Edwar Saliba Júnior

Proposta de Práticas de Gestão do Conhecimento no Contexto de Processos de Desenvolvimento de Software.

Belo Horizonte
2006

Edwar Saliba Júnior

Proposta de Práticas de Gestão do Conhecimento no Contexto de Processos de Desenvolvimento de Software.

Monografia apresentada à Universidade FUMEC – FACE, como parte dos requisitos necessários para conclusão do Curso de Pós-Graduação em Gerência da Tecnologia da Informação, sob a orientação do Professor Rodrigo Baroni Carvalho.

Belo Horizonte
2006

A toda minha família, que sempre participou ativamente na conquista de meus objetivos.

AGRADECIMENTOS

Aos professores do curso de Gerência da Tecnologia da Informação, pela transmissão de conhecimento.
A todos os funcionários da FACE, que possibilitaram um ambiente saudável para o acontecimento do curso.
Agradecimento especial ao Sr. Edwar Saliba Moreira e Sra. Maria das Graças Silva Saliba, meus pais. Ao meu irmão Samir Silva Saliba e minha irmã Cibele Aparecida Silva Saliba. À minha namorada, Alessandra Aki Honda, ao seu filho Victor Honda Cristinni, ao seu irmão Fábio Hideki Honda e seus pais, Sr. Hiroshi Honda e Sra. Luisa Mitsue Honda. Pela paciência, carinho, compreensão e tudo mais que todos me proporcionaram neste, e em todos os períodos da minha vida em que estivemos juntos.

“Qualquer que seja sua forma, formato ou onde residem, as empresas estão cada vez mais percebendo que o recurso conhecimento se tornou a chave para estabelecer a sobrevivência, o auto-reforço e a vantagem competitiva.”

Dr. José Cláudio Cyrineu Terra

SUMÁRIO

Introdução 7
1 Metodologia de desenvolvimento de software 10
1.1 Visão Geral 10
1.2 Requisitos 15
1.2.1 Requisitos de Software 15
1.2.1.1 Estudo de Viabilidade 15
1.2.1.2 Requisitos - Definição 16
1.2.2 Processos de Engenharia de Requisitos 17
1.2.2.1 Levantamento e Análise de Requisitos 17
1.2.3 Modelos de Sistema 19
1.2.4 Prototipação de Software 21
1.2.4.1 Prototipação no Processo de Software 21
1.2.4.2 Prototipação de Interface com o Usuário 22
1.2.5 Especificação Formal 23
1.2.5.1 Especificação Formal no Processo de Software 24
1.3 Projeto 27
1.3.1 Projeto de Arquitetura 27
1.3.1.1 Estruturação do Sistema 28
1.3.1.2 Modelos de Controle 30
1.3.1.3 Decomposição em Módulos 31
1.3.2 Arquiteturas de Sistemas Distribuídos 33
1.3.3 Projeto Orientado a Objetos 35
1.3.4 Projeto de Software de Tempo Real 35
1.3.5 Projeto com Reuso 37
1.3.5.1 Desenvolvimento Baseado em Componentes 38
1.4 Sistemas Críticos 39
1.4.1 Confiança 39
1.4.1.1 Sistemas Críticos 39
1.4.2 Especificação de Sistemas Críticos 40
1.4.3 Desenvolvimento de Sistemas Críticos 42
1.4.3.1 Minimização de Defeitos 42
1.5 Verificação e Validação 44
1.5.1 Verificação e Validação 44
1.5.1.1 Planejamento de Verificação e Validação 45
1.5.2 Testes de Software 47
1.5.2.1 Testes para Detecção de Defeitos 47
1.5.3 Validação de Sistemas Críticos 48
1.6 Gerenciamento 49
1.7 Evolução 51
2 Análise do processo de desenvolvimento de software e mapeamento daqueles que contribuem para geração de conhecimento 55
2.1 Requisitos 56
2.2 Projeto 58
2.3 Sistemas Críticos 62
2.4 Verificação e Validação 63
2.5 Gerenciamento e Evolução 63
3 Apresentação Práticas para a Gestão do Conhecimento em Softwarehouses 65

3.1 Apresentação das Práticas Propostas para Melhoria na Produção de Software:.....66
3.1.1 Resumo das práticas propostas:.....67
3.1.2 Prática 1:.....68
3.1.3 Prática 2:.....69
3.1.4 Prática 3:.....69
3.1.5 Prática 4:.....71
3.1.6 Prática 5:.....71
3.1.7 Prática 6:.....72
3.1.8 Prática 7:.....73
3.1.9 Prática 8:.....74
3.1.10 Prática 9:.....75
3.1.11 Prática 10:.....76
3.1.12 Prática 11:.....77
3.1.13 Prática 12:.....78
3.1.14 Prática 13:.....79
3.1.15 Prática 14:.....80
3.1.16 Prática 15:.....81
3.1.17 Prática 16:.....81
3.1.18 Prática 17:.....82
3.1.19 Prática 18:.....83
3.1.20 Prática 19:.....84
3.1.21 Prática 20:.....85
3.2 Apresentação das Práticas de Gestão do Conhecimento no Contexto de Desenvolvimento de Software 86
3.2.1 Práticas Genéricas.....87
3.2.2 Requisitos 88
3.2.3 Projeto / Sistemas Críticos.....88
3.2.4 Desenvolvimento de Software 88
3.2.5 Verificação e Validação 89
3.2.6 Práticas Auxiliares 89
Conclusão 92
Bibliografia.....94

Introdução

Este estudo é uma proposta de práticas de gestão do conhecimento no contexto de processos de desenvolvimento de software.

Com o avanço da tecnologia, alavancada pelo aumento da capacidade intelectual do homem, o conhecimento se dissemina em grande quantidade e alta velocidade sobre o planeta.

> As demandas, cada vez maiores, por informações, sejam escritas, orais, audiovisuais ou em multimídia, manifestam-se tanto nos setores científicos e produtivos quanto junto ao grande público. Surgem as redes globais de telemática, cujo exemplo de maior repercussão é a internet, que abre perspectivas ainda imprevisíveis para a mais ampla disseminação de informações. As mudanças que se verificaram nesse campo, tanto de ordem tecnológica quanto social e econômica, propiciaram o surgimento da ciência da informação, que estuda a construção, comunicação e uso da informação.[1]

Esse bombardeio de informação que nos é apresentado através dos diversos meios de comunicação hoje existentes, proporcionar-nos-ia um verdadeiro caos se não pudesse ser tratado, organizado e gerenciado de forma a podermos escolher, obter e analisar, apenas o que for de nosso real interesse em momento adequado.

Em determinado período da história, o homem começou a armazenar dados, e os chamou de informação. Mais tarde, descobriu que a informação, assim denominada, está embutida em um conceito muito mais amplo do que o simples fato de tê-la em forma de dados armazenados organizadamente em um ambiente adequado.

Hoje, o homem sabe que, respeitados os devidos requisitos e restrições, os dados podem ser transformados em informação, e esta, pode ser transformada em conhecimento. Essa nova abordagem deu origem a diversas idéias, práticas e produtos que mudaram conceitos e criaram perspectivas e possibilidades.

Diariamente, pessoas e empresas travam lutas incessantes, na tentativa de trabalhar, gerenciar e reter o conhecimento que lhes é transmitido através das informações que lhes chegam ou são adquiridas, em atividades realizadas.

---

[1] LE COADIC, Yves-François. **A Ciência da Informação.** Brasília, 2003. Disponível em: <http://www.briquetdelemos.com.br/editora/biblio16.htm>. Acesso em: 28 ago. 2005.

O conhecimento tomou uma nova dimensão no âmbito empresarial, e hoje é tratado como parte dos “Ativos Intangíveis” por diversos autores e estudiosos do assunto. A capacidade de produção de conhecimento é tratada como moeda de negociação nas transações comerciais entre as empresas, destacando-se, em princípio, as que produzem tecnologia. Assim sendo, os “Ativos Intangíveis” tornaram-se um dos principais fatores de valorização das empresas no mercado mundial[2].

“Ativos intangíveis podem ser facilmente calculados pela diferença entre o valor contábil e o valor de mercado da empresa” (GUTHERIE apud Biblioteca TerraForum Consultores, 2005, p. 1).[3]

Este trabalho foi estruturado em três capítulos como descrito a seguir:

- Capítulo 1 – Neste capítulo é apresentado um resumo da metodologia de desenvolvimento de software proposta por Ian Sommerville em seu livro intitulado “Engenharia de Software”.

- Capítulo 2 – Nesta parte é feita uma análise dos processos utilizados no desenvolvimento de software apresentado no capítulo 1, onde procura-se mapear aqueles que contribuem significativamente para geração de conhecimento.

- Capítulo 3 – Neste último capítulo é feita a apresentação das práticas desenvolvidas para Gestão do Conhecimento em *softwarehouses*[4]. Também é apresentada a aplicação destas práticas na metodologia de desenvolvimento de software descrita no capítulo 1.

---

[2] Os métodos contábeis vigentes se baseiam, em boa medida, no método das partidas dobradas, que foi descrito por Frei Luca Pacioli em 1494. Estes métodos foram desenvolvidos com o intuito de se controlar e reportar os objetos e ativos tangíveis de uma organização (produtos, estoques, caixa). Esse modelo não foi elaborado para abordar o fator de produção de conhecimento e tratar da gestão do capital intelectual de uma organização. (ESTIMANDO O VALOR DE EMPRESAS: A importância e os desafios de mensuração dos ativos intangíveis, Biblioteca TerraForum Consultores. Disponível em: <http://www.terraforum.com.br/lib/pages/viewdoc.php?from=map&l_intDocCod=233> Acesso em: 15 set. 2005.)

[3] GUTHRIE, J.; PETTY. R.; JOHANSON, U.; **Sunrise in the Knowledge Economy:** anaging, measuring and reporting intellectual capital, [S.I.: s.n.], 2001 apud BIBLIOTECA TERRAFORUM CONSULTORES, 2005.

[4] Empresa produtora de softwares. (Nota do autor)

Nas considerações finais desta monografia, procura-se mostrar as melhorias advindas da gestão do conhecimento nas empresas, e ressaltar sua importância neste período do tempo, onde o conhecimento pode alcançar valores mais expressivos do que bens materiais.

A “Proposta de Práticas de Gestão do Conhecimento no Contexto de Processos de Desenvolvimento de Software”, vem abordar uma parte desta nova realidade vivida pelas empresas, com a intenção de melhorar a produtividade com qualidade, menor tempo e custo às empresas.

1 Metodologia de desenvolvimento de software

Este capítulo apresenta uma versão resumida da metodologia de desenvolvimento de software descrita por Sommerville (2003)[5]. Em seu livro, o autor explica a engenharia de software dividida em sete partes:

- Visão Geral
- Requisitos
- Projeto
- Sistemas Críticos
- Verificação e Validação
- Gerenciamento
- Evolução

Resumiremos os principais assuntos que compõem estas partes do livro e que fazem jus ao escopo deste trabalho, onde buscaremos identificar os pontos mais intensos em geração de conhecimento. Estudá-los-emos e faremos proposições de práticas de Gestão do Conhecimento para melhoria geral no processo de desenvolvimento como um todo.

1.1 Visão Geral

Nesta parte Sommerville (2003:3-78), esclarece que a Engenharia de Software é uma disciplina que se ocupa de todo o processo de desenvolvimento de software. Além disto, ele aborda superficialmente, em caráter unicamente explicativo, questões como: Engenharia de Sistemas com Base em Computadores, Processos de Software e Gerenciamento de Projetos.

Os assuntos citados são pré-requisitos de extrema importância para se entender e utilizar a Engenharia de Software. Estes devem estar bem moldados nas

[5] SOMMERVILLE, Ian; **Engenharia de Software**; Addison Wesley, São Paulo, SP; 2003;

mentes dos analistas que se embrenharem na arte de construir softwares com qualidade.

> A engenharia de sistemas é a atividade de especificar, projetar, implementar, validar, implantar e manter os sistemas como um todo. Os engenheiros de sistemas não se ocupam apenas com o software, mas com as interações de software, hardware, e sistemas com os usuários e seu ambiente. Eles devem pensar sobre os serviços que o sistema fornece, as restrições dentro das quais o sistema deve ser construído e operado e as interações do sistema com seu ambiente. Os engenheiros de software necessitam de uma compreensão sobre a engenharia de sistemas, porque os problemas de engenharia de software são, frequentemente, o resultado de decisões da engenharia de sistemas. Há muitas definições de um sistema, desde a muito abstrata até a mais concreta, mas uma definição útil seria: Um sistema é uma coleção significativa de componentes inter-relacionados, que trabalham em conjunto para atingir algum objetivo. (SOMMERVILLE, 2003, p. 18)

O resultado final de toda a aplicação dos conceitos de engenharia de software é o que chamamos de Produtos de Software, ou seja, pode ser um ou mais programas desenvolvidos com suas respectivas documentações, e que, em conjunto ou exclusivamente (para o caso de ser apenas um), formam o sistema ou o produto de software.

Um Produto de software, para ser denominado bom, deve seguir alguns pré-requisitos. São estes: Facilidade de uso e manutenção, boa aparência, segurança, confiança, dentre outros.

Para desenvolvermos cada um dos Produtos de Software, entramos em uma etapa denominada de Processo de Software;

> Um processo de software é um conjunto de atividades e resultados associados que levam à produção de um produto de software. Esse processo pode envolver o desenvolvimento desde o início, embora, cada vez mais, ocorra o caso de um software novo ser desenvolvido mediante a expansão e a modificação de sistemas já existentes. (SOMMERVILLE, 2003, p. 36)

Para a construção de softwares, são definidos Métodos. Os quais vão tentar reger a construção do software de forma organizada, definindo sugestões sobre processos a serem seguidos, notações, regras e as diretrizes do projeto.

Sommerville (2003:57) cita que como pontos-chave sobre os Processos de Software, modelos diversos, destacando: O modelo em cascata, o desenvolvimento evolucionário, o desenvolvimento formal de sistemas e o desenvolvimento orientado a reuso.

Neste contexto, ele afirma que os modelos iterativos apresentam o processo de software em ciclos de atividades. Tendo a vantagem de evitar compromissos prematuros como especificação ou projeto.

Ainda dentro de Processos de Software, podemos destacar a Engenharia de Requisitos, que consiste basicamente em apurar, levantar dados suficientes para o entendimento de como o sistema deverá funcionar, e o que ele deverá produzir. Uma Engenharia de Requisitos, ou, também conhecida como levantamento de requisitos, bem feita, é fundamental para o desenvolvimento de um bom trabalho com resultados satisfatórios.

Já a atividade de Projeto de Software se encarregará da geração de toda análise e documentação pertinente ao desenvolvimento do mesmo. Para auxílio dos engenheiros de software, existem no mercado, as chamadas ferramentas CASE, que são softwares projetados para facilitar os Processos de Software, ou seja, construção de diagramas como DER's (Diagrama de Entidade e Relacionamento)[6], Casos de Uso, Classes e também, geração de código em linguagem de programação que facilitará bastante no processo de Implementação.

A atividade de Implementação de Software, nada mais é do que a transcrição, de todos os dados coletados e transformados em documentos (gráficos ou escritos), para uma linguagem que o sistema operacional de um computador possa interpretar e executar.

E, implementado o sistema ou parte dele (variará de acordo com o modelo adotado), passa-se ao processo de validação, ou seja; é aqui que o sistema será testado e re-escrito, caso necessário, até que fique conforme as especificações de projeto.

Já em um passo mais avançado, passamos para a atividade de evolução do software, que acontece, muitas vezes, depois que o software já está funcionando em algum cliente. E é ali, principalmente nos primeiros dias ou semanas de uso, que o cliente descobrirá que se o software deixa a desejar ou não. Algumas vezes faltam recursos, alguns simples, outros mais complexos; processos que talvez foram esquecidos de ser relatados na etapa de levantamento de requisitos ou descartados por um motivo qualquer. Contudo, o principal objetivo da atividade de Evolução de

[6] DER – Diagrama de Entidade e Relacionamento. Neste diagrama visualiza-se o mapeamento das entidades (tabelas) e os relacionamentos existentes entre elas, em um determinado banco de dados, de um ou mais sistemas. Este documento é de caráter essencial no desenvolvimento de qualquer sistema que envolva armazenamento de dados em estrutura de banco de dados. (Nota do autor).

Software dentro da Engenharia de Software, é o aprimoramento e/ou criação de novos recursos necessários ao bom funcionamento do sistema ou a ampliação de sua versatilidade.

Como toda e qualquer engenharia, a de software também tem a necessidade de passar por um controle rigoroso e constante, no intuito de promover a realização do projeto, cumprindo-se as estimativas de prazos e custos e, atendendo aos requisitos do cliente.

> O fracasso de muitos grandes projetos de software, na década de 60 e no início da década de 70, foi a primeira indicação das dificuldades de gerenciamento de software. O software era entregue com atraso, não era confiável, custava várias vezes mais do que previam as estimativas originais e, muitas vezes, exibia características precárias de desempenho (BROOKS, 1975 apud SOMMERVILLE, 2003, p. 60). Esses projetos não fracassaram porque os gerentes ou os programadores eram incompetentes. Ao contrário, esses projetos grandes e desafiadores atraíam pessoas de capacitação acima da média. A falha residia na abordagem de gerenciamento utilizada. Técnicas de gerenciamento provenientes de outras disciplinas da engenharia eram aplicadas e mostravam-se ineficazes para o desenvolvimento de software. (SOMMERVILLE, 2003, p. 60)

A produção de software se difere de outros produtos por motivos óbvios. Sommerville (2003:60) explicita alguns destes motivos que tornam este tipo de produto tão singular e tão complexo para ser desenvolvido: O produto é intangível. Não há processo de software-padrão e, Grandes projetos de software são, freqüentemente, projetos únicos.

Estas, dentre outras diferenças na fabricação do produto software, dão uma singularidade especial ao seu desenvolvimento. Tornando-o complexo e desafiador.

O gerenciamento de projetos de software envolve controles rigorosos, principalmente de prazos, requisitos e custos.

“Requisitos, prazos e custos formam os vértices de um triângulo crítico. Aumento de requisitos levam a aumentos de prazos ou de custos, ou de ambos. Reduções de requisitos podem levar a reduções de prazos ou de custos (mas nem sempre).”[7]

[7] PAULA FILHO, Wilson de Pádua; **Engenharia de Software:** Fundamentos, Métodos e Padrões; LTC, Rio de Janeiro, RJ; 2001; p. 6.

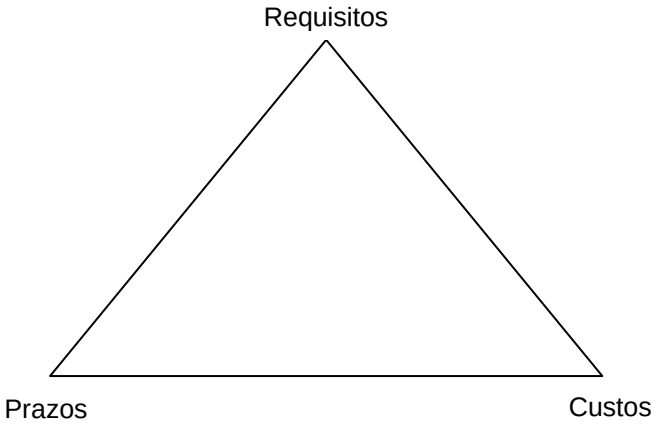

Fig. 1.0 – Um triângulo crítico da Engenharia de Software.
Fonte: Paula Filho (2001:7)

Através da visualização do Triângulo Crítico da Engenharia de Software, podemos verificar que não existe fórmula mágica para melhoria de processos (requisitos) sem alterarmos prazos e/ou custos.

"Para cumprir compromissos de prazos e custos, esses compromissos têm de ser assumidos com base em requisitos bem levantados, analisados e documentados. E os planos dos projetos têm de ser feitos com boas técnicas de estimativa e análise de tamanho, esforços, prazos e riscos." (PAULA FILHO, 2001, p. 7)

No gerenciamento de projetos, existem outras variáveis que precisam ser analisadas como: Restrições, Premissas, Riscos, Benefícios, Escopo, Orçamento e etc. Contudo, não nos aprofundaremos no mérito da questão. Essas outras variáveis foram citadas, apenas para que se tenha noção de que, o gerenciamento de projetos é algo bem mais complexo do que o que foi apresentado até aqui.

"O gerenciamento de projeto de software é um tópico amplo e não pode ser abordado em um único capítulo." (SOMMERVILLE, 2003, p. 60)

Segundo Sommerville (2003:59-78), faz-se necessário um bom gerenciamento de projeto de software para que se possa cumprir os prazos e custos previstos no projeto do sistema.

Sommerville (2003) destaca também que, diferentemente de qualquer outra engenharia, a de software se torna um pouco mais complexa devido a vários fatores. A começar pelo fato do software ser um produto intangível, seguido de que os projetos podem ser inovadores (isto sem levar em consideração que cada projeto de

software é diferente de outro), ou seja, não há como basear o gerenciamento em experiências anteriores.

No gerenciamento de software, pode e deve haver marcos que definem as partes do projeto, um exemplo disto seria a entrega de um módulo do sistema. Os marcos geralmente são pré-definidos em documentos formais e devem respeitar um prazo para serem concluídos. O avanço nas entregas dos marcos pré-definidos, dentro dos prazos também pré-definidos, sinaliza o bom andamento do projeto.

"A programação de projeto envolve a criação de várias representações gráficas de parte do plano de projeto. Isso inclui diagramas de atividades, que mostram o inter-relacionamento de atividades de projeto, e diagramas de barras, que mostram a duração das atividades". (SOMMERVILLE, 2003, p. 76).

Todo projeto está sujeito a riscos, e em todos eles os ricos devem ser tratados com especial atenção, principalmente aqueles que podem causar prejuízos irreversíveis ao projeto.

