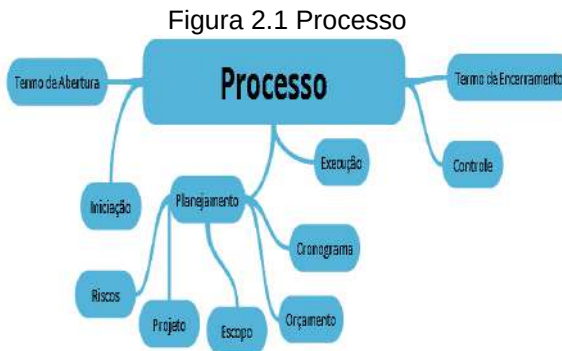


Observar pela imagem o que foi entregue não atendia a demanda solicitada e para manter evolutivamente, corretivamente e adaptativamente o exemplo acabamos nos distanciando ainda mais do que é realmente desejado.

2 Introdução à Processos de Desenvolvimento

Bom, antes de tudo, vamos conhecer as diferenças entre os termos metodologia, processo, modelo e técnica, que alguns fazem seu uso como se fossem sinônimos, o que não são.

Metodologia é um conjunto de práticas. Já as práticas são um conjunto de ações estruturadas que podem ser repetidas várias vezes durante um processo de desenvolvimento de *software*. Assim sendo uma metodologia é um conjunto que envolve várias práticas como gerenciamento de projetos, engenharia de requisitos, análise e modelagem, codificação, testes, gerencias, como as de configuração, ambiente, risco, qualidade, aquisição, entre várias outras, que vão depender do processo adotado..



Fonte: Próprio Autor

Agora vamos definir Processo e Modelo. Apesar de se confundirem, não se trata da mesma coisa. Podemos dizer que processo de desenvolvimento de *software* é um conjunto de atividades, parcialmente ordenadas, com a finalidade de obter um produto de *software*. Realizando o mais corretamente possível as etapas de uma construção de *software*. Os modelos são as atividades que fazem parte de um processo de engenharia de *software*. É o primeiro passo para se construir um processo, pois o mesmo deve ser baseado em um modelo. Concluindo as definições, podemos definir Técnica como sendo um procedimento ou mesmo um conjunto de procedimentos, que tem como objetivo alcançar um determinado resultado. E segundo os estudiosos do assunto, se caracteriza por ser consciente reflexiva e inventiva.

Em uma análise mais básica, teríamos. Metodologia como um conjunto de técnicas (práticas) que repetidas sempre tendem a chegar ao mesmo resultado esperado. E um processo é a forma de organizar essas técnicas, onde se conhece quem faz como faz, e quando faz utilizado no desenvolvimento de um Projeto. Nos dias atuais, normalmente se classificam as metodologias em

tradicionais ou pesadas e ágeis. No próximo tópico vamos falar sobre a evolução dos processos.

2.1 Evolução

Como se pode observar na figura 3.2 entre as décadas de 50 e 80, o que existia era a chamada análise estruturada que posteriormente evoluiu para análise essencial. Sendo a primeira forma organizada de desenvolvimento, focada na qualidade da construção de *software*, naturalmente muitos problemas eram inerentes a essa forma, mas inegavelmente, era sim, a melhor forma de se construir uma aplicação.



Fonte: Próprio Autor

Mas, o grande problema desse modelo, consistia na grande quantidade de simbologias para as diversas fases de construção. Explicando melhor, um analista fazia fluxogramas para retratar a solução de um problema e uma série de outro diagrama, a exemplificar, diagrama de contexto, macro-diagrama, diagrama de fluxo de dados, e por aí vai.

O grande problema é que para banco de dados, existia outra diagramação, o modelo de entidade relacionamento, e para desenvolvimento, a codificação propriamente dita, não se tinha uma simbologia padrão. Vale salientar, que nesse período, vários estudiosos passaram a desenvolver suas próprias simbologias, criando verdadeiros núcleos, dificultando o entendimento entre os profissionais de escolas diferentes.

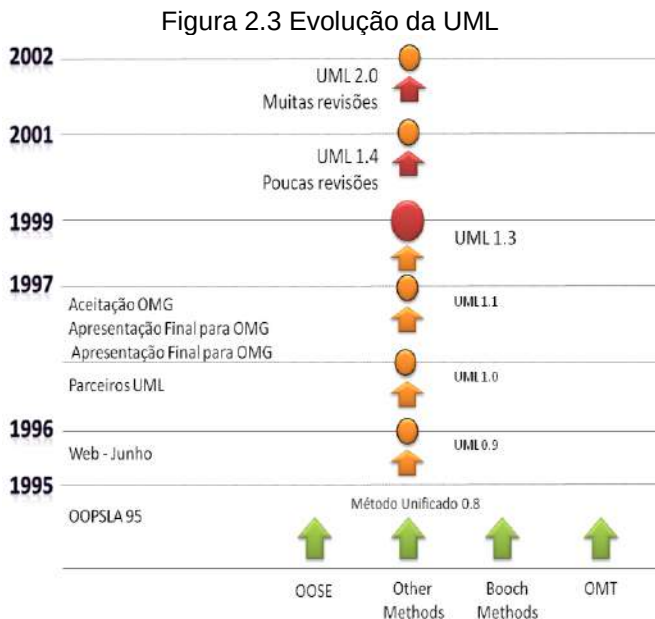
Esse período, onde se encontravam as linguagens de estruturadas e textuais, como COBOL, Fortran, pascal, Clipper e tantas outras, havia uma lacuna muito grande entre os analistas de sistemas e os programadores.

É entre as décadas de 80 e 90 que se começa a se tornar comum o uso da orientação a objetos. Esse novo paradigma de desenvolvimento surge no mundo profissional quase em paralelo com os sistemas multitarefa, que viriam a substituir os monotarefas com suas linguagens estruturadas e textuais. Essa

nova forma de programar veio com a orientação a eventos e interface gráfica a GUI. Então, definiram-se novos conceitos como classe, objeto, herança, encapsulamento e polimorfismo. Assim, os estudiosos desse tema, começam a construir suas simbologias, correndo o risco de surgirem novos núcleos e escolas independentes que sem dúvida, iriam complicar ainda mais o entendimento dessa árdua tarefa que é construir *software*.

Mas, em meados de 90, A SEI cria um grupo chamado OMG, que tem como responsabilidade evitar que esses núcleos surgissem. Assim, nessa época os principais estudiosos do tema foram convidados para participar desse grupo. Esses nomes são: Grady Booch que manifesta a necessidade de unificar critérios, James Rumbaugh se une a Booch em outubro de 1994 e elaboram a versão 0.8 do *Unified Method* e por fim Ivar Jacobson que completa o trio. Esses três nomes passam a serem considerados os pais da orientação a objetos, definindo os padrões da Linguagem de Modelagem Unificada, que na sua sigla em inglês é conhecida como UML. Passando a ser o modelo aceito como padrão pela OMT e assim pela SEI e por consequência pelo mercado.

Com a simbologia padronizada, evitando aquela chuva de diagrama que variavam de escola para escola, precisava-se agora definir um padrão de desenvolvimento para esse novo paradigma. Mas, padrões já existiam e vamos falar sobre eles.



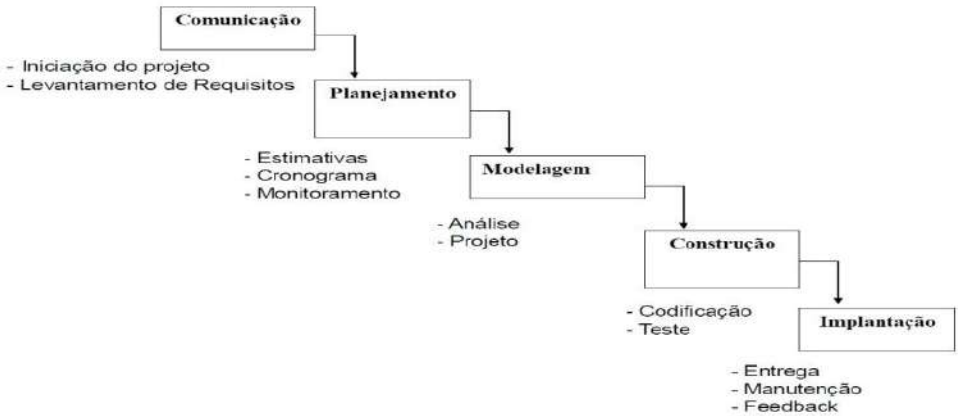
Fonte: Internet

2.2 Modelo Cascata

O modelo cascata é uma forma sequencial de desenvolvimento de *software*, onde o ciclo de vida de um projeto começa pelos requisitos seguindo pelas especificações e modelagem, passando pela codificação e testes, até chegar à implantação e manutenção.

O problema desse modelo é que todas essas etapas são executadas, uma após a outra.

Figura 2.4 Modelo Cascata



Fonte: Internet

Com a observação de que cada etapa de ser exaurida antes de passar para a próxima, ocasionando um grande lapso de tempo entre análise, projeto e implantação. Essa distância entre as fases macros geram inevitavelmente problemas nos requisitos, já que esses podem mudar pela própria natureza do negócio ou pelo problema mais comum, a comunicação entre os envolvidos. Algumas características desse modelo são: Execução sequencial das atividades; Levantamento de todos os requisitos no início do projeto (algo que a experiência mostrou ser muito difícil); Talvez a característica mais negativa desse modelo, seja a de que o usuário é envolvido apenas no início do projeto e na entrega do produto final, aumentando exponencialmente as chances de falha durante o projeto; e as mudanças nos requisitos não são bem vistas, pois gera retrabalho e conseqüentemente aumento dos custos. Mas, mudança de requisitos é inevitável na construção de qualquer *software*, por mais estável que seja.

2.3 Modelo Prototipação

Uma forma de lidar com a impaciência dos usuários no modelo cascata, foi à criação, ou melhor, a adaptação de uma técnica ao modelo que rendeu a denominação de prototipação e assim, surgiu um novo modelo. Que na verdade

era o modelo em cascata, mas servido da técnica de fazer protótipos para que o cliente durante o processo tivesse condições de acompanhar o andamento, visualizando como o *software* ficaria. Mas, o problema desse modelo, era que, sua premissa dizia que o protótipo deveria ser descartável, isto é, apresentado ao cliente, este faria as observações, e no local de desenvolvimento (fábrica de *software*) esse protótipo deveria ser descartado e o desenvolvimento, seria continuado no modelo real. Bom, pura ilusão, pois, o que os desenvolvedores faziam era continuar evoluindo o protótipo que virava o entregável. Além dos problemas já citados acima, esses persistiam, já que a forma de construir o *software* não mudara muito.

Figura 2.5 Modelo Prototipação

A imagem mostra uma interface web de prototipação para um sistema chamado 'WEBSUPORTE'. No topo, há uma barra de endereço com o URL 'http://www.websuporte.com.br'. Abaixo, o título 'WEBSUPORTE' é exibido em letras grandes e negritadas. Segue-se a mensagem 'Seja bem vindo ao WebSuporte'. A interface é dividida em duas colunas principais. A coluna da esquerda, intitulada 'CRIE A CONTA', contém campos para: Nome, CPF, Nome do login desejado, Escolha uma senha, Confirme a senha, E-mail, E-mail secundário, Bairro, CEP, Rua e número da casa, e Cidade. A coluna da direita, intitulada 'TERMOS DE SERVIÇO', contém um texto de aviso e um botão 'Aceito. Criar minha conta'. Abaixo da seção de termos de serviço, há campos para: DDD, Telefone, DDD, Celular, e Complemento. No rodapé da interface, há um campo para UF (Estado) com uma seta para baixo.

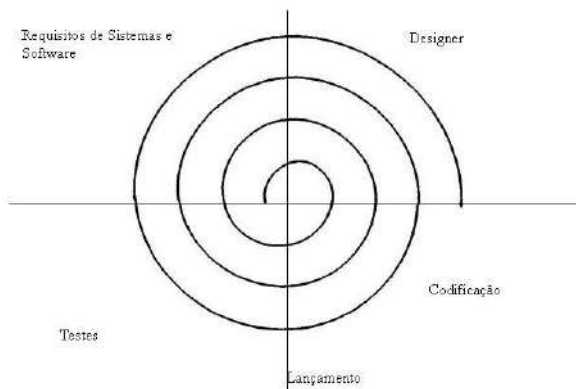
Fonte: Internet

2.4 Modelo Espiral

Até que surge o modelo espiral, isto é, você faz os levantamentos dos requisitos de *software*. Importante, não de todo o projeto, mas do que foi planejado, isto é, em etapas, normalmente do que é mais relevante para o que menos. Porém, dependo muito das necessidades do projeto. Depois desse levantamento, faz-se a diagramação do que foi requisitado, codifica-se, testa-se e implanta-se. Pronto.

Nesse momento o cliente interage novamente com o *software*, fazendo que o lapso de tempo seja bem menor, muito menor. O que ao longo do processo de amadurecimento dos modelos, viu-se ser muito proveitoso para o sistema. Afinal de contas, é para o cliente que se faz o *software*. Nesse momento, existem ajustes a serem combinadas e nova repetição do fluxo se faz, e se fará tantas vezes se faça necessário ou se tenha planejado.

Figura 2.6 Modelo Espiral



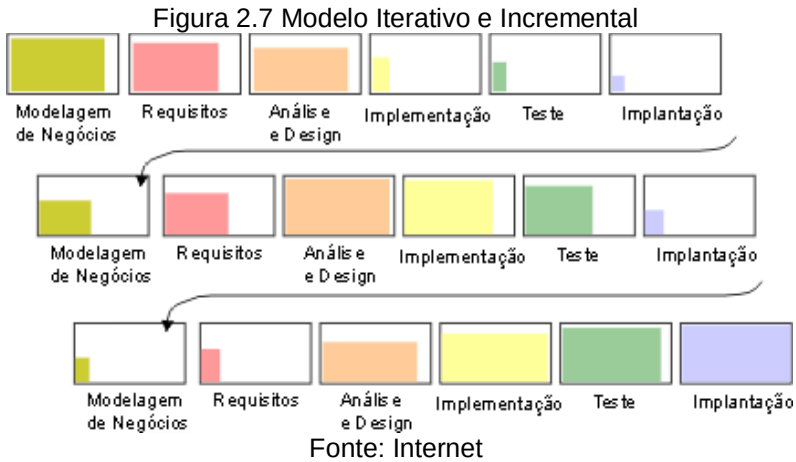
Fonte: Internet

Esse modelo se mostrou muito mais próximo as reais necessidades de um bom desenvolvimento, e dele surgem o que efetivamente ganhou o mercado. Mas, por que esse não se consolidou. Simples, não se fazia ideia que permitir a interação com o cliente entre as fases de desenvolvimento daria tanto problema, assim, foi necessário planejar essa interação, surgindo às iterações.

2.5 Modelo Iterativo e Incremental

O modelo iterativo e incremental consiste em fazer o desenvolvimento do *software* no modelo espiral, mas sabendo que em cada iteração, isto é, nova rodada de fases, talvez se tenha que corrigir falhas das rodadas anteriores, além de implementar as novas funcionalidades da rodada corrente. Simples, direto e objetivo.

Esse modelo se mostrou o mais próximo da realidade do desenvolvimento de projetos de *software*. Se tornando a base para uma grande gama de processos hoje vigentes.



2.6 Metodologia Tradicional

São métodos que enfatizam o processo com todo o formalismo necessário para garantir o controle das atividades de desenvolvimento de *software*, dando muita importância a documentos escritos. As pessoas envolvidas diretamente e indiretamente com o *software* devem utilizar o processo definido.

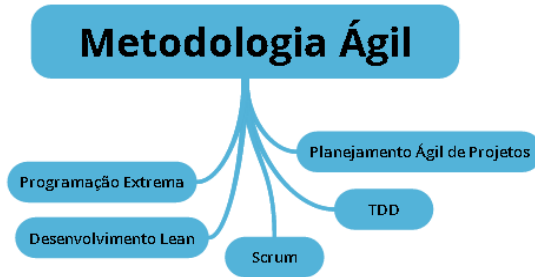


Fonte: Próprio Autor

2.7 Metodologia Ágil

São métodos que enfatizam a comunicação em tempo real entre as pessoas envolvidas, dando menos importância a documentos escritos. As pessoas envolvidas diretamente com o *software* devem estar preferencialmente disponíveis em relação a tempo e local da melhor forma possível.

Figura 2.9 Metodologia Ágil



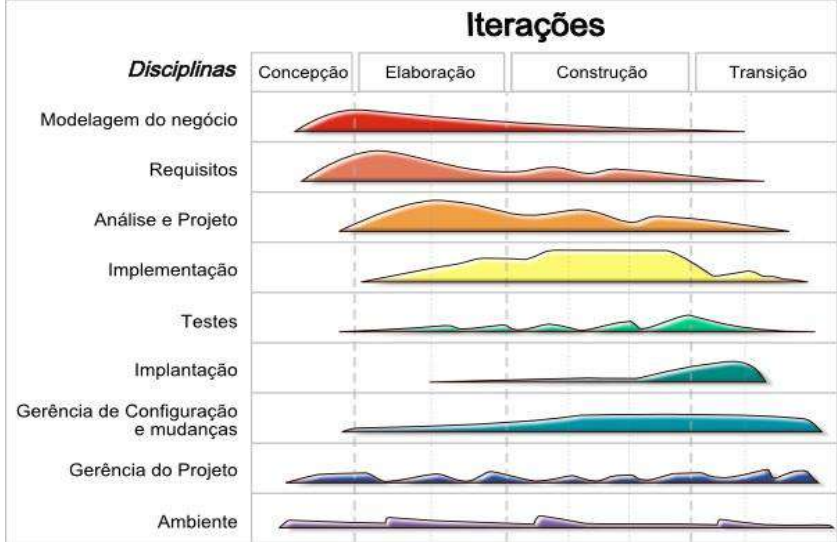
Fonte: Próprio Autor

2.8 Processo RUP

De todos os processos criados o *Rational Unified Process* (RUP), sem dúvida é um dos mais famosos. Esse modelo, criado nos "primórdios" da UML, trás todas as atividades, os papéis, os artefatos, e os fluxos de trabalhos, que se deve ter em um desenvolvimento pesado de *software*. Isto é, ele prevê praticamente tudo que se deve fazer.

E se tornou uma referência de mercado. Sendo "vendido" como um framework, que deve ser adaptado para a realidade do projeto em desenvolvimento. Bom, é aí que começam os problemas. Para projetos de médio e grande porte - excelente. Se existem prazo e custo aceitáveis para a contratação de mão de obra qualificada, e necessidade de controle de toda a produção. Esse é o modelo. Mas, mesmo ele sendo declarado pela seu fabricante Rational, hoje IBM, como "customizável" para projetos de pequeno porte.

Figura 2.10 Processo RUP



Fonte: Internet

O conhecimento desse processo, a experiência do "customizador" e as restrições orçamentárias e de prazo são complicadíssimos para sua adoção. Mas, por um período de tempo, esse foi o processo utilizado para qualquer empresa de TIC que se deseja construir algo com qualidade. Até que grupos de pensadores começaram a discutir se o uso desses tipos de processos, se justificavam para qualquer tipo de projeto de *software*. Nesse ponto entraremos na nossa próxima aula - as metodologias ágeis, já que os processos que passaram a ser chamados de pesados receberam a denominação de metodologias tradicionais ou mesmo pesadas.

2.9 Processo Scrum

Scrum é um processo ágil que tem como objetivo focar na entrega dos requisitos de maior valor agregado ao cliente no menor intervalo de tempo possível. Permitindo assim que, adequações sejam realizadas no *software* de maneira rápida e contínua, fazendo com que o produto final esteja em conformidade com as necessidades do referido cliente.

Esses intervalos de entregas devem ser de duas a quatro semanas, para que as necessidades do negócio com as suas prioridades sejam entendidas, desenvolvidas, testadas, entregues e usadas pelo cliente.

Figura 2.11 Processo Scrum



Fonte: Internet

Para isso as equipes devem possuir uma auto-organização para poder definir a melhor maneira de realizar as entregas dos requisitos de maior prioridade. Nesses intervalos de duas a quatro semanas todos podem ver o real *software* em produção, decidindo se o mesmo deve ser liberado ou não e isso é denominado de *Sprint*. Resumindo as características dos Scrum são: Equipes que se auto-organizam; O produto evolui em uma série de *Sprints* mensais; Os requisitos são listados em um *Product Backlog*; Não há uma prática específica de engenharia de *software*, pois o Scrum pode ser adequar a todas; Usa regras genéricas na criação de um ambiente ágil para a entrega de projetos; e é uma metodologia ágil.

2.10 Metodologias-Modelos-Processos-Técnicas

Agile Software Development (ASD). Disp.: <

http://pt.wikipedia.org/wiki/Agile_Software_Development >

Cleanroom. Disponível em < <http://pt.wikipedia.org/wiki/Cleanroom> >

CRUD. Disponível em < http://pt.wikipedia.org/wiki/Matriz_CRUD >

Crystal. Disponível em <

http://en.wikipedia.org/wiki/Crystal_Clear_%28software_development%29 >

Domain-Specific Modelling (DSM). Disponível em < <http://goo.gl/sU1Jfs> >

Driven Design de Domínio (DDD). Disponível em < <http://goo.gl/wDBnjk> >

Dynamic Systems Development Method (DSDM). Disponível em <

<http://pt.wikipedia.org/wiki/DSDM> >

eXtreme Programming (XP). Disponível em < <http://goo.gl/691P2> >

Feature Driven Development (FDD). Disp.: <

http://pt.wikipedia.org/wiki/Feature_Driven_Development >

IBM Rational Unified Process (IRUP). Disp.: <

http://pt.wikipedia.org/wiki/Rational_Unified_Process >

Microsoft Solution Framework (MSF). Disp.: <

http://pt.wikipedia.org/wiki/Microsoft_Solution_Framework >

Model Driven Engineering (MDE). Disponível em < <http://goo.gl/iDq3RW> >

Model-Driven Architecture (MDA). Disponível em < <http://goo.gl/bgKPxp> >
Modelo V. Disponível em < http://pt.wikipedia.org/wiki/Modelo_V >
PMBOK. Disponível em < <http://pt.wikipedia.org/wiki/PMBOK> >
Pragmatic Programmer. Disponível em < http://en.wikipedia.org/wiki/The_Pragmatic_Programmer >
PRAXIS. Disponível em < <http://goo.gl/ag09tD> >
Rapid Application Development (RAD). Disp.: < http://pt.wikipedia.org/wiki/Rapid_Application_Development >
Scrum. Disponível em < <http://pt.wikipedia.org/wiki/Scrum> >
Service-Oriented Architecture (SOA). Disponível em < <http://goo.gl/ESXj3U> >
SWEBOK. Disponível em < http://pt.wikipedia.org/wiki/Software_Engineering_Body_of_Knowledge >
Test Driven Development (TDD). Disponível em < <http://www.improveit.com.br/xp/praticas/tdd> >

2.11 Ferramentas

Astah: modelagem UML. Disponível em < <http://astah.change-vision.com/en/index.html> >
Project-open: gerenciador de projetos. Disponível em < <http://www.project-open.org/> >
StarUML: modelagem UML. Disponível em < <http://staruml.sourceforge.net/en/> >

2.12 Sugestões de Leitura

Engenharia de Software e Sistemas de Informação. Disponível em < <http://goo.gl/N9f446> >
Gerenciando Projetos de Desenv. Software com PMI, RUP e UML. Disp.: < <http://goo.gl/GcXUe4> >
Guia prático de engenharia de software. Disponível em < <http://goo.gl/ppe5ff> >

2.13 Referências Web

Análise de Domínio wiki. Disp.: < http://pt.wikipedia.org/wiki/An%C3%A1lise_de_Dom%C3%ADnio >
Análise Essencial wiki. Disponível em < http://pt.wikipedia.org/wiki/An%C3%A1lise_Essencial >
Análise Estruturada wiki. Disponível em < http://pt.wikipedia.org/wiki/An%C3%A1lise_Estruturada >
Análise Orientada a Objetos wiki. Disponível em < <http://goo.gl/N0Ktwf> >
BROD Dicas. Disponível em < <http://www.dicas-l.com.br/brod> >
BROD. Disponível em < <http://www.brod.com.br/> >
ControlChaos. Disponível em < <http://www.controlchaos.com/> >
Engenharia de Software wiki. Disponível em < http://pt.wikipedia.org/wiki/Engenharia_de_software >
IBM RUP. Disponível em < <http://www-01.ibm.com/software/awdtools/rup/> >

Iterativo e Incremental wiki. Disp.: <

http://pt.wikipedia.org/wiki/Desenvolvimento_iterativo_e_incremental >

Metodologias de Software wiki. Disp.: <

http://pt.wikipedia.org/wiki/Metodologia_%28engenharia_de_software%29 >

Modelo Espiral wiki. Disponível em <

http://pt.wikipedia.org/wiki/Modelo_em_espiral >

Modelos Ágeis wiki. Disp.: <

http://pt.wikipedia.org/wiki/Desenvolvimento_%C3%A1gil_de_software >

Modelos Ciclo de Vida wiki. Disponível em <

http://pt.wikipedia.org/wiki/Modelos_ciclo_de_vida >

RUP. Disp.: <

[http://www.wthreex.com/rup/portugues/index.htm%20\(Excelente%20site%20sobre%20RUP\)](http://www.wthreex.com/rup/portugues/index.htm%20(Excelente%20site%20sobre%20RUP)) >

Scrum Alliance. Disponível em < <http://www.scrumalliance.org/> >

Scrum. Disponível em < <http://www.mountangoatsoftware.com/scrum> >

SEI. Disponível em < <http://www.sei.cmu.edu/> >

SWEBOK. Disponível em < <http://www.computer.org/portal/web/swebok> >

2.14 Referências Bibliográficas

PRESSMAN, Roger. Engenharia de Software: uma abordagem profissional.

7ªed. Porto Alegre: Bookman, 2011.

RUMBAUGH, James; JACOBSON, Ivar; BOOCH, Grady. UML: Guia do Usuário.

1ªed. RJ: Campus, 2006.

SOMMERVILLE, Ian. Engenharia de Software. 8ªed. Pearson, 2007.

3 Introdução à Engenharia de Requisitos

Essa disciplina tem o objetivo de explicar as técnicas para se identificar o problema de um cliente e assim criar uma solução para o mesmo. Entender o verdadeiro problema é o que chamamos de análise de requisitos ou engenharia de requisitos, e é de suma importância na engenharia de *software*, pois todo o esforço de construção de um *software* só terá verdadeiro valor se atender as expectativas do cliente.

Analisar os requisitos de um cliente é identificar o real problema, isto é, propor uma solução, o que chamamos de projeto, porque para solucionar um problema precisamos construir a solução. Mas, como entender o verdadeiro problema do cliente? Aí entramos na análise de requisitos ou também chamada de engenharia de requisitos.

Na metodologia tradicional, o processo unificado, define quatro fases, as quais são: concepção, elaboração, construção e transição. Em todas as quatro sempre haverá algum esforço no entendimento das necessidades do cliente. Já que, negócios mudam, ou pela sua própria natureza, ou por falta de entendimento, ou por falta de detalhamento, ou até mesmo por todas elas juntas.

Mas, o maior esforço no entendimento das necessidades do cliente, na grande maioria dos casos, ocorrerá nas duas primeiras fases. Isso é lógico, já que na primeira, estamos tentando entender o negócio, e na segunda levantando as necessidades do cliente, e essas, devendo ser esmiuçadas ao máximo.

Agora, independente da fase, se faz crucial para o sucesso do sistema, o entendimento do problema, para que se atinja o objetivo desejado, que é obter um produto de *software* de melhor qualidade e que satisfaça a real necessidade do cliente, dentro de um prazo e orçamento adequados. Assim, nasce a engenharia de requisitos, que tem como premissa, conhecer todas as atividades usadas para a produção da documentação necessária ao entendimento do problema, como também a manutenção desses no decorrer do projeto.

Mas, o que é exatamente um requisito? Podemos entender requisito como uma função, restrição ou propriedade que deve ser fornecida, encontrada ou atendida para satisfazer às necessidades do usuário do sistema, descrevendo um serviço ou uma limitação.

