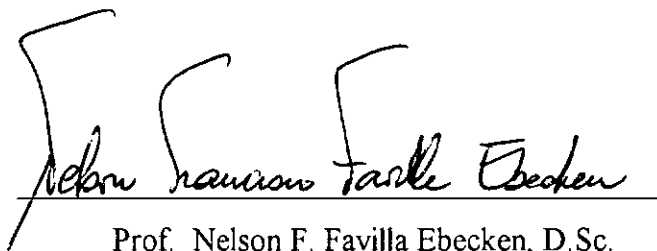


**ESTUDO DA QUALIDADE DE UM SISTEMA BASEADO
EM CONHECIMENTO PARA A ENGENHARIA**

Darlene Figueiredo Borges

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA CIVIL.

Aprovada por:

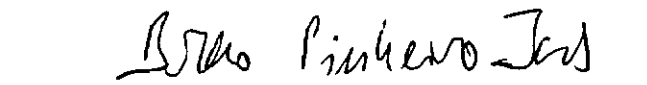


Prof. Nelson F. Favilla Ebecken, D.Sc.

(Presidente)



Eng. José Antonio Bogarin Geymayr, D.Sc.



Eng. Breno Pinheiro Jacob, D.Sc.

BORGES, DARLENE FIGUEIREDO

Estudo da Qualidade de um Sistema Baseado em Conhecimento para a Engenharia [Rio de Janeiro] 1995

VIII, 110 p. 29,7cm (COPPE/UFRJ), M.Sc., Engenharia Civil , 1995

Tese - Universidade Federal do Rio de Janeiro

1. Sistema Baseado em Conhecimento

2. Qualidade Sistemas Especialistas

3. Qualidade de “software”

3. Performance e Qualidade

4. Avaliação

I. COPPE/UFRJ

II Título(série)

Aos meus pais, Oderlino e Darcy

À minha tia, Aída

Aos meus avós

AGRADECIMENTOS

Ao professor Nelson Francisco Favilla Ebecken pela orientação, pela total colaboração, pelo incentivo e especialmente, pela amizade.

Aos meus pais pela educação, pela formação, pela dedicação, pelo carinho e especialmente pelo amor.

A minha tia Aída pela educação, pela dedicação e principalmente pelo carinho e compreensão.

Aos amigos Márcia Gottgroy e Paulo César Gottgroy pelo irrestrito apoio, incentivo, colaboração e amizade, por tudo, um especial obrigado.

Aos amigos Mônica Caruso e Ricardo Padilha pela amizade, pela força e colaboração irrestrita para a realização deste trabalho.

Aos amigos Mário Ribeiro, Edson Teixeira, Lenora Menezes, Guilherme Terra e Valda Jassus pelo total apoio, pela força, pelo incentivo e pela amizade.

À minha família e aos sinceros amigos, pelo carinho, pela compreensão e pela amizade.

A todos os colegas da UFRJ, pela colaboração, pela amizade e pelos ótimos momentos compartilhados.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.).

ESTUDO DA QUALIDADE DE UM SISTEMA BASEADO EM CONHECIMENTO PARA A ENGENHARIA

Darlene Figueiredo Borges

MARÇO, 1995

Orientador: Nelson Francisco Favilla Ebecken

Programa: Engenharia Civil

Os sistemas baseados em conhecimento tiveram um rápido crescimento nesta última década, e a necessidade de avaliação da qualidade tem demonstrado ultimamente que uma maior atenção deve ser dada nessa área. Este tópico vem se tornando cada vez mais importante e tem recebido uma grande atenção, tanto de forma teórica como prática.. Entretanto, avaliar um SBC é uma tarefa muito complexa.

O objetivo básico deste trabalho é apresentar os principais enfoques utilizados na tentativa de se analisar a performance e qualidade de um SBC. A principal proposta é mostrar uma visão da qualidade de um sistema baseado em conhecimento, incluindo a definição de um conceito sobre performance e qualidade, uma avaliação geral de uma metodologia para se desenvolver um SBC e um conjunto de critérios para suportar a sua aplicação prática.

Procurou-se ainda definir vários conceitos envolvidos na área de qualidade de “software”, assim como descrever métodos e técnicas que vem sendo utilizados, focalizando desse modo diretrizes para a sua utilização, de forma a contribuir para uma melhor aplicação dos mesmos e conseqüentemente no crescimento da eficiência do produto de “software”.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

**STUDY OF THE QUALITY OF A KNOWLEDGE BASED
SYSTEM FOR ENGINEERING**

Darlene Figueiredo Borges

MARCH, 1995

Thesis Supervisors : Nelson Francisco Favilla Ebecken

Department : Civil Engineering

The Knowledge Based Systems (KBS) had a quick growth over the last decade, and the need for an evaluation of its quality lately demonstrated that greater attention must be payed to this area. Quality is a topic wich is becoming increasingly important and has been given greater attention, both teorically and pratically. To evaluate a KBS is, however, a very complex task.