"A gerência pro-ativa de riscos significa que a equipe de projeto possui um processo visível, mensurável e repetível para o tratamento dos riscos. Este processo deve identificar, analisar e endereçar os riscos de forma pro-ativa."[8]

O pouco que foi dito, sobre gerência de projetos, até então é apenas um pequeno resumo de algumas atividades praticadas na gerência de projetos. Como foi dito anteriormente, este assunto não faz parte do escopo deste trabalho, assim sendo, encerraremos por aqui este tópico. Com ele, também a parte que, em seu livro, Sommerville (2003) chamou de "Visão Geral" da engenharia de Software. Deste ponto em diante, iniciaremos com o conteúdo direcionado ao escopo do projeto e, estaremos não só, detalhando os métodos utilizados na engenharia de software para o desenvolvimento de um produto, mas também, dando início as principais partes que serão as fontes de estudo do próximo capítulo deste trabalho.

1.2 Requisitos

1.2.1 Requisitos de Software

1.2.1.1 Estudo de Viabilidade

[8] GATTONI, Roberto Luís Capuruçu. *Gerência da Tecnologia da Informação*: Gerência de Projetos em Tecnologia da Informação. FUMEC; Belo Horizonte; MG; 2005; p. 58.

Antes de ser começado, todo novo projeto de sistema deve passar por um estudo de viabilidade.

> A entrada para o estudo de viabilidade é uma descrição geral do sistema e de como ele será utilizado dentro de uma organização. Os resultados do estudo de viabilidade deve ser um relatório que recomenda se vale a pena ou não realizar o processo de engenharia de requisitos e o processo de desenvolvimento de sistemas. (SOMMERVILLE, 2003, p. 103-104)

Algumas questões simples podem ajudar no estudo de viabilidade de um sistema de informação. Segundo Sommerville (2003:104), um estudo de viabilidade não deve ser um estudo demorado, deve ser simples e direcionado. Para tanto, sugere que sejam feitos os questionamentos a seguir:

> 1. O sistema contribui para os objetivos gerais da organização?
> 2. O sistema pode ser implementado com a utilização de tecnologia atual dentro das restrições de custo e de prazo?
> 3. O sistema pode ser integrado com outros sistemas já em operação? (SOMMERVILLE, 2003, p. 104)

As questões acima, são simples de serem respondidas. Podem parecer inúteis, mas são de caráter fundamental para que se construa um sistema que seja realmente necessário e viável para a organização.

A coleta e análise dos dados referentes às questões anteriores devem ser tratadas com muita responsabilidade pelos analistas que estiverem envolvidos no estudo de viabilidade.

1.2.1.2 Requisitos - Definição

Considerada uma das principais etapas na construção de um software com qualidade, a engenharia de requisitos deve ser uma tarefa bem formulada, bem direcionada e bem executada.

Segundo Sommerville (2003:82), a indústria de software não define com precisão o termo “requisito”, que pode ser entendido como uma declaração abstrata de alto nível, sobre algo, procedimento ou função que o sistema deverá executar; ou ainda, o requisito pode ser tomado como uma definição detalhada, matematicamente formal, de uma função do sistema.

Devido a esta falta de consistência, Sommerville (2003:82), faz as seguintes classificações:

- **Requisitos do usuário:** Que são expressos em linguagem natural ou através de diagramas. Estes tentam especificar funções e restrições ao sistema.

- **Requisito de Sistemas:** Este documento deve detalhar as funções e/ou restrições do sistema. Pode receber o nome de Especificação Funcional. Deve ser preciso, podendo servir como contrato entre o fornecedor do software e o cliente.

- **Especificação do projeto de software:** Descrição abstrata de um projeto de software, tomada como base para projeto e implementação mais detalhados.

"Os requisitos para um sistema de software estabelecem o que o sistema deve fazer e definem restrições sobre sua operação e implementação." (SOMMERVILLE, 2003, p. 99)

Conforme colocado anteriormente, a engenharia de requisitos é considerada uma das principais etapas para se construir um bom sistema. Como podemos ver nesta definição sobre requisitos (parágrafo anterior), estes são a base do que o sistema deve ser, ou seja, a alma do sistema. Podem-se variar nas formas como as tarefas, funções ou procedimentos serão feitas, mas o resultado final deverá ser aquele especificado no requisito correspondente àquela tarefa. Se assim não o for, então o sistema não estará funcionando corretamente, ou caso esteja, a especificação de requisitos provavelmente não foi bem feita ou não foi transcrita corretamente para o "papel".

1.2.2 Processos de Engenharia de Requisitos

1.2.2.1 Levantamento e Análise de Requisitos

Terminado o estudo de viabilidade e dando continuidade ao processo de construção do software, passamos para a fase de levantamento e análise de requisitos.

“Compreensão do domínio: Os analistas devem desenvolver sua compreensão do domínio da aplicação. Por exemplo, se for exigido um sistema para um supermercado, o analista deverá descobrir como operam os supermercados.” (SOMMERVILLE, 2003, p. 105)

A forma de adquirir estes requisitos é diversificada. Para se construir um sistema, o analista não precisa ter o conhecimento prévio da regra-de-negócio[9] deste, se tiver, melhor! Mas se não tiver, para isto servem os requisitos, ou seja, a partir da construção de funções ou subsistemas que atendam os requisitos, o sistema é gerado. Isto sem o analista ser um exímio conhecedor da regra-de-negócio do sistema.

Uma das principais formas de se obter os requisitos de um sistema é o diálogo direto com aquele que será o usuário do sistema.

Geralmente um sistema vem informatizar algumas ou muitas tarefas que ainda estão sendo feitas manualmente; ou, em alguns casos, substituir algum sistema antigo que já não atende mais as necessidades como atendia no início de sua utilização.

Na primeira hipótese, a de obter os requisitos diretamente com o futuro usuário do sistema, um analista se põe diante daquela pessoa que faz o trabalho braçal e, observa, questiona e anota tudo o que achar interessante e necessário sobre o processo que está sendo desenvolvido pela pessoa.

Em um segundo momento, acontece um diálogo mais apurado com esta pessoa para que ela possa validar o que foi anotado e entendido pelo analista. Nesta hora, a pessoa deverá colocar suas críticas e sugestões para a melhoria e/ou correção dos processos anotados.

Muitas vezes, ao entrevistarmos uma pessoa, que faz um determinado tipo de serviço, que será informatizado, esta poderá ter receio de passar todas as informações pertinentes e necessárias para o bom funcionamento do sistema. Isso acontece pelo fato das pessoas criarem em suas mentes o medo de “perderem o emprego para o computador” (neste caso o sistema que será construído).

Por isto, em entrevistas deste tipo, onde o analista observa, pergunta e depois valida os processos com o futuro usuário do sistema, deve-se ter muita habilidade para tentar transmitir à(s) pessoa(s) entrevistada(s) a segurança de que o sistema

[9] Uma regra de negócio é uma declaração que controla ou define alguns aspectos de um negócio. (Nota do autor)

não será construído para tirar o emprego de ninguém. E, caso a resistência em passar informações, da pessoa entrevistada seja muita, dever-se-á tentar, de maneira sutil obter as informações necessárias. Se ainda assim não for possível, o jeito é substituir a pessoa entrevistada por outra que também tenha conhecimento do processo, saiba e queira transmiti-lo.

E é dessa forma que se adquirem os Requisitos de Usuário.

Uma outra coisa que pode ajudar muito é o conhecimento prévio sobre o assunto, por parte do analista que fará o levantamento de requisitos, pois, assim ele poderá se interagir muito melhor com as pessoas entrevistadas e, transcreverá com mais clareza os requisitos para serem implementados.

“Os requisitos de usuário se destinam às pessoas envolvidas no uso e na aquisição do sistema. Eles devem ser escritos utilizando-se linguagem natural, tabelas e diagramas, de modo que sejam compreensíveis.” (SOMMERVILLE, 2003, 99).

Por outro lado, em se tratando de Requisitos de Sistemas, esses não são “bichos de sete cabeças”, pois são especificados pelos próprios analistas, que a partir de um refinamento nos requisitos de usuários, geram os de sistema. Neste caso, muitas vezes já aplicando um pouco de metodologia para descrição dos requisitos, como: Diagramas de Casos de Usos, de Fluxos de Dados, Linguagens Estruturadas e etc.

> Os requisitos de sistema se destinam a comunicar, de modo preciso, as funções que o sistema tem de fornecer. Para reduzir a ambigüidade, eles podem ser escritos em uma linguagem estruturada de algum tipo, que pode ser um formulário estruturado de linguagem natural, uma linguagem com base em uma linguagem de programação de alto nível ou uma linguagem especial para a especificação de requisitos. (SOMMERVILLE, 2003, p. 99)

Feito o levantamento de requisitos, partimos então para os Modelos de Sistema. Estes devem ser criados para melhorar a compreensão técnica sobre o sistema.

1.2.3 Modelos de Sistema

Os modelos são representações gráficas, dos requisitos que foram colhidos e documentados em uma linguagem natural, ou seja, é a transcrição do que foi escrito

em linguagem natural, para uma outra linguagem representada por símbolos (desenhos, losângulos, retângulos e etc.). Para isto, podemos usar qualquer metodologia já conhecida ou aquela que nos for mais conveniente. Contudo, recomendamos as metodologias reconhecidas pelo mercado, pois estas, se estão sendo utilizadas pelas empresas, é porque já foram testadas e aprovadas. Uma metodologia que está sendo muito utilizada e que está se tornando padrão de mercado é a UML[10].

Antes da criação de qualquer sistema, a construção de modelos é extremamente necessária para aumentar a compreensão do sistema, reduzindo-se drasticamente a representação de detalhes, coletadas no levantamento de requisitos e, que não são expressas nos símbolos utilizados nos modelos ou diagramas.

Como exemplos de modelos que podem ser criados, Sommerville (2003:124-143), cita e explica os seguintes: Modelos de Contexto, Modelos de Comportamento, Modelos de Dados e Modelos de Objeto.

Dentre os modelos citados por Sommerville, podemos destacar o Modelo de Dados e o Modelo de Objetos, como os mais populares nos desenvolvimentos de projetos de arquitetura.

Uma das grandes vantagens na adoção de Modelos de Sistemas é no sentido de que, estes modelos ajudam a diminuir ou eliminar completamente as ambigüidades descritas nos requisitos, devido às limitações da linguagem natural. Uma outra grande vantagem é a facilitação da compreensão do que o sistema deve fazer, pelo técnico (analista de sistemas ou programador), afinal de contas, como já foi dito anteriormente, os diagramas reduzem os níveis de detalhes, proporcionando uma visão Top-Down[11] do sistema.

A partir do término de alguns modelos de sistema, estes poderão ser passados para o programador para que ele os codifique, ou seja, escreva o sistema. Se analisarmos com atenção, perceberemos que está tudo encadeado, ou seja, se não fizermos um levantamento de requisitos muito bem feito, automaticamente a construção dos modelos de sistemas não serão bem feitas, pois, estes modelos se

[10] UML – Unified Modeling Language, do inglês: Linguagem de Modelagem Unificada. Esta metodologia surgiu da união de outras três metodologias distintas, de três autores renomados no mercado da informática. Esta metodologia está se tornando padrão de mercado, sendo extremamente utilizada na especificação de sistemas informacionais. (Nota do autor)

[11] Top-Down – Do inglês: De cima para baixo, ou seja, no contexto, uma visão de alto nível do sistema. Desprezando-se os detalhes e minúcias sem importância para uma compreensão geral do sistema como um todo. (Nota do autor)

baseiam nos requisitos coletados e, para piorar a situação, não teremos nenhuma das outras etapas que virão, bem feitas e; tampouco um software bem feito, pois, este é escrito baseado nos diagramas gerados a partir dos requisitos que foram especificados lá no início do processo.

Até aqui podemos notar que, quanto mais tarde no processo de desenvolvimento de software, descrobirmos um erro; mais caro e demorada será sua correção. Portanto, os trabalhos realizados deverão ser feitos sempre com o intuito de minimizar ao máximo, a possibilidades de erros, falhas ou divergências nos materiais gerados.

1.2.4 Prototipação de Software

Para auxiliar nesta etapa do desenvolvimento, existe ainda um recurso chamado de “Prototipação de Software”. Este pode ser considerado um dos recursos mais realistas e “palpáveis” que o cliente tem. Pois é através da utilização deste recurso que o cliente começa a ver os primeiros sinais de vida, do software que esta comprando.

1.2.4.1 Prototipação no Processo de Software

Em se tratando de prototipação de software, existem duas abordagens que podem ser utilizadas. Uma delas é completamente descartável, ou seja, você constrói um mini-software para ser apresentado ao cliente, e depois de validado pelo mesmo, este é “jogado no lixo”, ou melhor, apagado. Por outro lado, a outra abordagem, constrói o mini-software e vai evoluindo a partir do que foi construído.

A essas duas abordagens, Sommerville (2003:147) deu os nomes de Prototipação Descartável e Prototipação Evolucionária, respectivamente.

> - O objetivo da prototipação evolucionária é fornecer um sistema funcional aos usuários finais. Isso significa que, normalmente, o processo se inicia com os requisitos do usuário que são mais bem compreendidos e com os que têm maior prioridade. Os requisitos de menor prioridade e os mais vagos são implementados quando e se eles forem solicitados pelos usuários. (SOMMERVILLE, 2003, p. 147)
>
> – O objetivo da prototipação descartável é validar ou derivar os requisitos do sistema. É preciso começar com aqueles requisitos que

> não são bem compreendidos, porque é preciso saber mais sobre eles. Os requisitos que são diretos podem nunca ser prototipados. (SOMMERVILLE, 2003, p. 147)

A prototipação, tanto a Evolucionária quanto a Descartável, são recursos que podem trazer grandes benefícios ao projeto como um todo. Como foi escrito anteriormente, quanto mais tarde descobrirmos um erro de requisito, mais caro será sua correção.

Ao interagir com o protótipo do sistema, o usuário vai colocar sua opinião com mais clareza e talvez até mudar algum requisito radicalmente. O usuário poderá também descrever outras formas para as telas, de modo que facilite seu serviço, poderá ainda sugerir mais coisas, que poderão vir a melhorar muito o sistema, e que talvez não tenham sido pensadas anteriormente.

1.2.4.2 Prototipação de Interface com o Usuário

Sommerville (2003:159), fala sobre Prototipação de Interface com o usuário, aqui ele coloca que as interfaces gráficas se tornaram padrão nos sistemas que são construídos atualmente. Afirma também que estas interfaces representam um percentual significativo em relação ao custo total do sistema.

A respeito deste contexto, podemos afirmar que, não só pelo percentual significativo que ele representa, mas também, por ser a parte do sistema que estará interagindo diretamente com o usuário do sistema, ao contrário do que muitos pensam, deve ser uma das tarefas a serem mais bem elaboradas, trabalhadas e revisadas. De preferência, após já estarem semi-acabadas, com o usuário interagindo em sua construção.

A opinião de um usuário é muito importante em qualquer fase do sistema, mas especialmente nesta, pois, poderá evitar dores-de-cabeça futuras para os analistas/programadores envolvidos na construção do sistema.

Algo que deve estar claro é: Nem sempre o usuário sabe exatamente o que ele quer; sendo este, um bom motivo para ele não estar presente em todas as etapas de construção do sistema, contudo, suas opiniões não devem ser descartadas, pois, equivocadamente, as pessoas responsáveis pelo levantamento de requisitos podem ter entendido mal algum requisito ou alguma outra parte deste processo.

Quando bem utilizado, não só este, mas quaisquer outros recursos da engenharia de requisitos poderão proporcionar altos ganhos em relação a custos e tempo de desenvolvimento. Damos um destaque maior a esta parte do processo, pois, é aqui o primeiro contato do usuário com o sistema.

Devemos ter em mente que até então, tudo estava muito abstrato para aquelas pessoas que estão fora do processo de desenvolvimento do software, e que neste momento do processo, para estas pessoas, é como se o sistema estivesse começando a ser construído.

Após a prototipação, as dúvidas sanadas e o cliente de acordo com o que está sendo construído, então, passa-se para o processo de Especificação Formal.

1.2.5 Especificação Formal

> Na década de 80, a especificação formal e os métodos formais mais gerais eram vistos por muitos pesquisadores como o caminho mais provável para importantes melhorias na qualidade de software. Eles argumentavam que a análise detalhada e rigorosa, que é uma parte essencial dos métodos formais, levaria a programas com menos erros e que fossem mais adequados às necessidades dos usuários. Além disso, eles previram que, no século XXI, grande parte dos produtos de software seria desenvolvida com a utilização dos métodos formais. É claro que essa previsão não foi cumprida e existe uma série de razões para isso. (SOMMERVILLE, 2003, p. 163)

A Especificação Formal é a última parte da etapa de Engenharia de Requisitos. É neste momento do processo de construção do software, que se tem a oportunidade de diminuir drasticamente possíveis erros que passaram despercebidos aos olhos dos analistas/especificadores, ou seja, isto não quer dizer que o sistema será concebido sem erros, porém, será gerado um sistema muito mais confiável e com menos erros.

A especificação formal, apesar de elevar o custo nas fases iniciais do desenvolvimento do software, com certeza, reduz bastante o número de erros nas fases finais, o que facilitará a vida dos programadores/analistas envolvidos na concepção do software e obviamente reduzirá o custo de produção deste em suas fases finais.

Através da especificação formal, pode-se refinar todo o conhecimento adquirido no processo de desenvolvido até agora, e ao analista, além de entender

melhor o sistema, poderá propor mudanças satisfatórias em partes ou no projeto como um todo.

A especificação formal pode ser feita através de várias linguagens existentes no mercado, dentre elas: Linguagens Orientadas ao Modelo e Linguagens Orientadas a Propriedades.

1.2.5.1 Especificação Formal no Processo de Software

> Existem várias linguagens de implementação de métodos formais. [...] Esta divisão tem em conta as características das linguagens, nomeadamente relacionadas com os fundamentos científicos em que se apóia a especificação. Como característica comum, salienta-se que todas as linguagens são constituídas por: uma notação, que especifica o domínio sintático; um universo de objetos, que se referem ao domínio semântico e uma regra que indica quais os objetivos que satisfazem cada especificação.[12]

Segundo Figueiredo et al. (2002:5), as linguagens de especificação formal se dividem conforme a figura 1.0:

Ainda segundo Figueiredo et al. (2002:5-7), as linguagens de especificação formal mais utilizadas atualmente são: VDM e Z.

Os diagramas e/ou modelos gerados na especificação formal, são pouco claros para pessoas que não possuem o conhecimento técnico no assunto. Por outro lado, para os profissionais da área, são como uma bússola que os norteia pelos caminhos mais rápidos e seguros para alcançarem seus objetivos.

[12] FIGUEIREDO, Carlos et al. **Especificação Formal de Software.** Disponível em: <http://paginas.fe.up.pt/~ei99030/trabalhos/EspecificacaoFormaldeSoftware.pdf>. Acesso em: 25 fev. 2006.

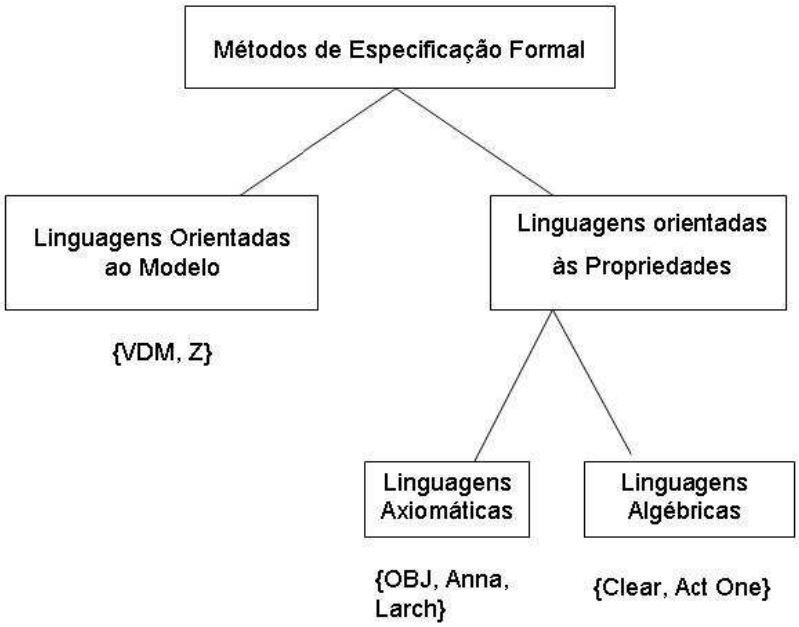

Fig. 1.0 – Divisão dos métodos de Especificação Formal
Fonte: Figueiredo et al. (2002:5)

> À medida que a especificação é desenvolvida detalhadamente, aumenta a compreensão do especificador sobre essa especificação. Criar uma especificação formal força uma análise detalhada dos sistemas, o que geralmente revela erros e inconsistências na especificação informal de requisitos. Essa detecção de erros é o argumento mais forte para o desenvolvimento de uma especificação formal (HALL apud SOMMERVILLE, 1990). De modo geral, é muito dispendioso corrigir os problemas com requisitos que são detectados nos estágios mais avançados do processo de software. (SOMMERVILLE, 2003, p. 164-165).

Conforme escrito no parágrafo imediatamente anterior, um erro descoberto em momentos avançados do processo de desenvolvimento de software, com certeza terá um custo muito mais alto, para sua correção, do que um erro descoberto ainda nas fases iniciais.

Os principais objetivos da Especificação Formal, na engenharia de software, são: reduzir ao máximo o número de erros e ambigüidades e aumentar a compreensão do sistema (por parte dos analistas).

Através do aumento da compreensão do que se está fazendo, e do uso de uma linguagem matemática para especificação formal dos requisitos, diversos erros que eventualmente poderiam existir nesta etapa, serão detectados e corrigidos, pois, aqui é o melhor momento para detecção e correção destes. Isto, levando-se em consideração o custo do re-trabalho.

Ainda assim, a adoção de um método de Especificação Formal, é extremamente cara e não se tem um resultado imediato, de redução de custos, em sua utilização. Os reflexos de seu uso só serão percebidos no término das fases finais de desenvolvimento do sistema.

“Devido ao conservadorismo existente entre os gestores de software, existe alguma relutância em adotar novas técnicas quando o lucro não é óbvio. É difícil demonstrar que o elevado custo de desenvolvimento da especificação formal, na fase inicial, irá tornar o preço final do projeto mais baixo”. (FIGUEIREDO et al., 2002, p. 3)

Desta forma, ao invés de trabalharem pró-ativamente na detecção e correção de erros, os gestores de software, contrariando as boas normas, preferem agir reativamente neste sentido.