Outra forma de definir requisito é a descrição de uma condição ou capacidade necessitada por um usuário para resolver um problema ou alcançar um objetivo. Em outras palavras, é o que o sistema deve fazer para implementar uma necessidade de automação requerida pela solução. Desde as necessidades básicas do cliente, premissas e restrições obtidas na fase de levantamento do projeto até as condições de negócio explicitadas no contrato com o fornecedor

da solução. Compreende um conjunto de definições que descreve como o sistema deve ser construído e testado, requisitos funcionais e não-funcionais, características operacionais e especificações técnicas, enfim, tudo o que o sistema tem que ter para atender plenamente ao propósito para o qual foi criado.

Segundo Soares (2005), a engenharia de requisitos é um processo baseado em quatro atividades, as quais são: Validação, Identificação; Análise; e Especificação. Mas, existe uma quinta atividade, inerente a qualquer processo de desenvolvimento, a de manutenção, seja ela corretiva, adaptativa ou evolutiva, e essa atividade recebe o nome na engenharia de requisitos de gestão de mudanças ou no inglês *changed request*.

3.1 Estudo de Viabilidade

Agora, como encontrá-los? Bom, em primeiro lugar é uma boa prática, realizar antes de qualquer coisa um estudo de viabilidade que, a partir das restrições do projeto, pode-se verificar se é ou não interessante fazer a identificação dos requisitos.

Mas, o que é um estudo de viabilidade? São a verificação da real necessidade de se levantar os requisitos, já que essa tarefa não é básica, e pode demandar muito tempo. Imagine que depois de todo o trabalho de elicitação realizado, verifica-se que esse não era necessário. Assim, com esse estudo realizado, pode-se verificar se em relação à tecnologia e a estrutura organizacional, o projeto é viável.

Uma técnica usada para verificar a viabilidade do projeto é realizar reuniões e/ou entrevistas, com aos *stakeholders* do projeto, para obter resposta às seguintes questões:

- O projeto proposto contribui para os objetivos da organização?
- As restrições tecnológicas, econômicas, políticas, ambientais, recursos e de prazo, torna o projeto viável?
- Se existe necessidade de integração entre diferentes sistemas, essa é possível?
- Caso o novo projeto não for construído, quais serão as consequências e o que se deve fazer?
- Baseado nos efeitos percebidos pelo cliente. Quais são os problemas?
- Como esses problemas podem ser resolvidos?
- Haverá colaboração e se sim de que forma?

Com essas questões respondidas, pode-se ter uma decisão mais “pé no chão”, para prosseguir com o levantamento dos requisitos. Após esse levantamento, deve-se produzir um relatório que irá determinar a continuação ou não do desenvolvimento do projeto.

Caso o estudo de viabilidade, mostre que é viável o desenvolvimento do projeto, deve-se então iniciar a identificação dos requisitos de *software*. E a primeira atividade deve ser o entendimento do domínio da aplicação, isto é, compreender o negócio que se pretende construir uma solução. Esse entendimento é vital para melhorar a comunicação entre os analistas e as partes interessadas. Um dos passos é identificar os *stakeholders*, isto é, todos os envolvidos no projeto, o outro é fazer um estudo do assunto a ser desenvolvido. Depois disso, iremos para o levantamento dos requisitos funcionais e não-funcionais.

3.2 Levantamento de Requisitos

A próxima fase é a chamada identificação dos requisitos. É baseada em questões do tipo. O cliente não sabe o verdadeiro problema, pois só tem conhecimento dos efeitos dele. Como consequência os requisitos elicitados, podem não ser relevantes ou mesmo reais, para a solução do problema. Caso existam vários *stakeholders*, esses podem expressar suas necessidades, em forma de requisitos diferentes, mas com a mesma finalidade.

Então, quais são as técnicas para levantamento de requisitos, para minimizar esses problemas? Existem diversas técnicas de identificação de requisitos, e que devem ser usadas de acordo com a necessidade.

Um delas é o uso de entrevistas e questionários. Nessa, se deve ter domínio sobre o assunto, para que se possa inferir nas respostas do *stakeholder*. É importante que o analista dê espaço ao entrevistado para que o mesmo possa expor as suas ideias. *Não o interrompa desnecessariamente*. Outro fator importante no uso dessa técnica é verificar a predisposição do entrevistado, pois caso, a atividade dele esteja sendo afetada pelo novo sistema, isso pode ocasionar dificuldades na obtenção das informações. Sempre construir um plano para realizar a entrevista, pois na ausência deste, pode se perder muito tempo com itens não relevantes ao assunto em questão.

Outra técnica é o uso de um *workshop* para requisitos, que consiste em realizar uma reunião na qual devem fazer parte o maior número de *stakeholders* possível. Nessa técnica deve-se fazer uso de momentos de descontração, para possibilitar um clima mais propício e ao mesmo tempo dinamizar o trabalho. Deve-se, quando possível fazer uso de facilitador cujo papel é coordenar as atividades a serem realizadas. No final do *workshop*, deve-se produzir um relatório que reflita os requisitos e decisões tomadas na mesma.

Mais uma técnica, que pode ser usada como uma atividade, nas técnicas anteriores, é a chamada tempestade de ideias ou no seu termo em inglês mais conhecido tecnicamente, o *brainstorming*. Que tem a finalidade de obter o maior número de ideias numa pequena quantidade de tempo. No final do *brainstorming*, deve-se produzir um relatório que reflita as ideias obtidas.

Cenários, é outra técnica muito interessante, pois é, uma forma de levar os *stakeholders* a imaginarem o comportamento do *software*. Assim, os futuros usuários podem interagir com o *software*, mesmo ele ainda não existindo, aumentando as possibilidades de requisitos obtidos. E para que essa técnica fique mais eficiente, os seguintes elementos devem ser considerados: O estado inicial do *software*; a sequência de eventos que devem ocorrer; Quais exceções podem ocorrer na execução dos eventos; Quais tratamentos devem ser dados quando as exceções ocorrerem; Quais as ações irão acontecer; e Qual o estado final do *software* na termino do cenário.

Mas, a técnica mais usada é a prototipagem, que consistem em criar um rascunho do *software*, baseando-se nos requisitos iniciais, e com esse, melhorar a comunicação entre os *stakeholders*, com o intuito de obter mais requisitos. Neste tipo de abordagem apenas algumas funcionalidades são desenvolvidas, já que a finalidade desses é ser descartado, pois não deve ser considerado o produto final. Apesar de em muitos casos o protótipo acaba sendo evoluído e passando a se tornar o produto final, sendo chamada essa técnica de prototipagem-evoluída.

A outra técnica é o estudo etnográfico. Nessa se analisa o componente social das atividades desempenhadas. Ela se baseia na observação direta das atividades realizadas durante um período de trabalho de um *stakeholder*, possibilitando encontrar requisitos. Esta observação pode ser realizada através de registros de áudio/vídeo.

3.3 Elicitação de Requisitos

A terceira fase é a de análise, que ocorre após a identificação dos requisitos, e consiste em classificar os requisitos em módulos para facilitar a visualização dos mesmos em uma visão macro, que na análise estruturada era chamado de macro-diagrama.

Outro ponto importantíssimo dessa fase é a resolução dos conflitos, que inevitavelmente ocorrem dadas a variedade de documentos produzidos na fase de identificação de requisitos.

O próximo é a priorização, que consiste na atribuição de um nível a cada requisito, onde normalmente se atribui os seguintes valores: alta; média; e baixa prioridade.

Também se podem usar os valores: essencial; importante; e recomendável.

Outra denominação que se pode utilizar são os valores essencial, mandatório e importante. Agora, como obter essa priorização? A melhor forma é fazer uso de métricas, um bom exemplo é a Análise de Ponto de Função. Mas, existem outras técnicas, tipo WidebandDelphi, que é mais leve

e rápida. Independente, da técnica, o importante é classificar, para identificarmos o que é mais ou menos relevante para o cliente.

Depois de modularizar e classificar, é importantíssima, a homologação, para que os envolvidos tenham ciência e forneçam o de acordo. Para que a completude dos requisitos, sua consistência e validade sejam aceitos por todos.

É importante salientar que essas fases não são independentes, já que uma informação obtida numa fase é entrada para outra e assim por diante. A identificação e análise de requisitos são processos iterativos que se iniciam como o entendimento do domínio da aplicação e culminam com a homologação dos requisitos, permitindo que se tenha o melhor entendimento possível do sistema.

Mas, claro, existem muitas dificuldades nesse processo e essas são de diversas naturezas, como fatores externos, do tipo legislação ou política socioeconômica, que podem influenciar nos requisitos. A decorrente de interesses particulares, que se advindo de um envolvido com poder de decisão, pode exigir requisitos específicos que sirvam aos seus interesses e não aos da aplicação, que tem como objetivo atender a organização. Outro fator é quando um envolvido com poder de decisão tenta forçar o seu ponto de vista em detrimento dos demais interessados. Existe também o ambiente, seja ele econômico e/ou organizacional, onde a análise realizada possui fatores dinâmicos, tornando os requisitos suscetíveis à mudança.

3.4 Análise de Requisitos

Seguindo temos, a fase de negociação é delicada e deve-se ter muito cuidado para que ocorra sem problemas. Para isso existe uma técnica, que deve ser aplicada aos que serão envolvidos nesse processo. Essa consiste em levantar as seguintes questões. Os envolvidos sabem: Lidar com ataques pessoais, evitando-os sempre que possível, remetendo a sua resolução para mais tarde, fora de reunião, de preferência nunca tomando partidos; Fomentar a justificação das posições negativas tomadas pelos intervenientes na negociação; Salientar e procurar encontrar os benefícios que uma solução apresenta para todos os envolvidos; e Relaxar restrições, quando se torna óbvio que as atuais não conseguem levar a um consenso.

A fase de especificação e documentação se dá no final, e consistem de descrever baseado nos padrões reconhecidos pelo mercado, os requisitos especificados. Mas, nessa etapa devem-se classificar esses requisitos em duas categorias. A dos funcionais, e a dos não-funcionais, que são normalmente representados pelas siglas RF e RNF, respectivamente.

Os requisitos funcionais servem para descrever as funcionalidades que se espera que o sistema tenha, isto é, aquilo que o usuário espera que o sistema

ofereça, atendendo aos objetivos requisitados. Esses requisitos especificam ações que um sistema deve ser capaz de executar, sem levar em consideração restrições físicas, portanto, especificam o comportamento de entrada e saída de um sistema.

Os requisitos não-funcionais servem como restrições e estão relacionados ao uso do sistema em termos de desempenho, usabilidade, confiabilidade, segurança, disponibilidade, manutenibilidade, tecnologias envolvidas, utilidade, suporte e escalabilidade. Normalmente não são obtidas com o usuário, já que são características mínimas de um *software* de qualidade, ficando a cargo de o desenvolvedor optar por atender esses requisitos. Segue alguns exemplos:

- Tempo de resposta do sistema não deve ultrapassar 15 segundos;
- *Software* deve ser operacionalizado no sistema Windows; e.
- O banco de dados usado deverá ser o Oracle.

É importante entender que os requisitos não-funcionais são críticos para o sucesso do *software* e estão diretamente relacionados com a satisfação dos usuários. Devido a essa importância, alguns requisitos funcionais podem ser sacrificados para atender às restrições impostas pelos requisitos não-funcionais.

Em um processo bem organizado, podem-se gerar diferentes tipos de documentos de especificação de *software*, os quais são: Especificação de requisitos do utilizador; Especificação de requisitos do sistema; e Especificação do design da aplicação. Essa organização permite que determinados tipos de especificação que serve para comunicar apenas um determinado tipo de informação, siga apenas para os interessados por ela, evitando assim, informação em demasia. Por exemplo, enquanto os requisitos do utilizador informam apenas a abordagem de alto nível das funcionalidades do sistema e suas restrições, usando para isso uma linguagem natural e eventualmente diagramas, os requisitos do sistema descrevem cada requisito com todos os detalhes introduzindo conceitos relativos à arquitetura do sistema, utilizando também diagramas de casos de uso para isso.

Seguem alguns cuidados para produção dessas documentações de requisitos:

Utilizador destinado aos usuários de alto nível. Deve se escrito com linguagem natural, com poucos formulários e diagramas simples. As dificuldades na sua produção são a ambiguidade, confusão e agrupamento de requisitos. Técnicas para sua produção são: Usar o mesmo formato em todos os requisitos; Distinguir claramente entre comportamentos esperados e desejáveis do sistema através do uso de expressões como "O sistema permitirá criar (...)" ou "Deverá ser possível criar (...)" respectivamente; Usar formatação de texto para salientar determinados aspectos do documento; e Evitar usar termos demasiado técnicos ou fora do âmbito do sistema, identificando-os e definindo-os de uma forma clara quando for absolutamente necessário usá-los.

Sistema destinado aos usuários mais técnicos. Deve conter uma descrição detalhada dos requisitos correspondentes ao uso, usando terminologia técnica quando preciso. Dificuldades: As mesmas palavras podem, de acordo com a interpretação de cada pessoa, designar conceitos diferentes; e O mesmo requisito pode ser descrito de formas diferentes, o que leva a que cada pessoa tenha de saber quando é que descrições diferentes se referem ou não a requisitos diferentes.

Design destinado aos desenvolvedores do sistema. Neste estão definidos os detalhes, em um nível totalmente técnico, acerca da implementação do sistema e sua arquitetura. A partir deste documento, um novo recurso que entre na equipe de desenvolvimento no meio do projeto, poderá facilmente entendê-lo. Compreendendo a forma de implementação.

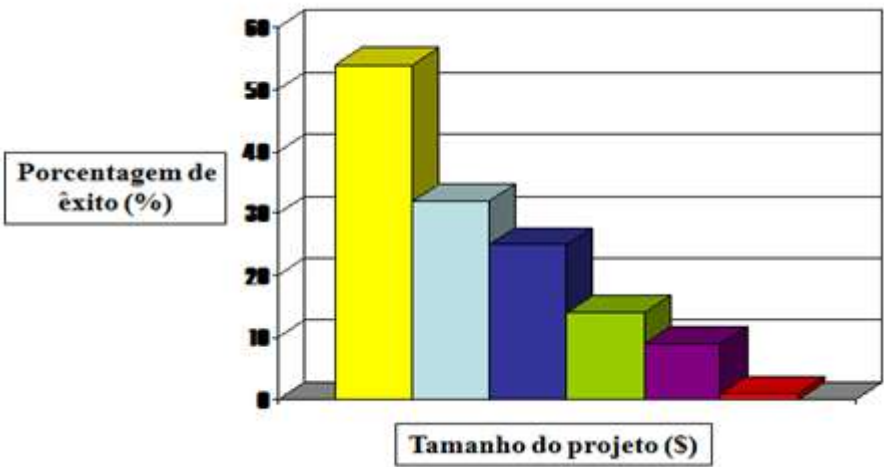
Especificação de Requisitos ou Documento de Requisitos de Sistema (DRS) pode ser entendida como uma descrição formal e oficial onde é descrita e comunicada os requisitos a todos os envolvidos no processo de desenvolvimento de *software* os *stakeholders*. Sendo basicamente composto dos serviços e funções que o sistema deve prover; As limitações sobre as quais o sistema deve operar; Definições de outros sistemas com os quais o sistema deve integrar; Limitações no processo usado para desenvolver o sistema; e Descrições sobre o *hardware* no qual o sistema irá executar. Esse é um documento completo, usado como declaração oficial dos requisitos do sistema. Este documento, normalmente conhecido na sua denominação em inglês *Software Requirements Specification* ou SRS, inclui uma combinação dos requisitos do utilizador e do sistema e tem diferentes utilidades, as quais são: Clientes, confirmar a completude dos requisitos e propor alterações; Gestores, orçar o sistema e planejar o processo de desenvolvimento; Engenheiros de *software*, compreender o sistema a desenvolver; Engenheiros de testes, desenvolver testes para validar o cumprimento dos requisitos; e Arquitetos de *software*, compreender o sistema e a ligação entre as suas partes. Para melhor entendimento desse documento, segue o template do mesmo, usado pelo RUP.

Existe uma técnica para produção do documento de requisitos, e essa consiste em demonstrar que o documento produzido corresponde, de fato, ao sistema que o cliente deseja.

A validação é especialmente importante em sistemas de grandes dimensões uma vez que erros encontrados demasiados tarde, durante o desenvolvimento ou já depois de o sistema estar sendo usado produzem um efeito desastroso, tanto a nível financeiro, como de credibilidade.

Segundo um estudo do *Standish Group*, quanto maior o *software*, maior é a possibilidade do insucesso. Veja o gráfico.

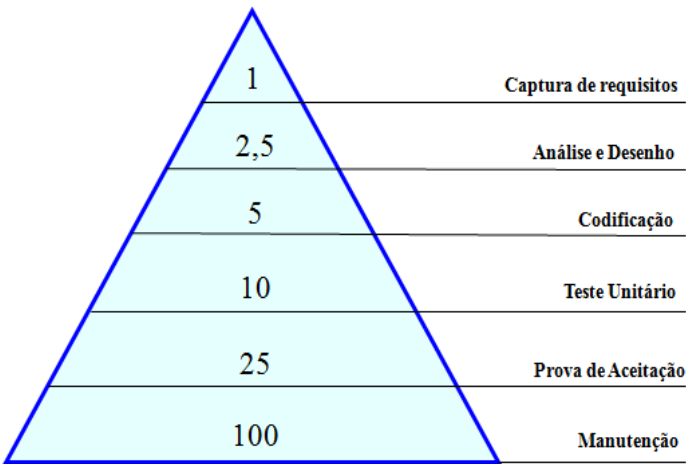
Figura 3.1 Percentual de sucesso



Fonte: Standish Group

Outro problema sério é quando se descobre uma falha. Observe o gráfico abaixo e veja qual é o incremento no custo de produção de um *software*, quando se detecta uma falha. Percebe-se que quanto mais tardio for encontrado essa falha muito maior será o seu custo de correção.

Figura 3.2 Custo de correção por tempo de identificação do erro



Fonte: Standish Group

Os estudos comprovam que, a maior parte dos problemas os de maior impacto negativo e os mais onerosos têm origem nas etapas iniciais do desenvolvimento de *software*. Justamente nas etapas de especificação dos requisitos, onde as principais atividades são definidas e onde os requisitos do produto devem ser identificados e mapeados com objetividade e clareza.

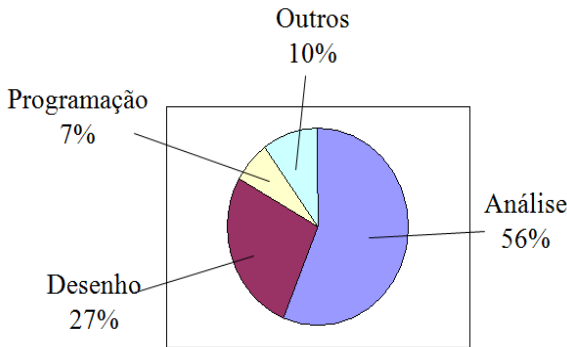
Podemos dizer que as principais causas para o fracasso dos projetos de *software* são: especificação de requisitos mal formulada e alterações constantes nos requisitos. Por serem atividades bases do processo de desenvolvimento de *software* as falhas cometidas nas atividades de definição e validação de requisitos irão originar documentos de requisitos inconsistentes afetando as etapas seguintes de projeto, implementação e testes e gerando produtos de *software* de baixa qualidade. Veja essa outra figura baseado em um estudo da *Standish Group*, sobre os principais originadores de falhas.

Para minimizar ou evitar isso, existe uma técnica que consiste em fazer um *checklist*, avaliando os seguintes aspectos:

- Validade, se as especificações levantadas são aceitas por todos os envolvidos; Consistências se existem conflitos entre os requisitos identificados; Compreensibilidades se são fáceis de serem compreendidos;
- Ambiguidade, se os requisitos estão em uma forma inequívoca pelas partes interessadas;
- Completude, se todas as funcionalidades pretendidas fazem parte da especificação do sistema;
- Realismo, se as restrições do projeto, sejam tecnológicas, financeiras e/ou temporais, são reais;
- Verificabilidade serve para evitar futuras discordâncias quanto à concretização dos requisitos;
- Rastreabilidade, se a origem dos requisitos foi identificada, isso é importantíssimo para facilitar a gestão futura dos requisitos; e

Conformidade com normas, se as normas foram usadas ao longo de todo o documento.

Figura 3.3 Percentual de inserção de erros



Fonte: Standish Group

Para completar, a engenharia de requisitos é tão importante, que fazer a gestão de requisitos, é crucial para fornecer qualidade na produção do *software*. A sua gestão torna o trabalho árduo de levantar requisitos em uma atividade mais segura, já que o surgimento de requisitos em um projeto de *software* é muito natural, devido a vários motivos.

Podemos citar a necessidade de resolução de conflitos entre requisitos resulta num compromisso que procura equilibrar as necessidades das diferentes partes interessadas, gerando novos requisitos. Outro fator é a orientação do negócio que se pode alterar, tipo, nova legislação ou regulamentação pode comprometer alguns dos requisitos do sistema. Alterações a nível tecnológico podem surgir na organização, afetando particularmente, no caso de alterações de *hardware*, os requisitos não-funcionais. Podem surgir novos sistemas que precisem de suporte, em nível de interação, por parte do sistema implementado, e toda uma série de alterações imprevisíveis pode surgir fazendo com que o sistema tenha de se adaptar a todos esses novos requisitos. Existem requisitos que, tipicamente, são mais voláteis, ou melhor, dizendo, mais dinâmicos, do que outros, como por exemplo, requisitos que dependam da entidade política governante num dado momento, enquanto que outros são relativamente estáveis, os que se referem à natureza do negócio propriamente dita.

Na prática, a gestão de requisitos acaba por coincidir com o início de novos processos de engenharia de requisitos, para os "novos" requisitos, isto é, os "antigos" que sofreram alterações. Como tal, o planeamento é uma parte importante da gestão de requisitos. Devem estar definidas desde o início da gestão de requisitos políticas para: Identificação de requisitos, identificação unívoca em especial para facilitar a rastreabilidade; Processo de gestão de mudanças a utilizar, conjunto de atividades que permitem avaliar o custo e impacto das alterações; Rastreabilidade, relações entre os requisitos e design, na qual estas podem precisar manter associadas a cada requisito informação

como a parte interessada que a propôs, dependências de outros requisitos e associação a módulos específicos do design do sistema; e Ferramentas a utilizar, para sistemas grandes, a quantidade de informação a processar podem se relevadas, sendo o uso de ferramentas CASE.

Para possibilitar a manutenção e consistência entre as várias alterações solicitadas e para estas serem feitas de um modo controlado, é importante que o processo de gestão de mudanças esteja definido de um modo formal, sendo que deverá incluir as seguintes três fases: Análise do problema e especificação da alteração a fazer, identificação do problema existente nos requisitos originais e proposta de alteração a fazer aos requisitos do sistema; Análise da alteração e seu impacto, através das políticas de rastreabilidade definidas previamente, avaliação do impacto da alteração no sistema; e Implementação da alteração, alteração no documento de requisitos e, conforme seja necessário, no design e implementação. É importante seguir este processo conforme foi enunciado já que cair na tentação de implementar a alteração diretamente e só depois, retrospectivamente, atualizar os requisitos pode levar a dessincronização entre requisitos e implementação.

Segue abaixo um exemplo de especificação básica de requisitos funcionais de um sistema.

Unidade 01 - Gerenciamento de usuário

1. Cadastro, edição e remoção de usuário
2. Login/logout dos usuários
3. Controle de permissão

Unidade 02 - Gerenciamento de cliente

1. Cadastro, edição e remoção de consumidor/fornecedor

Unidade 03 - Controle de Almoxarifado

1. Cadastro, edição e remoção de matéria-prima
2. Adição e subtração da matéria-prima no almoxarifado

Unidade 04 - Controle de Estoque

1. Cadastro, edição e remoção de produto
2. Adição e subtração do produto no estoque

Unidade 05 - Produção

1. Conversão de matéria-prima em produto

Unidade 06 - Venda

1. Saída do produto do estoque
2. Contabilizar o pagamento
3. Atualizar o histórico de vendas

Unidade 07 - Relatório

1. Relatório de clientes, almoxarifado, estoque, pedido, pagamento
2. Utilizar filtro na geração de relatórios

Unidade 08 - Nota fiscal eletrônica

1. Geração de nota fiscal
2. Impressão de nota fiscal

3. Exportação de nota fiscal

Unidade 09 - Boleto bancário

1. Geração de boleto bancário
2. Impressão de boleto bancário
3. Exportação de boleto bancário

Outro exemplo de especificação de requisitos de um sistema, mais detalhado: Controle de Acesso e Cadastro Básico. Esta seção agrupa os requisitos funcionais associados ao controle de acesso e cadastros básicos do sistema.

RF001 Cadastrar/Alterar Usuário Módulo: M01.01

O administrador do sistema será o próprio responsável, que poderá cadastrar outros responsáveis para utilizar o sistema. Serão cadastrados o nome, endereço residencial, telefone/email de contato, documentos de identificação, profissão com nº de registro, login e senha inicial. Para alteração, o sistema permitirá alterar todos os dados cadastrais do usuário, com exceção do nome de login

Prioridade: Essencial

RF002 Acessar sistema Módulo: M01, M02, M03

Para acessar o sistema, deve-se informar o login e a senha corretos. O login e a senha serão validados para permitir o acesso ao sistema. No 1º acesso do usuário, o sistema deverá solicitar que seja trocada a senha inicial.

Prioridade: Essencial

RF003 Sair do sistema Módulo: M01, M02, M03

Será disponibilizada opção para sair do sistema, encerrando a sessão do usuário conectado.

Prioridade: Essencial

RF004 Cadastrar/Alterar Paciente Módulo: M01.02

O sistema permitirá cadastrar ou alterar os dados pessoais, plano de saúde, familiares, histórico médico e situação no sistema (ativo ou inativo) do paciente.

Prioridade: Essencial

RF005 Cadastrar/Alterar Profissionais Módulo: M01.03

O sistema permitirá cadastrar ou alterar os profissionais que estarão diretamente ligados ao tratamento/acompanhamento do paciente. O sistema deverá registrar a data e usuário que realizou o cadastro ou alteração.

Prioridade: Essencial

RF006 Cadastrar/Alterar Serviços e Estabelecimentos Módulo: M01.04

O sistema permitirá cadastrar ou alterar os serviços ou estabelecimentos ligados ao tratamento/acompanhamento do paciente, como farmácias e hospitais. O sistema deverá registrar a data e usuário que realizou o cadastro ou alteração.

Prioridade: Essencial

RF007 Cadastrar Anamnese do Paciente Módulo: M01.05

O sistema permitirá cadastrar a Anamnese do paciente, contendo as informações de história pessoal, história de doenças, história patológica pregressa, história fisiológica e história psico-social. O sistema deverá registrar a data e usuário que realizou o cadastro.

Prioridade: Essencial

RF008 Complementar Anamnese do Paciente Módulo: M01.05

O sistema permitirá complementar as informações da Anamnese do paciente. O sistema deverá registrar a data e usuário que realizou o complemento.

Prioridade: Essencial

Acompanhamento do Paciente - Esta seção agrupa os requisitos funcionais associados ao registro de informações de acompanhamento do paciente.

RF009 Cadastrar/Alterar Lançamentos Diários Módulo: M02.01

O sistema permitirá incluir ou alterar o registro de informações diárias sobre o acompanhamento do paciente, sendo estas informações do tipo "TEXTO". O sistema também deverá registrar a data e usuário que realizou o cadastro ou alteração.

Prioridade: Essencial

RF010 Cadastrar/Alterar Lembretes e Avisos Módulo: M02.02

O sistema permitirá incluir ou alterar lembretes com as informações de Tipo do Lembrete, Data e Hora do Lembrete, Descrição do Lembrete e Frequência de Repetição do Lembrete, onde assim que o horário programado para lembrete for atingido, o sistema emitirá uma mensagem com a descrição do lembrete.

Prioridade: Essencial

RF011 Visualizar Proximos Lembretes Módulo: M02.02

O sistema permitirá visualizar os próximos lembretes ativos ordenados pela data/hora de aviso.

Prioridade: Importante

RF012 Alterar Tipo de Alerta Módulo: M02.02

Será possível alterar o tipo de alerta dos lembretes, como alerta sonoro ou textual.