The primary scope of this work is to present the major approaches used, in order to analyse the performance and the quality of a KBS. Our major proposal is to present a general approach of the quality of a KBS, including the definition of a concept of performance and quality, a general evaluation of methodology to develop a KBS and a set of criteria to support its practical application.

We also aimed to define a number of concepts involved in the software quality area, as well as describing methods and techniques wich have been used to focus a guideline for their utilization, contributing to a better application of such methods and techniques and, as a consequence, to obtain an improvement of the effectiveness of the software product.

ÍNDICE

Capítulo I - Introdução	1
I.1 - Qualidade - Histórico	1
I.2 - Conceito de Qualidade	10
I.3 - Dimensões Múltiplas de Qualidade	12
I.4 - Organização da Tese	15
 Capítulo II - Qualidade de “Software”	 16
II.1 - Introdução	16
II.2 - O Que é um Sistema de Qualidade?	16
II.3 - Perfil das Análises de Qualidade de “Software”	18
II.3.1 - Intenções da Filosofia de Qualidade em “Software”	18
II.3.2 - Visão da “Gerência de Qualidade Total” em Desenvolvimento de “Software”	23
II.3.3 - Melhoria da Qualidade de “Software”	26
II.3.4 - Evolução do “Software”	30
II.3.5 - Importância da Padronização da Qualidade de “Software”	38
II.3.6 - Análise das Características de Qualidade de “Software”	41
II.3.7. - Análise das Influências de Falhas de “Software”	43
II.4 - Comentários	50
 Capítulo III - Qualidade de Sistema Especialista	 51
III.1 - Introdução	51
III.2 - Segurança da Qualidade	53
III.3 - Avaliação de um Sistema Especialista	55
III.4 - Validação de um Sistema Especialista	61
III.5 - Verificação de um Sistema Especialista	64
III.6 - Comentários	83

Capítulo IV - Estudo de Caso	84
IV.1 - Introdução	84
IV.2 - Descrição do Sistema	85
IV.3 - Ciclo de Vida do Sistema Programador	89
IV.3.1 - Fase 1 - Estudo das Necessidades	90
IV.3.2 - Fase 2 - Construção de um Sistema de Demonstração	92
IV.3.3 - Fase 3 - Desenvolvimento do Protótipo	92
IV.3.4 - Fase 4 - Implementação e Instalação do Sistema	99
IV.3.5 - Fase 5 - Manutenção e Extensão	107
IV.4 - Comentários	107
Capítulo V - Conclusões	109
Referências Bibliográficas	112

CAPÍTULO I

INTRODUÇÃO

I. 1 - Qualidade - Histórico

A qualidade é um conceito notavelmente escorregadio, de fácil visualização, mas exageradamente difícil de se definir. É um motivo de grande confusão para os gerentes, levando a alegação frequente, mas vazia: sei o que ela é quando a vejo. Além do mais, mesmo quando a qualidade é definida com precisão, os programas carecem de um impacto na concorrência. Muitos programas tem se concentrado restritamente na fábrica ou tem confiado basicamente em métodos tradicionais de controle da qualidade. Tem-se prestado pouca atenção às fontes que explicam a qualidade superior: as contribuições relativas do projeto dos produtos, da seleção e gerenciamento de fornecedores e do gerenciamento da produção e da força de trabalho. O que tem dominado são os instrumentos e as técnicas, perseguindo-se, muitas vezes, projeto de melhoria a curto prazo às custas do planejamento da qualidade a longo prazo. Os vínculos com a estratégia de concorrência tem sido poucos e distanciados.

Como conceito, a qualidade é conhecida há milênios. Só recentemente é que ela surgiu como função de gerência formal. Em sua forma original, ela era relativa e voltada para a inspeção; hoje as atividades são consideradas essenciais para o sucesso estratégico.

Segundo David Garvin [referência 6], a evolução da qualidade aconteceu em 4 eras: a da inspeção, a do controle estatístico, a da garantia e a da gestão estratégia, sendo que o próprio Garvin é o precursor desta última.

Inspeção

Nos séculos XVIII e XIX, quase tudo era fabricado por artesãos ou trabalhadores experientes. Produziam-se pequenas quantidades de cada produto; as peças eram ajustadas umas às outras manualmente e a inspeção era feita, após os produtos prontos, para assegurar uma alta qualidade. Era informal, quando feita.

A inspeção formal só passou a ser necessária com o aparecimento da produção em massa e a necessidade de peças intercambiáveis. Com o aumento dos volumes de produção, as peças não podiam ser encaixadas uma nas outras manualmente: o processo exigia um grande grupo de mão de obra qualificada, era oneroso e demorado. Os preços eram altos, principalmente no caso das máquinas e equipamentos.

No início do século XIX apareceu um sistema racional de medidas, gabaritos e acessórios que contribuíram para produzir “modelo-padrão” do produto e assegurar a sua uniformidade. O controle de qualidade passou muitos anos limitado à inspeção e a atividades restritas, como a contagem, a classificação pela qualidade e os reparos.

Controle Estatístico da Qualidade

O ano de 1931 representou um marco no movimento da qualidade. Shewhart deu uma definição precisa e mensurável de controle de fabricação, criou poderosas técnicas de acompanhamento e avaliação da produção diária, e propôs diversas maneiras de se melhorar a qualidade.