> Desenvolver e analisar uma especificação formal sobrecarrega os custos do desenvolvimento de software. [...] Quando um processo convencional é utilizado, os custos de validação são aproximadamente 50 por cento dos custos de desenvolvimento e os custos de implementação e projeto, duas vezes maiores do que os custos de especificação. Com a especificação formal, os custos de especificação e implementação são equivalentes e os custos de validação de sistemas são significativamente reduzidos. Como o desenvolvimento de especificação formal descobre problemas com os requisitos, evita-se retrabalho necessário para corrigir esses problemas, depois que o sistema foi projetado. (SOMMERVILLE, 2003, p. 165)

Como pode ser observado, o imediatismo na procura da redução de custos de produção, pode causar perdas desnecessárias no desenvolvimento de software. E o que se vê de mais grave nisto, é que geralmente os gestores de softwares, que são em princípio, um dos lados mais interessados na melhoria da qualidade de seus produtos, ignoram a existência deste recurso simples.

E posteriormente, estes mesmos gestores que economizaram nas fases iniciais, gastarão rios de dinheiro com investimentos em ferramentas, equipamentos e pessoas especializadas para testar o sistema que está sendo gerado. Este sistema que, além de gerar muito retrabalho, para correção dos erros encontrados, será corrigido da maneira mais cara, pois, o sistema já se encontrará em suas etapas finais.

1.3 Projeto

1.3.1 Projeto de Arquitetura

O projeto pode ser considerado uma das etapas mais complexas do processo de desenvolvimento como um todo.

> Os grandes sistemas são sempre decompostos em subsistemas, que fornecem algum conjunto de serviços relacionados. O processo inicial de projeto, que consiste em identificar esses subsistemas e estabelecer um framework[13] para o controle e a comunicação de subsistemas, é chamado de projeto de arquitetura e a saída desse processo de projeto é uma descrição da arquitetura do software. (SOMMERVILLE, 2003, p. 182)

A identificação de subsistemas dentro de um sistema pode vir a ser uma tarefa um tanto quanto difícil, pois, muitas vezes a fronteira entre um e outro subsistema não está bem definida, o que leva ao analista responsável a ter que usar, ao invés da lógica, o bom senso na divisão dos subsistemas.

Segundo Sommerville (2003:182), o processo de Projeto de Arquitetura, além de estabelecer uma framework, também é responsável pela identificação dos principais componentes do sistema e a comunicação destes entre si.

O sentido desta colocação é obvio, pois, a partir do momento que o analista se torna o responsável pela criação da estrutura do sistema, este automaticamente assume também a responsabilidade pela coordenação de como será o fluxo e transformação de dados dentro do sistema. Afinal, a estrutura ou o projeto de arquitetura será totalmente influenciado por estes itens em sua concepção.

Segundo Sommerville (2003:182), apesar do modo subjetivo de cada projetista enfocar o processo de projeto de arquitetura, ainda assim, as seguintes atividades são comuns a todos:

1. Estruturação de Sistema: Onde é feita a divisão do sistema em subsistemas e, cada qual é uma unidade independente com comunicação entre si.

[13] Framework – Do inglês: Estrutura. Na informática, mais especificamente na área de desenvolvimento de sistemas, esta palavra define exatamente uma estrutura de componentes (partes de códigos ou formulários), construída com o objetivo de uso e re-uso em partes ou em todo o sistema a ser criado. A framework tem o objetivo de facilitar a programação (construção) de sistemas, visto que uma vez construído, um componente, objeto, função ou procedimento poderá ser usado em uma ou mais partes do sistema (onde for necessário) sem a necessidade de ser reescrito pelo programador/analista de sistemas. (Nota do autor).

2. Modelagem de Controle: Aqui se estabelece um modelo geral dos relacionamentos de controle entre os subsistemas.
3. Decomposição modular: Onde os subsistemas são divididos em módulos, para facilitar a criação do subsistema.

Na tentativa de se criar uma padronização de projetos de software, da mesma forma que é feita, por exemplo, na Engenharia Civil; diversos estudiosos da área publicaram metodologias para análise e documentação de sistemas. Dentre as diversas existentes no mercado, uma das mais bem aceitas foi a UML, que está se tornando um padrão para análise e documentação de sistemas no mundo inteiro.

1.3.1.1 Estruturação do Sistema

A Estruturação do Sistema, de certo modo, segue alguns tipos de padrões pré-definidos ou um conjunto destes. Dentre esses padrões, podemos destacar os seguintes:

- O Modelo de Repositório, que conforme Sommerville (2003:185) são aqueles onde se tem um banco de dados central, acessado por diversos sistemas ou subsistemas. Ou ainda, pode ser que cada subsistema mantenha seu próprio Banco de Dados. E o intercâmbio de informação entre estes subsistemas é feito através de mensagens[14].

- O Modelo Cliente-servidor, que conforme Sommerville (2003:187) tem uma composição formada por diversos processadores que solicitam serviços/informações a um ou mais servidores, através de uma rede.

- O Modelo de Máquina Abstrata, ainda conforme Sommerville (2003:188) é organizado em camadas onde, cada camada define uma máquina abstrata cuja linguagem de máquina ou serviços fornecidos pela camada, é utilizada para implementação de um próximo nível.

[14] Mensagem – No contexto de linguagem de programação, pode ser definida como um serviço solicitado por um sistema a outro, no qual obtem-se uma resposta, que é retornada ao sistema que solicitou o serviço. (Nota do autor)

Das estruturas descritas anteriormente, o que se nota é uma mistura destas para composição dos sistemas informacionais modernos. Como forma de exemplificar o que queremos dizer, podemos citar a linguagem de programação Java da Sun Microsystems. Esta é uma linguagem que usa uma máquina abstrata instalada no sistema operacional do computador cliente. Completando a estrutura para suportar tais sistemas, gerados com esta linguagem de programação, que são em grande parte, para utilização em redes. Associa-se a esta estrutura de máquina abstrata, uma mistura das estruturas de cliente servidor e de repositório.

Já há algum tempo, as tecnologias de programação evoluíram da estrutura Cliente-servidor, para a estrutura conhecida como Três Camadas. Essa estrutura de três camadas pode ser representada conforme figura 2.0:

Na figura 2.0, podemos ver claramente uma rede de computadores clientes. Estes processam apenas informações visuais e solicitações de serviços. As solicitações de serviços são enviadas ao servidor de aplicação. O servidor de aplicação conhece a regra de negócio do sistema, processa os serviços solicitados pelas máquinas clientes e se precisar de alguma informação, solicita esta ao servidor de banco de dados. O servidor de banco de dados responde por todas as transações envolvendo armazenamento e manutenção de dados; podendo também, em alguns casos, assumir o processamento de dados através do recurso conhecido como “Procedimento Armazenado" (Stored Procedure).

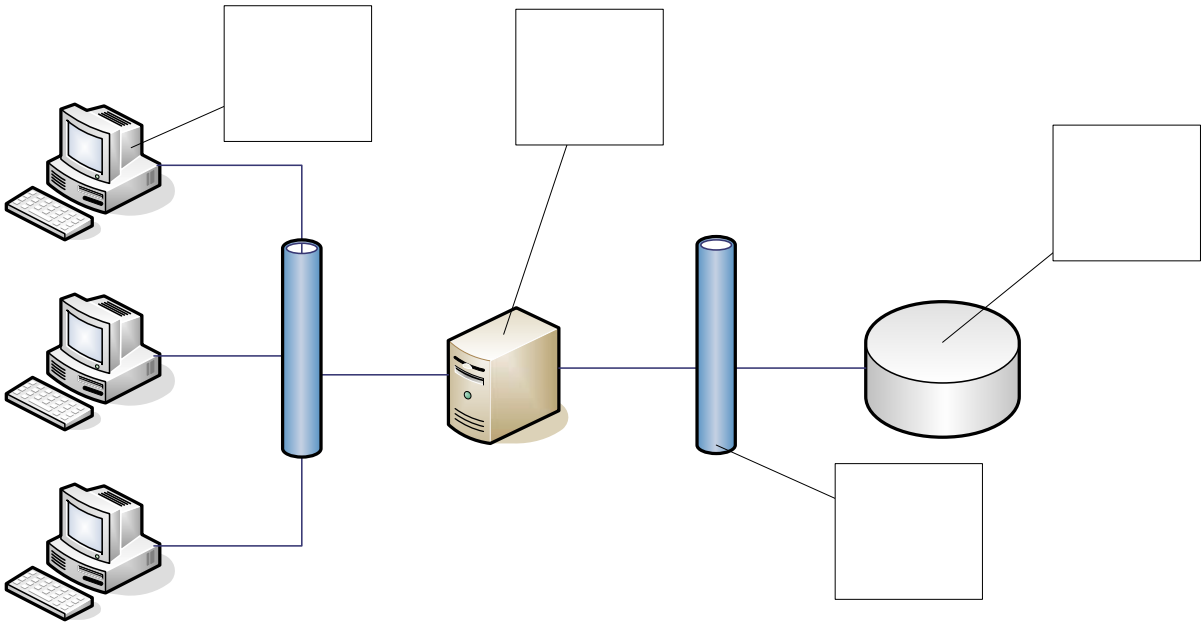

Fig. 2.0 – Representação de Estrutura de Três Camadas (Figura do Autor)

Além da estrutura de três camadas, exposta anteriormente, as linguagens de programação mais modernas (Ex: Java da Sun Microsystems e .Net da Microsoft) ainda utilizam o recurso da máquina abstrata que, como explicado anteriormente, é uma camada do sistema que fica instalada no sistema operacional das máquinas clientes.

A utilização desta mistura de estruturas proporcionou melhorias incontestáveis nas aplicações (sistemas) utilizados em grandes redes de computadores, sendo a principal delas a internet.

1.3.1.2 Modelos de Controle

Em seu livro, Sommerville (2003:189) cita dois possíveis modelos de controle. Estes são: Controle Centralizado e Controle Baseado em Eventos. Ele explica que em um sistema de Controle Centralizado, um subsistema é criado para controlar os demais. Esse controle se dá a partir de chamadas a sub-rotinas que também chamam sub-rotinas.

> Nos modelos de controle centralizado, as decisões de controle são normalmente determinadas pelos valores de algumas variáveis de estado do sistema. Por outro lado, os modelos de controle orientados a eventos são baseados em eventos gerados externamente. O termo evento, nesse contexto, não significa somente um sinal binário; ele pode ser um sinal capaz de assumir uma gama de valores. A distinção entre um evento e uma entrada simples é que a ocorrência do evento está fora do controle do processo que manipula esse evento. (SOMMERVILLE, 2003, p. 191)

O Modelo de Controle Baseado em Eventos revolucionou, juntamente com outras tecnologias, a maneira de construção dos sistemas informacionais.

A grande maioria dos sistemas modernos, construídos com linguagens modernas, é baseada em eventos. Quem controla o acontecimento dos eventos é o sistema operacional. O software criado para trabalhar neste sistema operacional, fica constantemente observando as ocorrências dos eventos e respondendo aos seus estímulos, de acordo com a programação do sistema.

Uma grande mudança de paradigma, que houve na Programação Orientada a Eventos, é que a execução do software deixou de seguir um fluxo seqüencial; um “lingüição”, como muitos programadores costumavam chamar (referindo-se aos

códigos-fonte gerados por linguagens de programação não orientadas a eventos), passando a seguir um fluxo determinado por eventos onde, qualquer parte do sistema pode ser executada a qualquer tempo, dependendo somente da ocorrência do evento a qual a mesma esteja subordinada.

1.3.1.3 Decomposição em Módulos

Outro fator importantíssimo no projeto de arquitetura é a Decomposição em Módulos ou modularização do sistema.

“Depois que uma arquitetura estrutural foi projetada, o próximo estágio do processo de projeto de arquitetura é a decomposição em módulos.” (SOMMERVILLE, 2003, p. 193)

Alguns dos principais motivos para utilização da decomposição do sistema em módulos são:

- Facilidade de programação da rotina.

- Facilidade de compreensão da rotina.

- Facilidade de depuração de possíveis erros que, por ventura estejam na rotina.

- A possibilidade de reutilização da rotina por outras rotinas deste ou de outros módulos do sistema.

Sommerville (2003:193) considera duas formas de decomposição de sistemas em módulos. Estas são: O Modelo Orientado a Objetos e o Modelo de Fluxo de Dados.

No primeiro modelo, Orientado a Objetos, o sistema é dividido em classes, que representam partes específicas do sistema, “assuntos”.

Este é um dos modelos mais utilizados na programação de sistemas. O benefício de sua utilização pode ser observado numa forma bem transparente de se

programar e separar as rotinas do sistema de acordo com seus “assuntos” específicos.

Existe também, neste modelo, uma melhora considerável no tempo de depuração de erros, pois, se estivermos procurando um erro referente, por exemplo, ao cadastro de um funcionário num sistema de recursos humanos; Verificaremos diretamente no objeto referente ao funcionário, pois, todos os dados sobre funcionário são processados unicamente no objeto de funcionários.

Por estes e alguns outros motivos, nas ferramentas de programação mais modernas, a decomposição em módulos através da construção de objetos, é praticamente obrigatória, sua utilização, pelos programadores.

Já o segundo modelo apontado por Sommerville, Modelo de Fluxo de Dados. Este já é mais antigo. Segue a mesma estrutura da chamada “Programação Estruturada”. Onde o sistema é decomposto em diversos módulos (funções, procedimentos ou simplesmente rotinas) que recebem alguns dados de entrada (também chamados de parâmetros), processa-os e retornam, de alguma maneira, os dados de saída.

Neste modelo, não há uma divisão do sistema por “assunto”, como no modelo anterior. Isto pode tornar a programação e a depuração de erros, muito mais complicadas, pois, os dados podem percorrer diversas rotinas de diversos módulos até chegar ao seu destino final.

Os analistas e programadores têm que ter em mente, que não existe um tipo de arquitetura que seja a ideal para todos os tipos de sistemas já inventados. Sempre haverá situações onde dever-se-á ponderar sobre a utilização de uma arquitetura ou outra, de uma tecnologia ou outra. Portanto, apesar de termos mostrado, falado e algumas vezes até destacado algumas arquiteturas e/ou tecnologias que estão sendo usadas pelas empresas; em hipótese alguma o leitor deverá pensar que julgamos estas como sendo as melhores para serem aplicadas aos sistemas que serão criados. Muito antes pelo contrário, sabe-se, cada caso é um caso, e deve ser analisado por pessoas competentes e aptas a darem uma opinião concreta sobre o que se deverá fazer e qual tecnologia usar.

Este trabalho, não tem como objetivo explicar nenhum método de desenvolvimento de software, portanto, para finalizar a parte de projeto, falaremos um pouco sobre alguns conceitos que são muito utilizados nos sistemas modernos e

que são passíveis de terem um lugar neste trabalho. Não nos aprofundaremos, serão apenas conceitos superficiais sobre as tecnologias que serão apresentadas.

1.3.2 Arquiteturas de Sistemas Distribuídos

O conceito de Arquitetura de Sistemas Distribuídos é extremamente utilizado na construção de sistemas informacionais modernos. Com a evolução dos sistemas de rede, a maneira de se construir softwares mudou completamente.

Em um futuro breve, o sujeito ao sair do trabalho, ver-se-á obrigado a passar no supermercado para comprar leite ou algum outro produto, devido à mensagem que recebeu em seu celular, enviada por sua geladeira. Isto se a geladeira não enviar a mensagem diretamente para o supermercado, que providenciará a entrega do leite, e debitará o valor da compra direto no cartão de crédito do sujeito.

Situações como esta, ainda hipotéticas ou futuristas, não estão longe de se tornarem realidade no dia-a-dia de muitos cidadãos. À medida que se desenvolve a tecnologia, a comunicação entre as máquinas é cada vez mais real. Principalmente com o aprimoramento das redes sem fio.

“Um sistema distribuído é aquele em que as informações em fase de processamento são distribuídas para vários computadores, em vez de ficarem confinadas a uma única máquina.” (SOMMERVILLE, 2003, p. 203)

Em se tratando de sistemas, conforme Sommerville (2003:203), existem três tipos distintos:

1. Sistemas Pessoais: Não são distribuídos, ou seja, são projetados para serem executados em computadores pessoais.
2. Sistemas Embutidos: Que podem ser executados em máquinas com um ou vários processadores, no entanto, servem especificamente para um determinado equipamento.
3. Sistemas Distribuídos: São softwares que podem ser executados em diversos processadores de diversas máquinas ao mesmo tempo. Ou seja, compartilhando recursos e/ou partes do software.

Dentre as diversas arquiteturas para sistemas distribuídos, podemos destacar as seguintes:

1. Arquiteturas de Multiprocessadores: Sistemas que são processados por mais de um processador ao mesmo tempo. Estes podem ou não estar em uma mesma máquina. Este tipo de sistema é um pouco mais complexo de ser construído, pois, nem toda parte do código poderá ser dividida em vários processamentos em paralelo. Para isto, em sistemas desta natureza, são colocados mecanismos que providenciam o processamento paralelo nas partes que são passíveis de tal.
2. Arquitetura Cliente-servidor: Esta arquitetura, já foi mencionada anteriormente, ela consiste em serviços que são fornecidos por um ou mais servidores às diversas máquinas que solicitam estes serviços (clientes). Foi mencionada também, a evolução da arquitetura Cliente-servidor, para a arquitetura conhecida como Três Camadas (Fig. 2.0), atualmente muito utilizada em aplicações de grandes redes de computadores, principalmente aplicações para internet.
3. Arquiteturas de Objetos Distribuídos: Uma das maneiras mais simples de se entender esta arquitetura é a seguinte: Num sistema cliente-servidor, os clientes solicitam serviços aos servidores. O oposto nunca é verdadeiro. Já na Arquitetura de Objetos Distribuídos, as máquinas que são clientes, também são servidoras. Ou seja, uma máquina servidora solicita serviços a uma máquina cliente, que por sua vez é servidora de outros serviços.

Uma desvantagem dos sistemas distribuídos, que deve ser mencionada é: Estes sistemas são bem mais complexos de serem construídos do que os sistemas cliente-servidor. Isto sem levar em consideração o fator vulnerabilidade, pois, dependendo de como estiver estruturada sua rede e a distribuição de serviços entre os servidores, o sistema poderá parar por completo na falta da rede.

A arquitetura a ser utilizada, variará para cada tipo e característica de aplicação a ser construída. Para tanto, voltamos a mencionar a necessidade de pessoas experientes e aptas para decidirem qual arquitetura dever-se-á utilizar.

1.3.3 Projeto Orientado a Objetos

> Um objeto é uma entidade que possui um estado e um conjunto definido de operações que operam nesse estado. O estado é representado por um conjunto de atributos de objeto. As operações associadas com o objeto fornecem serviços para outros objetos (clientes), que requisitam esses serviços quando alguma computação é necessária. (SOMMERVILLE, 2003, p. 222)

Os Projetos Orientados a Objetos (POO) podem ser considerados como a segunda parte da construção de um software orientado a objetos. Ao optar por desenvolver um POO, os responsáveis devem ter em mente que, toda a estrutura de construção do software deverá ser Orientada a Objetos.

Segundo Sommerville (2003:221), o POO é parte do Desenvolvimento Orientado a Objeto, onde a metodologia de Orientação a Objetos será usada no processo de desenvolvimento do software como um todo: Análise Orientada a Objetos, Projeto Orientado a Objetos e Programação Orientada a Objetos.

Grande parte dos projetos desenvolvidos são feitos usando os conceitos de Orientação a Objetos. Neste tipo de abordagem, os analistas deixam de pensar em funções, procedimentos e operações e passam a pensar em objetos, fechados e indivisíveis, para a construção do sistema.

Grande parte das linguagens de desenvolvimentos de sistemas e das ferramentas CASE existentes no mercado, disponibilizam a utilização desta metodologia.

“Uma importante vantagem do Projeto Orientado a Objetos é que ele simplifica a evolução do sistema.” (SOMMERVILLE, 2003, p. 239)

A Orientação a Objetos já mostrou ao mundo da informática, que é um conceito que veio pra ficar. Já existem nos meios acadêmicos, outras idéias, conceitos futurísticos voltados para a programação. Como exemplo podemos citar a Programação Orientada a Aspecto, que trás uma visão de programação tri-dimensional para o mundo da informática. Mas este conceito ainda não saiu dos meios acadêmicos, então o que ainda está valendo é a Orientação a Objetos.

1.3.4 Projeto de Software de Tempo Real

Sistemas de Tempo Real são sistemas que não podem gastar muito tempo para tomar uma decisão diante de um determinado evento. São sistemas geralmente de controle, ou seja, sistemas feitos para controlarem outros sistemas e avisar quando quaisquer destes sistemas, inclusive ele próprio, falharem.

> Os sistemas de tempo real são diferentes de outros tipos de sistemas de software; seu funcionamento correto depende da resposta do sistema aos eventos dentro de um determinado intervalo de tempo (geralmente, curto). [...] Um sistema de tempo real é um sistema de software cujo funcionamento correto depende dos resultados produzidos por ele e do tempo em que esses resultados são produzidos. Um sistema de tempo real leve é aquele cuja operação será degradada, se os resultados não forem produzidos de acordo com os requisitos de tempo especificados. Um sistema de tempo real rígido é aquele cuja operação será incorreta se os resultados não forem produzidos de acordo com a especificação de tempo. Uma maneira de ver um sistema em tempo real é como um sistema de estímulo/resposta. (SOMMERVILLE, 2003, p. 242)

Estes estímulos ou eventos que ocorrem, são fornecidos pelo meio que está sendo controlado ou monitorado. Geralmente, ambientes que requerem sistemas de tempo real, são ambientes que necessitam de monitoramento constante, não só de equipamentos mas também de seres humanos.

Em um Projeto de Software de Tempo Real, os equipamentos utilizados (hardware) devem possuir um tempo de resposta tão bom ou até melhor do que o do software.