Prioridade: Importante

RF013 Cadastrar/Alterar Alimentação Módulo: M02.03

O sistema permitirá cadastrar ou alterar a alimentação específica por paciente, informando o menu semanal e o valor nutricional dos alimentos.

Prioridade: Essencial

RF014 Cadastrar/Alterar Vacinação Módulo: M02.04

O sistema permitirá cadastrar ou alterar vacinação específica por paciente, informando o tipo de vacinação, a data em que foi aplicada e seu vencimento.

Prioridade: Essencial

RF015 Visualizar Proximas Vacinações Módulo: M02.04

O sistema permitirá visualiaras próximas vacinações do paciente, baseando-se na data de vencimento registrado no módulo de cadastro da vacinação.

Prioridade: Importante

RF016 Cadastrar/Alterar Medicação Módulo: M02.04

O sistema permitirá cadastrar ou alterar medicação específica por paciente, informando a prescrição de cada medicamento e sua posologia.

Prioridade: Essencial

Consulta de Informações - Esta seção agrupa os requisitos funcionais associados à consulta de informações registrada no sistema.

RF017 Consultar/Imprimir Anamnese Módulo: M03.01

O usuário poderá consultar na tela ou através da opção de impressão, todas as informações da Anamnese do paciente. No caso de impressão, automaticamente sairá descrito no rodapé do relatório "Documento informativo. Não possui valor médico".

Prioridade: Essencial

RF018 Consultar/Imprimir Relatórios - Módulo: M03.02

O usuário poderá consultar na tela ou através da opção de impressão, todas as informações registradas no módulo de Acompanhamento (M02), como os Lançamentos Diários, Relação de Lembretes Ativos, Relação da Alimentação Atual e Relação da Medicação Atual. No caso de impressão, automaticamente sairá descrito no rodapé do relatório "Documento informativo. Não possui valor médico".

Prioridade: Essencial

Embora não exista um modelo consagrado para especificação de requisitos o mais usado é baseado nos padrões propostos pelo RUP. O importante é garantir a qualidade do processo e a Norma ISO/IEC 9126 define seis características de qualidade de *software* que devem ser avaliadas:

- Funcionalidade (finalidade do produto) - Conjuntos de recursos; Habilidades; e Segurança;
- Usabilidade (esforço para utilizar, aprender o produto) - Fatores humanos; Estética; Consistência na interface do usuário; Ajuda on-line e contextual; Assistentes e agentes; Documentação do usuário; e Materiais de treinamento;

- Confiabilidade (frequência de falhas, recuperabilidade) - Frequência e gravidade de falha; Possibilidade de recuperação; Possibilidade de previsão; Exatidão; e Tempo médio entre falhas;
- Eficiência (característica relacionada ao desempenho) - Velocidade; Eficiência; Disponibilidade; Exatidão; Taxa de transferência; Tempo de resposta; Tempo de recuperação; e Uso de recurso;
- Suportabilidade (esforço necessário para modificar) - Possibilidade de teste; Extensibilidade; Adaptabilidade; Compatibilidade; Possibilidade de configuração; Possibilidade de serviço; Possibilidade de instalação; Possibilidade de localização (internacionalização); e
- Portabilidade (capacidade de transferir o produto para outros ambientes).

Vimos que sem uma boa gerencia de requisitos, a possibilidade de fracasso na construção de um produto de *software* é bem maior. Mas, para reforçar essa engenharia, surge o Paradigma Orientado a Objetos (POO), no qual os requisitos podem ser modelados e validados através de casos de uso que incluem o diagrama de casos de uso e a especificação do caso de uso. Um caso de uso representa uma funcionalidade completa, conforme percebida pelo ator e é definido como um conjunto de sequências de ações que um sistema executa e que produzem um resultado observável por um particular ator. Os casos de uso são uma das técnicas usadas para descrever claramente todos os requisitos de um dado sistema, esta técnica foi proposta por Ivar Jacobson em sua metodologia de desenvolvimento de sistemas orientados a objetos, visando identificar os requisitos de um sistema.

3.5 Atividades

Baseado no documento de especificação de requisitos de *software* do RUP. Disponível em: <

http://www.wthreex.com/rup/portugues/webtmpl/templates/req/rup_srs.htm >

Publique aqui um requisito de *software* baseado em uma necessidade sua.

Disponível em: < <https://tarefas-apoo.4shared.com/> >

3.6 Artefatos

Plano de Negócio. Disponível em < <http://goo.gl/Jfw2VK> >

Modelo de Negócio. Disponível em < <http://goo.gl/O3O6Ya> >

Glossário de Negócio. Disponível em < <http://goo.gl/SRgCx8> >

Visão. Disponível em < <http://goo.gl/liXVDn> >

Plano de Requisitos. Disponível em < <http://goo.gl/sLM5Lu> >

Requisitos de *Software*. Disponível em < <http://goo.gl/Bi5pJ> >

Proposta Comercial. Disponível em < <http://goo.gl/DI17e4> >

Termo de Abertura do Projeto. Disponível em < <http://goo.gl/CYnUmT> >

3.7 Sugestões de Leitura

Análise e Projeto com UML. Disponível em < <http://goo.gl/Q5Hp7q> >

Análise e Projeto de Sistemas de Informação. Disponível em < <http://goo.gl/ITZiYS> >

Engenharia de Software. Disponível em < <http://goo.gl/xk0CqE> >

UML Guia do Usuário. Disponível em < <http://goo.gl/av5BTh> >

3.8 Referências Web

Técnicas para Coleta de Dados. Disponível em < <http://goo.gl/rX7MSA> >

Técnicas para Levantamento de Requisitos. Disponível em < <http://goo.gl/3uO5Ji> >

Técnicas para Levantamento de Requisitos Método Volere. Disponível em < <http://goo.gl/z2vm1D> >

Técnicas para Levantamento de Requisitos Não Funcionais. Disponível em < >

Standish Group. Disponível em < <http://goo.gl/9HaAst> >

Standish Group Brazil. Disponível em < <http://goo.gl/NCpTzD> >

Standish Group Report Chaos. Disponível em < <http://goo.gl/KOyh4> >

Standish Group Manifesto 2013. Disponível em < <http://goo.gl/ShD7HW> >

3.9 Referências Bibliográficas

RUMBAUGH, James; *et al.* Modelagem e Projetos Baseados em Objetos com UML. 2ªed. RJ: Campus, 2006.

RUMBAUGH, James; JACOBSON, Ivar; BOOCH, Grady. UML: Guia do Usuário. 1ªed. RJ: Campus, 2006.

SOARES, Antônio. Introdução, Identificação e Análise em Engenharia de Requisitos, 2005.

SOMMERVILLE, Ian. Engenharia de Software. 8ªed. Pearson, 2007.

4 Introdução à Análise e Modelagem

A disciplina de análise e modelagem de software tem o objetivo de descrever o desenvolvimento de um produto de *software* fazendo uso do paradigma orientado a objetos, explicando os fundamentos da engenharia de requisitos empregando os casos de uso para capturar requisitos e a simbologia da Linguagem de Modelagem Unificada (LMU) com sua aplicação nos projetos, com o intuito de evidenciar a necessidade de se fazer a formalização através da modelagem da análise dos requisitos.

Como apresentado no capítulo 1 o artigo Um Mundo Feito (quase completamente) de *Software* que expõe de uma forma bem elucidativa a consequência do não uso de processos de software seguimos neste novo capítulo com a apresentação dos conceitos de análise e modelagem, essenciais para um bom desenvolvimento de software.

4.1 Desenvolvimento de Software

O *Standish Group* com os seus *Chaos Report* nos permite examinar o andamento do mundo dos projetos de TIC e assim podemos perceber que apesar das melhoras em relação ao período da crise do *software*, ainda não são boas. Com a finalidade de coletar e analisar a taxa de sucesso para projetos no setor de TI o relatório *Chaos*, nos fornece um panorama do mercado.

Os indicadores utilizados para essa mensuração são:

- *Successful* (sucesso) - representa o % de projetos aprovados, isto é, entregues no prazo, no orçamento e com as funcionalidades requeridas pelo cliente atendidas;
- *Challenged* (desafio) - representa os projetos que passaram do prazo, do orçamento e incluíram menos requisitos dos solicitados pelo cliente; e
- *Failed* (falha) - são os projetos que foram cancelados antes da conclusão, nunca entregues ou mesmo nunca usados.

Existem algumas críticas quanto os dados coletados, pois alguns alegam que só se mede orçamento, prazo e escopo, não se avaliando risco, qualidade e satisfação. Mas, mesmo assim esse relatório é uma das melhores referências acompanhar a evolução do desenvolvimento de projetos na área de TIC.

Observe a próxima figura de número 5.1 para ter uma ideia em percentuais das situações dos projetos de *software* comparados desde 1994 até 2009 utilizando os parâmetros analisados: sucesso, desafio e falha.

Figura 4.1 Chaos



Fonte: Standish Group

4.2 Adhoc (Desenvolvimento por Demanda)

Desenvolver sem um processo, baseado em uma metodologia, seja tradicional ou ágil, é um grande risco. Para ser mais acadêmico é um desenvolvimento *Adhoc*, isto é, um desenvolvimento por demanda, sem planejamento. O sucesso no projeto nesse caso está normalmente relacionado ao fator sorte. Para não ficar na dependência desse fator fazer uso de um processo é aconselhável. Mas, por que desenvolver "sem qualidade" é desenvolver *Adhoc*. Vamos primeiro entender esse termo.

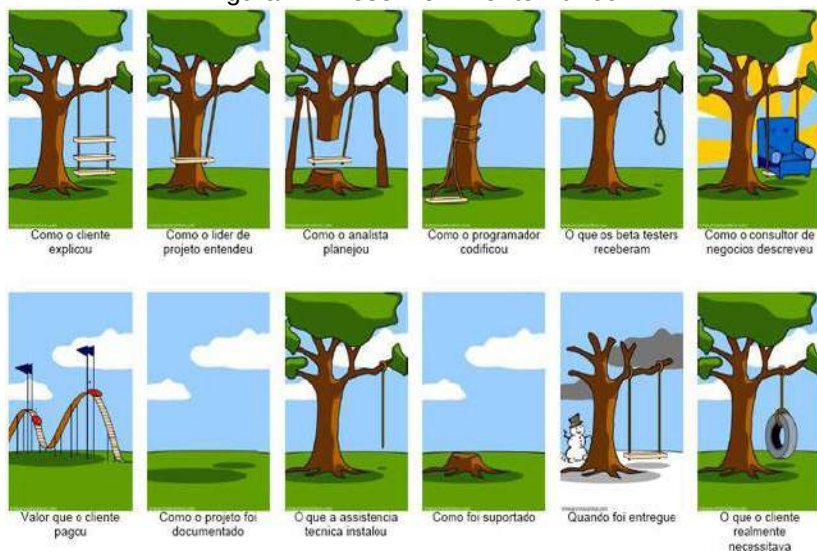
Adhoc é uma expressão de origem latina, que pode ser traduzida literalmente como sendo "para esta finalidade". Na Administração esse termo designa uma forma de administrar, chamada de adocracia, para quando o trabalho é realizado por demanda, diferenciando de outras formas de administrar chamadas de burocracia, meritocracia, etc. Na engenharia de *software*, o uso da expressão *Adhoc* serve para designar trabalhos de construção de *software* que não foram planejados para atendimento de uma demanda do usuário, sendo assim não se tendo um planejamento, consequentemente sem estimativas reais de prazo e custo, que pode ser muito abaixo ou acima da realizada, já que conta com a experiência para fornecer esses valores e não com uma métrica para cálculo dos prazos e custos.

O processo de qualidade CMMI nível 1 faz referencia dessa expressão para designar modelos informais utilizados pelos desenvolvedores de *software*, não oferecendo uma linguagem padronizada para comunicação de ideias, gerando dificuldade no compartilhamento dessas informações entre os envolvidos.

Resumindo, desenvolver *Adhoc*, é fazê-lo por demanda, sem planejamento. E se com planejamento já é complicado se estar entre os projetos que possuem ter sucesso, imagine, se depender do fator sorte. Para um melhor entendimento do

que normalmente acontece no desenvolvimento de *software* sem a preocupação com qualidade segue a figura 5.2 para exemplificar esse tipo de desenvolvimento.

Figura 4.2 Desenvolvimento *Adhoc*



Fonte: Internet

Percebe-se que pela primeira imagem (como o cliente explicou) para a última imagem (o que o cliente realmente necessitava) existe uma grande diferença, e essa é resultante de falha na comunicação. Isso, esse componente é vital para o sucesso no desenvolvimento de *software*. Para isso existe uma área de pesquisa chamada de engenharia de requisitos. Sim, elicitar requisitos é vital para o sucesso, isto é, entender realmente o que o cliente deseja e poder atendê-lo, no prazo e custo aceitáveis é um dos fatores de sucesso. Mas, isso não é uma tarefa fácil.

Então o que é um processo *Adhoc*? É um processo imaturo! Isto é:

- Processo improvisado por desenvolvedores e gerentes;
- Não é rigorosamente seguido e o cumprimento não é controlado;
- Dependente de pessoas;
- Visão do progresso e da qualidade deficientes;
- Prazos sobrepõem satisfação do cliente e qualidade;
- Custos de manutenção são excessivos; e
- Qualidade é difícil de prever.

Segundo DeMarco (1999). "Não se consegue controlar o que não se consegue medir". Talvez fique mais claro o que esse grande pesquisador de gestão de

projetos e autor de várias obras na área, quis dizer, é que sem planejamento, não se consegue ter parâmetros (medidas) para acompanhar, e assim, não se consegue verificar, se o que foi planejado, foi realizado.

E pior não se tem gerencia sobre sua execução, resultando em respostas típicas a perguntas frequentes em desenvolvimento:

- Como está o projeto? Andando!
- A atividade que você está responsável, como está? Estou fazendo!
- Quando fica pronta? Estou terminado!

Finalizando, usar os verbos no gerúndio é sinônimo de falta de planejamento.

4.3 Projetar (Desenvolvimento com Planejamento)

Planejar é o primeiro grande passo. Esse planejamento passa por definir uma metodologia e escolher um processo maduro e aceito pelo mercado. Mas, isso é só o primeiro passo. Então planejar não é tudo, mas um dos pontos necessários para aumentar as chances de sucesso.

Outro é fazer uma boa análise. E o que é fazer análise? Um exemplo que gosto de usar nas minhas aulas é fazer a analogia a um atendimento médico. Vamos lá. Imagine que existe um indivíduo com uma forte dor de cabeça. Dor essa que persiste há muito tempo, tanto tempo que até já se esqueceu quando começou. E nesse período esse nosso indivíduo, já fez de tudo para resolvê-la se consultar um especialista.

Já tomou remédio por conta própria. Já fez simpatias. Mas, nada disso surtiu efeito, e o pior cada vez a dor de cabeça aumentava. Assim, um dia. Sem mais condições de continuar a vida com essa não desejada acompanhante, ele foi ao médico.

Existem dois tipos de médicos (especialistas). Um ruim e outro bom. No nosso exemplo, o primeiro a ser consultado foi o ruim. Claro nosso paciente não sabia, mas o foi. Por quê? Obvio, ele era o mais barato. Esse, por sua vez, quando atendeu nosso fictício paciente, nem mesmo levantou o rosto para vê-lo. Estava escrevendo e assim continuo. Caso essa descrição se assemelhe a algo conhecido. É mera coincidência. Escrevendo estava, escrevendo continuou, sabe-se lá o quê. E perguntou:

- (especialista) Qual o seu problema?
- (paciente) Doutor. Estou com uma dor de cabeça muito forte.
- (especialista) Hum. Trabalha muito?
- (paciente) Sim, como todo bom brasileiro, que tem que pagar as contas!
- (especialista) Hum. Estresse. Tome esse remédio!

- (paciente) Vou ficar bom?
- (especialista) Sim!

Bom, é claro que não ficou e sim piorou. Então, nosso paciente foi para o segundo (o bom). Claro, mais caro. Mas, como se diz "às vezes o barato, sai mais caro". E lá chegando veja a diferença:

- (Especialista) Qual o seu problema?
- (Paciente) Doutor. Estou com uma dor de cabeça muito forte.
- (Especialista) E o que já fez para tentar resolvê-la?
- (Paciente) Dê tudo. O Senhor nem vai querer saber.
- (Especialista) Bom, infelizmente vou. Para ter uma exata proporção do que já aconteceu!
- (Paciente) Ok. Blá-blá-blá.
- (Especialista) Agora, conhecendo o seu histórico, preciso solicitar alguns exames!
- (Paciente) Ok. Quais?
- (Especialista) Sangue, como é na cabeça - uma ressonância, ...

Tempos depois: Feitos os exames; Exames entregues; Solicitados outros; Mais exames entregues; Mais dias solicitados para realizar análises; e Solicitações de parecer de outros especialistas. É claro que, com todo esse tempo, o paciente estava irritado, pois ainda estava com a dor de cabeça e a mesma aumentando, e claro, indignado com a demora. Mas, no dia marcado para saber o resultado, o médico lhe diz:

- (Paciente) O que o Doutor pode fazer para resolver o meu problema da dor de cabeça?
- (Especialista) Senhor a sua dor de cabeça é apenas o sintoma, o efeito do seu problema!
- (Paciente) Humm.
- (Especialista) O Senhor possui um tumor na cabeça! Esse é seu problema!
- (Paciente) HUMMMM – Nossa!
- (Especialista) Não! Existe solução.
- (Paciente) Hum.
- (Especialista) São três as alternativas:
- (Especialista) 1a. Cirurgia, cara, mas cura rápida;
- (Especialista) 2a. Quimioterapia, não tão caro, mas mais lenta; e
- (Especialista) 3a. Remédio para dor de cabeça, muito barato, mas o Senhor vai ter uma vida curta!
- (Paciente) Nossa?!
- (Especialista) A decisão é sua!

Essa pequena história mostra a importância dos exames, das análises e das possíveis soluções e consequências. Agora, substitua o paciente por cliente, o

especialista pelo analista, a dor de cabeça pela falta de sucesso na empreitada, o sintoma pelo efeito, a causa pelo fator, os exames pelas análises, a solução pelo projeto, as alternativas pelas propostas, e pronto temos o mesmo quadro para os projetos de TIC. Inclusive com as mesmas consequências, pois caso a opção escolhida seja a terceira, (a possibilidade de falência do primeiro exemplo é de uma pessoa física a do segundo é de uma pessoa jurídica), mas as consequências são tão desastrosas quanto..

Então, o que fazer para aumentar as chances de sucesso? Planejar, analisar e projetar soluções.

4.4 Modelagem

Os projetos seguem processos de desenvolvimento e um dos processos mais maduros é o Processo Unificado (UP) que possui 4 marcos: Marco dos objetivos do ciclo de vida; Marco da arquitetura do ciclo de vida; Marco da capacidade operacional inicial; e Marco do lançamento do produto.

Passado pelo primeiro marco tem que começar a modelar os requisitos especificados. O cliente necessita entender o que os projetistas estão construindo e precisam ter condições de inferir os seus conhecimentos nos projetos para atender plenamente suas necessidades.

Para isso é necessário estabelecer um canal formal de interação onde a linguagem natural do cliente é transformada em linguagem técnica para a equipe de desenvolvimento. Com esse objetivo a UML se propõe a ser uma linguagem padrão aceita e compreendida por todos os *stakeholders*.

A modelagem utilizando a Linguagem de Modelagem Unificada, mais conhecida através da sigla em inglês UML que significa *Unified Modeling Language* é focada no paradigma Orientado a Objetos (OO) cujos conceitos são classe, objeto, herança, polimorfismo, encapsulamento de atributos e métodos, alta coesão e baixo acoplamento. É usado para a análise não focando na codificação do *software* ou *hardware* e sim no entendimento do problema (análise) e na sua solução (projeto).

A UML representa símbolos, esses usados em diagramas que assim representam uma linguagem simbólica com regras claras e precisas para utilização desses símbolos nos diversos diagramas. O objetivo dos diagramas é apresentar múltiplas visões do sistema chamado de modelo. Assim, um modelo UML é um conjunto de diagramas que servem para compreender e desenvolver um projeto de *software*, descrevendo o que o *software* deve fazer.

4.5 Linguagem de Modelagem Unificada (UML)

Durante a década de 80, várias linguagens de programação se popularizaram, entre elas uma que se destacou foi à chamada de Smalltalk, mas outras linguagens também participaram deste movimento, tais como, Objective C, C++ e Eiffel, todas projetadas para o paradigma orientado a objetos. Seguindo esta onda, os métodos orientados a objetos começaram a ser publicados. Como exemplo temos em 1988 os de Shlaer, em 1990 os de Wirfs-Brock, em 1991 os de Coad e Yourdon, ainda em 1991 de Booch, e também em 1991 os de Rumbaugh.

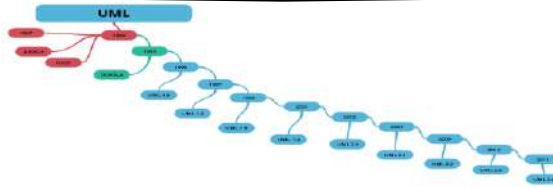
A primeira tentativa de sucesso de se criar uma linguagem unificada aconteceu quando Rumbaugh se juntou a Booch na *Rational Software Corporation* em 1994. Eles começaram a combinar conceitos de OMT com os métodos de Booch, produzindo um método unificado. Neste mesmo momento Jacobson também se juntou a *Rational* e começou a trabalhar com Booch e Rumbaugh. O trabalho conjunto resultou no que foi chamado de *Unified Modeling Language* (UML). Este foi um momento único, quando os autores dos três principais métodos trabalharam juntos para unificar suas propostas. Veja na figura 5.3 como se deu essa evolução nas suas versões.

Em 1996, a *Object Management Group* (OMG) fez uma chamada por propostas com o objetivo de criar um padrão para modelagem orientada a objetos. Aos autores da UML começaram a trabalhar com metodologistas e desenvolvedores de outras empresas para produzir uma proposta atrativa para os membros da OMG, assim como para as empresas de ferramentas, metodologistas e desenvolvedores em geral. No final de 1997, se tornou um padrão da OMG e até hoje vem sendo mantida e evoluída, sendo hoje bem aceita por toda a comunidade de desenvolvimento de *software*.

A *Unified Modeling Language* (UML) é a sucessora de uma onda de métodos para análise e projeto orientados a objetos que surgiram no fim da década de 80. Ela unificou os métodos proposto por Booch, Rumbaugh e Jacobson. Atualmente a UML é um padrão mantido pela OMG, o que dá a ela um status de linguagem padrão para modelagem de sistemas, isto é, uma linguagem padrão para visualizar, especificar, construir e documentar artefatos de um sistema de *software* e combina diversos aspectos como:

- Modelagem de Dados
- Modelagem de Negócios
- Modelagem de Objetos
- Modelagem de Componentes

Figura 4.3 Evolução da UML

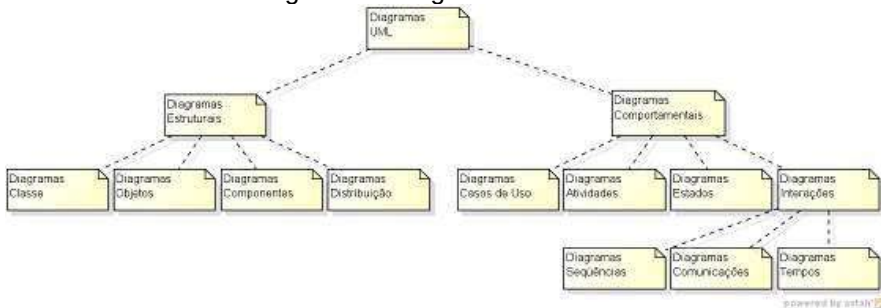


Fonte: Próprio Autor

É importante enfatizar que UML é uma linguagem de modelagem, ou seja, é uma forma de especificar um sistema de maneira visual. Ela é a linguagem padrão de modelagem adotada pelo *Rational Unified Process* (RUP) e seu grande sucesso se deu em conjunto com a popularização do RUP. Por isso, é muito comum que as pessoas associem UML ao RUP.

É importante distinguir muito bem, visto que as duas representam conceitos complementarmente diferentes. O RUP é um processo, ou seja, um conjunto de atividades, com seus respectivos artefatos e papéis, para se desenvolver produtos de *software* e a UML é uma linguagem para se modelar o comportamento e estrutura de um sistema. Na figura 5.4 se pode verificar quais são os principais diagramas desse linguagem.

Figura 4.4 Diagramas da UML

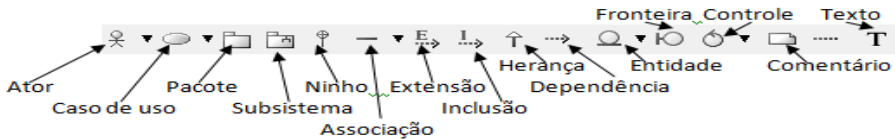


Fonte: Próprio Autor

4.5.1 Diagrama de Casos de Uso

O diagrama de caso de uso segundo Ivan Jacobson, um caso de uso é “um documento narrativo que descreve a sequência de eventos de um ator que usa um sistema para completar um processo”. Um caso de uso é uma técnica de modelagem que descreve o que o novo sistema deve fazer e se baseia em um processo de licitação de requisitos entre os *stakeholders* na condução de uma especificação na qual todos interagem. Na figura 5.5 temos os elementos utilizados nesse diagrama e na figura 5.6 os dois principais símbolos.

Figura 4.5 Elementos do Diagrama de Casos de Uso



Fonte: Próprio Autor

Um caso de uso descreve quais comportamentos o sistema deverá responder para cada um dos usuários do mesmo, servindo de formalização das ações que precisão ser desenvolvidas. Retratando uma lista de eventos entre os usuários e o sistema em uma visão abstrata, onde essa lista de eventos relatados abstratamente descreve as interações desde o início da atividade até o fim da mesma.

Figura 4.6 Símbolos da UML



Fonte: Próprio Autor

Casos de uso têm que ser compreensíveis pelos usuários por que só eles sabem o que o sistema precisa fazer. Os casos de uso permitem verificar se o desenvolvedor e o usuário concordam sobre o que o sistema deve fazer. Isso é um problema importante no desenvolvimento de *software*. No mesmo tempo, casos de uso podem servir de "contratos" entre os usuários e os desenvolvedores. Sendo assim os casos de uso devem:

- Descrever os requisitos funcionais do sistema;
- Fornecer uma descrição clara do que se deve fazer; e
- Definir os comportamentos das classes.

É importante salientar que casos de uso não são requisitos. E sim uma visão abstrata desses os quais são compostos pelos seguintes componentes:

- Ator, que é um papel que solicita eventos ao sistema e recebe feedbacks, onde cada ator pode participar de vários casos de uso; e
- Casos de uso são documentos narrativos que descrevem a sequência dos eventos feitos por um ator no uso do sistema.

Os elementos que descrevem esses papéis são exibidos abaixo. Vale salientar que o nome dos casos de uso deve ser composto por frases curtas, preferencialmente verbo + substantivo. Exemplo: matricular aluno.

Para especificar um caso de uso pode-se usar a estrutura que se segue, porém não existe uma estrutura rígida:

- Código do caso de uso;
- Nome do caso de uso;
- Descrição do caso de uso (um parágrafo);
- Lista dos nomes dos atores com descrição curta;
- Prioridade, caso de uso muito importante no projeto ou acessório;
- Pré-condições;
- Pós-condições;
- Fluxo de eventos principal;
- Fluxos de eventos alternativos;
- Fluxo de eventos de exceções;
- Cenários; e
- Dicionário de Dados.

Para identificar os atores que vão participar do modelo uma das técnicas é fazer as seguintes perguntas:

- Quem usa o sistema?
- Quem inicia o sistema?
- Quem fornece os dados?
- Quem usa as informações?

Na descrição dos atores geralmente se faz o uso de:

- Nome do caso de uso;
- Tipo de uso se é frequente, ocasional, etc;
- Descrição de seu papel no sistema.

Para identificar os casos de uso, normalmente a técnica usada é:

- Verificar as interações entre os atores e o sistema; e
- Verificar as ações do ator e do sistema.

É importante salientar que os atores sempre iniciam a ação. Para entendimento imagine o requisito em que um aluno necessita fazer um curso, onde existe a necessidade de um sistema de matrículas. Então:

- Quais são os atores?
- Quem usa o sistema?
- Como podemos identificar o caso de uso?

Podemos chamar este caso de uso de: Matricular Aluno (verbo+substantivo). Para fazer uma descrição textual do caso de uso, que possui dois atores Aluno e Administração. Sua especificação poderia ser:

- Caso de uso: Matricular Aluno.
- Atores: Aluno, Administrador.
- Descrição: Este caso de uso começa quando um aluno chega à administração que faz a matrícula do aluno no curso.

Fluxo:

- P1-O sistema registra a matrícula;
- P2-Recebe o pagamento;
- P3-Emite uma nota fiscal; e
- P4-O aluno é matriculado.

Na UML temos o diagrama de caso de uso que pode ser representado para o caso acima como mostrado na figura 5.7.

Figura 4.7 Diagrama de Caso de Uso



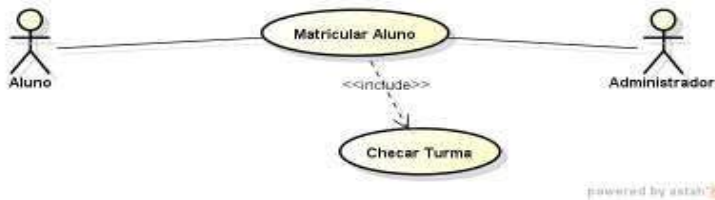
Fonte: Próprio Autor

Importante: o caso de uso deve começar com um verbo, enfatizando o processo. Exemplo: Matricular Aluno ou Cadastrar Curso. Deve-se identificar o ator que inicia o evento, começando a descrição da sequência de um caso de uso usando o esquema - este caso de uso começa quando um aluno faz uma matrícula em um curso. Verifique que um ator é um elemento externo ao sistema. Um ator é um papel que interage com o sistema, mas não faz parte dele.

O primeiro passo é identificar os casos de uso básicos, o próximo passo é identificar o relacionamento entre os casos de uso através das generalizações, inclusões e extensões.

As inclusões são um caso de uso que inclui o comportamento de outro. A finalidade básica da inclusão é realizar uma decomposição funcional e reduzir a complexidade de um caso de uso.

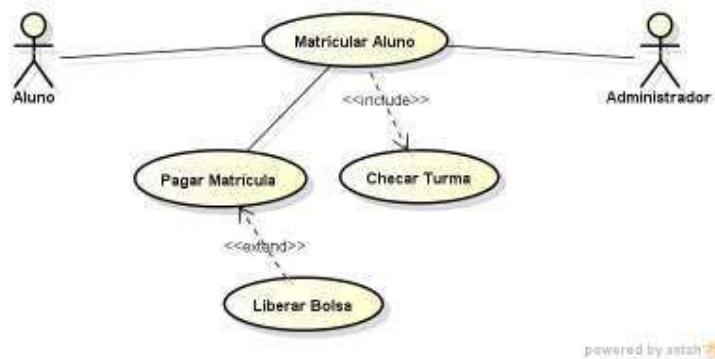
Figura 4.8 Diagrama de Casos de Uso com Inclusão



Fonte: Próprio Autor

As extensões definem pontos que adicionam um comportamento opcional, assim o caso de uso base pode ser executado mesmo que a extensão não o seja.

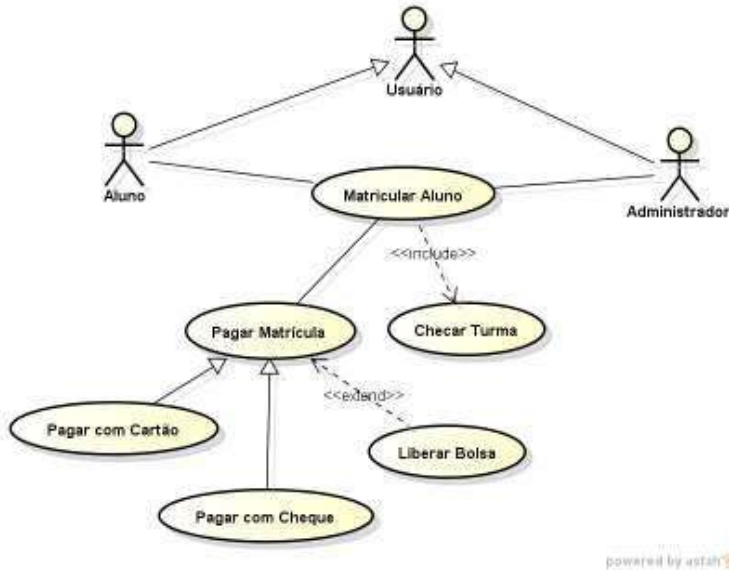
Figura 4.9 Diagrama de Casos de Uso com Extensão



Fonte: Próprio Autor

De princípio as generalizações são mais indicadas para as características dos atores, porém podem ser usadas com cautela para indicar especializações de comportamentos.

Figura 4.10 Diagrama de Casos de Uso Realizar Matrícula



Fonte: Próprio Autor

4.5.2 Diagrama de Classes

O diagrama de classes na Programação Orientada a Objetos (POO), os problemas são pensados e expressados como objetos, ao contrário da análise tradicional os quais eram em rotinas e dados, que aqui foram substituídos por métodos (comportamento) e atributos (propriedades). Assim, quando é colocado o problema de desenvolver um sistema acadêmico na análise orientada a objetos, deve-se pensar como dividir esse problema em objetos. Assim teríamos Alunos, Administradores, Professores, Cursos, Turmas, etc. E a melhor maneira de conceituar estes termos é considerar um objeto do mundo real e mostrar como podemos representá-lo em termos conceitos em POO.

Assim, um objeto é um conceito que usamos para representar uma entidade do mundo real. Exemplificando, um Aluno possui nome, data de nascimento, identidade, etc. E além dessas características (propriedades), possuem ações (métodos) como frequentar as aulas, fazer as avaliações, etc. Em termos de POO para podermos tratar os objetos temos que criar classes. Assim, uma classe representa um conjunto de objetos que possuem comportamentos e características comuns.

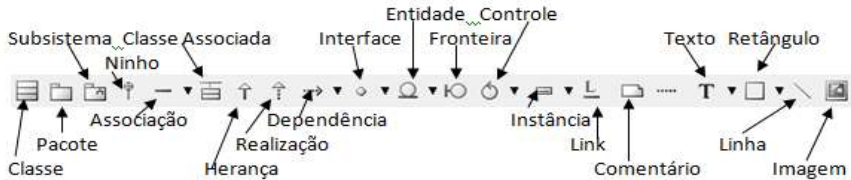
Na UML em primeiro lugar denomina-se uma classe e recomenda-se nomeá-la capitalizando-a e deixando no singular. E em segundo lugar denominam-se as propriedades, informações específicas relacionadas a uma classe de objeto, isto é características dos objetos que as classes representam. Em terceiro e última parte, métodos, que são ações que os objetos de uma classe podem realizar.

Assim tem-se um modelo para instanciar quantos objetos forem necessários. Dessa forma os objetos instanciados possuirão todas as características e comportamentos definidos pela classe. Então as classes especificam a estrutura (propriedades) e os comportamentos (operações) dos objetos, que são instâncias das classes.

Geralmente em um sistema de médio porte serão identificadas diversas classes que compõem o sistema. Neste contexto a UML surgiu como uma proposta de ser uma linguagem para modelagem de dados que usava diversos artefatos para representar o modelo de negócio; um destes artefatos é o diagrama de classes.

Observe a figura 5.11 para verificar os elementos que são utilizados nos diagramas de classes e na figura 5.12 a simbologia de um diagrama de classe definido na UML.

Figura 4.11 Elementos do Diagrama de Classes



Fonte: Próprio Autor

Os diagramas de classes registram atributos e operações de uma classe e as restrições de como os objetos podem ser conectados, descrevendo também os tipos de objetos no sistema e os relacionamentos entre eles e esse podem ser associações e abstrações. Para poder representar a visibilidade dos atributos e operações, a UML usa os seguintes símbolos: + público, visível em qualquer classe; - privado, visível somente dentro da classe; # protegido, na classe e suas subclasses.

Figura 4.12 Classe



Fonte: Próprio Autor

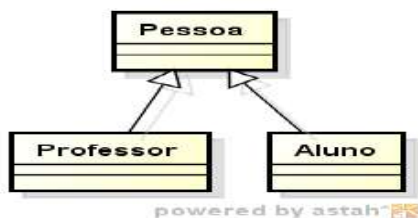
O relacionamento entre classes retrata as relações entre os objetos. Exemplo: um professor ministra uma disciplina para alunos numa sala. A UML reconhece

três tipos mais importantes de relações: dependência, associação e generalização ou herança. Geralmente as classes não estão isoladas (coesas) e se relacionam entre si (acoplamento). O relacionamento e a comunicação entre as classes se definem em três tipos: Associações que podem ser de Agregação ou Composição; Generalização ou herança; e Dependências.

As associações são relacionamentos estruturais entre instâncias e especificam que objetos de uma classe estão ligados a objetos de outras classes, podendo ser unária, binária, etc. As associações podem existir entre classes ou entre objetos. Uma associação entre a classe Professor e a classe Disciplina (um professor ministra uma disciplina) significa que uma instância de Professor (um professor específico) vai ter uma associação com uma instância de uma Disciplina. Esta relação significa que as instâncias das classes são conectadas, seja fisicamente ou conceitualmente. Dependências são relacionamentos de utilização no qual uma mudança na especificação de um elemento pode alterar a especificação do elemento dependente. A dependência entre classes indica que os objetos de uma classe usam serviços dos objetos de outra classe. Generalização ou herança, que pode ser simples ou múltipla (composta), serve para relacionar um elemento mais geral e um mais específico, onde o elemento mais específico herda as propriedades e métodos do elemento mais geral. Como a relação de dependência, ela existe só entre as classes. Um objeto particular não é um caso geral de outro objeto, apenas classes podem receber esse conceito. Agregação é um tipo de associação (é parte de ou todo-parte) onde o objeto parte é um atributo do todo, onde o objeto parte somente são criados se o todo ao qual estão agregados seja criado. Pedidos é composto por itens de pedidos. Composição é o relacionamento entre um elemento (o todo) e outros elementos (as partes) onde as partes só podem pertencer ao todo e são criadas e destruídas com ele. A seguir são mostradas as formas de relacionamento na UML.

Generalização é um relacionamento de especialização ou generalização, nos quais os objetos dos elementos especializados (os filhos) são substituíveis por objetos do elemento generalizado (os pais).

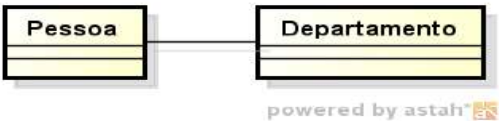
Figura 4.13 Diagrama de Classes com Herança



Fonte: Próprio Autor

Associação é um relacionamento estrutural que descreve um conjunto de ligações, em que as ligações são conexões entre objetos.

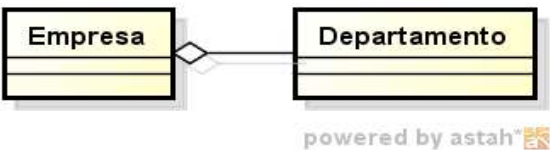
Figura 4.14 Diagrama de Classes com Associação



Fonte: Próprio Autor

Agregação é um tipo especial de associação representando um relacionamento estrutural entre o todo e sua parte.

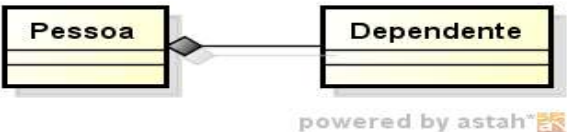
Figura 4.15 Diagrama de Classes com Agregação



Fonte: Próprio Autor

Composição é outro tipo especial de associação representando um relacionamento estrutural entre um elemento (o todo) e outros elementos (as partes) onde as parte só podem pertencer ao todo e são criadas e destruídas com ele.

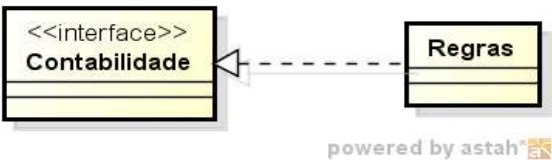
Figura 4.16 Diagrama de Classes com Composição



Fonte: Próprio Autor

Realização é um relacionamento semântico entre classes, em que uma classe especifica um contrato que outra classe garante executar.

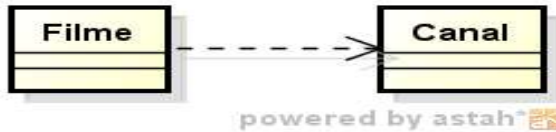
Figura 4.17 Diagrama de Classes com Realização



Fonte: Próprio Autor

Dependência é um relacionamento semântico entre duas classes, nos quais a alteração de um (a classe independente) pode afetar a semântica da outra (a classe dependente).

Figura 4.18 Diagrama de Classes com Dependência



Fonte: Próprio Autor

O diagrama de classes lista todos os conceitos do domínio que serão implementados no sistema e as relações entre os conceitos. Ele é muito importante, pois define a estrutura do sistema a desenvolver. O diagrama de classes é consequência do prévio levantamento de requisitos, definição de casos de usos e classes. Como exemplo tem os passos de elicitação de requisitos com os *stakeholders* do sistema a ser desenvolvido, usando a técnica de entrevista com os administradores, professores, etc, que a partir desses os objetos do sistema são definidos: Alunos, Professores, Turmas, Cursos, etc. Definição dos atores do sistema: aluno, professor, administrador, etc. Definição e detalhamento dos casos de uso: Matricular Aluno, Pagar Matrícula, etc. Definição das classes: alunos, professor, etc.

Figura 4.19 Diagrama de Classes com Relacionamento



Fonte: Próprio Autor

Atributo representa uma propriedade que todos os objetos da classe têm, porém cada objeto terá valores particulares para seus atributos. A UML, o nome de um atributo é um texto que deve capitalizar todas as primeiras letras de cada palavra no nome menos a primeira palavra.

Todos os métodos têm que respeitar exatamente a assinatura que é composta pelo nome, número de parâmetros, tipos de dados e ordem. Um método não pode acrescentar ou cortar um parâmetro. Para mandar a mensagem corretamente, devesse saber qual é a classe do objeto, já que cada classe tendo método com assinatura diferente.

Mais uma informação. Devido aos questionamentos de alguns alunos sobre a classificação em relação ao tipo de generalização, complementei essa aula

especificando essa tipificação. Ela pode ser basicamente de dois tipos: Total ou Parcial. Se Total temos para cada ocorrência da uma classe genérica sempre uma ocorrência em uma das classes especializadas. Se Parcial temos que nem sempre uma ocorrência da classe genérica possui uma ocorrência correspondente em uma classe especializada. Essa tipificação é muito interessante para a camada de persistência e pode ser também utilizada na estereotipação de entidades. Para melhor entender veja o exemplos na figura 5.20.



Fonte: Próprio Autor

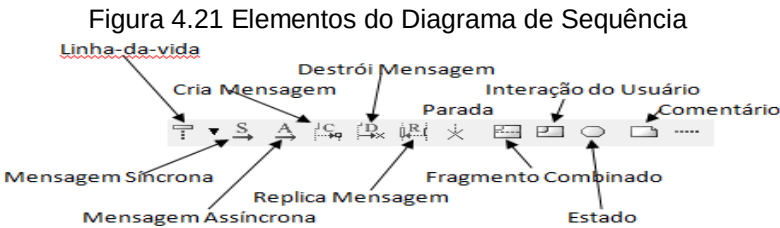
4.5.3 Diagrama de Objetos

Segundo os pais da UML (Rumbaugh, Jacobson e Booch) um objeto é uma instância de uma classe, isto é, trata-se de uma cópia da classe na memória em tempo de execução (*runtime*), sendo assim, um elemento específico que possui valores (estados) nos atributos (campos).

A UML representa um objeto usando um retângulo que o representa, onde o nome da instância do objeto é seguido de dois pontos, seguido do nome da classe, com essa formação sublinhada.

4.5.4 Diagrama de Sequência e Comunicação ou Colaboração

O diagrama de sequência serve para exibir as interações entre os vários componentes de um sistema em especial os objetos e como seus métodos interagem entre si e em qual ordem (sequência).



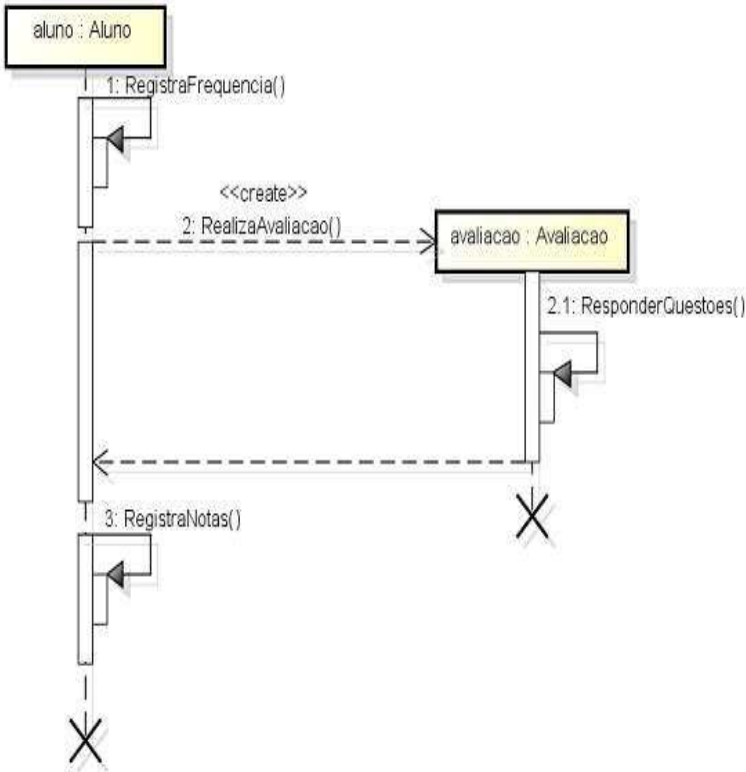
Fonte: Próprio Autor

O diagrama de comunicação serve para representar os elementos de um sistema que trabalham em conjunto. Esse diagrama expressa essa comunicação, mostrando a troca de mensagens entre os objetos.

Ele é similar ao diagrama de sequência, porém mais simples.

Os diagramas de sequência e de comunicação mostram as interações entre os objetos, por este motivo a UML se refere a estes diagramas como diagramas de interação.

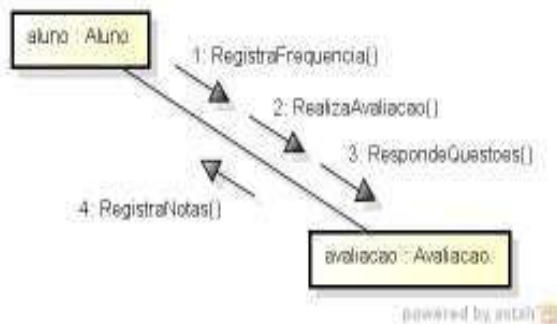
Figura 4.22 Diagrama de Sequência



powered by astah

Fonte: Próprio Autor

Figura 4.23 Diagrama de Comunicação

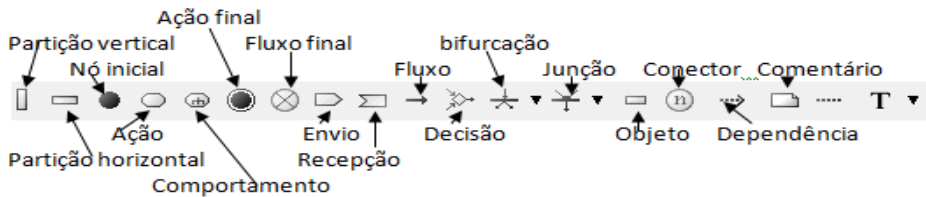


Fonte: Próprio Autor

4.5.5 Diagrama de Atividades

O diagrama de atividade exibe a forma que um objeto executa suas ações em um único processo, representando-os passo a passo, isto é, seu fluxo. As atividades que ocorrem em um caso de uso ou no comportamento do objeto seguem a sequência conforme descrito do diagrama.

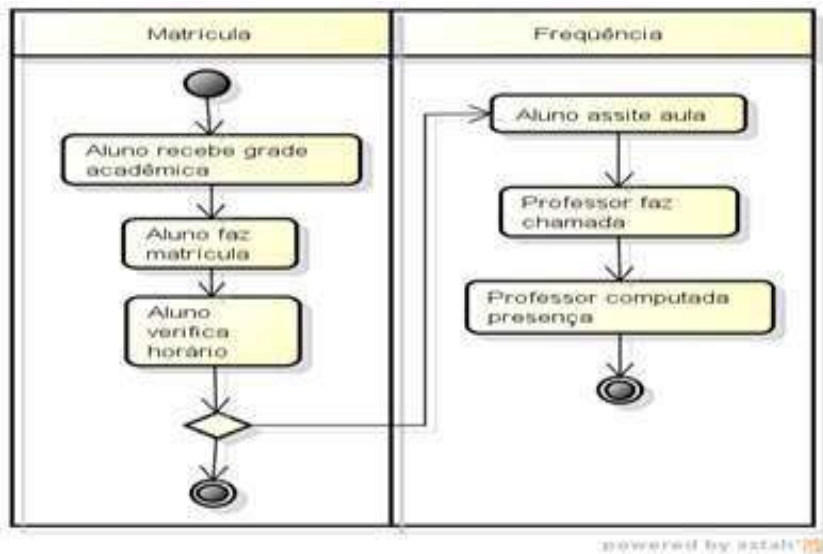
Figura 4.24 Elementos do Diagrama de Atividades



Fonte: Próprio Autor

Na figura de número 5.25 é exibido um exemplo de um diagrama de atividades que para os mais antigos podem observar o quanto é semelhante a um fluxograma, e isso se deve ao fato de ser uma herança deste último, pois a finalidade de ambos é a mesma. Representar a lógica de um processamento.

Figura 4.25 Diagrama de Atividades



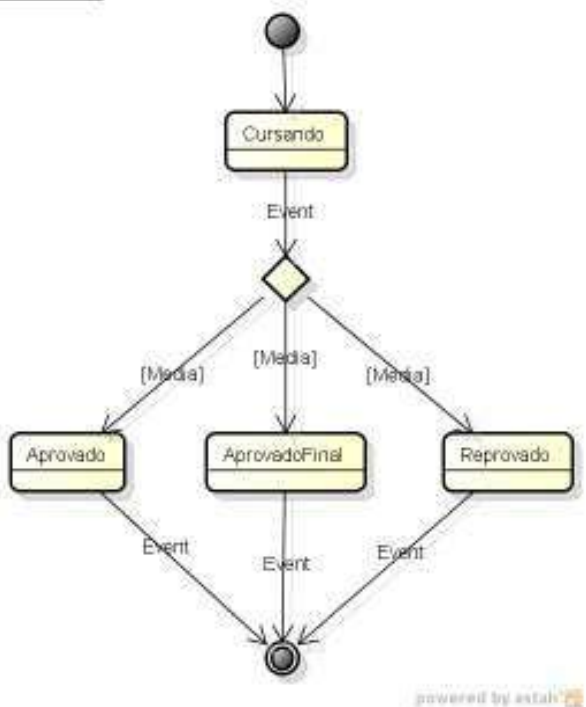
Fonte: Próprio Autor

4.5.6 Diagrama de Estados

O diagrama de estados tem a finalidade de exibir como um objeto realiza uma determinada operação num determinado momento da execução, representando um estado particular. Exemplo: um aluno pode estar com a situação de aprovado, aprovado na final ou reprovado.

O símbolo no topo da figura representa o início do estado e o símbolo na base da figura representa o fim do estado, entre elas são exibidas as transições de um estado para outro.

Figura 4.26 Diagrama de Estados



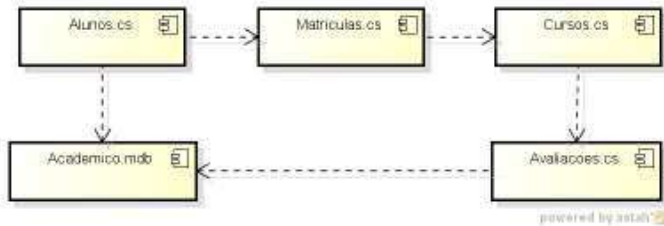
Fonte: Próprio Autor

4.5.7 Diagrama de Componentes

O diagrama de componentes mostra a organização dos componentes na implantação do sistema, mostrando os elementos (componentes) reutilizáveis de *software* e suas interdependências. Vale salientar que um componente é composto por um conjunto de classes, isto é, um *package* em Java ou *namespace* em C#. Na maioria das vezes um componente depende de outros, da mesma forma que as classes que ele possui, dependem das funcionalidades de outras classes do mesmo ou de outro componente. Desta forma o diagrama de componentes mostra esta dependência.

O diagrama de componentes também pode servir para exibir a configuração de um projeto de *software*, expondo a dependência entre os artefatos que compõem o projeto. Assim, as relações de dependência servem para informar a finalidade dos diversos artefatos que podem compor o projeto. E esses artefatos podem ser estereotipados para melhor identificá-los. Podendo ser um executável, um pacote, uma tabela, ou qualquer outro tipo de arquivo, estereotipado como documento ou às vezes arquivo.

Figura 4.27 Diagrama de Componentes

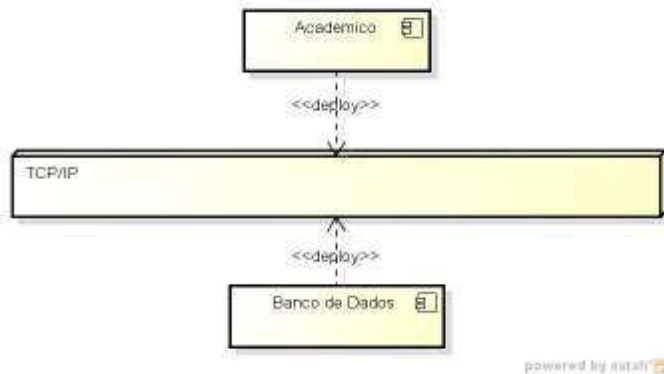


Fonte: Próprio Autor

4.5.8 Diagrama de Distribuição

O diagrama de implantação nos mostra a configuração física sobre qual o sistema será instalado. Esses diagramas servem para exibir a distribuição de *hardware* do projeto de *software*, identificando os recursos computacionais como pontos (nós) além da rede que os relaciona. Dessa forma, esse diagrama permite exibir a topologia de uma rede usada, como também os recursos (máquinas).

Figura 4.28 Diagrama de Distribuição



Fonte: Próprio Autor

4.5.9 Diagrama de Tempo

O diagrama de tempo serve para exibir a relação de tempo de execução dos objetos. Esse diagrama foi incluído na versão 2.0 para apresentar os comportamento dos objetos e suas interações em uma escala de tempo, focando as condições que mudam no decorrer desse período.

4.6 Exemplos

Para melhor entender como realizar uma análise orientada a objetos fazendo uso da Linguagem de Modelagem Unificada, segue um exemplo baseado na

necessidade de se controlar as solicitações de serviços da empresa ABC.
Especificação de Requisitos Funcionais:

- RF01. A empresa ABC necessita de uma aplicação que controle: o seu cadastro de seus clientes; o seu cadastro dos técnicos; e as solicitações de serviços realizadas.
- RF02. Para cada cliente são cadastrados os seguintes dados: código (que deve ser gerado automaticamente); o nome; endereço completo com o logradouro, número, complemento, bairro, cidade, e estado; e telefones.
- RF03. Quando o cliente fizer a primeira chamada por telefone, seus dados devem ser atualizados.
- RF04. Para o técnico deve se cadastrar: o nome; o CPF; endereço residencial completo; telefones; e data de admissão. Quando o técnico for desligado da empresa, deve se cadastrar a data da rescisão contratual.
- RF05. Quando o cliente solicitar um atendimento, deve-se cadastrar a identificação do cliente; a data e hora da solicitação; e o problema a ser solucionado; o status da solicitação deve ficar em “S” que significa Solicitado.
- RF06. Após a solicitação realizada ela deve ser atendida por um dos técnicos colocando a mesma no status “A” de atendimento e informando qual o técnico está fazendo esse atendimento, sendo registrada a data e hora do início do atendimento.
- RF07. Depois de realizado o atendimento o cliente deve ser avisado sobre o parecer deste escrito pelo técnico e o status deverá passar para “E” de encerrado sendo registrada a data e hora do encerramento.
- RF08. Caso alguma irregularidade ocorra nessa solicitação, o status deverá passar para “C” que indica cancelado, registrando o motivo do cancelamento.