E, no projeto do software, o que terá maior peso, obviamente, será a velocidade do sistema. Portanto, recursos tecnológicos a serem utilizados neste tipo de sistema, devem ser testados e comparados com alternativas que possibilitem os mesmos resultados. Isto faz-se necessário para averiguar os tempos gastos em um e outro recurso. E, para que posteriormente uma tomada de decisão sobre qual recurso será utilizado.

Só para se ter uma idéia da importância do tempo de resposta em sistemas deste tipo, existem casos, em que o sistema de tempo real tem que ser desenvolvido em linguagem Assembly[15].

[15] Assembly – Nome de uma linguagem de programação. Esta linguagem de programação é a mais leve que existe para o processamento do computador. Esta linguagem carrega o titulo de linguagem de “baixo nível”, pois é a linguagem mais próxima da linguagem realmente entendida pelo computador. Não é muito utilizada no desenvolvimento de outros tipos de sistemas por ser de difícil entendimento humano e escassa em recursos. (Nota do autor)

Tentamos deixar claro, nestas poucas linhas que escrevemos sobre os sistemas de tempo real, que estes são sistemas complexos e completamente diferentes dos sistemas convencionais. A maneira de análise, projeto, construção e teste deste tipo de sistema não foge completamente aos objetivos deste trabalho, contudo, ainda assim, uma abordagem completa sobre este assunto é pauta para um outro projeto de monografia.

Ressaltamos que, não é todo dia que alguém tem que projetar ou desenvolver um sistema de tempo real para ser colocado no mercado. É verdade que velocidade de processamento é um fator importante nos sistemas informacionais. Mas essa necessidade de minimização de tempo de processamento, ainda está muito aquém da necessidade dos sistemas de tempo real.

1.3.5 Projeto com Reuso

Reuso, em se tratando de informática, nada mais é do que escrever partes genéricas de um determinado sistema, e depois, reutilizá-las no próprio ou em outros sistemas.

Essa é uma tecnologia já meio antiga, que começou a ser melhor explorada com o advento das Linguagens de Programação Estruturadas. Isto não quer dizer que antes destas linguagens não se reutilizava código já escritos. Muito antes pelo contrário, o reuso era feito sim, mas de maneira complicada e de pouco eficiência.

Com a criação do conceito de programação estruturada, os programas começaram a ser modularizados em termos de funções e procedimentos que eram responsáveis por receber parâmetros, processá-los e devolver o resultado do processamento de alguma forma.

Com a modularização, veio também a possibilidade de reuso de código de sistema, desde que estes fossem escritos de maneira genérica.

Após esta fase, veio o advento das linguagens Orientadas a Objetos, que vieram fortalecer mais ainda este conceito de reutilização de código.

As linguagens de programação modernas fazem uso deste conceito por motivos óbvios. Dentre eles podemos destacar:

1. O reuso de código e componentes, gera uma economia palpável no custo de desenvolvimento de sistemas. Afinal de contas, menos código deverá ser escrito.

2. Redução no tempo de desenvolvimento.
3. A confiabilidade do software aumenta, visto que as partes utilizadas já funcionam em outro sistema.
4. Um fator importantíssimo em softwares comerciais é a Padronização, que pode ser conseguida com muito mais facilidade a partir do reuso de código e componentes.

1.3.5.1 Desenvolvimento Baseado em Componentes

Nas linguagens de programação modernas o desenvolvimento de sistemas ocorre baseado, completamente, em componentes.

"O desenvolvimento baseado em componentes, ou a engenharia de software baseada em componentes, emergiu no final da década de 90 como uma abordagem baseada no reuso para o desenvolvimento de sistemas de software." (SOMMERVILLE, 2003, p. 263)

Com o surgimento das linguagens de programação intituladas de "Visuais", a reutilização de código passou a ser peça chave no desenvolvimento de sistemas informacionais.

Nesta época, muitas empresas do ramo de desenvolvimento converteram seus projetos de linguagens de ambiente caracter[16] para linguagens de ambiente gráfico[17].

Atualmente, é pouco provável que sejam feitos investimentos em sistemas convencionais que não sejam baseados em linguagens visuais. Apesar dos sistemas gráficos serem muito mais pesados, em termos de processamento de telas. Essa tecnologia dominou o mercado e trouxe mais recurso para o desenvolvimento de softwares. O que facilitou e flexibilizou o acesso à informática por pessoas de todos os ramos de trabalho e níveis de escolaridade.

---

[16] Linguagem de Ambiente Caracter ou Linguagem Não Visual, são linguagens de programação que não apresentam a facilidade de utilização de componentes visuais e tampouco o recurso de uso do mouse para fazer alguns procedimentos. Exemplos de ambientes (sistemas operacionais) caracter: D.O.S., Unix, Xenix, dentre outros. Exemplos de linguagens de programação caracter: Pascal, C, C++, Clipper, Cobol, Basic e etc. (Nota do autor)

[17] Linguagem de Ambiente Gráfico ou Linguagem Visual, são linguagens de programação que apresentam a facilidade de utilização de componentes visuais. Exemplos de ambientes gráficos: Windows, Linux. Exemplos de linguagens de programação visual: Delphi, .Net, Java e etc. (Nota do autor)

Uma outra colocação importante para o fator reutilização, e que ficou muito mais fácil de ser implementado com as linguagens visuais, foram as frameworks. Já fizemos menção a esta tecnologia neste trabalho, portanto, ela reaparece aqui somente para complementar o contexto da reutilização de código.

Diante de todas as facilidades e vantagens, apresentadas neste tópico, para as linguagens de programação visual, não podemos esquecer que estas vantagens como qualquer outra em informática, tem seu custo, ou seja, essa tecnologia de reutilização e padrões gráficos de tela tem um custo caro de processamento. O que talvez a torne inviável, por exemplo, para desenvolvimento de sistemas de tempo real.

1.4 Sistemas Críticos

1.4.1 Confiança

Item de extrema importância em qualquer sistema, a confiança é um quesito básico para o sucesso de um aplicativo.

> A confiança em um sistema de computador é uma propriedade do sistema, que equivale à sua integridade. A integridade significa, essencialmente, o grau de confiança dos usuários de que o sistema operará como eles esperam e não 'falhará' em uso normal. (SOMMERVILLE, 2003, p. 300)

Essa integridade, muitas vezes é perdida por pequenas falhas que ocorrem nos sistemas informacionais.

1.4.1.1 Sistemas Críticos

> A falha de muitos sistemas controlados por software causa inconveniência, mas não danos sérios e prolongados. Contudo, há certos sistemas em que as falhas podem resultar em perdas econômicas significativas, danos físicos ou ameaças à vida humana. Esses sistemas, em geral, são chamados de sistemas críticos. A confiança é um atributo essencial dos sistemas críticos e todos os aspectos da confiança (disponibilidade, confiabilidade, segurança e proteção) podem ser importantes. Atingir um alto nível de confiança é normalmente, o requisito mais importante para os sistemas críticos. (SOMMERVILLE, 2003, p.302)

Falhas, são normais de acontecerem e, às vezes, até aceitáveis. Mas isto em sistemas convencionais. Em se tratando de Sistemas Críticos, como foi dito no parágrafo anterior, algumas delas podem causar muitos problemas, prejuízos e até mesmo perdas irreparáveis, como vidas.

Segundo Sommerville (2003:302), os sistemas críticos se subdividem em três tipos principais. São eles: Sistema Crítico de Segurança, Sistema Crítico de Missão e Sistema Crítico de Negócios. Cada qual com suas respectivas particularidades, mas ambos com a singularidade de serem para missões críticas. Como exemplo deste tipo de sistema temos: Sistema de Controle para Fábrica de Produtos Químicos, que em caso de falhas pode resultar em ferimentos, perdas de vidas ou danos ambientais. Sistema de Navegação de Aeronaves, cujas falhas podem causar prejuízos em relação às metas determinadas ou ainda prejuízos maiores dependendo do tipo de falha apresentada. Sistema de Contas de Clientes em um Banco, este, em caso de falhas, pode trazer enorme prejuízo financeiro não só aos correntistas mas também ao banco.

Sommerville (2003:307) dá uma explicação interessante sobre a terminologia usada em relação à confiabilidade de Sistemas:

- Falha de Sistema: Seria denominada por um evento que ocorre em algum momento, quando o sistema não fornece o serviço como é esperado por seus usuários.

- Erro de Sistema: Seria o comportamento errôneo do sistema, ou seja, o comportamento do sistema não está atendendo à sua especificação.

- Defeito de Sistema: Pode ser definido como um estado incorreto; um estado do sistema que é totalmente inesperado pelos seus projetistas.

- Erro ou engano humano: Comportamento humano que resulta na introdução de defeitos no sistema.

1.4.2 Especificação de Sistemas Críticos

Neste tópico, ressaltamos apenas alguns pontos-chave, assim considerados por Sommerville (2003:330).

Dentre eles, destacaremos:

- Os requisitos que determinam confiabilidade devem ser definidos quantitativamente na especificação de requisitos do sistema, ou seja, as especificações dos requisitos devem ser claras e não deixarem dúvidas, incoerências ou subjetividade.

- Existem várias métricas de confiabilidade diferentes que devem ser usadas. A probabilidade de falhas sob demanda, a taxa de ocorrência de falhas, o tempo médio para a ocorrência de falhas e a disponibilidade. Cada uma dessas pode ser aplicada a um ou mais tipos de sistemas diferentes. Quem deve determinar sua aplicação é o analista responsável pela especificação do sistema.

- As especificações não funcionais[18] da confiabilidade podem levar a requisitos funcionais do sistema que definam recursos do sistema cuja função seja reduzir seu número de falhas e, portanto, aumentar sua confiabilidade. Este dentre outros motivos torna este tipo de especificação, essencial nos projetos de sistemas críticos.

- Outra atividade considerada fundamental é a análise de perigos no processo de especificação de segurança. Ela envolve identificar condições perigosas que possam comprometer de alguma forma a segurança do sistema. Os requisitos do sistema devem tentar prever situações de perigo, para assegurar que estas não venham a ocorrer; ou caso ocorram não resultem em acidentes.

- Deve ser feita a Análise de Riscos, que é o processo de avaliação da probabilidade de que um perigo possa resultar em um acidente. Esta análise pode identificar situações perigosas importantes, que devem ser evitadas. Daqui saem também a classificação dos riscos do sistema de acordo com sua gravidade.

[18] Especificações Não Funcionais ou Requisitos Não Funcionais, são restrições sobre os serviços ou as funções oferecidos pelo sistema. Entre eles destacam-se restrições de tempo, restrições sobre o processo de desenvolvimento, padrões, entre outros. (SOMMERVILLE, 2003, p. 83)

- Para especificação dos requisitos de proteção, é preciso primeiramente, identificar os ativos[19] que devem ser protegidos. A partir daí, definir como as técnicas e a terminologia de proteção serão utilizadas na proteção destes ativos.

Em se tratando de sistemas críticos, o processo de desenvolvimento se diferencia um pouco dos sistemas convencionais no quesito “Tratamento de Erros, Falhas, Defeitos ou Enganos”, onde em sistemas convencionais emite-se uma simples mensagem para o usuário. Em sistemas críticos, os erros, as falhas e os enganos devem ser previstos e tratados de maneira que o sistema não venha a fazer, ou não induza o usuário a fazer algo que possa gerar qualquer tipo de risco, dano ou prejuízo sem o consentimento e intervenção de pessoas responsáveis. No caso dos Defeitos, estes devem ser sanados o mais rápido possível e, no projeto do sistema, deve haver uma solução paliativa para ser usada enquanto o defeito é consertado, se este defeito for um serviço essencial ao funcionamento do sistema.

1.4.3 Desenvolvimento de Sistemas Críticos

Para desenvolver um sistema crítico, além se ter ferramentas de desenvolvimento confiáveis, equipamentos confiáveis e pessoas competentes; deve-se ter também uma especificação de requisitos confiável, ou seja, uma especificação que tenha sido bem feita, revisada, passada para especificação formal e de preferência revisada novamente (com especialistas no assunto do qual o sistema se propõe tratar).

1.4.3.1 Minimização de Defeitos

> Um bom processo de software deve ter o objetivo de desenvolver softwares livres de defeitos. O software livre de defeito é aquele que atende exatamente a suas especificações. Contudo, isso não significa necessariamente que o software sempre se comportará como esperado pelos seus usuários. (SOMMERVILLE, 2003, p. 334)

Um software livre de defeitos não quer dizer um software livre de falhas. Um software pode atender fielmente a todas as especificações ou requisitos, e estes

[19] Dados e programas. (SOMMERVILLE, 2003, p. 329)

terem sido mal escritos, entendidos ou especificados de maneira incorreta. Assim sendo, apesar de apresentar uma inconsistência, não será um defeito, e sim uma falha (de acordo com as definições expressas anteriormente).

Sommerville (2003:334) cita alguns requisitos para o desenvolvimento de software livre de defeitos. São eles:

1. Deve-se fazer especificação precisa do sistema (de preferência, formal). Bem, além de evitar erros, a especificação formal, como já mencionado anteriormente, reduz os custos de implementação.
2. A organização que desenvolve o sistema precisa ter uma cultura de qualidade organizacional. O ideal e o que há de mais moderno neste sentido são as certificações do tipo CMM[20] ou mpsBr[21], as quais padronizam e direcionam a forma de desenvolvimento de software.
3. Deve-se fazer uso de linguagens orientadas a objeto.
4. Linguagens de programação fortemente tipadas devem ser usadas na construção do software. Em linguagens fortemente tipadas, vários erros podem ser detectados pelo compilador da linguagem.
5. Uso de construções de programas que estão potencialmente sujeitas a erros de programação devem ser evitadas.
6. O processo de desenvolvimento deve ser definido e os desenvolvedores treinados neste processo. Deve haver uma verificação rígida, por parte dos gerentes de qualidade, da conformidade com o processo.

Algo a se ressaltar é que, neste tipo de sistema, quanto maior for o grau de confiabilidade requerido, maior também será o custo com ferramentas e pessoas especializadas em testes. Isto devido aos riscos que já foram expostos anteriormente, e que se pretende minimizar.

---

[20] Capability Maturity Model - é um modelo para avaliação da maturidade dos processos de software de uma organização e para identificação das práticas-chave que são requeridas para aumentar a maturidade desses processos. Ele prevê cinco níveis de maturidade: inicial, repetível, definido, gerenciado e otimizado. (CAPABILITY MATURITY MODEL, Ministério da Ciência e Tecnologia, Disponível em: <http://www.mct.gov.br/SEPIN/Dsi/qualidad/CMM.htm> Acesso em: 04 mar. 2006)

[21] mpsBr é um projeto estruturante que vai promover a qualificação de um grupo amplo de empresas compatível com os padrões de qualidade aceitos internacionalmente pela comunidade de software, a custos acessíveis para a grande maioria das empresas brasileiras, sendo adequado ao perfil e cultura das empresas de software do país. (MELHORIA DE PROCESSO DO SOFTWARE BRASILEIRO, SOFTEX, Disponível em: <http://www.softex.br/cgi/cgilua.exe/sys/start.htm?sid=191> Acesso em: 04 mar. 2006)

1.5 Verificação e Validação

1.5.1 Verificação e Validação

É nesta etapa do processo de construção do software, que verificamos, validamos e testamos tudo o que foi ou esta sendo feito.

Conforme Sommerville (2003:358), Verificação e validação é o nome dos processos de verificação e análise que tentam assegurar que o software esteja dentro do que foi especificado nas etapas iniciais e, principalmente, que este atenda às necessidades do cliente. Sommerville explica que verificar e validar não são a mesma coisa, ou seja, são conceitos diferentes que podem ser entendidos da seguinte forma:

> Validação: estamos construindo o produto certo?
> Verificação: estamos construindo certo o produto? (BOEHM apud SOMMERVILLE, 2003, p. 358)

Ainda segundo Sommerville (2003:358), tais definições nos esclarecem que o papel da “verificação” envolve checar se o software realmente está cumprindo com as suas especificações. Já a “validação”, trata-se de um processo mais genérico, onde será checado se o software satisfaz às necessidades do cliente, isto vai além de checar se o que foi implementado está de acordo com o que foi especificado na etapa de levantamento de requisitos.

Na etapa de Verificação e Validação algumas técnicas podem ser utilizadas:

> - As inspeções de software, que analisam e verificam as representações do sistema, como o documento de requisitos, os diagramas de projeto e o código-fonte do programa. As inspeções podem ser aplicadas em todos os estágios do processo e ser complementadas por alguma análise automática do texto de origem de um sistema ou dos documentos associados. As inspeções de software e as análises automatizadas são técnicas estáticas de Verificação e Validação, uma vez que não requerem que o sistema seja executado. (SOMMERVILLE, 2003, p. 358)
>
> - Os testes de software, que envolvem executar uma implementação do software com os dados de teste e examinar as saídas dele e seu comportamento operacional, a fim de verificar se ele está sendo executado conforme o esperado. Os testes são uma técnica dinâmica de verificação e validação porque trabalham com uma representação executável do sistema. (SOMMERVILLE, 2003, p. 358)

No mercado mineiro é pouco comum a utilização da técnica de inspeção de software pelas empresas produtoras. Por outro lado, a utilização da técnica de testes de software é amplamente utilizada pelas empresas. A técnica de inspeção de software, não é totalmente descartada pelas empresas mineiras, ou seja, em partes mais complicadas do código às vezes a técnica é utilizada.

Feitos os testes e detectados defeitos, entrarão em ação os programadores, que vão depurar o código do software para localizar e corrigir o(s) erro(s) encontrado(s).

Conforme Sommerville (2003:360), a verificação e validação são processos para verificação e detecção de erros em um processo de software. Já a depuração é um processo para localização e correção de defeitos.

1.5.1.1 Planejamento de Verificação e Validação

O processo de verificação e validação de software é extremamente caro. Dependendo do tamanho e da complexidade do sistema, são necessárias pessoas especializadas para se testar e/ou julgar os resultados apresentados pelo software às diversas entradas que são realizadas nos testes.

> O processo de verificação e validação é dispendioso. Para alguns sistemas grandes, como os sistemas de tempo real, com complexas restrições não funcionais, metade do orçamento do desenvolvimento do sistema pode ser gasto em verificação e validação. Um planejamento cuidadoso é necessário para obter o melhor das inspeções e dos testes e para controlar os custos do processo de verificação e validação. (SOMMERVILLE, 2003, p. 361)

Para que seja feito um bom trabalho, tanto de verificação quanto de validação do software construído ou ainda em construção, é necessário um roteiro, ou seja, um plano de teste que possa ser seguido, passo-a-passo pelos testadores ou homologadores do sistema. É importante que este plano interfira o mínimo possível no trabalho dos desenvolvedores, exceto no caso da descoberta de alguma falha, caso contrário, as interferências poderão provocar atrasos no cronograma de desenvolvimento.

Um modelo de plano de testes, segundo Sommerville (2003:362), pode ser composto com os seguintes itens:

- O processo de teste: Uma descrição das principais fases do processo de teste, se for possível, a inclusão de um fluxograma descrevendo os passos a serem realizados, pode melhorar muito o entendimento do que deve ser feito e, em que momento.

- Facilidade de Rastreamento dos Requisitos: Os requisitos que foram levantados para construção do software devem estar disponíveis para a equipe de testes e, os testes devem ser planejados de modo que todos os requisitos sejam testados.

- Itens a serem testados: Os produtos do processo de software devem estar especificados para que possam entrar no plano de testes.

- Cronograma de testes: Um cronograma geral dos testes que deverão ser realizados deve ser montado, inclusive com os recursos que serão utilizados. Essa é a melhor maneira de se verificar o tempo que será despendido e se os recursos disponíveis serão suficientes.

- Procedimentos de registro de testes: Segundo Sommerville (2003:361), não é suficiente simplesmente executar os testes. Os resultados dos testes devem ser sistematicamente registrados.

Na prática, a utilização deste recurso é inviável, pois exigirá uma estrutura gigantesca para se guardar esses resultados e, dependendo do quão dinâmicas são as mudanças feitas nestes softwares, o registro do resultados apurados nos testes seriam inviáveis e inúteis, pois, raramente fariam jus a real condição do software em determinado momento. Lembramos que a utilização deste recurso seria útil de duas formas: a primeira para auditoria, ou seja, para apuração se os testes foram bem aplicados e quem os aplicou. A segunda seria tentar detectar qual parte do código foi alterada para estar produzindo novos resultados (com a aplicação dos mesmos testes anteriormente feitos). Contudo, para esta segunda opção, depurar ou comparar versões dos códigos-fonte, talvez ainda seja a forma mais barata, rápida e segura de se detectar alterações.

- Requisitos de hardware e software: Previamente, deve-se estabelecer as ferramentas de software que serão necessárias para o bom desempenho dos trabalhos a serem realizados.

Um fator importante é: as pessoas que utilizaram as ferramentas devem ter bom conhecimento de como fazê-lo. Caso contrário deverão ser substituídas ou treinadas nestas.

- Restrições: As restrições que afetam o processo de testes devem ser previstas.

Para este último item, o cronograma é ferramenta fundamental a ser utilizada, não esquecendo de que deve-se deixar uma margem de segurança nos prazos computados e também no número de pessoas disponíveis para desempenhar as tarefas.

1.5.2 Testes de Software

1.5.2.1 Testes para Detecção de Defeitos

A principal finalidade destes testes é a detecção de defeitos no software, para que estes sejam corrigidos antes que o software seja disponibilizado ao mercado ou ao(s) cliente(s).

Para aplicação destes testes, como já foi exposto anteriormente, é necessário que sejam elaborados: planejamento, cronograma e etc.

Existem diversas formas de se aplicar testes de detecção de defeitos em um software. Vamos expor algumas das mais utilizadas a seguir:

Testes de Caixa Preta: Também conhecido como Teste de Funcionalidade, talvez este seja o mais popular entre os testes, caracterizando-se por se testar um software pronto. Gera-se um arquivo executável e, diante de um ambiente preparado (base de testes de homologação), são feitos diversos testes no software com o intuito de detectar defeitos.

Testes de estruturas: Os testes de estruturas são aplicados diretamente nas estruturas de código construídas. Para tanto, é necessário conhecimento de programação e acesso aos códigos-fonte. Muitas vezes, este tipo de teste, quando utilizado, é feito pelo próprio analisa/programador que implementou a função/procedimento. Para criação destes tipos de testes, existem ferramentas especializadas em facilitar este trabalho de criação de testes. No mercado existem diversas, tais como: JUnit para linguagem Java, DUnit para linguagem Delphi e etc.