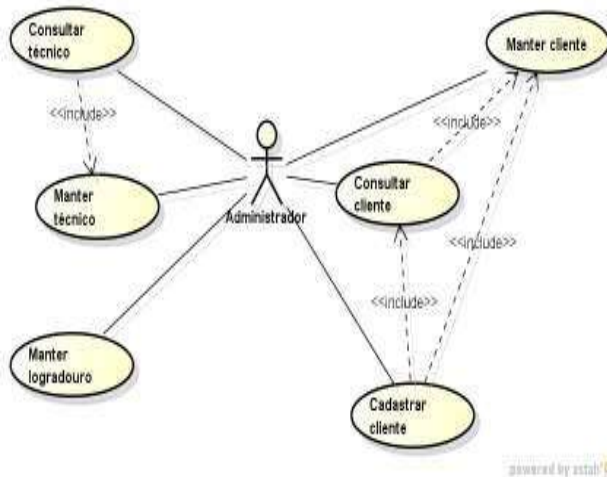
Figura 4.29 Especificação Resumida de Casos de Uso

Caso de Uso Consultar Cliente
Código: UC01
Nome: Consultar Cliente
Descrição: Tem a finalidade de exibir os clientes cadastrados possibilitando as opções de inclusão, alteração e exclusão.
Ator: Administrador
Fluxo Principal
1. Exibição de lista de todos os clientes cadastrados.
2. Oferece ao usuário:

- 2.1. Selecionar um cliente, para alterar seu cadastro;
- 2.2. Localizar um cliente ou conjunto de clientes por meio de pesquisa;
- 2.3. Escolher a opção de “ inserir cliente”.
3. Pesquisar. cliente
- 3.1. Encontrar cliente através do nome ou parte dele.
- 3.2. O sistema exibe a lista de clientes que satisfação ao critério, exibindo:
 - 3.2.1. Código de identificação;
 - 3.2.2. Nome do cliente;
 - 3.2.3. Telefone.
4. Inserção de Cliente
- 4.1. Include Caso de Uso Manter Cliente
5. Seleção de Cliente
- 5.1. Selecionar um cliente habilitando as opções de Alterar Cliente e Excluir Cliente.
- 5.2. Se selecionado usar Include Caso de Uso Manter Cliente.

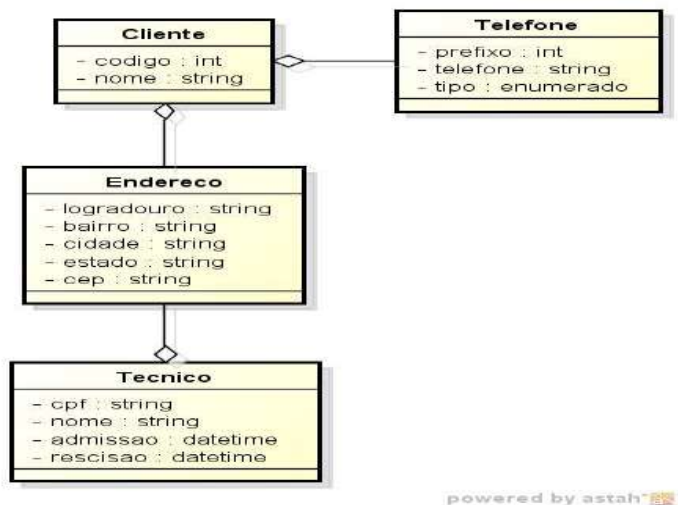
Fonte: Próprio Autor

Figura 4.30 Exemplo de Diagrama de Casos de Uso Consultar Cliente



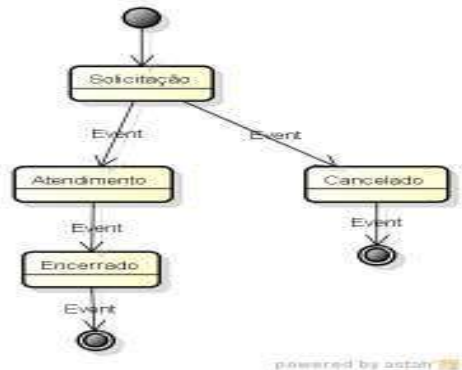
Fonte: Próprio Autor

Figura 4.31 Exemplo de Diagrama de Classes Consultar Cliente



Fonte: Próprio Autor

Figura 4.32 Exemplo de Diagrama de Estados Consultar Cliente



Fonte: Próprio Auto

4.7 Ferramentas

Astah Change-Vision. Disponível em < <http://astah.change-vision.com/en/index.html> >
Astah Community. Disponível em < <http://astah.change-vision.com/en/product/astah-community.html> >
Astah Profissional. Disponível em < <http://astah.change-vision.com/en/product/astah-professional.html> >
Astah Share. Disponível em < <http://astah.change-vision.com/en/product/astah-share.html> >
Astah UML. Disponível em < <http://astah.change-vision.com/en/product/astah-uml.html> >

CASE tools. Disponível em < <http://case-tools.org/uml.html> >
Manual do Astah. Disponível em < <http://goo.gl/oxXADk> >
Plugin de modelagem UML no NetBeans 6.9. Disponível em < <http://goo.gl/cKByrP> >
Plugin diversos para NetBeans 6.9. Disponível em < <http://dlc.sun.com.edgesuite.net/netbeans/6.9/rc/zip.html> >

4.8 Sugestões de Leitura

Análise e Projeto de Sistemas. Disponível em < <http://goo.gl/kQVR7L> >
Desenvolvendo Aplicações com UML 2.2. Disponível em < <http://goo.gl/vVW6IL> >
Engenharia de Software. Disponível em < <http://goo.gl/qZVIYT> >
Exercitando Modelagem com UML. Disponível em < <http://goo.gl/r7toX5> >
UML Essencial. Disponível em < <http://goo.gl/cdLkBT> >
O Homem A Administração A Sociedade. Disponível em < <http://goo.gl/MSkGE9> >

4.9 Referências Web

A Theory of Object-Oriented Design. Disponível em < [http://www.eden-study.org/articles/2002/isf4\(4\).pdf](http://www.eden-study.org/articles/2002/isf4(4).pdf) >
Aulas de UML. Disponível em < <http://goo.gl/2TRbUF> >
Design Patterns Languages for Object-Oriented Programs (DP). Disp.: < <http://www.vincehuston.org/dp/> >
Exemplo de Modelagem UML. Disponível em < <http://goo.gl/zwOYCF> >
Exemplo de Modelagem UML com a ferramenta ASTAH. Disponível em < <http://goo.gl/JvqhSt> >
Guia Básico sobre UML. Disponível em < <http://goo.gl/eGhizM> >
IBM Rational Unified Proces. Disponível em < ftp://public.dhe.ibm.com/software/pdf/br/RUP_DS.pdf >
Mais sobre relacionamentos de Inclusão e Extensão. Disponível em < <http://goo.gl/uH8X6Z> >
Manuais de Referência OMG-UML. Disponível em < <http://www.omg.org/spec/UML/> >
Modelagem UML. Disponível em < <http://goo.gl/iwtZh3> >
Object Management Group (OMG). Disponível em < <http://www.omg.org/> >
Object-Oriented Programming (OOP). Disp.: < <http://download.oracle.com/javase/tutorial/Java/concepts/index.html> >
Rational Unified Process (RUP). Disponível em < <http://www-01.ibm.com/software/br/rational/> >
Standish Group. Disponível em < <http://www.standishgroup.com/> >
Standish Group Report Chaos 2009. Disp.: < http://www1.standishgroup.com/newsroom/chaos_2009.php >
Template Mais Detalhado para Especificação de Casos de Uso. Disp.: < <http://goo.gl/5yKpBD> >

Template Casos de Uso. Disp.: <

http://www.wthreex.com/rup/portugues/webtml/templates/req/rup_ucspec.htm >

Unified Modeling Language (UML). Disponível em < <http://www.uml.org/> >

Um Mundo Feito (quase completamente) de Software. Disponível em < <http://goo.gl/YE26CS> >

4.10 Referências Bibliográficas

BENNETT, Simon; *et al.* Schaum's Outline of UML. 1ªed. UK: McGraw-Hill, 2005.

BEZERRA, Eduardo. Princípios de Análise e Projetos de Sistemas com UML. 1ªed. Rio de Janeiro: Campus, 2007.

CARDOSO, Caíque. UML na Prática: do problema ao sistema. 1ªed. São Paulo: CM, 2003.

DEMARCO, Tom; LISTER Timothy. Peopleware Productive Projects and Teams. US: Dorset House Pub, 1999.

DENNIS, Alan. Análise e Projeto de Sistemas. 2ªed. São Paulo: LTC, 2011.

DRUCKER, Peter. Management: Tasks, Responsibilities, Practices. 1ªed. US: Transaction Publishers, 2007.

FOWLER, MARTIN. UML Essencial. 2ªed. Porto Alegre: Bookman, 2004.

KRUCHTEN, Philippe. Introdução ao RUP Rational Unified Process. 1ªed. Rio de Janeiro: Ciência Moderna, 2003.

MEDEIROS, Ernani. Desenvolvendo Software com UML 2.0: Definitivo. 1ªed. São Paulo: Makron Books, 2004.

MELO, Ana. Desenvolvendo Aplicações com UML 2.2: do conceitual à implementação. 3ªed. RJ: Brasport, 2010.

PENDER, Tom. UML - A Bíblia. 1ªed. RJ: Campus, 2004.

RUMBAUGH, James; *et al.* Modelagem e Projetos Baseados em Objetos com UML. 2ªed. RJ: Campus, 2006.

RUMBAUGH, James; JACOBSON, Ivar; BOOCH, Grady. UML: Guia do Usuário. 1ªed. RJ: Campus, 2006.

SOMMERVILLE, Ian. Engenharia de Software. 8ªed. Pearson, 2007.

5 Introdução à Análise e Projeto Orientado a Objetos

Esse capítulo descreve a construção de um software fazendo uso do paradigma orientado a objetos, processo de desenvolvimento, engenharia de requisitos, modelagem UML, medidas de esforço de desenvolvimento e padrões de projetos. Mas, ressaltando que para construir um *software* que atenda de fato as necessidades de um usuário precisamos entender bem suas necessidades para então desenvolver uma solução adequada, isto é, analisar e projetar um produto de *software*.

Vamos aqui introduzir os conceitos básicos desse paradigma. Lembrando que no dicionário, paradigma significa um sistema de regras e regulamentos, que podem ser escritos ou não, e que servem para estabelecer fronteiras ou limites, determina como se comportar dentro dos limites para alcançar o sucesso. Simplificando, uma forma de pensar e agir, mas um paradigma de desenvolvimento de *software* serve para fornecer uma visão sobre as estruturas da programação, isto é, as formas de se desenvolver um *software*. E para isso existem vários paradigmas, por exemplo, o imperativo, o lógico, o funcional, o orientado a objetos, o orientado a aspectos, entre outros. Porém, nosso foco nesse capítulo é o Paradigma Orientado a Objetos (POO).

Os paradigmas servem para estabelecer um padrão de linguagem para um determinado nicho de problemas, por isso existem diversos paradigmas, além claro, da própria evolução da ciência da computação, que de tempos em tempos, através de pesquisas, disponibilizam novas metodologias e paradigmas de programação. Então se entende que certos grupos de linguagens foram e são desenvolvidas para determinado paradigma. Para o POO temos por exemplo C++, CSharp, Java, Object Pascal, Delphi, Delphi.NET, VB.NET, Python, Ruby, Smalltalk, além do PHP a partir da versão quatro e do *Perl* a partir da versão cinco, entre tantas outras atualmente utilizadas.

A motivação de criação e evolução desse paradigma se deu com a necessidade da área de engenharia de *software* realizar a manutenção corretiva e evolutiva de forma melhor, a que era usada nos paradigmas anteriores. Focado na criação de classes e na instância de objetos dessas classes, a construção do *software* passaram a ser mais criteriosas, pensando na possibilidade de reuso, isto é, reutilizar código pronto e testado. Além, de fazer as correções e implementações, de forma mais organizada e eficiente. Uma grande vantagem desse paradigma, é que ele une as três áreas básicas do desenvolvimento de aplicações - análise, programação e banco de dados. Permitindo que a mesma simbologia possa ser usada nessas três áreas. Nos dias atuais, quase todas as linguagens permitem esse paradigma, umas criadas, outras adaptadas para ele.

5.1 Análise Orientada a Objetos

Na Análise Orientada a Objetos os problemas são pensados e expressados como objetos, ao contrário da análise tradicional os quais eram em rotinas e dados e que foram substituídos por métodos (comportamento) e atributos (propriedades). Assim, quando é colocado o problema de desenvolver um sistema na análise orientada a objetos, deve-se pensar como dividir esse problema em objetos.

Tomando como exemplo um sistema acadêmico para uma análise e modelagem teria Alunos, Administradores, Professores, Cursos, Turmas, etc. E a melhor maneira de conceituar estes termos é considerar um objeto do mundo real e mostrar como podemos representá-lo em termos conceitos em POO. Assim, um objeto é um conceito que usamos para representar uma entidade do mundo real. Exemplificando, um Aluno possui nome, data de nascimento, identidade, etc. E além dessas características (propriedades), possuem ações (métodos) como frequentar as aulas, fazer as avaliações, etc. Em termos de POO para podermos tratar os objetos temos que criar classes. Assim, uma classe representa um conjunto de objetos que possuem comportamentos e características comuns.

Na UML em primeiro lugar denomina-se uma classe e recomenda-se nomeá-la capitalizando-a e deixando no singular. E em segundo lugar denominam-se as propriedades, informações específicas relacionadas a uma classe de objeto, isto é características dos objetos que as classes representam. Em terceira e última parte, métodos, que são ações que os objetos de uma classe podem realizar. Assim tem-se um modelo para instanciar quantos objetos forem necessários. Dessa forma os objetos instanciados possuirão todas as características e comportamentos definidos pela classe. Então as classes especificam a estrutura (propriedades) e os comportamentos (operações) dos objetos, que são instâncias das classes.

Geralmente em um sistema de médio porte serão identificadas diversas classes que compõem o sistema. Neste contexto a UML surgiu como uma proposta de ser uma linguagem para modelagem de dados que usava diversos artefatos para representar o modelo de negócio; um destes artefatos é o diagrama de classes. Os diagramas de classes registram atributos e operações de uma classe e as restrições de como os objetos podem ser conectados, descrevendo também os tipos de objetos no sistema e os relacionamentos entre eles e esse podem ser associações e abstrações. Para poder representar a visibilidade dos atributos e operações, a UML usa os seguintes símbolos: + público, visível em qualquer classe; - privado, visível somente dentro da classe; # protegido, na classe e suas subclasses.

Agora que sabemos o conceito e como codificar uma classe, vamos entender sobre objeto. Segundo os pais da UML (Rumbaugh, Jacobson e Booch) um objeto é uma instância de uma classe, isto é, trata-se de uma cópia da classe na memória em tempo de execução (*runtime*), sendo assim, um elemento específico que possui valores (estados) nos atributos (campos). A UML

representa um objeto usando um retângulo que o representa, onde o nome da instância do objeto é seguido de dois pontos, seguido do nome da classe, com essa formação sublinhada.

O relacionamento entre classes que retrata as relações entre os objetos, por exemplo, um professor que ministra uma disciplina para alunos numa sala, deve ser modelada em um dos tipos de relações da UML, as quais são: dependência, associação e generalização ou herança. Geralmente as classes não estão isoladas (coesas) e se relacionam entre si (acoplamento). O relacionamento e a comunicação entre as classes se definem em três tipos: Associações que podem ser de Agregação ou Composição; Generalização ou herança; e Dependências. As associações são relacionamentos estruturais entre instâncias e especificam que objetos de uma classe estão ligados a objetos de outras classes, podendo ser unária, binária, etc. As associações podem existir entre classes ou entre objetos. Uma associação entre a classe Professor e a classe Disciplina (um professor ministra uma disciplina) significa que uma instância de Professor (um professor específico) vai ter uma associação com uma instância de uma Disciplina. Esta relação significa que as instâncias das classes são conectadas, seja fisicamente ou conceitualmente.

Outro aspecto importante da POO é o polimorfismo que é a capacidade de classes mais abstratas possuírem comportamentos diferentes das classes concretas, isto é, duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação (assinatura), mas comportamentos distintos, especializados para uma das classes derivadas. A decisão sobre qual o método que deve ser selecionado, de acordo com o tipo da classe derivada, é tomada em tempo de execução, através do mecanismo de ligação tardia. Mas para ser considerado polimorfismo, é necessário que os métodos tenham exatamente a mesma identificação, quer dizer, o mesmo nome de método, sendo utilizado o mecanismo de redefinição de métodos, isto é, sobrescrita dos métodos. A redefinição ocorre quando um método cuja assinatura já tenha sido especificada recebe uma nova definição, ou seja, uma nova implementação em uma classe derivada. O mecanismo de redefinição, juntamente com o conceito de ligação tardia, é a chave para a utilização do polimorfismo. É importante observar que, quando polimorfismo está sendo utilizado, o comportamento que será adotado por um método só será definido durante a execução. A outra forma é a sobrecarga, isto é, um método aplicado a um objeto é selecionado para execução através da sua assinatura e da verificação a qual classe o objeto pertence. Dois ou mais métodos de uma mesma classe podem ter o mesmo nome, desde que suas listas de parâmetros sejam diferentes, constituindo assim uma assinatura diferente. Tal situação não gera conflito, pois o compilador é capaz de detectar qual método deve ser escolhido a partir da análise dos tipos de argumentos do método.

Por fim e não menos importante temos o encapsulamento, que consiste na separação de aspectos internos e externos de um objeto. Este mecanismo é

utilizado amplamente para impedir o acesso direto ao estado de um objeto (seus atributos), disponibilizando externamente apenas os métodos que alteram estes estados.

5.2 Modelagem Orientada a Objetos

O Paradigma Orientado a Objetos tem como processo mais completo para representá-lo o RUP e esse processo possui 4 marcos, os quais são: Marco dos objetivos do ciclo de vida; Marco da arquitetura do ciclo de vida; Marco da capacidade operacional inicial; e Marco do lançamento do produto.

Para começar a modelar os requisitos especificados. O cliente necessita entender o que os projetistas estão construindo e precisam ter condições de inferir os seus conhecimentos nos projetos para atender plenamente suas necessidades. Para isso é necessário estabelecer um canal formal de interação onde a linguagem natural do cliente é transformada em linguagem técnica para a equipe de desenvolvimento. Com esse objetivo a UML se propõe a ser uma linguagem padrão aceita e compreendida por todos os *stakeholders*.

A modelagem utilizando a Linguagem de Modelagem Unificada, mais conhecida através da sigla em inglês UML que significa *Unified Modeling Language* é focada no paradigma Orientado a Objetos (OO) cujos conceitos são classe, objeto, herança, polimorfismo, encapsulamento de atributos e métodos, alta coesão e baixo acoplamento. É usado para a análise não focando na codificação do *software* ou *hardware* e sim no entendimento do problema (análise) e na sua solução (projeto).

A UML representa símbolos, esses usados em diagramas que assim representam uma linguagem simbólica com regras claras e precisas para utilização desses símbolos nos diversos diagramas. O objetivo dos diagramas é apresentar múltiplas visões do sistema chamado de modelo. Assim, um modelo UML é um conjunto de diagramas que servem para compreender e desenvolver um projeto de *software*, descrevendo o que o *software* deve fazer.

A *Unified Modeling Language* (UML) é a sucessora de uma onda de métodos para análise e projeto orientados a objetos que surgiram no fim da década de 80. Ela unificou os métodos proposto por Booch, Rumbaugh e Jacobson. Atualmente a UML é um padrão mantido pela OMG, o que dá a ela um status de linguagem padrão para modelagem de sistemas, isto é, uma linguagem padrão para visualizar, especificar, construir e documentar artefatos de um sistema de *software* e combina diversos aspectos como: Modelagem de Dados; Modelagem de Negócios; Modelagem de Objetos; e Modelagem de Componentes

É importante enfatizar que UML é uma linguagem de modelagem, ou seja, é uma forma de especificar um sistema de maneira visual. Ela é a linguagem padrão de modelagem adotada pelo *Rational Unified Process* (RUP) e seu

grande sucesso se deu em conjunto com a popularização do RUP. Por isso, é muito comum que as pessoas associem UML ao RUP. É importante distinguir muito bem, visto que as duas representam conceitos complementarmente diferentes. O RUP é um processo, ou seja, um conjunto de atividades, com seus respectivos artefatos e papéis, para se desenvolver produtos de *software* e a UML é uma linguagem para se modelar o comportamento e estrutura de um sistema.

Na UML o diagrama de caso de uso segundo Ivan Jacobson, um caso de uso é "um documento narrativo que descreve a sequência de eventos de um ator que usa um sistema para completar um processo". Um caso de uso é uma técnica de modelagem que descreve o que o novo sistema deve fazer e se baseia em um processo de licitação de requisitos entre os *stakeholders* na condução de uma especificação na qual todos interagem.

Um caso de uso descreve quais comportamentos o sistema deverá responder para cada um dos usuários do mesmo, servindo de formalização das ações que precisam ser desenvolvidas. Retratando uma lista de eventos entre os usuários e o sistema em uma visão abstrata, onde essa lista de eventos relatados abstratamente descreve as interações desde o início da atividade até o fim da mesma.

Casos de uso têm que ser compreensíveis pelos usuários por que só eles sabem o que o sistema precisa fazer. Os casos de uso permitem verificar se o desenvolvedor e o usuário concordam sobre o que o sistema deve fazer. Isso é um problema importante no desenvolvimento de *software*. No mesmo tempo, casos de uso podem servir de "contratos" entre os usuários e os desenvolvedores. Sendo assim os casos de uso devem: Descrever os requisitos funcionais do sistema; Fornecer uma descrição clara do que se deve fazer; e Definir os comportamentos das classes.

É importante salientar que casos de uso não são requisitos. E sim uma visão abstrata desses os quais são compostos pelos seguintes componentes: Ator, que é um papel que solicita eventos ao sistema e recebe feedbacks, onde cada ator pode participar de vários casos de uso; e Casos de uso são documentos narrativos que descrevem a sequência dos eventos feitos por um ator no uso do sistema. Os elementos que descrevem esses papéis são exibidos abaixo. Vale salientar que o nome dos casos de uso deve ser composto por frases curtas, preferencialmente verbo + substantivo. Exemplo: matricular aluno.

Para especificar um caso de uso pode-se usar a estrutura que se segue, porém não existe uma estrutura rígida: Código do caso de uso; Nome do caso de uso; Descrição do caso de uso (um parágrafo); Lista dos nomes dos atores com descrição curta; Prioridade, caso de uso muito importante no projeto ou acessório; Pré-condições; Pós-condições; Fluxo de eventos principal; Fluxos de eventos alternativos; Fluxo de eventos de exceções; Cenários; e Dicionário de

Dados. Segue o *template* para especificação de Casos de Uso, [clique aqui](http://www.wthreex.com/rup/portugues/webtmpl/templates/reg/rup_ucspec.htm) que está disponível no *link* que se segue < http://www.wthreex.com/rup/portugues/webtmpl/templates/reg/rup_ucspec.htm > e para se obter o *template* mais detalhado da especificação de casos de uso, [clique aqui](http://goo.gl/5yKpBD) ou acesso o *link* disponível no site wthreex para baixá-lo < <http://goo.gl/5yKpBD> >

Para identificar os atores que vão participar do modelo uma das técnicas é fazer as seguintes perguntas: Quem usa o sistema? Quem inicia o sistema? Quem fornece os dados? Quem usa as informações? Na descrição dos atores geralmente se faz o uso de: Nome do caso de uso; Tipo de uso se é frequente, ocasional; e Descrição de seu papel no sistema. Para identificar os casos de uso, normalmente a técnica usada é: Verificar as interações entre os atores e o sistema; e Verificar as ações do ator e do sistema.

É importante salientar que os atores sempre iniciam a ação. Para entendimento imagine o requisito em que um aluno necessita fazer um curso, onde existe a necessidade de um sistema de matrículas. Então: Quais são os atores? Quem usa o sistema? Como podemos identificar o caso de uso? Podemos chamar este caso de uso de: Matricular Aluno (verbo+substantivo). Para fazer uma descrição textual do caso de uso, que possui dois atores Aluno e Administração. Sua especificação poderia ser: Caso de uso: Matricular Aluno. Atores: Aluno, Administrador. Descrição: Este caso de uso começa quando um aluno chega à administração que faz a matrícula do aluno no curso. Fluxo: P1-O sistema registra a matrícula; P2-Recebe o pagamento; P3-Emite uma nota fiscal e P4-O aluno é matriculado.

Importante: o caso de uso deve começar com um verbo, enfatizando o processo. Exemplo: Matricular Aluno ou Cadastrar Curso. Deve-se identificar o ator que inicia o evento, começando a descrição da sequência de um caso de uso usando o esquema - este caso de uso começa quando um aluno faz uma matrícula em um curso. Verifique que um ator é um elemento externo ao sistema. Um ator é um papel que interage com o sistema, mas não faz parte dele.

O primeiro passo é identificar os casos de uso básicos, o próximo passo é identificar o relacionamento entre os casos de uso através das generalizações, inclusões e extensões. As inclusões são um caso de uso que inclui o comportamento de outro. A finalidade básica da inclusão é realizar uma decomposição funcional e reduzir a complexidade de um caso de uso. As extensões definem pontos que adicionam um comportamento opcional, assim o caso de uso base pode ser executado mesmo que a extensão não o seja. De princípio as generalizações são mais indicadas para as características dos atores, porém podem ser usadas com cautela para indicar especializações de comportamentos.

No diagrama de classes na Programação Orientada a Objetos (POO), os problemas são pensados e expressados como objetos, ao contrário da análise tradicional os quais eram em rotinas e dados, que aqui foram substituídos por métodos (comportamento) e atributos (propriedades). Assim, quando é colocado o problema de desenvolver um sistema acadêmico na análise orientada a objetos, deve-se pensar como dividir esse problema em objetos. Assim teríamos Alunos, Administradores, Professores, Cursos, Turmas, etc. E a melhor maneira de conceituar estes termos é considerar um objeto do mundo real e mostrar como podemos representá-lo em termos conceitos em POO.

Assim, um objeto é um conceito que usamos para representar uma entidade do mundo real. Exemplificando, um Aluno possui nome, data de nascimento, identidade, etc. E além dessas características (propriedades), possuem ações (métodos) como frequentar as aulas, fazer as avaliações, etc. Em termos de POO para podermos tratar os objetos temos que criar classes. Assim, uma classe representa um conjunto de objetos que possuem comportamentos e características comuns.

Na UML em primeiro lugar denomina-se uma classe e recomenda-se nomeá-la capitalizando-a e deixando no singular. E em segundo lugar denominam-se as propriedades, informações específicas relacionadas a uma classe de objeto, isto é características dos objetos que as classes representam. Em terceiro e última parte, métodos, que são ações que os objetos de uma classe podem realizar. Assim tem-se um modelo para instanciar quantos objetos forem necessários. Dessa forma os objetos instanciados possuirão todas as características e comportamentos definidos pela classe. Então as classes especificam a estrutura (propriedades) e os comportamentos (operações) dos objetos, que são instâncias das classes.

Geralmente em um sistema de médio porte serão identificadas diversas classes que compõem o sistema. Neste contexto a UML surgiu como uma proposta de ser uma linguagem para modelagem de dados que usava diversos artefatos para representar o modelo de negócio; um destes artefatos é o diagrama de classes.

Os diagramas de classes registram atributos e operações de uma classe e as restrições de como os objetos podem ser conectados, descrevendo também os tipos de objetos no sistema e os relacionamentos entre eles e esse podem ser associações e abstrações. Para poder representar a visibilidade dos atributos e operações, a UML usa os seguintes símbolos: + público, visível em qualquer classe; - privado, visível somente dentro da classe; # protegido, na classe e suas subclasses.

O relacionamento entre classes retrata as relações entre os objetos. Exemplo: um professor ministra uma disciplina para alunos numa sala. A UML reconhece três tipos mais importantes de relações: dependência, associação e

generalização ou herança. Geralmente as classes não estão isoladas (coesas) e se relacionam entre si (acoplamento). O relacionamento e a comunicação entre as classes se definem em três tipos: Associações que podem ser de Agregação ou Composição; Generalização ou herança; e Dependências.

As associações são relacionamentos estruturais entre instâncias e especificam que objetos de uma classe estão ligados a objetos de outras classes, podendo ser unária, binária, etc. As associações podem existir entre classes ou entre objetos. Uma associação entre a classe Professor e a classe Disciplina (um professor ministra uma disciplina) significa que uma instância de Professor (um professor específico) vai ter uma associação com uma instância de uma Disciplina. Esta relação significa que as instâncias das classes são conectadas, seja fisicamente ou conceitualmente. Dependências são relacionamentos de utilização no qual uma mudança na especificação de um elemento pode alterar a especificação do elemento dependente. A dependência entre classes indica que os objetos de uma classe usam serviços dos objetos de outra classe. Generalização ou herança, que pode ser simples ou múltipla (composta), serve para relacionar um elemento mais geral e um mais específico, onde o elemento mais específico herda as propriedades e métodos do elemento mais geral. Como a relação de dependência, ela existe só entre as classes. Um objeto particular não é um caso geral de outro objeto, apenas classes podem receber esse conceito. Agregação é um tipo de associação (é parte de ou todo-parte) onde o objeto parte é um atributo do todo, onde o objeto parte somente são criados se o todo ao qual estão agregados seja criado. Pedidos é composto por itens de pedidos. Composição é o relacionamento entre um elemento (o todo) e outros elementos (as partes) onde as partes só podem pertencer ao todo e são criadas e destruídas com ele.

Resumindo, temos as seguintes formas de relacionamentos: Generalização: É um relacionamento de especialização/generalização, nos quais os objetos dos elementos especializados (os filhos) são substituíveis por objetos do elemento generalizado (os pais); Associação: É um relacionamento estrutural que descreve um conjunto de ligações, em que as ligações são conexões entre objetos.

Agregação: É um tipo especial de associação representando um relacionamento estrutural entre o todo e sua parte; Composição: É outro tipo especial de associação representando um relacionamento estrutural entre um elemento (o todo) e outros elementos (as partes) onde as parte só podem pertencer ao todo e são criadas e destruídas com ele; Realização: É um relacionamento semântico entre classes, em que uma classe especifica um contrato que outra classe garante executar; e Dependência: Relacionamento semântico entre duas classes, nos quais a alteração de um (a classe independente) pode afetar a semântica da outra (a classe dependente).

O diagrama de classes lista todos os conceitos do domínio que serão implementados no sistema e as relações entre os conceitos. Ele é muito importante, pois define a estrutura do sistema a desenvolver. O diagrama de classes é consequência do prévio levantamento de requisitos, definição de casos de usos e classes. Como exemplo tem os passos de elicitação de requisitos com os *stakeholders* do sistema a ser desenvolvido, usando a técnica de entrevista com os administradores, professores, etc. que a partir desses os objetos do sistema são definidos: Alunos, Professores, Turmas, Cursos, etc. Definição dos atores do sistema: aluno, professor, administrador, etc. Definição e detalhamento dos casos de uso: Matricular Aluno, Pagar Matrícula, etc. Definição das classes: alunos, professor, etc.

Atributo representa uma propriedade que todos os objetos da classe têm, porém cada objeto terá valores particulares para seus atributos. A UML, o nome de um atributo é um texto que deve capitalizar todas as primeiras letras de cada palavra no nome menos a primeira palavra.

Todos os métodos têm que respeitar exatamente a assinatura que é composta pelo nome, número de parâmetros, tipos de dados e ordem. Um método não pode acrescentar ou cortar um parâmetro. Para mandar a mensagem corretamente, devesse saber qual é a classe do objeto, já que cada classe tendo método com assinatura diferente.

Mais uma informação. Devido aos questionamentos de alguns alunos sobre a classificação em relação ao tipo de generalização, complementei essa aula especificando essa tipificação. Ela pode ser basicamente de dois tipos: Total ou Parcial. Se Total temos para cada ocorrência da uma classe genérica sempre uma ocorrência em uma das classes especializadas. Se Parcial temos que nem sempre uma ocorrência da classe genérica possui uma ocorrência correspondente em uma classe especializada. Essa tipificação é muito interessante para a camada de persistência e pode ser também utilizada na estereotipação de entidades.

Os demais diagramas são tão importantes quanto os anteriores, porém menos usados, pois sua aplicação se dá para esclarecer outros aspectos da modelagem e só quando se fazemos necessidade real são desenvolvidos. Esses são os diagramas de sequência que serve para exibir as interações entre os vários componentes de um sistema em especial os objetos e como seus métodos interagem entre si e em qual ordem (sequência). O diagrama de comunicação que serve para representar os elementos de um sistema que trabalham em conjunto. Esse diagrama expressa essa comunicação, mostrando a troca de mensagens entre os objetos. Ele é similar ao diagrama de sequência, porém mais simples. Os diagramas de sequência e de comunicação mostram as interações entre os objetos, por este motivo a UML se refere a estes diagramas como diagramas de interação. O diagrama de atividade que exibe a forma que um objeto executa suas ações em um único processo, representando-os passo

a passo, isto é, seu fluxo. As atividades que ocorrem em um caso de uso ou no comportamento do objeto seguem a sequência conforme descrito do diagrama. O diagrama de estados que tem a finalidade de exibir como um objeto realiza uma determinada operação num determinado momento da execução, representando um estado particular. Exemplo: um aluno pode estar com a situação de aprovado, aprovado na final ou reprovado. O diagrama de componentes que mostra a organização dos componentes na implantação do sistema, mostrando os elementos (componentes) reutilizáveis de *software* e suas interdependências. Vale salientar que um componente é composto por um conjunto de classes. Na maioria das vezes um componente depende de outros, da mesma forma que as classes que ele possui, dependem das funcionalidades de outras classes do mesmo ou de outro componente. Desta forma o diagrama de componentes mostra esta dependência. E o diagrama de implantação que nos mostra a configuração física sobre qual o sistema será instalado. Esses diagramas servem para exibir a distribuição de *hardware* do projeto de *software*, identificando os recursos computacionais como pontos (nós) além da rede que os relaciona. Dessa forma, esse diagrama permite exibir a topologia de uma rede usada, como também os recursos (máquinas).

5.3 Projeto de Software

Vamos iniciar começando com o entendimento do que é um projeto. A definição mais simples e conhecida de projeto é "um conjunto de atividades que representam algo único e que possui início e fim". Mas existem outros conceitos e níveis de administração de atividades, por exemplo, um conjunto de projetos interligados compõe um programa, um conjunto de programas interligados compõe um portfólio. Para exemplificar, temos um portfólio de governo que engloba vários programas, como: Fome Zero, Escola para Todos, Minha Casa Minha Vida, etc. Cada programa, a exemplo Fome Zero, necessita de vários projetos para ser realizado, como: Identificação dos Pontos Necessitados, Captação de Recursos, Logística, etc. Cada projeto para ser concluído terá que realizar várias atividades únicas com prazo, recursos e custos planejados. Essas tarefas podem ser agrupadas em fases, subfases, etc.

Resumindo o parágrafo anterior temos de forma simplificada a seguinte classificação: Portfólio = conjunto de programas; Programa = conjunto de projetos; Projeto = conjunto de fases; Fase = conjunto de atividades; e Atividade = conjunto de subatividades ou tarefa única a ser executada.

Com o passar dos anos várias instituições foram surgindo para criar e padronizar as teorias de administração de projetos, e algumas se consolidaram. Podemos citar, o ISO, ITIL, COBIT, PRINCE2, SWEBOK, entre outras, mas a que vem se destacando em várias áreas é o PMI. Porém, independente o processo que define como fazer, quando fazer, quem fazer e onde fazer a gestão de atividades, todas elas servem para definir um corpo de conhecimentos

sobre gestão, o que é fundamental, para se ter um local de acesso para se obter um compêndio de boas práticas de gerenciamento.

Aqui faremos o uso do corpo de conhecimento de gerenciamento de projetos, mais conhecido na sua sigla em inglês como PMBOK que significa *Project Management Body of Knowledge*. Abordando as suas 9 áreas que são integração, escopo, tempo, custos, qualidade, recursos humanos, comunicações, riscos e aquisições, e como utilizar o MS-Project da Microsoft como ferramenta para aplicar os conhecimentos administrado pelo instituto de gestão de projeto, também mais conhecido como *Project Management Institute (PMI)*.

Um projeto pode ser definido como uma sequência de atividades ou eventos com início e fim definidos, dirigidos por pessoas que se destinam a alcançar um dado objetivo dentro de parâmetros de custo, tempo, recursos e qualidade. MS-Project é um dos mais modernos aplicativos voltados para o gerenciamento de projetos, no qual é possível planejar, especificar, implantar e acompanhar o desenvolvimento de qualquer tipo de projeto. As informações podem estar representadas graficamente ou através de relatórios customizados.

A forma mais simples de definir um projeto, e usando a definição do PMI, é a administração de atividades fazendo uso de conhecimentos, habilidades e técnicas no seu planejamento, relacionando-as para atingir um conjunto de objetivos pré-definidos, num certo prazo, com certo custo e qualidade, através da mobilização de recursos técnicos e humanos.

O *Project Management Institute (PMI)* é uma organização sem fins lucrativos que se destina a criar e manter boas práticas na gestão de portfólios, programas e projetos. Ela foi criada por 5 gerentes de projetos em 1969 na cidade da Filadélfia no estado da Pensilvânia.

Nos dias atuais, o PMI já possui mais de 200.000 associados e é considerado uma das principais organizações de gestão de projetos, com representações em 185 países. Apesar de inicialmente o PMI ser orientado a projetos de *software*, hoje suas práticas são utilizadas nas mais diversas áreas como: aeroespacial, automotiva, negócios, construção, engenharia, serviços financeiros, farmacêutico, saúde, etc.

A seguir são exibidas as diversas certificações oferecidas pelo instituto: *Certified Associate in Project Management (CAPM)*, que serve para demonstrar uma base comum do conhecimento e dos termos no campo da gerência de projeto. Ele requer 1500 horas do trabalho em uma equipe de projeto ou 23 horas/aula em gerência de projeto; *Project Management Professional (PMP)* contém curso de especialização e experiência, concordando em aderir a um código da conduta profissional, e aprovação para avaliar e medir objetivamente o conhecimento da gerência de projeto. Este certificado deve ser periodicamente renovado, caso

contrário implica na perda da certificação; e *Program Management Professional* (PgMP), equivalente ao anterior, porém focando na certificação em programa.

O principal produto gerado pelo PMI é o guia chamado de *Project Management Body of Knowledge* (PMBOK) que é um compêndio de boas práticas gerenciais para as 9 áreas que a organização orienta na administração. O Guia PMBOK serve para oferecer um vocabulário comum na gestão de portfólios, programas e projetos. Permitindo assim a discussão no mesmo nível entre os participantes, disseminando o conhecimento adquirido por todos. Possibilitando assim o intercâmbio eficiente de informações entre os profissionais de gerência de projetos.

O guia é baseado em processos e subprocessos para descrever de forma organizada as atividades de gestão de projetos que se relacionam e interagem. E isso é feito através de artefatos de entradas que podem ser documentos, planos, desenhos, etc.; de ferramentas e técnicas, por exemplo, o MS-Project; e de saídas, como documentos, produtos, etc. A versão 2004 do guia, cita 44 processos agrupados em cinco grupos e nove áreas de conhecimento, que definem o ciclo de vida e da organização de um projeto.

As áreas de conhecimentos de acordo com o PMBOK são:

- Gerenciamento de integração do projeto;
- Gerenciamento do escopo do projeto;
- Gerenciamento de tempo do projeto;
- Gerenciamento de custos do projeto;
- Gerenciamento da qualidade do projeto;
- Gerenciamento de recursos humanos;
- Gerenciamento das comunicações;
- Gerenciamento de riscos do projeto; e
- Gerenciamento de aquisições do projeto.

Os grupos de processos de gerenciamento de projetos são:

- Iniciação;
- Planejamento;
- Execução;
- Monitoramento e controle; e
- Encerramento.

Gestão de projetos ou gerenciamento de projetos ou ainda administração de projetos é a aplicação de conhecimentos, habilidades e técnicas na elaboração de atividades relacionadas no cumprimento de objetivos pré-definidos, no prazo, no custo e na qualidade planejados, fazendo uso de recursos humanos e

materiais. O planejamento do projeto na gestão de projetos é o processo que serve para quantificar o tempo e custo de um projeto, com o intuito de criar um plano de projeto para que o gestor possa utilizar como guia no acompanhamento do progresso dos recursos humanos.

Passos de um planejamento:

- Identificar o principal objetivo do projeto;
- Determinar as condições de finalização do projeto;
- Fazer um inventário das atividades a serem realizadas no projeto;
- Identificar os recursos necessários para execução do projeto;
- Definir as dependências entre tarefas;
- Determinar quais tarefas será impossível de estimar o prazo com precisão;
- Colocar as tarefas sem prazo fora do caminho crítico do projeto;
- Criar um cronograma do projeto; e
- Obter comprometimento da organização na execução do projeto.

Ser um gerente de projeto é ser um profissional que raramente participa das atividades diretas do projeto, isto é, "botar a mão na massa", pois sua função é gerenciar o progresso através das variáveis prazos, custo e qualidade, chamada de tríade da gestão, para verificar os desvios nessas. Assim, podemos dizer que o principal objetivo é evitar e tratar das falhas que venham acontecer e assim minimizá-las ou eliminá-las. Um gerente de projetos é um maestro que deve entender as necessidades do cliente e planejar um projeto para atendê-lo da melhor forma possível, do começo ao fim, assegurando que tudo que foi planejado será cumprido. Esta tarefa pode ser dividida entre a gerencia de supervisão - o gerente de projetos e a gerencia de execução - o gerente operacional. O gerente de projetos precisa monitorar e comunicar o desempenho do projeto. Os resultados do trabalho que estiverem abaixo de um nível de desempenho aceitável precisam ser ajustados com ações corretivas para que o projeto volte a estar em conformidade com as linhas de base de custo, prazo e escopo. A comunicação do desempenho do projeto é um dos principais elementos para o gerenciamento de projetos bem sucedido.

5.4 Artefatos de um Projeto de Software (Templates)

Avaliação da Organização de Desenvolvimento. Disponível em < <http://goo.gl/n4Agyq> > Acesso: Jan/15.

Avaliação da Organização-alvo. Disponível em < <http://goo.gl/n5bKph> >

Avaliação de Iteração. Disponível em < <http://goo.gl/YIRRBv> >

Avaliação de Status. Disponível em < <http://goo.gl/546iYI> >

Caso de Desenvolvimento. Disponível em < <http://goo.gl/xT7kXd> >

Caso de Negócios. Disponível em < <http://goo.gl/9euZmQ> >

Documento de Arquitetura de Negócios. Disponível em < <http://goo.gl/9AuSQa> >

Documento de Arquitetura de Software. Disponível em < <http://goo.gl/BsuHyq> >

Especificação de Caso de Uso de Negócios Nome do Caso de Uso de Negócios. Disp.:< <http://goo.gl/XklLXt> >

Especificação de Caso de Uso Nome do Caso de Uso. Disponível em < <http://goo.gl/8OsCF8> >

Especificação de Realização de Casos de Uso Nome do Caso de Uso de Negócios. Disp.:< <http://goo.gl/PoKB2f> >

Especificação de Realização de Casos de Uso Nome do Caso de Uso. Disponível em < <http://goo.gl/NHNreG> >

Especificação dos Requisitos de Software Para Subsistema ou Recurso. Disponível em < <http://goo.gl/mnOYe5> >

Especificação dos Requisitos de Software. Disponível em < <http://goo.gl/dfnJMy> >

Especificação Suplementar de Negócios. Disponível em < <http://goo.gl/G52y77> >

Especificação Suplementar. Disponível em < <http://goo.gl/aGdgmj> >

Glossário de Negócio. Disponível em < <http://goo.gl/SRqCx8> >

Glossário de Negócios. Disponível em < <http://goo.gl/yjDhRQ> >

Glossário. Disponível em < <http://goo.gl/UVhQ2H> >

Lista de Materiais. Disponível em < <http://goo.gl/A0sTGO> >

Lista de Riscos. Disponível em < <http://goo.gl/Vyy1js> >

Modelo de Negócio. Disponível em < <http://goo.gl/O3O6Ya> >

Notas de Release. Disponível em < <http://goo.gl/e0FC0h> >

Plano de Aceitação do Produto. Disponível em < <http://goo.gl/RH4JDj> >

Plano de Desenvolvimento de Software (Projeto Pequeno). Disponível em < <http://goo.gl/J9qsvB> >

Plano de Desenvolvimento de Software. Disponível em < <http://goo.gl/5KRFCJ> >

Plano de Garantia de Qualidade. Disponível em < <http://goo.gl/D1M0Gp> >

Plano de Gerenciamento de Configuração. Disponível em < <http://goo.gl/almYbn> >

Plano de Gerenciamento de Requisitos. Disponível em < <http://goo.gl/1iwQYG> >

Plano de Gerenciamento de Riscos. Disponível em < <http://goo.gl/kvwGyW> >

Plano de Implantação. Disponível em < <http://goo.gl/zK4sDV> >

Plano de Integração do Build. Disponível em < <http://goo.gl/J5eP3T> >

Plano de Iteração. Disponível em < <http://goo.gl/1f4yLQ> >

Plano de Métricas. Disponível em < <http://goo.gl/64uo2r> >

Plano de Negócio. Disponível em < <http://goo.gl/Jfw2VK> >

Plano de Requisitos. Disponível em < <http://goo.gl/sLM5Lu> >

Plano de Resolução de Problemas. Disponível em < <http://goo.gl/pwvd6l> >

Plano de Teste. Disponível em < <http://goo.gl/XM0cNO> >

Proposta Comercial. Disponível em < <http://goo.gl/DI17e4> >

Regras de Negócios. Disponível em < <http://goo.gl/Zx5CWY> >

Requisitos de Software. Disponível em < <http://goo.gl/Bi5piJ> >

Solicitações dos Principais Envolvidos. Disponível em < <http://goo.gl/XGUuGR> >

Sumário de Avaliação de Testes. Disponível em < <http://goo.gl/ZVLVb6> >

Termo de Abertura do Projeto. Disponível em < <http://goo.gl/CYnUmT> >

Visão. Disponível em < <http://goo.gl/liXVDn> >

Visão (Projeto Pequeno). Disponível em < <http://goo.gl/x8YMS7> >

Visão do Negócio. Disponível em < <http://goo.gl/wUKpZc> >

Visão. Disponível em < <http://goo.gl/fBmJq8> >

5.5 Ferramentas

Astah Change-Vision. Disponível em < <http://astah.change-vision.com/en/index.html> >

Astah Community. Disponível em < <http://astah.change-vision.com/en/product/astah-community.html> >

Astah Profissional. Disp.:< <http://astah.change-vision.com/en/product/astah-professional.html> >

Astah Share. Disponível em < <http://astah.change-vision.com/en/product/astah-share.html> >

Astah UML. Disponível em < <http://astah.change-vision.com/en/product/astah-uml.html> >

CASE tools. Disponível em < <http://case-tools.org/uml.html> >

Manual do Astah. Disponível em < <http://goo.gl/z9GrFB> >

Plugin de modelagem UML no NetBeans 6.9. Disponível em < <http://goo.gl/tYFMDL> >

Plugin diversos para NetBeans 6.9. Disp.:<

<http://dlc.sun.com.edgesuite.net/netbeans/6.9/rc/zip.html> >

5.6 Apresentações (pptx)

UML. Disponível em < <https://pt.slideshare.net/alvarofpinheiro/alvarofpinheouruml> >

5.7 Vídeos

Introdução a Compiladores. Disponível em < <http://vimeo.com/112860457> >

5.8 Sugestões de Leitura

Análise e Projeto de Sistemas. Disponível em < <http://goo.gl/kQVR7L> >

Desenvolvendo Aplicações com UML 2.2. Disponível em < <http://goo.gl/vVW6lL> >

Engenharia de Software. Disponível em < <http://goo.gl/qZVIYT> >

Exercitando Modelagem com UML. Disponível em < <http://goo.gl/r7toX5> >

Fundamentos de Programação. Disponível em < <http://goo.gl/xwYOfc> >

Introdução a Informática. Disponível em < <http://goo.gl/LPT4Sc> >

UML Essencial. Disponível em < <http://goo.gl/cdLkbT> >

UML Guia do Usuário. Disponível em < <http://goo.gl/av5BTh> >

Utilizando UML e padrões. Disponível em < <http://goo.gl/1YPJ0l> >

5.9 Referências Web

A Theory of Object-Oriented Design. Disp.:< [http://www.eden-study.org/articles/2002/isf4\(4\).pdf](http://www.eden-study.org/articles/2002/isf4(4).pdf) >
ANSI site oficial. Disponível em < <http://www.ansi.org/> >
ANSI wiki. Disponível em < http://pt.wikipedia.org/wiki/American_National_Standards_Institute >
Aulas de UML. Disponível em < <http://goo.gl/2TRbUF> >
Design Patterns Languages for Object-Oriented Programs (DP). Disp.:< <http://www.vincehuston.org/dp/> >
Exemplo de Modelagem UML com a ferramenta ASTAH. Disponível em < <http://goo.gl/JvghSt> >
Exemplo de Modelagem UML. Disponível em < <http://goo.gl/zwOYCF> >
Guia Básico sobre UML. Disponível em < <http://goo.gl/eGhjzM> >
Guia da UML. Disponível em < <http://goo.gl/3Wtx0b> >
Guia de Modelagem de Casos de Uso. Disponível em < <http://goo.gl/PtwH3L> >
Guia de Modelagem de Negócios. Disponível em < <http://goo.gl/W2U0B6> >
Guia de Referência de Convenções de Codificação Java. Disponível em < <http://goo.gl/PfvJcb> >
Guia de Design. Disponível em < <http://goo.gl/OpvFEC> >
Guia de Programação. Disponível em < <http://goo.gl/NL4sX9> >
Guia de Teste. Disponível em < <http://goo.gl/hKmKE1> >
IBM Rational Unified Process. Disponível em < ftp://public.dhe.ibm.com/software/pdf/br/RUP_DS.pdf >
ISO site oficial. Disponível em < <http://www.iso.org/iso/home.html> >
ISO wiki. Disponível em < http://en.wikipedia.org/wiki/International_Organization_for_Standardization >
Langpop. Disponível em < <http://www.langpop.com/> >
Mais sobre relacionamentos de Inclusão e Extensão. Disponível em < <http://goo.gl/uH8X6Z> >
Manuais de Referência OMG-UML. Disponível em < <http://www.omg.org/spec/UML/> >
Modelagem UML. Disponível em < <http://goo.gl/iwtZh3> >
Object Management Group (OMG). Disponível em < <http://www.omg.org/> >
Object-Oriented Programming (OOP). Disponível em < <http://goo.gl/rIYFI5> >
Ohloh. Disponível em < <http://www.ohloh.net/> >
Paradigma (wiki). Disponível em < <http://pt.wikipedia.org/wiki/Paradigma> >
Paradigma de Desenvolvimento (wiki). Disponível em < <http://goo.gl/rIYFI5> >
Rational Unified Process (RUP). Disponível em < <http://www-01.ibm.com/software/br/rational/> >
Standish Group Report Chaos 2009. Disp.:< http://www1.standishgroup.com/newsroom/chaos_2009.php >
Standish Group. Disponível em < <http://www.standishgroup.com/> >
Template Mais Detalhado para Especificação de Casos de Uso. Disp.:< <http://goo.gl/5yKpBD> >
Template RUP para Especificação de Casos de Uso. Disponível em < <http://goo.gl/GwnXCn> >

Tiobe. Disponível em < <http://www.tiobe.com/index.php/content/company/Home.html> >
Um Mundo Feito (quase completamente) de Software. Disponível em < <http://goo.gl/YE26CS> >
Unified Modeling Language (UML). Disponível em < <http://www.uml.org/> >

5.10 Referências Bibliográficas

BENNETT, Simon; *et al.* Schaum's Outline of UML. 1ªed. UK: McGraw-Hill, 2005.
BEZERRA, Eduardo. Princípios de Análise e Projetos de Sistemas com UML. 1ªed. Rio de Janeiro: Campus, 2007.
CARDOSO, Caíque. UML na Prática: do problema ao sistema. 1ªed. São Paulo: CM, 2003.
DEMARCO, Tom; LISTER Timothy. Peopleware Productive Projects and Teams. US: Dorset House Pub, 1999.
DENNIS, Alan. Análise e Projeto de Sistemas. 2ªed. São Paulo: LTC, 2011.
DRUCKER, Peter. Management: Tasks, Responsibilities, Practices. 1ªed. US: Transaction Publishers, 2007.
FOWLER, MARTIN. UML Essencial. 2ªed. Porto Alegre: Bookman, 2004.
GAMMA, Erich; *et al.* Padrões de Projeto. Porto Alegre: Bookman, 2007.
KRUCHTEN, Philippe. Introdução ao RUP Rational Unified Process. 1ªed. Rio de Janeiro: Ciência Moderna, 2003.
MEDEIROS, Ernani. Desenvolvendo Software com UML 2.0: Definitivo. 1ªed. São Paulo: Makron Books, 2004.
MELO, Ana. Desenvolvendo Aplicações com UML 2.2: do conceitual à implementação. 3ªed. RJ: Brasport, 2010.
PENDER, Tom. UML - A Bíblia. 1ªed. RJ: Campus, 2004.
PRESSMAN, Roger. Engenharia de Software: uma abordagem profissional. 7ªed. Porto Alegre: Bookman, 2011.
RUMBAUGH, James; *et al.* Modelagem e Projetos Baseados em Objetos com UML. 2ªed. RJ: Campus, 2006.
RUMBAUGH, James; JACOBSON, Ivar; BOOCH, Grady. UML: Guia do Usuário. 1ªed. RJ: Campus, 2006.
SOMMERVILLE, Ian. Engenharia de Software. 8ªed. Pearson, 2007.

6 Introdução à Medidas de Esforço

Uma dúvida que me perseguiu por muito tempo foi, como determinar o prazo e custo do desenvolvimento de um produto que ainda não temos todos os requisitos levantados e sim a previsão do que deverá ser feito. A resposta foi chegando aos poucos, com as tentativas frustradas de estimar prazos e custos de algo que na sua totalidade era desconhecido. Assim, fui percebendo que não há como determinar o tamanho de um *software*, por mais que se possua experiência, e por mais que as conversas iniciais com o cliente sejam esclarecedoras. Sem fazer o levantamento de requisitos, o que poderemos ofertar aos contratantes é apenas uma estimativa ou a técnica do "chutômetro". Pode-se até acertar, se o fator experiência, expertise do negócio e sorte, estiverem do seu lado. Mas, por mais veterano que você e sua equipe possam ser, o domínio do negócio a ser desenvolvido não é uma obrigatoriedade, e infelizmente o fator sorte é uma incógnita. Então, o que fazer?

Bom, quando disse que a resposta chegou, ela veio com o nome métrica. Resumindo, não há como estimar com boa aproximação, ou determinar o tamanho de um *software*, se não fizermos o uso de uma métrica. E para colaborar com o que digo, segue a citação de Drucker (2007) "A melhor estrutura não garantirá os resultados nem o rendimento. Mas a estrutura errada é a garantia do fracasso." Podemos entender que, caso não use uma métrica apropriada ou não use nenhuma métrica, a possibilidade de mensurar errado é muito alta, caso use a métrica certa, a possibilidade de acerto já passa a existir e num percentual aceitável.

Métrica pode ser: na literatura um conceito relacionado ao ritmo de um poema; na matemática um conceito ligado à distância; na música um conceito sobre a divisão de uma linha musical em compassos; e na Engenharia de *Software* (ESW) um conceito relacionado à gerência de projetos.