Partição de equivalência: Segundo Sommerville (2003:380), este tipo de teste, parte do princípio que os programas se dividem em uma série de classes com características comuns, tipo: Números positivos, números negativos, strings e etc. A partir deste ponto, define-se o escopo de entradas válidas e inválidas para a função; E feito isto, parte-se para uma massa de testes de entradas válidas e inválidas, esperando-se saídas verdadeiras e falsas respectivamente.

1.5.3 Validação de Sistemas Críticos

Os sistemas críticos, tais quais os convencionais, devem ser testados, verificados e validados. Os testes aplicados nos sistemas críticos são basicamente os mesmos aplicados em sistemas convencionais.

> A verificação e validação de um sistema crítico têm, obviamente, muito em comum com a validação de qualquer outro sistema. Os processos de Verificação e Validação devem mostrar que o sistema cumpre com sua especificação e que os serviços e o comportamento do sistema são compatíveis com os requisitos do cliente. Contudo, a natureza dos sistemas críticos é tal que geralmente é necessário aumentar a análise e os testes normais com processos adicionais, que são projetados para produzir evidências de que o sistema merece confiança. (SOMMERVILLE, 2003, p. 399)

Ainda segundo Sommerville (2003:399), existem dois bons motivos para quais esses procedimentos são necessários:

1. Custo das falhas: O custo da ocorrência de uma falha em um sistema crítico é potencialmente maior do que em sistemas convencionais. Por isso, é melhor investir mais recursos em testes, verificação e validação de sistemas críticos.
2. Validação dos atributos de confiança: Os clientes precisam estar convencidos que os atributos de confiança (disponibilidade,

confiabilidade, segurança e proteção) estão funcionando perfeitamente.

Por estes motivos, são necessários investimentos altíssimos em verificação e validação de sistemas críticos. Além do investimento maior, é necessário também que os testes feitos neste tipo de sistemas, sejam bem mais acurados que os praticados em sistemas convencionais.

Como podemos notar, as semelhanças entre sistemas críticos e convencionais são enormes, ou seja, um e outro são: concebidos, gerados e testados da mesma forma. Contudo, o que pesa nos sistemas críticos é que em todas as etapas do desenvolvimento do software, os mecanismos de controle, produção e testes devem ser mais precisos e, consequentemente bem mais caros, que nos sistemas convencionais.

1.6 Gerenciamento

A parte de gerenciamento abordada por Sommerville (2003), não é diretamente voltada ao desenvolvimento de software, por isto, faremos apenas um breve menção aos assuntos por ele descrito nos capítulos de seu livro a seguir descritos em tópicos:

Gerenciamento de Pessoal

Nesta parte, Sommerville (2003:417-435) aborda questões relacionadas às questões de administração de pessoal.

Questões essenciais para uma boa administração como, motivação de funcionários e ambiente de trabalho.

Neste capítulo, uma passagem que se destaca é a seguinte:

“O gerenciamento de software se ocupa principalmente de gerenciar pessoal. Os gerentes devem, portanto, ter alguma compreensão dos fatores humanos, de modo que não façam exigências irrealistas de si mesmos ou de seu pessoal.” (SOMMERVILLE, 2003, p. 434)

Resumidamente falando ele aborda a questão do excesso de exigências e pressão que, quando impostas aos funcionários ou mesmo à chefia, tende a trazer perdas substanciais na qualidade do software que está sendo produzido.

Estimativa de Custo de Software

Aqui Sommerville (2003:436-456) aborda os fatores que influenciam na produtividade e destaca:

> Os fatores que afetam a produtividade incluem a aptidão individual (o fator dominante), a experiência no domínio, o processo de desenvolvimento, o tamanho do projeto, o suporte a ferramentas e o ambiente de trabalho. (SOMMERVILLE, 2003, p. 455)

Além do fator produtividade, ele ainda aborda algumas métricas e modelo de algoritmos para gerenciamento e estimativas de custos.

Gerenciamento de qualidade

Segundo Sommerville (2003:457-475) o gerenciamento de qualidade software se ocupa em garantir que o software tenha o mínimo de inconsistências possíveis.

Este processo de garantir a qualidade é construtivo e estende-se desde a engenharia de requisitos até a fase de testes de software. Não adiante aplicar o controle de qualidade nas últimas fases do projeto, pois, como já foi exposto várias vezes nesse trabalho, quanto mais tarde se encontrar os erros, mais caros serão seus ajustes.

Sommerville (2003:474) diz também que os padrões de software são importantes para garantir qualidade, e que o processo de controle de qualidade deve verificar constantemente se o processo de software e o de desenvolvimento estão em conformidade com esses padrões.

Um importante mecanismo para garantir a padronização é a reutilização de código, telas e componentes de software em geral. Este mecanismo deve ser destacado entre os demais, pois, além da padronização, ele gera produtividade e segurança para o software que esta sendo desenvolvido.

Melhoria de Processo

Segundo Sommerville (2003:476-493) os pontos-chave para a melhoria de processo envolvem a análise de processo, a padronização, a medição e as modificações. Também, o treinamento dos envolvidos é ferramenta essencial para que a melhoria do processo seja eficaz.

Sommerville (2003:492) destaca que a melhoria nos processos de desenvolvimento de software, é de fundamental importância para o desenvolvimento das empresas e melhoria de expectativas de abrangência de um mercado cada vez mais competitivo. Para tal capacitação, ele cita o modelo de maturidade do SEI[22].

Como expusemos anteriormente neste trabalho, a certificação CMM e a mpsBr são os atuais mecanismos utilizados por diversas entidades, tanto civis como militares, para qualificar as empresas produtoras de software. Sendo que a certificação CMM é conhecida internacionalmente e a mpsBr já é uma adaptação da certificação CMM para o mercado brasileiro, principalmente em relação ao custo de implantação e certificação.

Como pode ser visto nos tópicos acima, fizemos uma passagem rápida, citando apenas alguns itens principais sobre os diversos assuntos do capitulo de gerenciamento. Como já foi exposto, por não se tratarem de questões ligadas diretamente ao desenvolvimento de software não nos aprofundaremos no assunto.

1.7 Evolução

Nesta parte, Sommerville (2003:497-567) aborda questões que não tratam diretamente do processo de desenvolvimento de software, por isso, aqui, semelhante à parte de Gerenciamento, faremos uma passagem rápida, no intuito de citarmos alguns tópicos principais e esclarecer pequenos detalhes.

Nesta parte, Sommerville (2003:497-567) dá destaque aos seguintes capítulos aqui referidos em forma de tópicos:

Sistemas Legados:

[22] O Software Engineering Institute (SEI) é um centro de pesquisa e desenvolvimento patrocinado pelo Departamento de Defesa dos Estados Unidos da América que provê uma prática avançada de engenharia de software qualificando graus de qualidade de software. (SOFTWARE ENGINEERING INSTITUTE, Wikipédia – A enciclopédia livre, Disponível em: < http://pt.wikipedia.org/wiki/Software_Engineering_Institute> Acesso em: 23 mai. 2006)

Neste capítulo, Sommerville (2003:497-567) explica o conceito de Sistemas Legados:

“Um sistema legado é um sistema antigo que ainda fornece serviços de negócios essenciais.” (SOMMERVILLE, 2003, p. 512)

Muitas empresas, principalmente as maiores, ainda usam esse tipo de sistema. São sistemas com telas um tanto quanto ultrapassadas, vistos aos padrões de hoje. São sistemas que não possuem banco de dados, muitas vezes armazenam seus dados em arquivos ou em softwares que simulam bancos de dados arcaicos.

Contudo, mesmo sem ter tecnologia de ponta, esses sistemas ainda funcionam. E pensar em substituí-los, torna-se uma tarefa difícil e muitas vezes impraticável. Os custos para troca de equipamento e de software, talvez não atendam as expectativas da empresa. Que é o que acontece em boa parte dos casos.

“O valor de negócios de um sistema é uma avaliação da eficácia do sistema em fornecer apoio aos objetivos do negócio.” (SOMMERVILLE, 2003, p. 512)

Os sistemas legados, só devem ser substituídos caso não estejam atendendo aos objetivos do negócio, mesmo assim, deve-se antes fazer uma análise de custo/benefício bem feita antes de se propor uma mudança.

Mudanças de Software:

“Entre as estratégias de modificação de software estão a manutenção de software, a evolução da arquitetura e a reengenharia de software” (SOMMERVILLE, 2003, 530)

Neste capítulo são expostos vários fatores que implicam na manutenção de software, dentre eles, podemos citar os custos para evoluir um software, que geralmente são altos devido ao fato dos softwares não serem construídos de modo a facilitar o processo evolutivo.

Ainda segundo Sommerville (2003:530), o custo de mudança de software geralmente excede os custos de desenvolvimento. Tendo em vista que as empresas possuem um número crescente de sistemas legados, grande parte de seu orçamento pra software é destinada a manutenção destes sistemas.

Mudanças de softwares, legados ou não, geralmente são complicadas, pois, envolve outra tecnologia, outra forma de se fazer os processos que os funcionários

já sabiam fazer de “olhos fechados” no sistema antigo. Às vezes faltam ou sobram recursos no novo software. Falta de treinamento do pessoal e etc.

Reengenharia de Software:

Para a reengenharia de software, vários processos podem ser utilizados, dentre eles podemos citar: Tradução de Código-fonte e Engenharia Reversa. A tradução de código-fonte implica em traduzir literalmente o código-fonte escrito na linguagem X para a linguagem Y.

O processo de engenharia reversa, pode se dar através do estudo do código-fonte, se este estiver disponível ou do executável, caso contrário.

Existem também neste capítulo, alguns textos sobre Melhoria de Estrutura do Programa e Modularização de Programa, que dizem respeito às dificuldades que serão encontradas nas tentativas de melhorar a estrutura de um software legado ou mesmo tentar modularizá-lo. Grande parte desta dificuldade se encontra devido às linguagens de programação antigamente utilizadas.

Gerenciamento de Configuração:

Este é o último processo a ser analisado. A geração de configuração, é super-importante para empresas de pequeno, médio e grande porte. Até mesmo para os softwares caseiros, se for possível fazer o controle de versões mediante uma ferramenta especializada, isto facilitará bastante para o proprietário do sistema lidar com os problemas de versões.

“O gerenciamento de configuração é o gerenciamento de mudanças no sistema.” (SOMMERVILLE, 2003, p. 566)

Em empresas com grande estrutura de desenvolvimento, ferramentas para controle de versão de software em grupo, são essenciais. Não só para o controle de versão, mas também para o controle de alteração no código-fonte. Principalmente quando se tem mais de um analista/programador trabalhando em um mesmo sistema simultaneamente.

Existem no mercado, ferramentas que possibilitam esse trabalho em grupo. Como exemplo de uma ferramenta de controle de versão, podemos citar o Visual Source Safe da Microsoft.

Durante todo este primeiro capítulo, foi apresentada resumidamente uma metodologia de desenvolvimento de software baseada num autor de renome

internacional. A metodologia utilizada em Sommerville (2003), não foi totalmente resumida neste trabalho, pois, muitos tópicos saiam do escopo deste e não fariam sentido se aqui fossem expostos.

2 Análise do processo de desenvolvimento de software e mapeamento daqueles que contribuem para geração de conhecimento

Neste capítulo, o foco principal será a análise distinta de cada uma das fases de desenvolvimento de software descritas no capítulo anterior. Onde procurar-se-á identificar os pontos-chave de geração de conhecimento que poderão ser contemplados por este trabalho, através das práticas que serão sugeridas no próximo capítulo.

> Na última década, centenas de empresas, literalmente, tentaram melhorar seus processos. Primeiro, usando abordagens de qualidade total e de aperfeiçoamento contínuo; depois, mais radicalmente, utilizando métodos de reengenharia. Uma ampla variedade de processos recebeu ênfase. No entanto, os métodos voltados à informação e ao conhecimento raramente entraram no foco dos programas de aperfeiçoamento rigorosos.[23]

O que aconteceu num passado não muito remoto, ainda acontece no presente, ou seja, as empresas produtoras de software, grande parte delas, não dão o devido valor ao conhecimento que têm, que é produzido, transferido e muitas vezes perdido.

Na formação dos profissionais da computação, não há uma disciplina que os ensine o real valor do conhecimento. E tampouco como mapear, arquivar e transmitir este conhecimento para que não se perca no tempo.

Ao se mapear, no processo de desenvolvimento de software, os pontos-chave na geração de conhecimento, estar-se-á dando um passo para que novas práticas de gestão do conhecimento sejam aplicadas em benefício mútuo. Portanto torna-se essencial que todos os profissionais envolvidos no processo de desenvolvimento de software, tenham conhecimento do que se pretende fazer e das práticas que serão aplicadas, pois, isto é fator primordial para garantir o sucesso do projeto com a aplicação das práticas.

Iniciando pela parte da metodologia chamada de Visão Geral, pode-se notar claramente a necessidade da gama de conhecimento prévio que um profissional deve possuir para começar seu trabalho dentro da engenharia de software. Nesta

[23] DAVENPORT, Thomas H.; **Ecologia da INFORMAÇÃO**; Futura, São Paulo, SP; 2000; p. 198.

parte da metodologia fica claro também a complexidade envolvida na produção de software, não somente pelo fato da transcrição da realidade para a lógica entendida pela máquina, mas também pela complexidade que é a própria máquina e os demais equipamentos que a compõem. Isto sem levarmos em consideração a possibilidade desta estar em uma rede se interagindo com milhares de outras máquinas.

2.1 Requisitos

Esta fase é peça-chave na produção e transcrição de conhecimento de forma estruturada e lógica.

Nesta parte, de requisitos, haverá grande geração e transformação de conhecimento.

Conhecimento este que, conforme (Nonaka e Takeuchi apud Rus et al, 2001)[24], passará por uma transição, onde o conhecimento tácito de uma pessoa será externalizado através dos requisitos descritos em linguagem natural, diagramas e prototipação, passando a ser compreendido como conhecimento explícito.

Em se tratando de requisitos de software, pode-se considerar que esta seja a parte mais importante de toda a engenharia de software, pois, aqui é definido tudo o que será ou não feito no produto final, o software. E mais importante, será definido se será ou não feito o software.

Quaisquer erros ou falhas na definição, levantamento e análise dos requisitos, poderá causar grande perda de tempo e/ou dinheiro nas diversas outras fases, da engenharia de software, que virão.

Em se tratando de modelos de sistemas, uma análise cabível para esta parte da metodologia é dizer que esta se faz essencial a partir do momento que ajuda no tratamento das ambigüidades que podem ser causadas pela linguagem natural. Obviamente estas ambigüidades podem não ser muitas ou mesmo podem até não existirem, contudo, a partir da transcrição da informação escrita em linguagem natural para uma simbologia conhecida, automaticamente estar-se-á facilitando a compreensão do que está sendo pedido aos analistas/programadores.

---

[24] RUS, Ioana; LINDVALL, Mikael; SINHA, Sachin Suman. **Knowledge Management in Software Engineering.** Disponível em: <http://192.73.45.130/techs/kmse/kmse.pdf>. Acesso em: 20 fev. 2006

O mesmo raciocínio do parágrafo anterior se aplica na prototipação de software. Tanto no modelo de “interface com o usuário” quanto no modelo de “processo de software”, além de ampliar o entendimento dos analistas/programadores quanto ao que se espera do sistema, ainda existe a vantagem do cliente poder dar palpites e confirmar se o que está sendo feito está de acordo com suas expectativas.

Quanto à especificação formal de software, esta técnica, apesar de, não se ter conhecimento de empresas que a usem para produção de softwares convencionais, sabe-se através do estudo da técnica, e também como foi apresentado no capítulo anterior, que pode facilitar nos processos finais do desenvolvimento e até mesmo reduzir os custos com re-trabalho. Isto sem levar em consideração que esta é de extrema importância para os softwares conhecidos como “sistemas críticos”. Visto que este tipo de software não pode apresentar erros, ou pelo menos deve reduzi-los a um número bem próximo de zero.

À etapa de requisitos como um todo, ao que se pode ver na realidade diária das empresas produtoras de softwares convencionais, é tratada como mais uma burocracia no desenvolvimento de software. Isto quando esta etapa existe, pois, na maioria dos casos o que se pode constatar é que mal... mal é rascunhado, pelos analistas/programadores, uma lista dos itens que devem ser implementados; e no muito, quando surgem as dúvidas de como fazê-los, estes analistas/programadores procuram pessoas com conhecimento do assunto para algum esclarecimento.

Desta forma, perde-se a oportunidade de se obter um bom nível de conhecimento sobre o sistema, a documentação do mesmo (visto que essa não será criada) e principalmente o histórico das decisões tomadas e o porquê destas tomadas de decisão. E quando em algum momento futuro for necessário saber por que algo no software foi feito de tal forma, essa informação não estará disponível e tampouco suas origens.

Esta mentalidade tende a mudar aos poucos, como já está acontecendo. Contudo, esta mudança caminha a passos lentos na realidade nacional.

> A indústria de software vem demonstrando crescente interesse em engenharia de requisitos, isto é, entender o que se deseja construir antes de começar a fazê-lo. Estão percebendo que o tempo utilizado no entendimento do problema é um excelente investimento. Os requisitos de software são a base a partir da qual a qualidade é

> medida. Desta forma, a falta de conformidade aos requisitos significa falta de qualidade.[25]

Isto sem levar em consideração o alto custo gerado e a perda de tempo com re-trabalho na correção do que foi programado incorretamente.

Por se tratar da fase onde identificar-se-á tudo o que o software deverá ou não fazer, é de extrema importância que façam parte deste processo: Pessoas com conhecimento aprofundado de como as coisas funcionam e/ou deveriam funcionar na empresa-cliente. Generalistas e especialistas nas diversas funcionalidades, processos e procedimentos realizados na empresa e que o sistema se propõe a tratar.

2.2 Projeto

Na parte de projeto, iniciando-se pela estruturação de sistemas, pode-se fazer a seguinte análise: Esta é totalmente depende da parte de requisitos. Os requisitos apurados dirigiram todo o processo de construção do projeto de software.

Na parte de projeto de arquitetura, mais especificamente em estruturação de sistema, podemos inferir que o responsável deverá ter em mente, os conceitos do novo sistema, bem definidos, pois, em se tratando de sistemas de médio e grande porte, a complexidade de se estruturar de forma clara, precisa e ordenada, é extrema. Vale destacar aqui que deve-se ter uma boa documentação dos requisitos.

A estruturação do sistema é extremamente importante e deve ser analisada, feita e avaliada junto a alguém que conheça bem dos processos que estão sendo mapeados.

Toda a documentação deve ser gerada, de forma a facilitar possíveis consultas futuras. E principalmente, esta deve ser refeita ou atualizada a toda ocorrência de mudanças no sistema.

Antecipadamente à estruturação, a análise/avaliação de: Com que tipo de arquitetura será desenvolvido o sistema (arquitetura distribuída, Orientada a Objetos

---

[25] CARVALHO, Ana Elizabete Souza de; TAVARES, Helena Cristina; CASTRO, Jaelson Brelaz. **Uma Estratégia para Implantação de uma Gerencia de Requisitos Visando a Melhoria Dos Processos de Software**. Disponível em: <http://www.inf.puc-rio.br/~wer01/Pro-Req-3.pdf>. Acesso em: 15 abr. 2006

e etc.) funcionará como um mecanismo norteador para o responsável pela estruturação.

A análise, os diagramas e toda a documentação que será gerada basear-se-á na arquitetura pré-definida e também no modelo de controle. Assim tende-se a aproveitar ao máximo os recursos que todo o ambiente pode proporcionar.

O modelo de controle que será definido para construção do sistema e a decomposição em módulos, estão sujeitos ao tamanho, às atribuições e também a estrutura utilizada no sistema.

O projeto da arquitetura como um todo, é bastante variável de projeto para projeto, exceto em caso de projetos de software semelhantes. É válido deixar claro que para adoção de qualquer ação em prol da construção de um projeto de software, todo um estudo deverá ser efetuado. Este estudo, como já foi visto, está previsto na parte de requisitos e leva o nome de “Estudo de Viabilidade”. Lá não serão feitos estudos de Estruturação de sistemas, Modelos de Controle ou tampouco de Decomposição em Módulos, contudo, lá serão definidos os gastos para que o software seja implementado, onde estará incluída também a parte física (computadores, redes, impressoras e etc.) que será necessária para suportar o software. E conforme for esta definição da parte física, que deverá respeitar às necessidades do cliente e o orçamento previsto, afetará diretamente no projeto de arquitetura do software.

Para a arquitetura de sistemas distribuídos, vale à mesma regra dos sistemas convencionais, descrito anteriormente. Pois a única diferença será na forma com este será construído. A documentação também será um pouco diferente, mas não deverá fugir aos padrões utilizados em sistemas convencionais. Um fator importante que deve ser mencionado é o seguinte: em se tratando de sistemas distribuídos, a parte de segurança do produto deve ser bem mais complexa do que em sistemas convencionais, inclusive pelo fato de que pode-se perder parte ou a rede por completo em determinado momento. Daí, ações corretivas deverão ser tomadas. O funcionamento e as ações corretivas, propriamente ditas, devem estar claramente especificados na documentação do sistema.

Em se tratando de sistemas distribuídos, muitas vezes, a continuidade do funcionamento do sistema deve ser garantida pela rede, para isto existem diversas alternativas que não serão discutidas aqui, contudo, mesmo que estas alternativas sejam de responsabilidade total da parte física da rede, é essencial que estas

informações constem na documentação do software, tanto a informação referente a delegação de responsabilidade quanto a possível solução. Caso não seja possível, deverá constar, no mínimo, a outra fonte onde a solução poderá ser conseguida.

Aos projetos orientados a objetos, vale fazer a seguinte análise: Deve-se pensar em estruturas orientadas a objeto desde o início do mesmo, inclusive na parte de requisitos, pois, assim o transcorrer do projeto não terá complicações referentes à metodologia. Vale lembrar também que a documentação é a mesma, variando apenas na simbologia aplicada nos diagramas e também nas descrições dos documentos.