Bom, se estamos falando em engenharia, isto é, construções de algo não podem deixar de falar em medição, que é algo intimamente ligado a criar ou construir algo. E um *software* também precisa ser medido. Porém, essa é uma tarefa não muito fácil, já que estamos falando da construção de algo abstrato, que é um *software*, e assim sendo, possuindo um grau de subjetividade alto. E tudo que é subjetivo, está à mercê de interpretação as mais diversas, já que nesse caso, as referências, a experiência, entre outros fatores, influenciam a interpretação. Dessa forma, não existe ainda na ESW uma medição padrão aceito amplamente pela comunidade e que forneça resultados sem subjetividade. Isso tornar o processo de medir, muito árduo e muitas vezes, gerador de conflitos do tipo, o que medir o que avaliar, e se os resultados são aceitáveis.

Mas, quando fiz o curso de Análise de Pontos de Função (APF) o professor Guilherme Simões autor do livro citado na referência, que é certificado em AFP

pelo IFPUG e utilizada essa métrica há muitos anos, nos disse que, ela não é a melhor métrica que existe, muito longe disso, mas em relação ao que existe hoje no mercado, ainda é a melhor. Assim, temos que ter consciência que medir o tamanho de um *software* é necessário, mas que qualquer métrica escolhida, terá suas limitações. Mas, sem uma métrica o processo de gerenciar a construção de um produto de *software*, planejando seu esforço, seu custo, seus recursos, suas atividades seu prazo, etc., torna-se uma caixa preta cheia de surpresas desagradáveis.

As métricas na ESW são divididas em duas categorias, as quais são: Direta. Focada na medição do custo e do esforço de construção, isto é, tanto o desenvolvimento com a manutenção, seja ela evolutiva ou corretiva. Analisando a quantidade de Linhas de Código Fonte (SLOC) produzidas e o total de defeitos encontrados. Exemplos: custo, esforço, linhas de código, velocidade de execução, memória, número de erros, e complexidade ciclomática, isto é, a quantidade de caminhos de execução independentes a partir dum código fonte; e Indireta. Focada na qualidade e na funcionalidade do produto de *software*, em relação à aplicabilidade do *software*. Exemplos: funcionalidade, qualidade, complexidade, eficiência, confiabilidade, e manutenibilidade.

Essas ainda podem ser divididas em outras duas subcategorias, as quais são:

- Produtividade. Focada nas saídas do processo de engenharia de *software*; e
- Qualidade. Focada no atendimento aos requisitos definidos pelo usuário.

Razões para Medir:

- Verificar a qualidade do *software*;
- Avaliar a produtividade dos desenvolvedores;
- Determinar as vantagens de novos métodos;
- Analisar as vantagens de novas ferramentas;
- Compor uma base para as estimativas;
- Fomentar oportunidades por refatoração;
- Justificar aquisições de recursos;
- Prever o custo do desenvolvimento;
- Estimar o prazo para entrega dos artefatos;
- Apoiar o planejamento do escopo do *software*;
- Mensurar os quantitativos de recursos; e
- Obter a produtividade dos recursos.

Métricas da Engenharia de *Software*:

- Pontos de Função (APF) que mede o tamanho funcional do *software*. Saber mais; e

- Pontos de Caso de Uso que mede o tamanho de um *software* a partir da utilização do mesmo pelos usuários. Saber mais.

6.1 Conceitos

Medida: A medição é a atividade básica de qualquer engenharia e não seria diferente para a engenharia de *software*, porém como esse campo é muito ressentido, está longe de desenvolver uma medição padrão que seja amplamente aceita e principalmente, tenha elementos para seus cálculos que não sejam baseados em fatores subjetivos. Consequentemente a comunidade de TIC possui muitas críticas sobre as medições hoje utilizadas e muita discordância sobre o que medir e como avaliar os resultados obtidos dessas medições.

Métrica: As métricas de *software* servem para realizar a tarefa fundamental do gerenciamento de projetos que é o planejamento. É com uma métrica que se pode identificar a quantidade de esforço necessário para o desenvolvimento, o custo para tal e o prazo para entrega dos artefatos. As métricas de *software* sob o ponto de vista de MEDIÇÃO são divididas em duas categorias: medidas diretas e indiretas.

Medidas Diretas: é composto por medições que objetivam descobrir o custo para produzir o *software*, o esforço necessário a ser aplicados ao desenvolvimento ou a manutenção do *software*, a quantidade de linhas de código produzidas e o total de *bugs* encontrados. Exemplos: Custo; Esforço; Linhas de Código; Velocidade de Execução; Memória; Número de *Bugs*; Complexidade Ciclômática.

Medidas Indiretas: utilizam medições mais complexas, pois objetivam verificar a qualidade do *software*, a funcionalidade, ou a sua capacidade de manutenção, sendo mais difíceis de serem percebidas e avaliadas. Exemplos: Funcionalidade; Qualidade; Complexidade; Eficiência; Confiabilidade; Manutenibilidade.

As métricas de *software* sob o ponto de vista de APLICAÇÃO são divididas em duas categorias: medidas produtividade e qualidade. Medidas de Produtividade: verificam as saídas do processo de engenharia de *software*, e tem como objetivo de avaliar próprio processo de desenvolvimento. Medidas de Qualidade: verifica o quanto o *software* atende aos requisitos definidos pelo usuário, isto é, as funcionalidades assim permitem indicar o nível de resposta do *software* às exigências explícitas e implícitas do cliente, com relação ao que foi definido como qualidade.

6.2 Terminologias

MEDIDA: quantidade, dimensão, capacidade ou tamanho do *software*.

MEDIÇÃO: ato de medir.

INDICADOR: métrica que fornece compreensão dos resultados do *software*.

MÉTRICA: as técnicas utilizadas para verificar a funcionalidade, a modularidade, a manutenibilidade, etc., e podem ser subclassificadas como Orientadas ao Tamanho, Orientadas a Função, Orientadas a Pessoas.

6.3 Análise de Ponto de Função

Como dito anteriormente à métrica que mesmo não sendo a melhor, mas a que produz resultados menos conflitantes é a APF e é essa que vamos usar.

A Análise de Pontos de Função (APF) ou *Function Point Analysis* (FPA) foi criada em 1979 na IBM, tornando-se uma referência mundial na década de 80. A APF é um conjunto de técnicas que objetivam medir as funcionalidades, sendo assim, uma unidade de medida para fins práticos, não importando a tecnologia utilizada na construção do *software*. Resumindo, é uma métrica capaz de mensurar o tamanho do *software*.

O órgão responsável pela manutenção e divulgação das diretrizes da APF é o *International Function Point Users Group* (IFPUG), que publicou e mantém o Manual de Práticas de Contagem de Pontos de Função – *Counting Practices Manual* (CPM).

6.3.1 Cálculo do Ponto de Função

O objetivo da Análise de Pontos de Função (APF) é obter o tamanho de um *software* em Pontos de Função (PF) analisando as funcionalidades a serem desenvolvidas, isto é, os requisitos funcionais.

Porém é importante salientar que essa funcionalidade é sob o ponto de vista do usuário (PVU). Mas, o que quer dizer sob o ponto de vista do usuário? Significa que um Requisito Funcional (RF) tipo o que se segue RF001 deve ser analisado aos olhos do usuário e não do desenvolvedor.

RF001 - [...] é necessário armazenar os dados do cliente como CPF; nome; endereços residencial, comercial e de contato; telefones residencial, comercial, contato, celular; e-mails; sexo; nascimento; filiação; emprego atual; tempo de trabalho; emprego anterior; e faixa salarial [...].

Se um desenvolvedor fizer a leitura desse requisito, obviamente ele saberá que terá trabalho pela frente. Visto que, na sua visão tecnicista, já visualiza a modelagem do banco, que normalizará essa entrada em diversas tabelas, caso a persistência dos dados seja em um banco de dados relacional, e que uma série de classes de domínio, de controle, de persistência e de apresentação, terão que ser criadas, caso do desenvolvimento for numa linguagem orientada a objetos. Além da usabilidade, que exigirá uma interface gráfica (GUI) bem

amigável, já que o cadastro poderá produzir várias visões para um único cadastrador.

Resumindo, sob ponto de vista do desenvolvedor (PVD), isso vai dar um trabalho danado. Porém, para o PVU se trata de um cadastro único. No qual serão cadastrados todos os dados acima levantados e simplesmente o utilizador irá clicar em um botão salvar e pronto. Para o PVU é muito simples. Qual o trabalho que tem isso? :)

Agora, por que se deve analisar pelo PVU? A resposta é: Para que a medição de um projeto de desenvolvimento de *software* e a manutenção deste não se preocupe com a tecnologia a ser utilizada na sua implementação e sim, apenas nas funcionalidades do sistema requisitadas pelo usuário. Explicando melhor, se você for desenvolver o RF001 em Clipper, ou em Pascal, ou em Delphi, ou Visual Basic, ou Java, ou CSharp, etc, o grau de esforço para cada linguagem é diferente, como também é diferente com o uso do tipo de arquitetura de banco de dados, se banco relacional é uma, se banco orientado a objetos é outra, assim a medição sofreria uma grande variação devido ao fator tecnologia empregada. Então, a mesma tem que ser isenta dessa preocupação, só focando nas necessidades do usuário.

6.3.2 Técnica

A primeira ação é determinar qual o tipo de contagem de pontos de função que deverá ser realizada. Podem ser de três tipos: Projeto de Desenvolvimento (PD); Aplicações Instaladas (AI); e Projetos de Manutenção (PM).

A segunda ação realiza a identificação do escopo de contagem, com a determinação da fronteira da aplicação, isto é, definir quais serão as funcionalidades que serão incluídas na contagem de PFs. Para isso a definição da fronteira da aplicação é fundamental, pois estabelece as fronteiras entre aplicação, usuário e outras aplicações.

A terceira ação é determinar a contagem de Pontos de Função Não Ajustados (PFNA), isto é, verificar requisitos funcionais levantados, e classificá-los em dados e transações.

A quarta ação é a contagem dos dados. Para contar os dados se leva em consideração os dados *internos* os chamados Arquivos Lógicos Internos (ALI) e os dados *externos* os denominados Arquivos de Interface Externa (AIE) da aplicação. Ambos representam um grupo de dados logicamente relacionados que foram identificados pelo usuário. A diferença é que os ALIs são mantidos dentro da fronteira e os AIEs são os mantidos fora da fronteira da aplicação. Simplificando, os ALIs são persistidos pelas operações da aplicação, enquanto os AIEs são referenciados pela aplicação, mas não persistidos por ela.

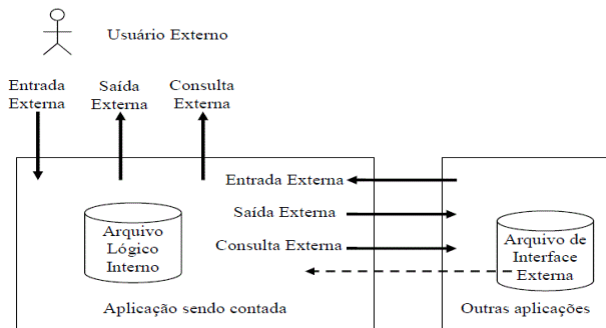
Entendido o que é uma métrica e o que se propõe a Análise de Ponto de Função, vamos aprender a contar PFs.

Exemplo, no RF001 terá que armazenar no banco de dados da aplicação os dados do cliente, e neste caso terá 1ALI, mas existe um requisito RF002 relacionado ao RF001 que diz:

RF002 - todos os endereços devem ser validados por CEP que deve ser consultado na base de dados dos correios.

Assim, para o referido cadastro do usuário, teremos 1AIE já que a persistência dos dados de CEP's é feita pelos correios que é outro sistema que se relaciona com o desenvolvido e está fora da fronteira da aplicação.

Figura 6.1 Análise de Ponto de Função



Fonte: Internet

A quinta ação consiste na contagem das transações. Para contar as transações se leva em consideração às funcionalidades de processamento de dados do sistema fornecidas para o usuário. Esses são classificados em Entradas Externas (EE), as Saídas Externas (SE) e as Consultas Externas (CE). As EEs são os processos responsáveis em manter os ALIs. As SEs são os processos responsáveis em exibir as informações recuperadas através de um processamento lógico, com a realização de cálculos. As CEs são os processos responsáveis em exibir as informações recuperadas através de um processamento lógico, sem a realização de cálculos.

A tabela a seguir define a complexidade dos ALI e os AIE:

Figura 6.2 Tabela Complexidade ALI e AIE

A L I A I E	TR TD	<20	20-50	>50
	1	Baixa	Baixa	Média
	2-5	Baixa	Média	Alta
	>5	Média	Alta	Alta

Fonte: IFPUG

A esta outra tabela define a complexidade do EE:

Figura 6.3 Tabela Complexidade EE

E E	TR TD	<5	5-15	>15
	<2	Baixa	Baixa	Média
	2	Baixa	Média	Alta
	>2	Média	Alta	Alta

Fonte: IFPUG

A próxima tabela define a complexidade dos SE e CE:

Figura 6.4 Tabela Complexidade SE e CE

S E C E	TR TD	<6	6-19	>19
	<2	Baixa	Baixa	Média
	2-3	Baixa	Média	Alta
	>3	Média	Alta	Alta

Fonte: IFPUG

A tabela ao lado é a define os pesos para a quantidade de ocorrências de dados e transações encontrados na análise dos requisitos funcionais:

Figura 6.5 Tabela de Contribuição

C o n t r i b u i ç ã o	Tipo	Baixa	Média	Alta
	ALI	7PF	10PF	15PF
	AIE	5PF	7PF	10PF
	EE	3PF	4PF	6PF
	SE	4PF	5PF	7PF
	CE	3PF	4PF	6PF

Fonte: IFPUG

A sexta ação serve para verificar as complexidades. Encontrados os ALIs, os AIEs, as EEs, as SEs e as CE, devem-se verificar as suas complexidades funcionais e classificá-las em baixa, média ou alta. Essas complexidades são definidas com base em dois conceitos, os Tipos de Registros (TR) e os Tipos de Dados (TD). Os TRs correspondem ao conjunto de dados dentro de um ALI/AIE, que foram reconhecidos pelo usuário, exemplo, imagine o cadastramento de uma venda, teríamos no mínimo três tabelas de dados envolvidas no processo, a de vendas, itens-da-venda e a de produtos, mas todas as três seriam consideradas apenas 1RL. Os TDs são os campos reconhecidos pelo usuário como únicos e não repetidos, exemplo, o cadastro de cliente do RF001, temos o CPF, o nome, os fones comercial, residencial e celular, neste caso, temos 3ID's, já que os fones devem ser contados como 1ID. Contando-se os RLs e os IDs de um ALI/AIE, pode-se chegar à sua complexidade.

A seguir temos um exemplo de uma planilha de contagem. Para obter essa planilha e as tabelas de complexidade clique [aqui](#).

Figura 6.6 Exemplo de Planilha de Contagem (PC)

[illegible]

Fonte: Próprio Autor

A sétima ação é realizar a contagem dos pontos de função, considerando-se o tipo de contagem definida na primeira ação. O processo de contagem é relativamente simples, consiste em separar os requisitos funcionais, analisar para cada RF os elementos básicos necessários, depois identificar nesses quantos ALIs, AIEs, EEs, SEs e CEs existem, para depois verificar em cada um deles quantos TRs e TDs são encontrados. Pronto, agora é só preencher a Planilha de Contagem (PC), tomando como base as Tabelas de Complexidade

acima exibidas. Por exemplo, supondo que para o requisito RF001 tenhamos encontrados dos elementos básicos incluir cliente, alterar cliente, excluir cliente, consultar cliente e imprimir cliente. Fazendo a análise de PF teríamos 1ALI, 1AIE, 3EE, 1SE e 1CE. Ok, agora contar quantos TRs e TDs serão encontrados para cada um dos dados e transações encontrados. Supondo novamente que tenhamos para o primeiro EE, 2TR e 25TD, para o segundo EE, 1TR e 15TD, e para o terceiro EE, 1TR e 32TD. Com esse levantamento, podemos fazer a verificação com as tabelas e com o cruzamento dos dados obtemos a complexidade, se baixa, média ou alta.

A oitava ação é obter o total de pontos de função não ajustados (PFNA), que é obtido pelo somatório dos pontos. Esse consiste de após realizar o preenchimento das colunas "Processo Elementar", "Tipo", "TD", "TR" e "Complexidade" da Planilha de Contagem, basta fazer o batimento com a Tabela de Contribuição e lançar os pesos na coluna "PF". Pronto, é só fazer o somatório dos PFs e se obterá o PFNA, com outras palavras, temos o tamanho do *software*, que diríamos para exemplificar 2500PFs. Com esse tamanho determinado já se pode obter o prazo e o custo para o esforço de construção, criando-se uma relação entre a complexidade e o tempo para resolvê-la, e o tamanho multiplicando por um valor equivalente a 1PF. Supondo que 1PF equivale a R\$80,00 então 2500PFs aplicando a regra de três equivale a R\$200.000,00. Temos aí o custo do desenvolvimento, sem levar em consideração os componentes indiretos da métrica, isto é, as 14 características já citadas.

A nona ação deve determinar o valor do fator de ajuste, isto é, avaliar 14 características gerais de sistemas, que avaliam a funcionalidade geral da aplicação que está sendo contada e seus Graus de Influência (GI). O GI é determinado com base na seguinte escala: 0=Nenhuma influência; 1=Influência mínima; 2=Influência moderada; 3=Influência média; 4=Influência significativa; e 5=Influência forte.

A décima ação é aplicar o fator de ajuste de influência nos PFNA que fica na ordem de +/- 35%, aplicando-se as 14 características gerais dos sistemas, a saber: 1=Comunicação de Dados; 2=Processamento de Dados Distribuído; 3=Desempenho; 4=Utilização do Equipamento (Restrições de Recursos Computacionais); 5=Volume de Transações; 6=Entrada de Dados On-line; 7=Eficiência do Usuário Final (Usabilidade); 8=Atualização On-line; 9=Processamento Complexo; 10=Reusabilidade; 11=Facilidade de Implantação; 12=Facilidade Operacional (Inicialização, Cópia, Recuperação, etc.); 13=Múltiplos Locais e Organizações do Usuário; e 14=Facilidade de Mudanças (Manutenibilidade).

Para cada uma dessas 14 características deve-se atribuir um dos GIs para determinar o nível de influência, que indicará o quanto determinado à

característica tem influência no sistema. Os 14 graus de influência (GIs) informados são somados, resultando no Nível de Influência Total (NIT).

A décima primeira ação consiste em determinar o Valor do Fator de Ajuste (VFA) pela fórmula $VFA = (NIT * 0,01) + 0,65$.

A décima segunda ação é o cálculo dos Pontos de Função Ajustados (PFA). Esse cálculo depende do tipo de projeto, se desenvolvimento, se manutenção ou se aplicações instaladas. Se for um projeto de desenvolvimento de sistemas então a fórmula é:

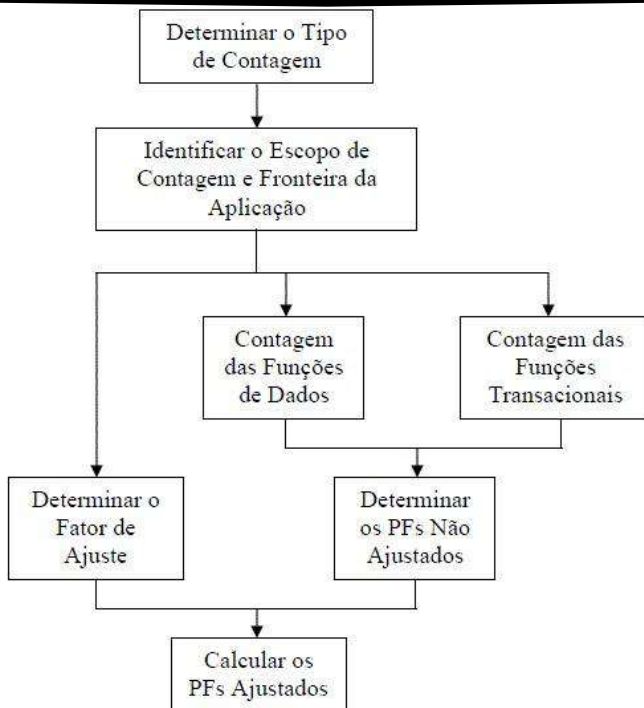
- $Projeto_Desenvolvimento = Ponto_Função_Não_Ajustado * Valor_Fator_Ajuste$;
- $Aplicação_Implantada = Projeto_Desenvolvimento - (Pontos_Função_Não_Ajustados * Fator_Ajuste)$;
- $Projeto_Manutenção = (Pontos_Função_Não_Ajustado + Ponto_Função_Incluído + Ponto_Função_Alterado_Atual - Ponto_Função_Alterado_Anterior - Ponto_Função_Excluído) * Fator_Ajuste_Atual$.

Com isso, se tem o tamanho do *software* e com esse tamanho (x PF's), pode-se determinar um valor a ser pago por cada ponto de função, e a partir dele determinar o valor total do *software*, além de determinar formas de pagamentos, relacionadas às entregas realizadas. Exemplo: se o tamanho do *software* é de 2.500 PFs e se paga R\$80,00 por todos os PF o valor total a ser pago é de R\$200.000,00. Mas, se no planejamento de entregas, ficou determinado que a contratada devesse entregar 50 PFs no mês e só entregou 25 PFs, o que será pago pela contratante a contratada, é o trabalho realizado, assim sendo, ela receberá R\$2.000,00 pelo trabalho feito, ficando os 25 PFs desse mês, para os próximos meses acrescidos dos outros 50PFs que já se tinham planejado. Controle efetivo do que se tem a produzir e do que é produzido.

Clicando aqui se pode obter uma [Tabela de Referência de PF](#) para cada linguagem de programação.

Segue um resumo de todas as ações pode ser visto a figura a seguir.

Figura 6.7 Passos da Contagem



Fonte: Internet

6.4 Ferramentas

APFPlus. Disponível em < <http://ziggi.uol.com.br/downloads/12202> >
Metric Studio for FPA. Disponível em < <http://www.tsaquality.com/> >

6.5 Apresentações

Análise. < <http://www.slideshare.net/alvarofpinheiro/medida-de-esforo-de-software-com-anlise-de-ponto-de-funo> >

6.6 Sugestões de Leitura

APF para Melhoria de Software. Disponível em < <http://goo.gl/oeEK8v> >
Custo de Software. Disponível em < <http://goo.gl/q6Vm8o> >
Engenharia de Software. Disponível em < <http://goo.gl/K55qXq> >
Engenharia de Software: Os Paradigmas Clássico e Orientado a Objetos. Disp.:< <http://goo.gl/I5G7Y2> >
Tecnologia da Informação - Questões Comentadas Cespe. Disponível em < <http://goo.gl/I57buQ> >

6.7 Referências Web

Brazilian Function Point Users Group (BFPUG). Disponível em <

<http://www.bfpug.com.br/> >

Fatto Consultoria e Sistemas - empresa especializada em métricas. Disponível

em < <http://www.fattocs.com.br/> >

International Function Point Users Group (IFPUG). Disponível em <

<http://www.ifpug.org/> >

Peruntas e Respostas sobre APF BFPUG. Disponível em <

<http://www.bfpug.com.br/FAQ.htm> >

Template da Planilha de Contagem de Pontos de Função. Disponível em <

<http://goo.gl/IESrhq> >

QSM. Disponível em < <http://goo.gl/3UzXCY> >

6.8 Referências Bibliográficas

PRESSMAN, Roger. Engenharia de Software: uma abordagem profissional.

7ªed. Porto Alegre: Bookman, 2011.

RUMBAUGH, James; *et al.* Modelagem e Projetos Baseados em Objetos com UML. 2ªed. RJ: Campus, 2006.

RUMBAUGH, James; JACOBSON, Ivar; BOOCH, Grady. UML: Guia do Usuário. 1ªed. RJ: Campus, 2006.

SOMMERVILLE, Ian. Engenharia de Software. 8ªed. Pearson, 2007.

VAZQUES, Carlos; SIMÕES, Guilherme; *et al.* Análise de Pontos de Função.

10ªed. São Paulo: Érica, 2010.

7 Referências Bibliográficas

- BENNETT, Simon; *et al.* Schaum's Outline of UML. 1ªed. UK: McGraw-Hill, 2005.
- BEZERRA, Eduardo. Princípios de Análise e Projetos de Sistemas com UML. 1ªed. Rio de Janeiro: Campus, 2007.
- CARDOSO, Caíque. UML na Prática: do problema ao sistema. 1ªed. São Paulo: CM, 2003.
- CHIAVENATO, Idalberto. Introdução à Teoria Geral da Administração. 7ªed. Rio de Janeiro: Campus, 2004.
- DEMARCO, Tom; LISTER Timothy. Peopleware Productive Projects and Teams. US: Dorset House Pub, 1999.
- DENNIS, Alan. Análise e Projeto de Sistemas. 2ªed. São Paulo: LTC, 2011.
- DRUCKER, Peter. Management: Tasks, Responsibilities, Practices. 1ªed. US: Transaction Publishers, 2007.
- FERNANDES, Daniel. Análise de Sistemas Orientada ao Sucesso. 1ªed. Rio de Janeiro: Ciência Moderna, 2005.
- FINLAY, Paul. Introducing decision support systems. 1ªed. US: NCC Education Services, 1998.
- FOWLER, Martin. UML Essencial. 2ªed. Porto Alegre: Bookman, 2004.
- KOSCIANSKI, André; SOARES, Michel. Qualidade de Software. 2ªed. São Paulo: Navatec, 2007.
- KRUCHTEN, Philippe. Introdução ao RUP Rational Unified Process. 1ªed. Rio de Janeiro: Ciência Moderna, 2003.
- LENDRIKE, Araújo. Projeto de Processos de Negócio visando à automação em BPMS, UNIRIO.
- MANZANO, Alves. Guia de Orientação e Desenvolvimento de Sites. 1ªed. São Paulo: Érica, 2008.
- MEDEIROS, Ernani. Desenvolvendo Software com UML 2.0: Definitivo. 1ªed. São Paulo: Makron Books, 2004.
- MELO, Ana. Desenvolvendo Aplicações com UML 2.2: do conceitual à implementação. 3ªed. RJ: Brasport, 2010.
- PENDER, Tom. UML - A Bíblia. 1ªed. RJ: Campus, 2004.
- PMI. A Guide to the Project Management Body of Knowledge: Pmbok Guide. 2011.
- PRESSMAN, Roger. Engenharia de Software: uma abordagem profissional. 7ªed. Porto Alegre: Bookman, 2011.
- RUMBAUGH, James; *et al.* Modelagem e Projetos Baseados em Objetos com UML. 2ªed. RJ: Campus, 2006.
- RUMBAUGH, James; JACOBSON, Ivar; BOOCH, Grady. UML: Guia do Usuário. 1ªed. RJ: Campus, 2006.
- SCHACH, Stephen. Engenharia de Software: Os Paradigmas Clássico e Orientado a Objetos. McGraw-Hill, 2008.
- SOARES, Antônio. Introdução, Identificação e Análise em Engenharia de Requisitos, 2005.
- SOMMERVILLE, Ian. Engenharia de Software. 8ªed. Pearson, 2007.

VAZQUES, Carlos; SIMÕES, Guilherme; *et al.* Análise de Pontos de Função. 10ªed. São Paulo: Érica, 2010.

WAZLAWICH, Raul. Análise e Projeto de Sistemas de Informação Orientados a Objetos. RJ: Campus, 2011.

8 Referências Web

A Theory of Object-Oriented Design. Disponível em < <http://goo.gl/fq3q5> >
Análise de Domínio wiki. Disp.: < http://pt.wikipedia.org/wiki/An%C3%A1lise_de_Dom%C3%ADnio >
Análise Orientada a Objetos wiki. Disponível em < <http://goo.gl/N0Ktwf> >
ANSI site oficial. Disponível em < <http://www.ansi.org/> >
ANSI wiki. Disponível em < <http://goo.gl/ZCrnXk> >
Association of Business Process Management Professionals (ABPMP). Disponível em < <http://www.abpmp-br.org/> >
Aulas de UML. Disponível em < <http://goo.gl/2TRbUF> >
Brazilian Function Point Users Group (BFPUG). Disponível em < <http://www.bfpug.com.br/> >
ControlChaos. Disponível em < <http://www.controlchaos.com/> >
Engenharia de Software wiki. Disponível em < http://pt.wikipedia.org/wiki/Engenharia_de_software >
Exemplo de Modelagem UML com a ferramenta ASTAH. Disponível em < <http://goo.gl/JvghSt> >
Exemplo de Modelagem UML. Disponível em < <http://goo.gl/zw0YCF> >
Fatto Consultoria e Sistemas - empresa especializada em métricas. Disponível em < <http://www.fattocs.com.br/> >
Gartner Group. Disponível em < <http://www.gartner.com/technology/home.jsp> >
GRASP. Disponível em < [http://en.wikipedia.org/wiki/GRASP_\(object-oriented_design\)](http://en.wikipedia.org/wiki/GRASP_(object-oriented_design)) >
Guia Básico sobre UML. Disponível em < <http://goo.gl/eGhizM> >
Guia da UML. Disponível em < <http://goo.gl/3Wtx0b> >
Guia de Modelagem de Casos de Uso. Disponível em < <http://goo.gl/PtwH3L> >
Guia de Modelagem de Negócios. Disponível em < <http://goo.gl/W2U0B6> >
Guia de Design. Disponível em < <http://goo.gl/OpvFEC> >
Guide to the Project Management Body of Knowledge (PMBOK Guide) wiki. Disponível em < <http://goo.gl/Oa0fh> >
IBM Rational Unified Process. Disponível em < <http://goo.gl/ugl9H7> >
IBM RUP. Disponível em < <http://www-01.ibm.com/software/awdtools/rup/> >
Informações Gerenciais Relevantes. Disponível em < <http://goo.gl/M3WKJJ> >
International Function Point Users Group (IFPUG). Disponível em < <http://www.ifpug.org/> >
ISO site oficial. Disponível em < <http://www.iso.org/iso/home.html> >
ISO wiki. Disponível em < <http://goo.gl/vjWG> >
Iterativo e Incremental wiki. Disp.: < http://pt.wikipedia.org/wiki/Desenvolvimento_iterativo_e_incremental >
Knowledge Management (KM) wiki. Disponível em < http://en.wikipedia.org/wiki/Knowledge_management >
Mais sobre relacionamentos de Inclusão e Extensão. Disponível em < <http://goo.gl/uH8X6Z> >

Manuais de Referência OMG-UML. Disponível em < <http://www.omg.org/spec/UML/> >
Modelagem UML. Disponível em < <http://goo.gl/iwtZh3> >
Modelo Espiral wiki. Disponível em < http://pt.wikipedia.org/wiki/Modelo_em_espiral >
Modelos Ágeis wiki. Disp.: < http://pt.wikipedia.org/wiki/Desenvolvimento_%C3%A1gil_de_software >
Modelos Ciclo de Vida wiki. Disponível em < http://pt.wikipedia.org/wiki/Modelos_ciclo_de_vida >
Object Management Group (OMG). Disponível em < <http://www.omg.org/> >
Object-Oriented Programming (OOP). Disponível em < <http://goo.gl/r1YFIs> >
Perguntas e Respostas sobre APF BFPUG. Disponível em < <http://www.bfpug.com.br/FAQ.htm> >
PMBOK 1996. Disponível em < <http://goo.gl/WD4ULJ> >
PMI Brasil wiki. Disponível em < <http://www.pmi.org.br/> >
PMI Internacional. Disponível em < <http://www.pmi.org/> >
PMI Pernambuco. Disponível em < <http://www.pmi.pe.org.br/v2/html/> >
Project Management Institute (PMI) wiki. Disponível em < http://pt.wikipedia.org/wiki/Project_Management_Institute >
QSM. Disponível em < <http://goo.gl/3UzXCY> >
Rational Unified Process (RUP). Disponível em < <http://www-01.ibm.com/software/br/rational/> >
RUP. Disp.: < [http://www.wthreex.com/rup/portugues/index.htm%20\(Excelente%20site%20sobre%20RUP\)](http://www.wthreex.com/rup/portugues/index.htm%20(Excelente%20site%20sobre%20RUP)) >
Scrum Alliance. Disponível em < <http://www.scrumalliance.org/> >
Scrum. Disponível em < <http://www.mountangoatsoftware.com/scrum> >
SEI. Disponível em < <http://www.sei.cmu.edu/> >
Standish Group Brazil. Disponível em < <http://goo.gl/NCpTzD> >
Standish Group Manifesto 2013. Disponível em < <http://goo.gl/ShD7HW> >
Standish Group Report Chaos 2009. Disp.: < http://www1.standishgroup.com/newsroom/chaos_2009.php >
Standish Group Report Chaos. Disponível em < <http://goo.gl/KOyh4> >
Standish Group. Disponível em < <http://www.standishgroup.com/> >
SWEBOK. Disponível em < <http://www.computer.org/portal/web/swebok> >
Técnicas para Coleta de Dados. Disponível em < <http://goo.gl/rX7MSA> >
Técnicas para Levantamento de Requisitos Método Volere. Disponível em < <http://goo.gl/z2vm1D> >
Técnicas para Levantamento de Requisitos Não Funcionais. Disponível em < >
Técnicas para Levantamento de Requisitos. Disponível em < <http://goo.gl/3uO5Ji> >
Template Casos de Uso. Disp.: < http://www.wthreex.com/rup/portugues/webtmpl/templates/reg/rup_ucspec.htm >
Template da Planilha de Contagem de Pontos de Função. Disponível em < <http://goo.gl/IESrhq> >

Template Mais Detalhado para Especificação de Casos de Uso. Disp.: < <http://goo.gl/5yKpBD> >

Template RUP para Especificação de Casos de Uso. Disponível em < <http://goo.gl/GwnXCn> >

Transição de Estados wiki. Disponível em < <http://goo.gl/QVb3F5> >

Um Mundo Feito (quase completamente) de *Software*. Disponível em < <http://goo.gl/YE26CS> >

Unified Modeling Language (UML). Disponível em < <http://www.uml.org/> >

9 Biografias

Alan Turing nasceu na Inglaterra em 1912, foi matemático, lógico, criptoanalista e cientista da computação. É um dos nomes importantes na Ciência da Computação. Formulou conceitos de algoritmo e computação com a máquina de Turing. Trabalhou no *National Physical Laboratory*. Saber mais.

Disponível em < http://en.wikipedia.org/wiki/Alan_Turing >

Alfred Vaino Aho é cientista da computação canadense. Conhecido internacionalmente como co-autor da linguagem de programação AWK, junto com Peter Weinberger e Brian Kernighan, e também pela sua co-autoria no Livro do Dragão como é mais conhecido o livro, *Compilers: Principles, Techniques, and Tools*, escrito em parceria com Ravi Sethi e Jeffrey Ullman. Suas contribuições para área foram às versões iniciais das ferramentas Unix egrep e fgrep. Também possui livros nas áreas de algoritmos e estruturas de dados. Saber mais.

Disponível em < http://pt.wikipedia.org/wiki/Alfred_Aho >

Avram Noam Chomsky nasceu nos Estados Unidos na Filadélfia em 1928, é um linguista e filósofo. É professor de Linguística no *Massachusetts Institute of Technology*. É autor de trabalhos fundamentais sobre as propriedades matemáticas das linguagens formais, sendo o seu nome associado à chamada Hierarquia de Chomsky. Seus trabalhos estão relacionados a uma abordagem matemática da linguagem. Saber mais.

Disponível em < http://pt.wikipedia.org/wiki/Noam_Chomsky >

Christopher Date formou-se no Reino Unido pela High Wycombe Royal Grammar School em 1958 e recebeu o bacharelado em Matemática pela Universidade de Cambridge em 1962. Iniciou as atividades na área da computação como programador na *Leo Computers Ltd* em Londres onde também começou a lecionar. Em 1967 começou a trabalhar na IBM na Inglaterra também como programador e instrutor de computação, ficando entre os anos de 1969 e 1974 como o principal instrutor da área educacional da IBM. Saber mais.

Disponível em < http://en.wikipedia.org/wiki/Christopher_J._Date >

Craig Larman é um canadense nascido em 1958 que trabalha como cientista da computação especializado em desenvolvimento iterativo e incremental, desenvolvimento de *software* ágil, análise orientada a objetos e modelagem ágil. Ele é o autor de vários textos sobre Scrum. Saber mais.

Disponível em < http://en.wikipedia.org/wiki/Craig_Larman >

Erich Gamma é um dos quatro autores do livro *Design Patterns: Elements of Reusable Object-Oriented Software*. Ele é também o principal projetista do JUnit e do Eclipse. Nascido em 1961, em Zurique é cientista da computação e co-autor influente de livros didáticos sobre engenharia de *software*. Co-escreveu o

JUnit framework de teste de *software* com Kent Beck e liderou o projeto do Eclipse da plataforma Java *Development Tools* (JDT). Também trabalhou na IBM Rational projeto Jazz. Saber mais.

Disponível em < http://en.wikipedia.org/wiki/Erich_Gamma >

Grady Booch nasceu em 1955 nos Estados Unidos, é engenheiro de software e cientista-chefe da IBM Research. Obteve seu grau de bacharel pela *United States Air Force Academy* e título de mestre em engenharia elétrica pela Universidade da Califórnia em Santa Barbara. Saber mais.

Disponível em < http://en.wikipedia.org/wiki/Grady_Booch >

Hal Varian, nascido em 1947, em Ohio, é economista especializado em microeconomia e economia da informação. É professor emérito nas universidades de Berkeley e Haas *School of Business*. Formou-se como bacharel em economia pelo e licenciado em matemática pelo MIT. Possui o doutorado em economia pela *Berkeley University*, e é doutor honorário pelo Instituto de Tecnologia de Karlsruhe, Alemanha. Já lecionou em *Stanford University*, *Oxford University*, *Michigan University*. Nos dias atuais é o economista-chefe da Google e publica regularmente artigos no *New York Times* e *Wall Street Journal*. Saber mais.

Disponível em < http://en.wikipedia.org/wiki/Hal_Varian >

Harvey Deitel tem mais de 40 anos de experiência na área de computação. Graduou-se e fez o mestrado no *Massachusetts Institute of Technology* e o doutorado na *Boston University*. Possuindo mais de 20 anos de experiência no ensino universitário. Possuindo diversas publicações traduzidas em diversos idiomas. Saber mais.

Disponível em http://www.informit.com/authors/author_bio.aspx?ISBN=9780137058426 <

Ian Sommerville, inglês nascido em 1951 é professor titular de engenharia de software na Universidade de St. Andrews na Escócia. Destacado pesquisador no campo da engenharia de sistemas, segurança e informática social. Saber mais.

Disponível em < http://en.wikipedia.org/wiki/Ian_Sommerville_%28academic%29 >

Idalberto Chiavenato, nascido em 1936 em São Paulo, é autor na área de administração de empresas e de recursos humanos. Atualmente, atua como conselheiro no Conselho Regional de Administração do Estado de São Paulo (CRA-SP) e é Presidente do Instituto Chiavenato de Educação. Saber mais.

Disponível em < http://pt.wikipedia.org/wiki/Idalberto_Chiaavenato >

Ivar Jacobson é um sueco nascido em 1939, se graduou no Instituto de Tecnologia de Chalmers, onde fez seu mestrado em Engenharia Elétrica e possuindo doutorado pela *Royal Institute of Technology de Estocolmo*. Saber mais.

Disponível em < http://en.wikipedia.org/wiki/Ivar_Jacobson >

James Gosling nasceu em Calgary no Canadá em 1955 é conhecido como o pai da linguagem de programação Java. Em 1977, formou Bacharel em Ciência da Computação pela Universidade de Calgary, e em 1983 tornou-se PhD em Ciência da Computação pela Universidade Carnegie Mellon. Enquanto estudava no doutorado, Gosling programou a versão original do editor de texto Emacs para UNIX, e antes de se juntar a Sun Microsystems ele construiu uma versão de UNIX para máquinas multiprocessadas, assim como diversos compiladores e sistemas de mensagens eletrônicas. Trabalhou de 1984 até 2010 na Sun Microsystems, atualmente está na Google. Saber mais.

Disponível em < http://pt.wikipedia.org/wiki/James_Gosling >

James Rumbaugh é Bacharel em Física pelo Massachusetts Institute of Technology (MIT), possui mestrado em Astronomia pela Caltech, e é Ph. D. em Ciência da Computação também pelo MIT. Trabalhou na *Digital Equipment Corporation* (DEC), como cientista. Foi um dos líderes na criação da OMT e em 1994 se juntou a *Rational Software*, e junto com Ivar Jacobson e Booch de Grady desenvolveram a *Unified Modeling Language* (UML). Mais tarde, eles fundiram as suas metodologias de desenvolvimento de software, OMT, OOSE e Booch em um único processo batizado de *Rational Unified Process* (RUP). Saber mais.

Disponível em < http://en.wikipedia.org/wiki/James_Rumbaugh >

John von Neumann nasceu na Hungria em 1903 naturalizado como estadunidense, foi um matemático. Membro do Instituto de Estudos Avançados em Princeton, *New Jersey*, do qual também faziam parte Albert Einstein. Professor na Universidade de Princeton e um dos construtores do ENIAC. Trabalhou no desenvolvimento da Física Nuclear e contribuiu com diversas teorias, a exemplo as dos conjuntos, análise funcional, ergódica, mecânica quântica, ciência da computação, economia, teoria dos jogos, análise numérica, estatística, entre outras. Saber mais.

Disponível em < http://pt.wikipedia.org/wiki/John_von_Neumann >

Kenneth Laudon é professor de Sistemas de Informação na New York University's Stern School of Business. Bacharel em Economia pela *Stanford University* e Ph. D pela *Columbia University*. Autor de vários livros que tratam do tema Sistemas de Informação, Organizações e Sociedade. Saber mais.

Disponível em < <http://pages.stern.nyu.edu/~klaudon/> >

Patrick Naughton americano nascido em 1965 é um desenvolvedor de software conhecido como sendo um dos criadores originais da linguagem de programação Java, pois trabalho na Sun Microsystems no Projeto Verde a origem do Java. Em 1994, saiu da Sun e foi para Starwave de propriedade de Paul Allen para desenvolver um servidor Java. É autor do livro O Manual do Java, Osborne, 1995 e co-autor do *Java: The Complete Reference*, com Herbert

Schildt, Osborne , 1996. Em 1998, torna-se vice-presidente executivo da Infoseek. Saber mais.

Paul Deitel é graduado em Tecnologia da Informação e pós-graduação em Administração pela Massachusetts *Institute of Technology*. Trabalha como consultor em diversas organizações internacionais. Saber mais.

Disponível em http://www.informit.com/authors/author_bio.aspx?ISBN=9780137058426 >

Peter Ferdinand Drucker, austríaco nascido 1909 e que faleceu recentemente em 2005, foi um filósofo e economista, que por suas contribuições a comunidade, passou a ser considerado o pai da administração moderna, se tornando um dos pensadores mais citados pela teoria da globalização em economia em geral. Saber mais.

Disponível em < http://pt.wikipedia.org/wiki/Peter_Drucker >

Roger Pressman é Bacharel em Engenharia de Software pela University of Connecticut, com Mestrado pela *University of Bridgeport* e PhD pela *University of Connecticut*. Pertence ao conselho editorial da *IEEE Software* e do *The Cutter TI Jornal*. Autor e consultor na área de Engenharia de Software. Trabalha na área de engenharia de software a mais de 30 anos, atuando como administrador, professor e consultor. É presidente da *R.S. Pressman & Associates, Inc.*, empresa de consultoria especializada em práticas de Engenharia de Software. Saber mais. Disponível em < <http://www.cutter.com/meet-our-experts/rpbio.html> >

Stephen Schach é professor emérito do curso de Ciência da Computação e Engenharia de Computação do Departamento de Engenharia Elétrica e Ciência da Computação da Universidade de Vanderbilt. Possui mestrado em Física pelo *Weizmann Institute of Science* e doutorado em Matemática Aplicada pela *University of Cape Town*. Saber mais.

Disponível em < <http://www.vuse.vanderbilt.edu/~srs/> >

Tim O'Reilly, nasceu em 1954 na Irlanda, é o fundador da O'Reilly Media e estudioso e fomentador de software livre, para muitos é o criador da expressão Web 2.0. É Bacharel em filologia clássica pela Harvard University em 1975. Saber mais.

Disponível em < http://pt.wikipedia.org/wiki/Tim_O'Reilly >

Tom DeMarco, nascido em 1940 na Pensilvânia, é autor, professor e palestrante em engenharia de software e é reconhecido como um dos criadores da Análise Estruturada nos anos 1980. É Bacharel em Engenharia Elétrica pela Universidade de *Cornell University*, Mestre pela *Columbia University* e Doutor pela University of Paris at the Sorbonne. Saber mais.

Disponível em < http://en.wikipedia.org/wiki/Tom_DeMarco >

10 Artefatos (Templates) para Projetos de *Software*

Avaliação da Organização de Desenvolvimento. Disponível em < <http://goo.gl/n4Aqyg> > Acesso:Jan/15.

Avaliação da Organização-alvo. Disponível em < <http://goo.gl/n5bKph> >

Avaliação de Iteração. Disponível em < <http://goo.gl/YIRRBv> >

Avaliação de Status. Disponível em < <http://goo.gl/546iYl> >

Caso de Desenvolvimento. Disponível em < <http://goo.gl/xT7kXd> >

Caso de Negócios. Disponível em < <http://goo.gl/9euZmQ> >

Documento de Arquitetura de Negócios. Disponível em < <http://goo.gl/9AuSOa> >

Documento de Arquitetura de Software. Disponível em < <http://goo.gl/BsuHyg> >

Especificação de Caso de Uso de Negócios Nome do Caso de Uso de Negócios.

Disp.:< <http://goo.gl/XklLXt> >

Especificação de Caso de Uso Nome do Caso de Uso. Disponível em < <http://goo.gl/8OsCF8> >

Especificação de Realização de Casos de Uso Nome do Caso de Uso de Negócios. Disp.:< <http://goo.gl/PoKB2f> >

Especificação de Realização de Casos de Uso Nome do Caso de Uso.

Disponível em < <http://goo.gl/NNHreG> >

Especificação dos Requisitos de Software Para Subsistema ou Recurso.

Disponível em < <http://goo.gl/mnOYe5> >

Especificação dos Requisitos de Software. Disponível em < <http://goo.gl/dfnJMy> >

Especificação Suplementar de Negócios. Disponível em < <http://goo.gl/G52y77> >

Especificação Suplementar. Disponível em < <http://goo.gl/aGdgmi> >

Glossário de Negócio. Disponível em < <http://goo.gl/SRqCx8> >

Glossário de Negócios. Disponível em < <http://goo.gl/yjDhRQ> >

Glossário. Disponível em < <http://goo.gl/UVhQ2H> >

Lista de Materiais. Disponível em < <http://goo.gl/A0sTGO> >

Lista de Riscos. Disponível em < <http://goo.gl/Vyy1js> >

Modelo de Negócio. Disponível em < <http://goo.gl/O3O6Ya> >

Notas de Release. Disponível em < <http://goo.gl/e0FC0h> >

Plano de Aceitação do Produto. Disponível em < <http://goo.gl/RH4JDj> >

Plano de Desenvolvimento de Software (Projeto Pequeno). Disponível em < <http://goo.gl/J9gsvB> >

Plano de Desenvolvimento de Software. Disponível em < <http://goo.gl/5KRFCJ> >

Plano de Garantia de Qualidade. Disponível em < <http://goo.gl/D1M0Gp> >

Plano de Gerenciamento de Configuração. Disponível em < <http://goo.gl/almYbn> >

Plano de Gerenciamento de Requisitos. Disponível em < <http://goo.gl/1iwQYG> >

Plano de Gerenciamento de Riscos. Disponível em < <http://goo.gl/kvwGyW> >

Plano de Implantação. Disponível em < <http://goo.gl/zK4sDV> >

Plano de Integração do Build. Disponível em < <http://goo.gl/J5eP3T> >

Plano de Iteração. Disponível em < <http://goo.gl/1f4yLQ> >

Plano de Métricas. Disponível em < <http://goo.gl/64uo2r> >

Plano de Negócio. Disponível em < <http://goo.gl/Jfw2VK> >

Plano de Requisitos. Disponível em < <http://goo.gl/sLM5Lu> >

Plano de Resolução de Problemas. Disponível em < <http://goo.gl/pwvd6l> >

Plano de Teste. Disponível em < <http://goo.gl/XM0cNO> >

Proposta Comercial. Disponível em < <http://goo.gl/DI17e4> >

Regras de Negócios. Disponível em < <http://goo.gl/Zx5CWY> >

Requisitos de Software. Disponível em < <http://goo.gl/Bi5piJ> >

Solicitações dos Principais Envolvidos. Disponível em < <http://goo.gl/XGUuGR> >

Sumário de Avaliação de Testes. Disponível em < <http://goo.gl/ZVLVb6> >

Termo de Abertura do Projeto. Disponível em < <http://goo.gl/CYnUmT> >

Visão. Disponível em < <http://goo.gl/liXVDn> >

Visão (Projeto Pequeno). Disponível em < <http://goo.gl/x8YMS7> >

Visão do Negócio. Disponível em < <http://goo.gl/wUKpZc> >

11 Ferramentas

APFPlus. Disponível em < <http://ziggi.uol.com.br/downloads/12202> >

Metric Studio for FPA. Disponível em < <http://www.tsquality.com/> >

Astah Change-Vision. Disponível em < <http://astah.change-vision.com/en/index.html> >

Astah Community. Disponível em < <http://astah.change-vision.com/en/product/astah-community.html> >

Astah Profissional. Disponível em < <http://astah.change-vision.com/en/product/astah-professional.html> >

Astah Share. Disponível em < <http://astah.change-vision.com/en/product/astah-share.html> >

Astah UML. Disponível em < <http://astah.change-vision.com/en/product/astah-uml.html> >

Astah. Disponível em < <http://astah.change-vision.com/en/index.html> >

Astah: modelagem UML. Disponível em < <http://astah.change-vision.com/en/index.html> >

Green UML. Disponível em < <http://green.sourceforge.net/index.html> >

Jude Community. Disponível em < <http://goo.gl/r2xvL2> >

Manual do Astah. Disponível em < <http://goo.gl/oxXADk> >

StarUML: modelagem UML. Disponível em < <http://staruml.sourceforge.net/en/> >

StylusStudio. Disponível em < http://www.stylusstudio.com/java_ide.html >

12 Apresentações e Vídeos

Análise. < <http://www.slideshare.net/alvarofpinheiro/medida-de-esforo-de-software-com-anlise-de-ponto-de-funo> >

UML. Disponível em < <https://pt.slideshare.net/alvarofpinheiro/alvarofpinheirouml> >

13 Sugestões de Leitura

Análise e Projeto com UML. Disponível em < <http://goo.gl/Q5Hp7q> >

Análise e Projeto de Sistemas de Informação. Disponível em < <http://goo.gl/ITZiYS> >

Análise e Projeto de Sistemas. Disponível em < <http://goo.gl/kQVR7L> >

APF para Melhoria de Software. Disponível em < <http://goo.gl/oeEK8v> >

Custo de Software. Disponível em < <http://goo.gl/q6Vm8o> >

Desenvolvendo Aplicações com UML 2.2. Disponível em < <http://goo.gl/vVW6lL> >

Engenharia de Software e Sistemas de Informação. Disponível em < <http://goo.gl/3m2OIh> >

Engenharia de Software. Disponível em < <http://goo.gl/K55qXq> >

Engenharia de Software: Os Paradigmas Clássico e Orientado a Objetos. Disp.:< <http://goo.gl/I5G7Y2> >

Exercitando Modelagem com UML. Disponível em < <http://goo.gl/r7toX5> >

Guia prático de engenharia de software. Disponível em < <http://goo.gl/ppe5ff> >

UML Essencial. Disponível em < <http://goo.gl/cdLkbT> >

UML Guia do Usuário. Disponível em < <http://goo.gl/av5BTh> >

Livros da série Fundamentos da Engenharia de Software



Fundamentos da Engenharia de Software: Conceitos Básicos é uma

coletânea de disciplinas que integradas servem para fundamentar o entendimento da construção de projetos de software com qualidade, isto é, baseado em processos maduros e reconhecidos pela comunidade tecnológica. O objetivo deste livro é fornecer ao leitor as bases necessárias para o desenvolvimento de aplicações sejam Desktop, Web ou Mobile. Iniciando a leitura na Teoria da Computação, passando por Processos, Linguagens, Bancos de Dados e finalizando com Sistemas de Informação e Colaboração. Este livro pode ser lido capítulo a capítulo ou somente a disciplina desejada, pois sua elaboração consiste na compilação das disciplinas fundamentais da Engenharia de Software que são independentes, mas ao mesmo tempo se



Introdução à Engenharia de Software. Neste são abordados os

conceitos básicos de bancos de dados e seus sistemas gerenciadores, mas com o foco na arquitetura relacional, porque ainda hoje o mercado faz uso em larga escala desses bancos de dados, mesmo que o paradigma predominante seja o orientado a objetos e que, já existam a um bom tempo bancos orientados a objeto, até mesmo os bancos objetos-relacionais que são um híbrido entre essas duas arquiteturas, o que predomina ainda é o relacional, assim, este material é focado na linguagem de consulta estruturada para os SGBD-Rs do mercado, com foco na comparação de cinco dos mais utilizados bancos relacionais, os quais são: Oracle, SQLServer, MySQL, SQLBase e Interbase.



Este livro é sobre processos de desenvolvimento de software, evidencia

ndo a necessidade de qualidade na construção de sistemas, conceituando a diferença entre desenvolvimento Adhoc e com processo. Para isso é realizada a introdução à engenharia de requisitos abordando as técnicas para a elicitação de requisitos que forneçam subsídios necessários para uma construção de software com maior qualidade, enfatizando a necessidade de se aplicar na construção de qualquer sistema as técnicas de análise e modelagem, evidenciando o uso da linguagem da Linguagem de Modelagem Unificada (UML) para diagramar um projeto de software, explicando a necessidade do uso de modelos na construção, entrando com detalhes na análise orientada a objetos, com o objetivo de explorar os seus conceitos de requisitos e modelagem integrados. Este material é finalizado

integram objetivando o desenvolvimento de aplicações.

com a introdução à medidas de esforço de desenvolvimento, técnica necessária para responder as perguntas básicas de qualquer desenvolvimento: Qual o prazo e custo? E para responder a essas questões é abordado o uso da métrica análise de ponto de função.



Este livro aborda os sistemas que são classificados como informação, a

exemplo, sistemas de apoio a decisão, sistemas estratégicos, sistemas gerenciais e sistemas transacionais. A produção deste material que compõe o volume 4 da coleção Fundamentos da Engenharia de Software é resultado da compilação das aulas produzidas nas disciplinas que compõem os capítulos deste livro.



A motivação deste livro é exemplificar os conceitos de

Padrões de Projetos utilizando a linguagem de programação Java, sendo a construção uma compilação das aulas produzidas com o intuito de facilitar o entendimento do assunto abordando os seguintes temas: Paradigma Orientado a Objetos que introduz o leitor nos conceitos do POO; Linguagem de Modelagem Unificada para apresentar a simbologia UML dos conceitos de POO; Linguagem de Programação Java apresentando essa poderosa linguagem de programação orientada a objetos para exemplificar os padrões de projeto; e



Este livro é o resultado do uso da ferramenta a MS Project da Microsoft

utilizada na aplicação dos conceitos de gestão de projetos do PMBOK com as premissas da engenharia de testes para aquisição de qualidade nos produtos de software.

Padrões de Projetos que neste livro aborda os mais referenciados nas academias, sendo eles o GRASP e GoF.



Este livro aborda basicamente os conceitos básicos de programa

ção como autômatos, tipos de linguagens, princípios dos compiladores, paradigmas de desenvolvimento e lógica de programação.



Este livro introduz nas tecnologias Web abordando os conceitos

básicos para desenvolvimento para Internet com a apresentação da plataforma Dot Net e exibindo dicas de codificação para a linguagem de marcação ASPX, para a linguagem de script mais utilizada pelos navegadores o JavaScript com exemplos de CSS e principalmente dicas de código para a linguagem de programação CSharp e de banco de dados SQL com foco no SQLServer.