Em projetos de softwares de tempo real, existem diferenças gigantescas na forma de se pensar, em relação ao software convencional. E estas diferenças não são aplicáveis somente ao projeto de arquitetura do software, mas também a toda estrutura física onde o software deverá funcionar. A documentação deste tipo de software, além de uma simbologia um pouco diferente, deverá ser bem mais detalhista contendo especificações precisas sobre as limitações e as necessidades do sistema (Ex. tempo de resposta ideal, suportável e inadequado para uma ação).

Já em projetos com reuso, é aconselhável que na documentação do software contenha também a parte do sistema que está sendo utilizada, ou seja, aquela parte que está sendo reutilizada de outro software ou de alguma biblioteca compartilhada. Assim, quando houver algum estudo em cima do projeto, não será necessário estar recorrendo a outros locais para se obter as informações desejadas. Caso não seja possível a adição da documentação das partes reutilizadas, torna-se imperativo que se especifiquem as fontes onde estas poderão ser encontradas.

As *softwarehouses* costumam fazer, principalmente nesta época de mudança de mentalidade, uma documentação parcial dos sistemas que vão ou que estão sendo desenvolvidos. Para isto, elas adotam uma metodologia existente e fazem uso de algumas práticas propostas, por esta metodologia. O grande problema é que, esta documentação quando existe, além de deficitária (por ser apenas parte da sugerida pela metodologia), na maioria das vezes, nunca é atualizada. Ou seja, faz-se a documentação no projeto do sistema. Durante o andamento do projeto, às vezes, esta documentação se mantém atualizada, porém, após sua finalização esta documentação é esquecida e as alterações ocorridas no software só estarão presentes no código-fonte. Onde, se a empresa não possuir um rigoroso controle de versões, todo o histórico de mudanças será perdido definitivamente.

Este é um problema duplo. Duplo porque, primeiramente gastou-se tempo e dinheiro fazendo uma documentação para um sistema. E em segundo, sendo o mais grave, esta documentação não veio sendo atualizada juntamente com o sistema e, dependendo do tempo que já se tenha passado, talvez seja mais fácil jogá-la no lixo e fazer outra. Como na prática é quase impossível que isto aconteça (Alguém fazer outra documentação ou atualizar a existente.), o sistema acabará ficando sem documentação.

> Em muitas empresas, as políticas de gerenciamento de informações lembram a maneira como lidamos com doenças. Gastamos enormes recursos para desenvolver medicamentos de alta tecnologia, mas os pacientes não os tomam, ou não seguem a receita de maneira adequada. Como um medicamento que não é tomado, a informação de nada servirá até que seja utilizada. (DAVENPORT, 2000, p. 194)

Ressalta-se que esta prática de documentação deficitária não é comum em sistemas mais complexos e tampouco em sistemas que não são convencionais, como por exemplo, sistemas de tempo real e sistemas críticos.

Um projeto bem feito é também um projeto bem documentado, isto não quer dizer que a recíproca seja verdadeira. Faz-se necessário, porém, que esta documentação acompanhe o projeto, a qual pertença, em todas as suas mudanças. Caso contrário esta tornar-se-á obsoleta e ficará cada vez mais sem utilidade, cairá em descrédito e muitas vezes, como já foi dito anteriormente, acabará no lixo.

As empresas devem ter em mente, que a documentação não é um adorno físico que acompanha um produto intangível. Ela vai muito além disto, sendo uma ferramenta indispensável na economia de tempo e dinheiro, principalmente quando se estiver falando de manutenção e aprendizado (geralmente por novos funcionários) do funcionamento do sistema.

Algo importante para se destacar é que, a parte de requisitos deve ter sido muito bem feita e avaliada, visto que esta é à base de todo o conhecimento pelo qual o sistema será concebido e, toda a documentação e o próprio sistema que será gerado a partir desta, basear-se-ão na documentação gerada na parte de requisitos.

Outra coisa importante é que, o analista responsável pela estruturação do projeto deve estar com os conceitos e informações sobre o projeto, adquiridos na parte de requisitos, bem definidos em sua mente. Este é um outro bom motivo para que a documentação de requisitos esteja impecável, pois, quando aparecerem dúvidas para os analistas/programadores estes recorrerão primeiramente, à

documentação de requisitos. Em último caso recorrerão às pessoas responsáveis, na empresa-cliente, pelas áreas dos requisitos onde surgiram as dúvidas.

2.3 Sistemas Críticos

Este é um tipo de sistema que requer muita atenção em todas as etapas de sua construção. Em quase cem por cento dos casos, devido às tarefas que serão executadas pelo sistema, a ocorrência de erros é inadmissível. Como já foi explicado anteriormente, vidas ou mesmo serviços que não podem sofrer interrupção podem estar subordinadas ao bom funcionamento deste tipo de sistema.

Num sistema crítico, todo o processo de construção do software segue os mesmos passos dos softwares convencionais, já descritos anteriormente, contudo, vale ressaltar que neste tipo de sistema, o que pesa é a confiabilidade. Assim sendo, grande parte da verba destinada a construção de sistemas críticos vão para a homologação do sistema. Isso ocorre no intuito de que o sistema, ao sair da fábrica de software, esteja realmente apto a assumir as responsabilidades a que se propõe.

Isto não quer dizer que as outras fases da construção do sistema não sejam privilegiadas como a homologação. Principalmente as fases iniciais, de apuração de requisitos (que devem ser especificados formalmente) e o projeto de arquitetura, também recebem tratamento especial, visto que, erros quanto mais tarde forem descobertos mais caros serão os acertos.

Empresas que desenvolvem este tipo de software são geralmente empresas com métodos de requisitos, projeto, testes e verificação bem definidos, e que têm bons resultados no produto final.

A questão que aqui deve ser colocada é: Mesmo com todo esse rigoroso esquema montado para se ter um produto final quase perfeito, os processos aplicados não podem ser melhorados? Afinal de contas, quanto menos erros forem encontrados na homologação, menores serão os custos com re-trabalho.

Neste caso, crê-se que o melhor a se fazer seria aplicação de métodos que melhorem e reduza, mais ainda, a possibilidade de erros na parte de engenharia de requisitos.

2.4 Verificação e Validação

Nesta parte da engenharia de software, defini-se a qualidade com que o produto chegará às mãos do cliente. Sua relevância é indiscutível.

Os métodos e ferramentas utilizadas para testar os diversos tipos de softwares existentes, são quase que padronizados, ou seja, não há muita diferença entre eles. Contudo, como já foi dito anteriormente, existem softwares que devem passar por rigorosos e criteriosos testes antes de serem colocados em funcionamento.

O processo de teste de software deve ser cauteloso, bem estruturado e bem feito. Deve possuir total independência em relação aos desenvolvedores. Mesmo por que estes têm suas tarefas e prazos a cumprir, da mesma forma que os homologadores.

Os processos de verificação e validação de software, muitas vezes são utilizados por pessoas inexperientes nas ferramentas de homologação ou mesmo sem conhecimento da regra de negócio que deve ser aplicada aos testes. Quase sempre, os testes são realizados através de uma lista de itens, ou seja, um passo-a-passo seguido pelos funcionários que estão incumbidos de realizá-los. Estes geralmente o fazem sem questionar.

Para se testar um software, pessoas experientes na ferramenta de testes e com conhecimento da regra de negócio do módulo em teste, trariam melhores resultados no que diz respeito à qualidade final do produto.

Estes profissionais auxiliados por uma metodologia bem definida aumentam a produtividade e, consequentemente a melhoria dos resultados do desempenho do projeto.

A definição de uma metodologia e também a capacitação dos funcionários é fator primordial para se criar/manter a qualidade do produto.

2.5 Gerenciamento e Evolução

Estes tópicos da metodologia utilizada para a construção deste trabalho, como explicado no capítulo anterior, não tratam diretamente do processo de

desenvolvimento de software. Por isto não serão analisados e tampouco apontados ou sugeridos processos que contribuem para a geração de conhecimento.

3 Apresentação Práticas para a Gestão do Conhecimento em Softwarehouses

Através de diversos estudos e experiência em análise e desenvolvimento de sistemas do autor, foram selecionadas práticas de gestão do conhecimento para aplicação na área de engenharia de software.

> O papel principal da tecnologia da informação na gestão do conhecimento consiste em ampliar o alcance e acelerar a velocidade de transferência do conhecimento. Os software de gestão do conhecimento pretendem auxiliar na captura e estruturação do conhecimento de grupos de indivíduos, disponibilizando esse conhecimento em uma base compartilhada por toda a organização. Objetiva construir uma tipologia baseada no uso das ferramentas de gestão de conhecimento.[26]

Este capítulo será apresentado da seguinte forma:

- A seguir, será apresentado um resumo contendo o nome de todas as práticas propostas para gestão do conhecimento. Estas serão enumeradas, sendo que para cada uma delas haverá uma introdução, objetivo, aplicação e possível melhoria esperada com a aplicação da prática no desenvolvimento de sistemas.

- Em seguida, serão apresentadas algumas considerações e logo as sugestões para aplicação das práticas propostas nas diversas partes da engenharia de software segundo a metodologia utilizada no primeiro capítulo. Junto a estas sugestões, serão analisados os efeitos das aplicações das práticas propostas e as possíveis melhorias que poderão ser conseguidas.

[26] CARVALHO, Rodrigo Baroni. **Aplicações de Software de Gestão do Conhecimento: tipologia e usos.** Disponível em: <http://www.eci.ufmg.br/pcionline/viewarticle.php?id=320>. Acesso em: 09 mai. 2006.

3.1 Apresentação das Práticas Propostas para Melhoria na Produção de Software:

Antes da apresentação das práticas propostas para a Gestão do Conhecimento no contexto de desenvolvimento de software, vale ressaltar que algumas destas práticas, não são exatamente voltadas para Gestão do Conhecimento como veremos a adiante.

As duas primeiras práticas apresentadas, 1 (*Design Patterns*[27]) e 2 (Especificação Formal), são práticas utilizadas em Engenharia de Software. Já as próximas duas, 3 (Padronização das Fases do Projeto) e 4 (Inspeção Final de Fase), são práticas utilizadas em Gestão de Projetos; contudo, apesar de não serem diretamente voltadas para a Gestão do Conhecimento, estas são práticas complementares essenciais para que os objetivos deste trabalho sejam alcançados em sua plenitude.

As demais práticas relatadas são todas pertencentes ao âmbito da Gestão de Conhecimento, sendo que a prática 10 (Portal Corporativo do Conhecimento) pode ser considerada uma prática atípica às demais, pois, sua aplicação dar-se-á na forma de um *container*[28] para as demais práticas que virão após esta, até a prática de 19 (Disponibilização de Diagramas).

É importante ressaltar que das idéias relatadas neste trabalho sobre as práticas que serão citadas e descritas mais adiante, além dos diversos autores que foram e que serão referenciados bibliograficamente no decorrer deste. Merecem créditos pela contribuição fundamental na concepção e construção deste trabalho, além do conhecimento tácito do autor, os diversos estudos e diálogos nas seguintes áreas:

- Das práticas de Engenharia de Software, o livro de Sommerville (2003), as aulas da professora Márcia Mônica Nogueira Mendes[29] e do professor João Luiz Silva Barbosa[30].

---

[27] Do inglês “Padrões de Projeto”. (Tradução nossa.)
[28] Recipiente, receptáculo, cofre de carga. In: Michaelis Pequeno Dicionário Inglês-Português/Português-Inglês. 50. ed. São Paulo: Melhoramentos, 1995. 794 p.
[29] MENDES, Márcia Mônica Nogueira. Engenharia de Software. Belo Horizonte: Universidade FUMEC, 2005. (Aulas ministradas no Curso de Pós-graduação em Gerência de Tecnologia da Informação da Universidade FUMEC)

- Das práticas de Gestão de Projetos, as aulas do professor Roberto Luis Capuruço Gattoni[31].

- Das práticas de Gestão do Conhecimento, as aulas e os diversos diálogos com o professor Rodrigo Baroni de Carvalho[32] e com o professor Roberto Luis Capuruço Gattoni, que ministrou o primeiro o módulo da disciplina de Ciência da Informação finalizada pelo professor Rodrigo Baroni Carvalho.

3.1.1 Resumo das práticas propostas:

- Prática 1 - *Design Patterns*
- Prática 2 - Especificação Formal
- Prática 3 - Padronização das Fases do Projeto
- Prática 4 - Inspeção Final de Fase
- Prática 5 - Classificação e Documentação de Reuniões
- Prática 6 - Reunião Inaugural de Fases de Projeto
- Prática 7 - Reunião de Finalização de Fase de Projeto
- Prática 8 - Universidade Corporativa
- Prática 9 - Aposentandos[33] como Professores
- Prática 10 - Portal Corporativo do Conhecimento
- Prática 11 - Zona de Auxílio Imediato
- Prática 12 - Fórum de Publicação de Dúvidas
- Prática 13 - Informações a Iniciantes
- Prática 14 - Documentação do Conhecimento
- Prática 15 - *Help*[34] Inteligente
- Prática 16 - *Check List*[35] – O que/Como fazer?
- Prática 17 - FAQ (*Frequency Asked Questions*[36])
- Prática 18 - Mapa do Conhecimento
- Prática 19 - Disponibilização de Diagramas
- Prática 20 - Avaliação e Revitalização das Práticas

---

[30] BARBOSA, João Luiz Silva. Arquitetura de Sistemas de Informação. Belo Horizonte: Universidade FUMEC, 2005. (Aulas ministradas no Curso de Pós-graduação em Gerência de Tecnologia da Informação da Universidade FUMEC)
[31] GATTONI, Roberto Luis Capuruço. Gerência de Projetos em Tecnologia da Informação. Belo Horizonte: Universidade FUMEC, 2005. (Aulas ministradas no Curso de Pós-graduação em Gerência de Tecnologia da Informação da Universidade FUMEC)
[32]CARVALHO, Rodrigo Baroni. Ciência da Informação. Belo Horizonte: Universidade FUMEC, 2005. (Aulas ministradas no Curso de Pós-graduação em Gerência de Tecnologia da Informação da Universidade FUMEC)
[33] Funcionários que estão prestes a aposentar. (Nota do autor)
[34] Do inglês, “Ajuda”. (Tradução nossa.)
[35]Lista. In: Michaelis Pequeno Dicionário Inglês-Português/Português-Inglês. 50. ed. São Paulo: Melhoramentos, 1995. 794 p.
[36] Do inglês, “Questões Perguntadas Frequentemente”. (Tradução nossa.)

A seguir, conforme mencionado anteriormente, serão apresentadas as práticas detalhadamente. Cada qual conterá introdução, objetivo, aplicação e possível melhoria esperada, no desenvolvimento de sistemas, a partir da sua efetiva aplicação.

3.1.2 Prática 1:

Design Patterns:

> *Design patterns* são geralmente definidos como soluções já testadas para problemas recorrentes. O termo refere tanto à descrição de uma solução que você pode ler quanto a uma instância daquela solução utilizada para resolver um problema particular (podemos pensar que *design patterns* poderiam ser tanto uma classe quanto uma instância daquela classe). *Design patterns* têm suas raízes no trabalho de Christopher Alexander, um engenheiro civil que escreveu sobre a sua experiência em resolver problemas de projeto que ele encontrava em construções e cidades. Ocorreu a Alexander que certas idéias de projeto, sempre que eram utilizadas, levavam ao efeito desejado. Ele documentou e publicou o conhecimento e a experiência que ele de forma que outros pudessem se beneficiar. Aproximadamente 15 anos atrás, profissionais de software começaram a incorporar os princípios de Alexander na criação das primeiras documentações de *design patterns* como um guia para desenvolvedores novatos.[37]

Com a definição dada a *Design Patterns*, podemos entendê-lo como um tipo de conhecimento que foi pensado, testado, aprovado e publicado para que outros não tivessem que fazer o mesmo toda vez que fossem resolver problemas corriqueiros do dia-a-dia.

O objetivo dos *design patterns* é realmente facilitar a resolução de problemas comuns e de constante acontecimento, de maneira simples e segura.

A utilização de *design patterns* no desenvolvimento de software, vem se tornando cada vez mais freqüente, visto que sua utilização é segura e de grande valia para os analistas/programadores que ganham tempo e segurança não tendo que “reinventar a roda”, ou seja, pensar numa solução que já foi pensada, programada, testada, aprovada e está publicada.

---

[37] O que são e de onde vieram os Design Patterns?, [s. n.]. Disponível em: <http://www2.fundao.pro.br/articles.asp?cod=144> Acesso em: 25 abr. 2006.

As melhorias advindas da utilização desta prática são claras, sendo que as principais já foram citadas no parágrafo anterior. A utilização de partes de código que já foram pensadas, testadas, aprovadas e que estão publicadas, é mais uma segurança de que não haverá erros nestas partes específicas que serão utilizadas. Isso reduz de fato o escopo de teste na fase de homologação, paralelamente produzindo ganho no tempo de implementação e naturalmente redução de custos. O tempo que seria gasto na implementação da função encontrada como *design patterns* poderá ser utilizado para adiantar outras partes do projeto em questão.

3.1.3 Prática 2:

Especificação Formal:

Como foi exposto anteriormente neste trabalho, por Sommerville (2003) e Figueiredo et al (2002), esta prática apesar de pouquíssimo usada, basicamente só em sistemas de missão crítica, pode ser a chave para uma enorme redução nos erros de projeto de sistemas.

O objetivo principal da especificação formal seria esse: Através da transcrição do conhecimento adquirido e registrado em linguagem natural, para uma linguagem de especificação formal; reduzir bastante o número de erros provenientes de deficiências e limitações da linguagem natural que podem gerar duplo entendimento ou mesmo más especificações.

A aplicação da especificação formal será feita diretamente na fase de engenharia de requisitos, pois além de ser o sugerido pela metodologia adotada para o desenvolvimento deste trabalho, logicamente opta-se, em qualquer tipo de projeto, por exterminar os erros o mais cedo possível. Principalmente quando esses podem se propagar para as demais fases que seguirão após a fase em questão.

A melhoria esperada com esta prática é o aumento da qualidade do software que sairá para o mercado. E também, estando esta parte em segundo plano, um aumento de produtividade nas fases de construção e homologação do software.

3.1.4 Prática 3:

Padronização das Fases do Projeto (PFP):

Ao iniciar uma fase de um projeto, os profissionais responsáveis costumam se perguntar como devem agir e o que deve vir primeiro. Bem, estas são perguntas bem comuns, a não ser que o que deverá ser feito seja um trabalho praticado diariamente, onde qualquer profissional possa facilmente decorar os passos a serem dados.

Obviamente, já existem diversos padrões de desenvolvimento de projetos de softwares já aprovados pelo mercado. Estes padrões levam o nome de Engenharia de Software, como foi exposto no primeiro capítulo deste trabalho. Contudo, deve-se levar em consideração, que cada empresa, diferente uma da outra, tem suas peculiaridades, ou seja, mesmo que se adote fielmente uma das metodologias existentes no mercado, ainda assim, é provável que a empresa faça alguns ajustes na metodologia adotada para que esta se enquadre bem à filosofia da empresa.

Tudo isto, sem levarmos em consideração que: Hoje a empresa poderá ter um profissional experiente, que saiba todos os passos para se iniciar um projeto de cor. Mas amanhã, este profissional pode não estar mais lá, por motivos diversos. E se assim for, o que deverá ser feito?

O objetivo desta prática é facilitar a aplicação do conhecimento tácito, nas fases de desenvolvimento de software, de maneira clara, padronizada e ordenada. Para que qualquer profissional habilitado possa iniciar um projeto com tranqüilidade e segurança de seus atos, respeitando as normas da metodologia e os padrões da empresa.

Isto não significa que o profissional poderá se abster de possuir o conhecimento prévio de engenharia de software. Muito antes pelo contrário, o profissional deverá ser um bom conhecedor de engenharia de software, pois o papel desta prática, nada mais é do que ser um norteador para que o profissional saiba quando e como aplicar seus conhecimentos.

Este mecanismo não deve ser tão simplificado como o exposto na prática 17 (*Check List* – O que/Como fazer?) e tampouco detalhista demais, como explicado na metodologia adotada pela empresa. Deve ser um meio termo entre ambos, capaz de prover o profissional com informações suficientes para guiá-lo em seu trabalho.

A aplicação da PFP se dá principalmente na iniciação de novas fases em projetos. Com um mecanismo norteador, o profissional saberá os passos a serem

seguidos. Vale ressaltar que a prática não deve possuir um caráter rígido, ou seja, em se tratando de desenvolvimento de projetos, sejam eles de sistema ou não, a flexibilidade na condução destes é algo muito importante de ser mantido. Contudo, tal flexibilidade deverá respeitar os limites do bom senso, que não há como ser medido e/ou especificado. Para isto, caberá muito diálogo com as pessoas responsáveis para que o que for necessário seja feito.

A melhoria esperada está na forma de condução dos projetos pelos profissionais da empresa, pois, havendo este tipo de padronização, a empresa terá como mensurar melhor o custo e o tempo de desenvolvimento de produtos. Não só isto, mas também com a utilização da PFP, os profissionais sentir-se-ão mais seguros e mais confiantes no trabalho que estarão desenvolvendo, podendo gerar um ganho de performance e qualidade no desenvolvimento de projetos.

3.1.5 Prática 4:

Inspeção Final de Fase (IFF):

A IFF é uma prática que deve trabalhar fundamentada na prática 17, onde os itens a serem inspecionados devem ser previamente especificados.

Esta prática tem por objetivo, averiguar se todos os itens pré-definidos para uma determinada fase foram cumpridos satisfatoriamente. Caso não tenham sido, deve-se determinar o motivo e automaticamente providenciar para que as inconsistências detectadas sejam corrigidas.

A aplicação da IFF deve se dar ao final de cada fase, primordialmente antes da realização da prática 4.

A melhoria esperada com a IFF é a extinção de pendências que costumam ficar de uma fase para outra, tornando as fases do processo de desenvolvimento de software mais concisas.

3.1.6 Prática 5:

Classificação e Documentação de Reuniões (CDR):

Em toda empresa, ao se fazer uma reunião costuma-se gerar uma ata. Esta ata geralmente é arquivada ou publicada no portal da empresa. Quando se precisa da informação e principalmente, quando as pessoas lembram que esta informação foi discutida em reunião e que foi gerada uma ata; então procura-se a tal ata para se saber mais sobre o que foi decidido.

Essa prática é bem semelhante à prática 2 (Documentação do Conhecimento), diferenciando-se apenas pelo fato de que na prática 2 (Documentação do Conhecimento) os documentos gerados não são unicamente oriundos de reuniões, pode até ser que sejam, mas não necessariamente. Já nesta prática, toda a documentação gerada, será originária de reuniões e debates que serão promovidos na empresa.

O objetivo desta prática é adicionar um item a mais nas reuniões que são feitas e que são documentas, ou seja, um item classificatório, de modo que venha a facilitar o acesso às informações quando necessário.

A aplicação da CDR dar-se-á a todas as reuniões realizadas com geração de documentação.

A partir desta classificação, a forma como será aplicada a gestão de conteúdo deverá obedecer ao índice classificatório para facilitar a pesquisa quando necessária.

O índice classificatório, como sugestão, poderá ser feito por assunto, por exemplo: Reunião Administrativa, Reunião para Resolução de Problemas de Desenvolvimento de Software, Reunião com Clientes, Reunião com Representantes e assim por diante. Poderá também conter um campo específico para colocação de palavras-chaves que facilitem a pesquisa.

A melhoria esperada com esta prática é proporcionar organização ao acervo de documentos contendo o conhecimento gerado nestas reuniões, para que este não se perca, possibilitando e facilitando sua localização e utilização em outros momentos.

3.1.7 **Prática 6:**

Reunião Inaugural de Fases de Projeto (RIFP):

Diante de uma infinidade de dados e informações, muitas vezes transformados em conhecimento de alto valor agregado, as empresas costumam se perder em seus próprios acertos. Desta forma, diante de tanto conhecimento acumulado, a empresa pode cometer erros que já foram acertados ou não cometidos em projetos anteriores.

A RIFP tem o objetivo de apresentar e conseqüentemente minimizar o acontecimento de erros que já possuem solução conhecida.

A aplicação da RIFP deve ser da seguinte forma: Uma reunião entre os membros participantes de cada fase do projeto. Esta deve acontecer como sendo a primeira etapa de cada fase, onde serão exibidos erros clássicos que podem acontecer no desenvolvimento de cada fase especificamente, e as possíveis práticas que poderão ser aplicadas para melhoria do processo. O conhecimento que será exibido, será proveniente dos que foram registrados com a utilização da prática 2 (Documentação do Conhecimento) e 15 (Classificação e Documentação de Reuniões), não impedindo porém, a manifestação de conhecimento tácito dos integrantes da equipe.

As melhorias esperadas se aplicam diretamente em fornecer conhecimento aos funcionários que entraram recentemente na empresa e também dar a oportunidade aos veteranos, de relembrarem os acontecimentos de projetos anteriores. Desta forma pretende-se minimizar o acontecimento de erros que já possuem solução conhecida.

3.1.8 Prática 7:

Reunião de Finalização de Fase de Projeto (RFFP):

Na RFFP se faz exatamente o oposto do que é feito na prática 3 (Reunião Inaugural de Fases de Projeto), ou seja, enquanto que na RIFP se expõem os erros e acertos ocorridos em determinadas fases de determinados projetos; na RFFP procura-se detectar e registrar os erros e acertos ocorridos, e também possíveis melhorias e/ou correções que possam ser úteis/necessárias nas práticas utilizadas na fase do projeto que está sendo finalizada.

O objetivo desta prática é a preservação da memória e do conhecimento aplicado para resolução de problemas que ainda não foram registrados, bem como a evolução das práticas utilizadas nas fases de desenvolvimento de software.

A aplicação da prática é simples. Ao final de cada fase do projeto, faz-se uma reunião com os membros da equipe para que sejam expostos e registrados, através das práticas 2 (Documentação do Conhecimento) e 15 (Classificação e Documentação de Reuniões), os erros e acertos relevantes que ainda não têm registro e que podem vir a ser úteis em consultas futuras; também as melhorias e/ou correções das práticas, se houverem.

A melhoria esperada com esta prática está na ampliação do acervo de conhecimento da empresa e automaticamente a redução de perda de conhecimento por motivos diversos, mantendo sempre atualizadas as práticas utilizadas nas fases de desenvolvimento.

3.1.9 Prática 8:

Universidade Corporativa (UC):

A UC pode ser vista como uma forma de incentivo ao aprendizado do funcionário, diante do surgimento de diversas tecnologias, quase que diárias. Diante das diversas modificações, a manutenção do conhecimento e o treinamento de pessoal é fator importantíssimo para aquelas empresas que trabalham com tecnologia de ponta.

O objetivo desta prática é ao mesmo tempo treinar, capacitar e, acima de tudo, motivar o funcionário a dar continuidade ao seu aprendizado e também fazê-lo enxergar que ali dentro, da empresa, ele terá a possibilidade de estar sempre renovando e atualizando seu conhecimento, com as práticas e ferramentas mais modernas.

Deve-se estipular um calendário, para que, sem prejudicar o dia-a-dia da empresa, haja, de preferência gratuitamente, treinamento interno dos funcionários nas diversas ferramentas e conhecimentos necessários para que possam prestar serviço com qualidade.

As melhorias esperadas com esta prática são: Aumento da qualidade e produtividade dos funcionários. O que é diretamente bom para a empresa e igualmente para os funcionários. Ao ampliar o conhecimento dos funcionários, automaticamente amplia-se com ele o fator motivação, que é essencial na vida das pessoas e que faz muito bem a qualquer empresa. Esta melhoria será bem-vinda não só para os funcionários, mas também, e principalmente, para os produtos da empresa, que receberão diretamente o reflexo do conhecimento transmitido entre os membros das equipes na UC.

3.1.10 **Prática 9:**

Aposentandos como Professores:

Esta prática, em princípio, pode parecer estranha, mas não é. As pessoas quando estão próximas às aposentadorias, geralmente, carregam consigo uma gama de conhecimento e experiência empresarial invejável.

Geralmente o que as empresas fazem é tentar usufruir ao máximo dos últimos momentos daquele ser, enquanto funcionário da empresa. As empresas costumam dar a ele as mais difíceis tarefas e os mais complicados problemas para serem resolvidos. Isso acabará consumindo o resto de tempo que o sujeito ainda terá como funcionário da empresa.

Fazendo isto, a empresa estará, obviamente, aproveitando o dinheiro que foi investido no funcionário durante toda sua estadia na empresa. Contudo, fazendo isto a empresa também estará usufruindo de seu conhecimento e experiência momentaneamente e, simultaneamente, estará os perdendo para o futuro.

O objetivo desta prática é evitar ao máximo, a perda do conhecimento tácito acumulado pelo funcionário durante toda sua estadia na empresa. Perda esta que dar-se-á com sua aposentadoria; que em muitos casos, pode vir a ser necessária e, caso seja, provavelmente terá um alto custo para a empresa através da contratação de consultoria especializada.

Ao invés de sobrecarregar os aposentandos com tarefas intermináveis, as empresas deveriam monitorar o tempo que lhe resta com tais funcionários e promover a criação de uma equipe para substituição deste.

Sugere-se que com pelo menos um ano de antecedência à aposentadoria do funcionário, este, se ausente totalmente da responsabilidade que lhe é delegada e comece a instruir a equipe de substituição em todas as suas, a partir de então, antigas tarefas. Entenda-se que este ainda será responsável pela equipe e por seus atos, que estarão todos subordinados a ele, ou seja, trabalharam sob sua orientação.

O prazo de um ano foi sugerido, visto que as contas da empresa, parte fiscal, parte contábil, gerencial e etc. podem passar por todas as variações possíveis neste período, tornando-se assim, uma ótima escola para o aprendizado de novatos e, principalmente, para funcionários com experiência no assunto, mas, não tanta quanto a do funcionário que está para se aposentar.

Para a parte administrativa da empresa, este prazo parece ser bem razoável. Contudo, para a área técnica e gerencial talvez esses devam ser re-avaliados.

As melhorias que são propostas são as seguintes: Primeiramente a continuidade de um trabalho que seria interrompido. E em segundo, sendo talvez o mais importante, a economia que será gerada com a manutenção do conhecimento tácito que será transferido de um funcionário para outro, o que dispensará futuras consultorias ou mesmo contratação de outros funcionários gabaritados o suficiente para substituir o que aposentou.

3.1.11 Prática 10:

Portal Corporativo do Conhecimento (PdCC):

Esta prática foi inspirada na leitura dos artigos do Dr. José Cláudio Cyrineu Terra[38,39].

Um PdCC, também chamado de Portal do Conhecimento por algumas organizações, pode ter mil e uma utilidades dentro de uma *softwarehouse*. Diversas

[38] TERRA, José Cláudio Cyrineu. **Fortalecendo Cadeias Produtivas Através de Portais do Conhecimento**, Disponível em: <http://www.terraforum.com.br/lib/pages/biblioteca.php> Acesso em: 15 Set. 2005.
[39] TERRA, José Cláudio. **Portais Corporativos e Gestão de Conteúdo**, Disponível em: <http://www.terraforum.com.br/sites/terraforum/Biblioteca/libdoc00000020v002Portais%20Corporativos%20e%20Gestao%20de%20Conteudo%20.pdf> Acesso em: 16 Mai. 2006.

empresas dentro ou fora do ramo da informática, já aplicaram a idéia às suas realidades e obtiveram resultados extremamente satisfatórios.

O objetivo principal da criação de um PdCC é que ele pode ser usado para gerir, transmitir, armazenar e disponibilizar o conhecimento dentro da organização de forma rápida e prática.

> Portais Corporativos aplicados à Gestão do Conhecimento provêem um ponto central, em muitos casos personalizados, de acesso aos recursos de conhecimento - as bases de dados, sistemas de informação de uma empresa e fontes de conhecimento tácito. Também podem incluir tecnologias avançadas para colaboração virtual, gerenciamento de conteúdo e comércio eletrônico. [...] Eles são mais complexos e encontram sua justificativa no apoio à missão, estratégias e objetivos da organização e colaboram para a criação de um modelo de negócio mais colaborativo e descentralizado.[40]

A aplicação do PdCC não se dá diretamente nos processos aplicados à engenharia de software, e sim como uma ferramenta para disponibilização de outras. O que faz desta, uma prática essencial para aplicação das demais que virão a seguir; tornando-a uma das principais a serem desenvolvidas.

Uma das principais melhorias seria uma mudança radical na estrutura utilizada para gestão do conhecimento na empresa e também a centralização do conhecimento em um só lugar. Com certeza, se a idéia for comprada por todos os funcionários, esta tende a dar bons resultados e trazer grandes benefícios à empresa.

3.1.12 Prática 11:

Zona de Auxílio Imediato (ZAI):

Várias vezes nos encontramos com um problema de prioridade urgente para ser resolvido e não sabemos como fazê-lo, ou estamos com alguma dúvida em algum procedimento que é peça chave para a resolução de todo o problema.

---

[40] TERRA, José Cláudio. **Portais Corporativos e Gestão de Conteúdo**, Disponível em: <http://www.terraforum.com.br/sites/terraforum/Biblioteca/libdoc00000020v002Portais%20Corporativos%20e%20Gestao%20de%20Conteudo%20.pdf> Acesso em: 16 Mai. 2006.

Se o problema tem prioridade urgente e você não consegue resolvê-lo, então uma solução prática seria pedir ajuda a quem possa, saiba ou tenha conhecimento de algo ou alguém que possa auxiliar-te.

O objetivo desta prática é disponibilizar um canal direto aos membros da equipe, para exposição de dúvidas de caráter emergencial, onde todos os funcionários, respeitadas às devidas restrições, receberiam a dúvida ou pedido de ajuda imediatamente e, eventualmente os que se enquadrarem na resolução da dúvida ou pedido de ajuda exposta na ZAI manifestar-se-ão publicando na própria ZAI a solução do problema ou se oferecendo para ajudar na resolução deste.

A ZAI deverá ser utilizada somente em caráter de extrema urgência, pois, é uma ferramenta que quando utilizada, tomará a atenção de diversas pessoas que eventualmente estarão trabalhando, e que interromperão seus afazeres para dar atenção ao problema que estará sendo publicado. Sua aplicação dar-se-á na empresa como um todo. Contudo, uma dúvida publicada pelo departamento de administração, não deve nunca chegar a um membro da equipe técnica, a não ser que seja uma exceção às regras.

A melhoria prevista com a utilização da ZAI é a minimização do tempo de resolução de problemas de caráter emergencial, e também a minimização do desgaste que se tem com a pesquisa de possíveis soluções.

3.1.13 **Prática 12:**

Fórum de Publicação de Dúvidas (FPD):

Muitas vezes nos deparamos com problemas os quais não têm necessidade de resolução imediata, mas que ainda assim, encontramos dificuldade em resolvê-los.

O objetivo desta prática é criar um espaço permanente para os funcionários postarem suas dúvidas, e estas serem publicadas para toda a empresa, respeitadas as devidas restrições, sem o compromisso de resposta imediata.

Esta prática é extremamente semelhante à prática 12 (Zona de Auxílio Imediato), diferenciando-se apenas pelo nível de urgência com o qual se espera uma resposta à dúvida ou problema publicado. Desta forma, deve-se levar em

consideração um fator primordial para a criação desta prática separada ou bem distinta da prática 12 (Zona de Auxílio Imediato), o fator urgência.

Uma dúvida publicada no FPD não vai parar, por exemplo, todo o setor de desenvolvimento de sistemas como uma dúvida publicada na prática 12 (Zona de Auxílio Imediato). Se analisarmos friamente, veremos que, o prejuízo causado por um setor parado ou dedicado a uma única tarefa (neste caso, problema) pode ser enorme de acordo com as proporções da empresa. Para um setor de 10 funcionários talvez não faça tanta diferença, mas para um setor com 200 ou mais funcionários, a diferença pode ser enorme.

O FPD será o canal pelo qual o funcionário poderá expor sua dúvida e obter respostas de outros funcionários sem caráter de urgência. A aplicação desta prática pode ser fomentada através da criação de fóruns de mensagens dentro do PdCC.

A melhoria esperada com a utilização do FPD é a redução de pendências que costumam existir em softwares, e muitas vezes levam muito tempo para serem resolvidas; quando são.

Uma outra melhoria seria proporcionar maior interação entre os membros da equipe para o compartilhamento do conhecimento tácito.

3.1.14 Prática 13:

Informações a Iniciantes:

Devido ao alto grau de rotatividade existente nas *softwarehouses* entre os profissionais de informática, gasta-se muito dinheiro com a contratação e instrução de novos profissionais constantemente. Isso sem contar o tempo de adaptação do funcionário às regras e forma de trabalho praticadas na empresa.

Esta prática é bem parecida com a prática 7 (Padronização das Fases do Projeto) e também com a prática 5 (Aposentandos como Professores), contudo, o objetivo principal é atingir diretamente as pessoas que estão em um estágio inicial como funcionário da empresa, ou seja, novatos.

A aplicação desta prática dar-se-á com a criação de documentos contendo o conhecimento específico que o novato precisa saber para iniciar seus trabalhos na empresa. Funcionará assim, como um mecanismo norteador.

Mas, além disto, deverá abranger não somente a parte de trabalho, mas também orientações de como as coisas funcionam dentro da empresa e os caminhos que deverão ser seguidos para se obter orientações, produtos e serviços que são disponibilizados da empresa para o funcionário.

A melhoria esperada com esta prática seria a redução da perda de tempo e dinheiro, pois, ao contratar um novato, este deverá estar subordinado a um funcionário que será seu professor e que constantemente estará sendo interrompido para o esclarecimento de dúvidas desta pessoa. Isto obviamente trará atrasos no cronograma do funcionário experiente ou um nível maior de stress na sua vida profissional. Ou ainda, caso esse não atenda prontamente ao novato, poderá gerar outras implicações diversas, tanto na vida profissional do novato, quanto na vida profissional dos demais funcionários que estão aguardando mais um companheiro que possa ajudá-los nas tarefas a serem desenvolvidas.

3.1.15 Prática 14:

Documentação do Conhecimento (DoC):

A DoC pode ser considerada uma das partes mais importantes de todas as práticas que aqui serão relatadas, pois ela conterá o conhecimento explicitado pelos funcionários e que auxiliarão em projetos futuros.

Fazendo uma analogia com os *design patterns*, a DoC poderia também ser considerada um item de *design patterns*, pois os diversos documentos que serão gerados comporão soluções pensadas, testadas, aprovadas e publicadas (no escopo empresarial) de problemas que ocorreram em projetos realizados, e que poderão ocorrer em novos projetos caso estas ocorrências não estejam claras nas mentes dos funcionários responsáveis pelo desenvolvimento da fase.

O objetivo da DoC é documentar os erros e acertos que aconteceram nos projetos em que a empresa está ou esteve engajada.

A aplicação da DoC baseia-se no registro, pelos membros da equipe, do tipo do projeto em questão, da fase do projeto, dos fatos que levaram ao acontecimento da inconsistência e da solução apresentada. Para isto pode-se desenvolver um

formulário padrão para que sirva de modelo para o relatório final e ao mesmo tempo, sirva de padrão para os documentos que serão criados.

É válido lembrar que, como diz o ditado: “Uma imagem vale mais do que mil palavras”, assim sendo, quanto mais rico em imagens significativas, for o relatório, melhor e mais fácil poderá ser o entendimento do conhecimento que se pretende explicitar.

A melhoria esperada com esta prática é a de que o conhecimento tácito comece a ser explicitado e compartilhado entre os funcionários e membros das diversas equipes. Assim, na perda de um funcionário, a empresa não perderá seu conhecimento por completo.

3.1.16 Prática 15:

Help Inteligente (HI):

O HI deverá ser construído em uma estrutura de árvore de diretórios, ordenada alfabeticamente, onde serão descritas todas as funcionalidades do software, onde o usuário poderá obter informações sobre o procedimento que procura (desde que o software tenha tratamento para este procedimento), como deve ser feito e onde encontrá-lo no software. O HI nada mais é do que um acervo de conhecimento sobre as ferramentas utilizadas para o completo funcionamento da empresa ou de softwares/ferramentas específicas.

O objetivo do HI é simplesmente o de auxiliar, principalmente os funcionários novatos, na utilização das diversas ferramentas para gestão do conhecimento aplicadas no desenvolvimento do projeto.

Ao tentar utilizar um software específico, por exemplo, um software de controle de versão de aplicativos, o funcionário consultará o HI para obter informações claras, sucintas e precisas sobre os principais recursos que estão disponíveis, e que são utilizados pela empresa em tarefas rotineiras.

3.1.17 Prática 16:

Check List (CL) – O que/Como fazer?

O CL deverá ser criado como forma de orientação dos passos que deverão ser dados para início, continuidade e até mesmo conclusão de tarefas, partes, fases e projetos. Poderá ser criado para cada área, representando o que se deve fazer ao iniciar algum tipo de trabalho. Conterá apenas os tópicos e não o conhecimento do que se deve fazer.

O objetivo desta prática é criar um mecanismo orientador. Este será um recurso útil, principalmente para tarefas que não são realizadas constantemente. Este tipo de mecanismo pode auxiliar tanto os funcionários experientes quanto os novatos.

A aplicação desta prática dar-se-á no projeto como um todo, ou seja: Para especificação de partes de projetos, especificação de fases, ou ainda especificação de tarefas que são compostas ou subdivididas em diversas outras. Funcionando, como já foi exposto no parágrafo anterior, como um mecanismo de orientação ao que deve ser feito.

A melhoria esperada com a utilização da CL é proporcionar maior segurança aos funcionários em relação ao que deve ser feito e simultaneamente a padronização do processo de produção.

Quando seguro de si, o ser humano trabalha com mais tranqüilidade, o que automaticamente gera um melhor resultado.

Exemplo:

Check List – Requisitos:

- Entrevista com o cliente
- Levantamento de requisitos
- Estudo de viabilidade

...

- Especificação formal

3.1.18 Prática 17:

FAQ (*Frequency Asked Questions*):

A FAQ é um recurso muito conhecido no mundo da informática e que é utilizado por milhares de pessoas no mundo inteiro, principalmente na Internet. FAQ nada mais é do que uma lista de questões com suas respectivas respostas, que aparecem constantemente na vida das pessoas a respeito de um serviço ou produto específico.

O objetivo desta prática é proporcionar um acesso rápido ao conhecimento básico que deve ser conhecido por todos, e que frequentemente é utilizado/solicitado.

A aplicação da FAQ dar-se-á principalmente nos processos complexos, que são difíceis de serem memorizados e que são de uso constante dos procedimentos rotineiros da empresa.

A melhoria esperada com esta prática é o aumento na eficiência para obtenção do conhecimento necessário, por parte dos funcionários, para conclusão/continuação das tarefas que estiverem sendo executadas.

3.1.19 Prática 18:

Mapa do Conhecimento (MC):

O MC já é uma prática conhecida por algumas empresas. Contudo, ainda assim, é pouco utilizado e às vezes acaba caindo no esquecimento da mesma forma que acontece com a documentação de sistemas.

O objetivo da utilização e manutenção do MC é prover a empresa com informação sobre quem possui que tipo de conhecimento e, consequentemente saber quando e quem deverá ser acionado em questões específicas.

> A utilização de mapas de conhecimento em organizações para a localização de ‘pessoas fonte’ que tem o conhecimento chave em um determinado assunto, tem se mostrado uma ferramenta muito útil. Tornar público quem sabe o quê na organização e facilitar o contato entre quem tem necessidade de conhecimento e quem pode ter as respostas adequadas é um dos objetivos do mapa. Independentemente da forma adotada (desde mapas reais que indicam onde estão pessoas com determinadas competências, até páginas amarelas numa intranet), vale destacar que um mapa de conhecimento assinala onde está o conhecimento, mas não o

> contém. O mapa de conhecimento se torna um guia, não um armazém.[41]

Esta é uma ferramenta que deve existir e ser utilizada somente por pessoas estrategicamente aptas para terem acesso a este tipo de informação, tais como líderes de equipes e etc. Assim, estas pessoas saberão para quem poderão enviar tarefas específicas de acordo com o conhecimento especificado no mapa, aumentando a eficiência de suas atividades.

As melhorias esperadas com a utilização do MC são: A redução de tráfego de tarefas entre pessoas, visto que estas serão direcionadas às pessoas que possuem o conhecimento necessário para solucioná-las.

Proporcionar aos gestores, informação suficiente para que possam tomar decisões sobre novos projetos, como por exemplo, quais recursos estão disponíveis e quais deverão ser contratados como mão-de-obra externa.

3.1.20 Prática 19:

Disponibilização de Diagramas:

Os sistemas que são construídos baseados em metodologias de desenvolvimento de aplicativos possuem geralmente vários diagramas.

Estes diagramas que se constroem antes do desenvolvimento do sistema, às vezes são pouco utilizados e acabam se perdendo ou ficam desatualizados. E o que era pra ser um facilitador acaba tornando-se um monte de dados sem valor devido à falta de atualização ou mesmo de esquecimento.

O objetivo desta prática se traduz na disponibilização destes diagramas para as pessoas as quais estes têm utilidade, pois, facilitam extremamente a compreensão e manipulação da lógica aplicada ao sistema, principalmente para os novatos.

A aplicação desta prática seria em todas as fases da engenharia de software onde se tenha e se faça necessária a sua utilização, sendo que a principal fase seria

---

[41] OLIVEIRA, Carla. **Mapas de Conhecimento.** Disponível em: < http://www.informal.com.br/pls/portal/docs/PAGE/GESTAODOCONHECIMENTOINFORMALINFORMATICA/ARTIGOSGESTAODOCONHECIMENTO/ARTIGOSGC/21082004_0.PDF>. Acesso em: 09 mai. 2006.

a de desenvolvimento, onde estes seriam não somente consultados, mas também criados e constantemente atualizados através de ferramentas CASE[42].

A idéia é: Depois de criados ou atualizados, esses gráficos ou diagramas seriam transformados em uma imagem (arquivo do tipo Bitmap[43], GIF[44] ou Jpeg[45], por exemplo.) e seriam disponibilizados através da prática 9 para toda a empresa ou restritamente às pessoas relevantes.

A melhoria esperada com esta prática seria uma facilitação de acesso a essas informações por parte de todas as pessoas envolvidas com o desenvolvimento do software em questão. Atualmente para se ter acesso a este tipo de informação, quando ela existe, é necessário se instalar uma ferramenta CASE no computador onde se pretende visualizar os gráficos ou diagramas. Além disto, ainda existe a necessidade de se ter o conhecimento prévio do funcionamento da ferramenta para se visualizar as informações desejadas.

3.1.21 Prática 20:

Avaliação e Revitalização das Práticas (ARP):

Nenhum processo ou metodologia criada é totalmente ideal ou aplicável a todas as empresas. Cada empresa possui uma necessidade, uma particularidade e uma visão diferente para negócios semelhantes.

A ARP tem o objetivo de promover a flexibilização no processo e na maneira de utilização das práticas propostas, de modo que estas possam evoluir, aumentar

[42] Software que possibilita a construção de diagramas para a área de informática, geralmente como um todo. (Nota do autor)
[43] Uma imagem raster ou bitmap - em tradução literal "mapa de bits" - é um padrão de representação de imagens formado por uma grade geralmente retangular de pontos de cor, ou pixels (do inglês, *picture element*) em um monitor de computador, papel ou outro dispositivo, ou mesmo em meios não visuais, como por exemplo, na memória RAM de um computador ou em disco magnético, sob a forma de arquivos. (BITMAP, Wikipédia – A enciclopédia livre, Disponível em: <http://pt.wikipedia.org/wiki/Bitmap> Acesso em: 29 abr. 2006)
[44] GIF (*Graphics Interchange Format*, que se pode traduzir como "formato para intercâmbio de gráficos") é um formato de imagem de mapa de bits muito usado na world wide web, quer para imagens fixas, quer para animações. (GIF, Wikipédia – A enciclopédia livre, Disponível em: <http://pt.wikipedia.org/wiki/GIF> Acesso em: 29 abr. 2006)
[45] Em informática, JPEG (pronuncia-se "jay-peg") é a sigla de Joint Photographic Experts Group, tratando-se de um formato de compressão, com perda de dados, aplicado em imagens fotográficas. A perda de dados é proporcional ao fator de compressão desejado. O arquivo (ou ficheiro) que usa este método de compressão é chamado normalmente por JPEG; as extensões de arquivos para este formato são .jpeg , .jfif , .jpe e .jpg , este último, o mais comum. (JPEG, Wikipédia – A enciclopédia livre, Disponível em: <http://pt.wikipedia.org/wiki/JPEG> Acesso em: 29 abr. 2006)

ou diminuir e se modificar de maneira que atenda às necessidades da empresa e das equipes, sem desviar do propósito para o qual estas foram criadas.

A aplicação da ARP dar-se-á na finalização de cada projeto, através da avaliação da documentação gerada nas RFFP's.

A melhoria esperada pela ARP é a manutenção contínua das práticas propostas, para que estas não fiquem obsoletas.

3.2 Apresentação das Práticas de Gestão do Conhecimento no Contexto de Desenvolvimento de Software

Daqui para frente serão apresentadas as práticas, descritas acima, dentro do contexto da metodologia de desenvolvimento de software utilizada no desenvolvimento deste trabalho. Para isto, destacar-se-ão as práticas genéricas que deverão ser aplicadas em todas as fases do desenvolvimento de software. As práticas específicas serão apresentadas subdivididas em suas respectivas fases; logo em seguida serão apresentadas as práticas auxiliares.

A seguir, algumas considerações que devem ser feitas para que o processo proposto possa funcionar em perfeita harmonia com o ambiente em que atuará:

- A prática 10 (Portal Corporativo do Conhecimento) propõe a utilização de uma ferramenta importantíssima para a gestão do conhecimento, ou seja, o PdCC. A maioria das práticas descritas são viavelmente aplicáveis em um PdCC, sendo que algumas são de uso exclusivo neste ambiente, inclusive a parte de gestão de conteúdo que deverá ser aplicada diretamente nas práticas 14 (Documentação do Conhecimento) e 5 (Classificação e Documentação de Reuniões).

- Para que as práticas acima funcionem, é necessário que previamente, antes do início dos trabalhos, a prática 3 (Padronização das Fases do Projeto) seja totalmente formalizada por uma equipe de profissionais competentes, no desenvolvimento de sistemas baseados na metodologia adotada pela empresa.

- Faz-se necessário também, que sejam definidos os CL's para que possam ser aplicados na prática 4 (Inspeção Final de Fase). Esses CL's, provavelmente

deverão ser definidos pela mesma equipe que definiu as PFP's da prática 3 (Padronização das Fases do Projeto).

- As práticas 13 (Informações a Iniciantes) e 15 (*Help* Inteligente) para que tragam bons resultados, deverão ser criadas e atualizadas constantemente, acompanhando as modificações realizadas no ambiente de trabalho seja por motivo de evolução das ferramentas utilizadas, mudanças na metodologia ou nas normas da empresa. Esta prática também deve ser especificada pela mesma equipe que definir a prática 3 (Padronização das Fases do Projeto), auxiliada pela equipe de analistas/programadores, se esta já não os for.

- As práticas 11 (Zona de Auxílio Imediato), 12 (Fórum de Publicação de Dúvidas), 13 (Informações a Iniciantes), 14 (Documentação do Conhecimento), 15 (*Help* Inteligente), 16 (*Check List* – O que/Como fazer?), 17 (FAQ (*Frequency Asked Questions*)), 18 (Mapa do Conhecimento) e 19 (Disponibilização de Diagramas), têm sua aplicação efetiva quando utilizadas como ferramentas disponibilizadas pela prática 10 (Portal Corporativo do Conhecimento). Sendo que, algumas destas práticas, são totalmente submissas à existência do PdCC, para que sejam viáveis suas aplicações em ambiente empresarial. Outras práticas também podem ser adicionadas ao PdCC, como por exemplo as práticas 3 (Padronização das Fases do Projeto) e 5 (Classificação e Documentação de Reuniões), tornando a ferramenta um órgão centralizador de todas as demandas de conhecimento da empresa.

3.2.1 Práticas Genéricas[46]

- Todas as fases de desenvolvimento do software deverão ser iniciadas pelas seguintes práticas obrigatórias: 6 (Reunião Inaugural de Fases de Projeto) e 3 (Padronização das Fases do Projeto), nesta ordem.

[46] As práticas genéricas são aquelas utilizadas em quaisquer das fases do desenvolvimento de software, estas são divididas em: Obrigatórias e Facultativas. Onde as obrigatórias, como o próprio nome já diz, deverão ser utilizadas obrigatoriamente conforme especificado. Já as facultativas, somente serão utilizadas quando necessárias. (Nota do autor)

- Todas as fases de desenvolvimento do software deverão ser finalizadas pelas seguintes práticas obrigatórias: 4 (Inspeção Final de Fase) e 7 (Reunião de Finalização de Fase de Projeto), nesta ordem.

- Em todas as fases de desenvolvimento do software, a aplicação da prática 14 (Documentação do Conhecimento) (prática facultativa) deverá ser feita à medida da conveniência, ou seja, quando necessário.

- Se houver novatos na equipe, utilizar as práticas 13 (Informações a Iniciantes) e 15 (*Help* Inteligente) (práticas facultativas).

3.2.2 Requisitos

- Aplicar a prática 2 (Especificação Formal) após a conclusão da especificação dos requisitos.

3.2.3 Projeto / Sistemas Críticos

- À medida que os diagramas do sistema ficarem prontos, ou após alguma atualização nestes, dever-se-á aplicar a prática 19 (Disponibilização de Diagramas).

- Para cada tarefa que exija atualizações diversas ou sub-tarefas, a prática 16 (*Check List* – O que/Como fazer?) deverá ser exigida.

3.2.4 Desenvolvimento de Software

- A prática 1 (*Design Patterns*) deve ser aplicada, sempre que possível, na construção do software.

- As práticas de auxílio, tais como: 11 (Zona de Auxílio Imediato), 12 (Fórum de Publicação de Dúvidas), 18 (Mapa do Conhecimento) e 17 (FAQ (*Frequency Asked Questions*)) podem ser utilizadas a qualquer momento que forem necessárias.

- Para cada tarefa que exija atualizações diversas ou sub-tarefas, a prática 16 (*Check List* – O que/Como fazer?) deverá ser exigida e possivelmente a prática 19 (Disponibilização de Diagramas) também.

3.2.5 Verificação e Validação

- Para cada tarefa que exija atualizações diversas ou sub-tarefas, a prática 16 (*Check List* – O que/Como fazer?) deverá ser exigida.

Finalizadas todas as fases de desenvolvimento do software, é hora de partir para a prática 20 (Avaliação e Revitalização das Práticas). Esta prática é de caráter essencial nesta proposta de práticas para melhoria do desenvolvimento de software.

Tem-se em mente que nenhum processo é totalmente correto ou totalmente aplicável a todos os tipos de empresa, mesmo por que, por mais semelhantes que sejam, ainda assim, cada empresa possui suas particularidades.

Por estes motivos, como já especificado anteriormente, a prática 20 (Avaliação e Revitalização das Práticas) foi criada para funcionar como um mecanismo flexibilizador, para adaptação e evolução desta proposta às reais necessidades e/ou particularidades de cada empresa.

O mundo do desenvolvimento de software é extremamente dinâmico. A evolução das técnicas, das ferramentas utilizadas e da forma de se trabalhar o conhecimento, se dá a todo o momento. Portanto, a adição, subtração e/ou evolução de práticas devem existir sempre e, principalmente, serem estimuladas pelas empresas para que seus processos estejam sempre atualizados, nunca obsoletos ou ultrapassados.

3.2.6 Práticas Auxiliares[47]

A prática 18 (Mapa do Conhecimento) é uma das que devem ser mais incentivadas e trabalhadas na empresa, para que não fique obsoleta, pois esta é uma ferramenta de grande valia e com diversas aplicabilidades.

[47] Denominou-se como práticas auxiliares, aquelas que não são aplicadas diretamente nas fases da metodologia de desenvolvimento de software. (Nota do autor)

Ao se ter a noção do conhecimento que cada funcionário possui, a empresa terá noção de seu potencial. Daí podem surgir negócios e oportunidades que nunca seriam possíveis sem o auxílio ou a existência desta prática.

A prática 9 (Aposentandos como Professores), como todas as outras aqui propostas, é de caráter importantíssimo para a empresa. Contudo, diferente das demais, esta tem sua particularidade, como já foi relatado em sua descrição. Esta particularidade que é correr contra o tempo para não perder conhecimento, é de fato algo pelo qual as empresas devem zelar e se esforçar de unhas e dentes. Pois, após a perda do conhecimento para o mercado ou para uma aposentadoria, como já foi dito anteriormente, o mesmo só voltará à empresa através de caríssimas horas de consultoria externa ou na contratação de um outro profissional a peso de ouro.

Uma sugestão, é que a aplicação da prática 9 (Aposentandos como Professores) fique a cargo do departamento pessoal, que pode arranjar para que, no prazo certo, a orientação do profissional que está prestes a se aposentar, comece.

A prática 8 (Universidade Corporativa) já é bem mais ampla e pode abranger a empresa como um todo. Todos os profissionais poderão, dentro de suas capacidades, ministrar cursos para os demais profissionais da empresa que tenham interesse.

Esses cursos devem ser planejados e bem estruturados, para que se faça um trabalho de primeira classe e desperte o interesse dos funcionários em estar participando destes.

Ao crescer intelectualmente, empresa e funcionário só têm a ganhar. Portanto, esta é uma prática que pode trazer grandes benefícios para todos. Os horários e como serão feitos os cursos, cabe a cada empresa elaborar o que melhor lhe convir.

E para finalizar, é importante também que, nestas práticas que não fazem parte do desenvolvimento de software diretamente aplicada à metodologia, também se faça uma avaliação para sua manutenção, atualização e/ou até mesmo sua extinção. Para tanto, fica a sugestão de que, da mesma forma que é feito para cada fase do processo de desenvolvimento, também seja feito para estas práticas auxiliares. Ou seja, tomemos como exemplo a prática 8 (Universidade Corporativa), antes de começar um curso da UC pode-se fazer uma espécie de RIFP, voltada exclusivamente para o curso e após o término do mesmo, pode-se concluir com uma RFFP, para que este seja avaliado pelos participantes.

Para a manutenção do curso, sugere-se uma ARP, que deverá ser aplicada em cima do material gerado a partir das RFFP’s.

Conclusão

Através deste trabalho, conclui-se que com o avanço da tecnologia e da necessidade de informação vivida na atualidade, os processos de desenvolvimento de software não podem ser tratados artesanalmente, como era e ainda vem sendo feito por muitas empresas.

Para se manterem e progredirem neste mercado crescente e cada vez mais competitivo, as *softwarehouses* deverão tratar o conhecimento de forma profissionalizada, estudando e mapeando os seus principais pontos de geração de conhecimento e, nestes promover o monitoramento para que o conhecimento produzido seja tratado e registrado para que não se perca no tempo.

Contudo, tal abordagem só será possível se as *softwarehouses* criarem/adotarem e mantiverem fielmente uma metodologia de desenvolvimento de software.

A partir da metodologia adotada pela *softwarehouse*, e da aplicação das práticas sugeridas neste trabalho, os benefícios esperados são:

- A melhoria na qualidade do produto gerado ao final de todo o processo. Visto que as práticas propostas vão desde a parte de levantamento de requisitos de softwares, passando pela parte de projeto, desenvolvimento e homologação do produto final.

- Criação/manutenção de um acervo do conhecimento adquirido, gerado e trabalhado no dia-a-dia da empresa, sendo de caráter essencial para que a empresa cresça, fortaleça-se e mantenha-se em bases sólidas de conhecimento que serão gerados, adquiridos, atualizados, registrados e transmitidos.

Este conjunto de melhorias visa promover o engrandecimento da organização e da equipe que a compõe. À medida que o conhecimento for sendo armazenado, de forma prática e ordenada de modo a promover sua fácil recuperação quando necessário, a empresa estará, ao mesmo tempo mantendo sua história e gerando uma gama de informações importantíssimas para a criação e manutenção de seus produtos.

Conclui-se também, que os funcionários, peça-chave para o sucesso da empresa, deverão trabalhar com afinco na implementação e manutenção de todas as ações propostas, e que sem a sua colaboração e comprometimento, nenhuma melhoria, qualquer que seja, trará resultado significativo.

Bibliografia

BITMAP, Wikipédia – A enciclopédia livre, Disponível em: <http://pt.wikipedia.org/wiki/Bitmap> Acesso em: 29 abr. 2006.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML Guia do Usuário.** Rio de Janeiro: Campus, 2000.

CAPABILITY MATURITY MODEL, Ministério da Ciência e Tecnologia, Disponível em: <http://www.mct.gov.br/SEPIN/Dsi/qualidad/CMM.htm> Acesso em: 04 mar. 2006.

CARVALHO, Ana Elizabete Souza de; TAVARES, Helena Cristina; CASTRO, Jaelson Brelaz.
Uma Estratégia para Implantação de uma Gerencia de Requisitos Visando a Melhoria Dos Processos de Software. Disponível em: <http://www.inf.puc-rio.br/~wer01/Pro-Req-3.pdf>. Acesso em: 15 abr. 2006.

CARVALHO, Rodrigo Baroni. **Aplicações de Software de Gestão do Conhecimento: tipologia e usos.** Disponível em: <http://www.eci.ufmg.br/pcionline/viewarticle.php?id=320>. Acesso em: 09 mai. 2006.

DAVENPORT, Thomas H. **Conhecimento Empresarial**: Como as organizações gerenciam o seu capital intelectual. Rio de Janeiro: Campus, 1998.

ESTIMANDO O VALOR DE EMPRESAS: A importância e os desafios de mensuração dos ativos intangíveis, Biblioteca TerraForum Consultores. Disponível em: <http://www.terraforum.com.br/lib/pages/viewdoc.php?from=map&l_intDocCod=233> Acesso em: 15 set. 2005.

FIGUEIREDO, Carlos et al. **Especificação Formal de Software.** Disponível em: <http://paginas.fe.up.pt/~ei99030/trabalhos/EspecificacaoFormaldeSoftware.pdf>. Acesso em: 25 fev. 2006.

GATTONI, Roberto Luís Capuruço. *Gerência da Tecnologia da Informação*: Gerência de Projetos em Tecnologia da Informação. FUMEC; Belo Horizonte; MG; 2005; p. 58.

GIF, Wikipédia – A enciclopédia livre, Disponível em: <http://pt.wikipedia.org/wiki/GIF> Acesso em: 29 abr. 2006.

JPEG, Wikipédia – A enciclopédia livre, Disponível em: <http://pt.wikipedia.org/wiki/JPEG> Acesso em: 29 abr. 2006.

LE COADIC, Yves-François. **A Ciência da Informação.** Brasília, 2003. Disponível em: <http://www.briquetdelemos.com.br/editora/biblio16.htm>. Acesso em: 28 ago. 2005.

MALDONADO, José Carlos; ROCHA, Ana Regina Cavalcanti da; **Qualidade de Software** : Teoria e Prática. São Paulo: Prentice-Hall, 2001.

MELHORIA DE PROCESSO DO SOFTWARE BRASILEIRO, SOFTEX, Disponível em: <http://www.softex.br/cgi/cgilua.exe/sys/start.htm?sid=191> Acesso em: 04 mar. 2006.

Michaelis Pequeno Dicionário Inglês-Português/Português-Inglês. 50. ed. São Paulo: Melhoramentos, 1995. 794 p.

O que são e de onde vieram os Design Patterns?, [s. n.]. Disponível em: <http://www2.fundao.pro.br/articles.asp?cod=144> Acesso em: 25 abr. 2006.

OLIVEIRA, Carla. **Mapas de Conhecimento.** Disponível em: < http://www.informal.com.br/pls/portal/docs/PAGE/GESTAODOCONHECIMENTOINFORMALINFORMATICA/ARTIGOSGESTAODOCONHECIMENTO/ARTIGOSGC/21082004_0.PDF>. Acesso em: 09 mai. 2006.

PAULA FILHO, Wilson de Pádua. **Engenharia de Software** : Fundamentos, Métodos e Padrões, 2ª. ed., Rio de Janeiro: LTC Editora, 2001.

RUS, Ioana; LINDVALL, Mikael. Knowledge Management in Software engineering. **IEEE Software.** Maryland, v. 2, n. 0740-7459, p. 26-38, maio/jun. 2002.

RUS, Ioana; LINDVALL, Mikael; SINHA, Sachin Suman. **Knowledge Management in Software Engineering.** Disponível em: <http://192.73.45.130/techs/kmse/kmse.pdf>. Acesso em: 20 fev. 2006.

SOARES, Kely Teixeira; RIBEIRO, Lílian; RODRIGUES, Walace de Almeida; **Uma Hierarquia de Classes para Construção de Autômatos Finitos.** Disponível em: <http://www.dcc.ufla.br/infocomp/artigos/v4.3/art10.pdf> Acesso em: 19 set. 2005.

SOFTWARE ENGINEERING INSTITUE, Wikipédia – A enciclopédia livre, Disponível em: <http://pt.wikipedia.org/wiki/Software_Engineering_Institute> Acesso em: 23 mai. 2006.

SOMMERVILLE, Ian. **Engenharia de Software.** São Paulo: Addison-Wesley, 2003.

TERRA, José Cláudio Cyrineu, **Fortalecendo Cadeias Produtivas Através de Portais do Conhecimento**, Disponível em: <http://www.terraforum.com.br/lib/pages/biblioteca.php> Acesso em: 15 Set. 2005.

TERRA, José Cláudio, **Portais Corporativos e Gestão de Conteúdo**, Disponível em: <http://www.terraforum.com.br/sites/terraforum/Biblioteca/libdoc00000020v002Portais%20Corporativos%20e%20Gestao%20de%20Conteudo%20.pdf> Acesso em: 16 Mai. 2006.

UM Software Assistente para Especificação de Regras de Negócio, [s.n.]. Disponível em: <http://www.inf.ufrgs.br/~kroth/papers/Klinger-CBCOMP.pdf> Acesso em 18 set. 2005.