Controle de Processo

A abertura inicial foi feita por Shewhart. Ele reconheceu que a variabilidade era um fato concreto na indústria e que ela seria entendida por meio dos princípios da probabilidade e da estatística.

A questão não era mais a existência de variação, mas como distinguir as variações aceitáveis das flutuações que indicassem problemas. Toda a análise derivou do conceito de controle estatístico de Shewhart:

“Dir-se-á que um fenômeno está sob controle quando, recorrendo-se à experiência passada, se puder prever, pelo menos dentro de certos limites, como o fenômeno deve variar no futuro. Entende-se , aqui , que previsão significa que se possa determinar, pelo menos aproximadamente, a probabilidade de que o fenômeno observado fique dentro de determinados limites.”

Amostragem

As técnicas de amostragem partem da premissa simples de que uma inspeção de 100% é uma maneira ineficiente de se separar os bons produtos dos maus.

Uma alternativa clara é verificar um número limitado de produtos de um lote de produção, e depois decidir se o lote é aceitável ou não. Mas este processo envolve certos riscos, tendo em vista que as amostras nunca são inteiramente representativas.

Com o advento da Segunda Guerra Mundial e com a necessidade de produzir armas em grande escala é que os conceitos de controle estatístico passaram a ter um público maior.

No fim dos anos 40, o controle da qualidade era reconhecido. Seus métodos eram, porém basicamente estatísticos e seu impacto confinou-se em grande parte as fábricas.

Nos anos de 50 e início da década de 60 foram introduzidas obras de qualidade, surgindo então, a era da garantia da qualidade.

Garantia da Qualidade

Foi fruto da necessidade de atender às rigorosas especificações e aos critérios de desempenhos exigidos pelos programas militares, eletrônicos e espaciais. O projeto de

produtos tornou-se mais exato, dando origem à engenharia de confiabilidade e à necessidade de uma melhor coordenação entre os departamentos antes de serem liberados novos produtos. Ao mesmo tempo, estavam surgindo inúmeras idéias novas no pensamento norte-americano sobre gerenciamento de recursos humanos.

No período da garantia da qualidade, a qualidade passou a ser uma disciplina com implicações mais ampla para o gerenciamento. A prevenção de problemas continuou sendo seu objetivo fundamental, mas os instrumentos da profissão expandiram para muito além da estatística. Havia quatro elementos distintos: quantificação dos custos da qualidade, controle total da qualidade, engenharia da confiabilidade e zero defeito.

Custo da Qualidade

Em 1951 Joseph Juran abordou esta questão no seu livro *Quality Control Handbook*, que discutia a economia da qualidade. Ele observou que os custos para atingir um determinado nível da qualidade podiam ser divididos em custos evitáveis e custos inevitáveis. Os inevitáveis eram associados à prevenção - inspeção, amostragem, classificação e outras iniciativas de controle da qualidade. Custos evitáveis eram os do defeitos e das falhas dos produtos - material sucateado, horas de trabalhos necessárias para refazer o produto e repará-lo, processamento de reclamações e prejuízos financeiros resultantes de fregueses insatisfeitos. Juran considerava que os custos das falhas poderiam ser drasticamente reduzidos investindo-se na melhoria da qualidade.

Controle da Qualidade Total

Em 1956, Armand Feigenbaum propôs o controle da qualidade total. Produto de alta qualidade, argumentava ele, não teriam probabilidade de serem produzidos se o departamento de fabricação fosse obrigado a trabalhar isoladamente:

“O princípio em que se assenta esta visão de qualidade total ... é que, para se conseguir uma verdadeira eficácia, o controle precisa começar pelo projeto do produto e só terminar quando o produto tiver chegado às mãos de um cliente que fique satisfeito... o primeiro princípio a ser reconhecido é o de que *qualidade é um trabalho de todos*”.

Engenharia da Confiabilidade

Na mesma época, outra ala da disciplina estava surgindo, com uma crença mais forte ainda na teoria da probabilidade e na estatística: a engenharia de confiabilidade, que tinha por objetivo garantir um desempenho aceitável do produto ao longo do tempo. Este campo esteve intimamente associado ao crescimento, após a guerra, da indústria aeroespacial e da indústria eletrônica nos Estados Unidos. Assim sendo, um de seus principais pontos de apoio foi a área militar. Em 1950, o Departamento de Defesa criou um Grupo Ad Hoc de Confiabilidade de Equipamentos Eletrônicos e em 1957 foi publicado um grande relatório sobre o assunto. Este relatório acabou levando a inúmeras especificações militares que estabeleciam os requisitos de um programa formal de confiabilidade.

Esses esforços foram estimulados pela queda da confiabilidade dos componentes e sistemas militares. Em 1950, apenas a terça parte dos dispositivos eletrônicos da Marinha estava funcionando adequadamente. Um estudo feito na época pela Rand Corporation estimou que cada tubo de vácuo que os militares tinham funcionando era acompanhado de outros nove no depósito ou já encomendados. Havia problemas da mesma gravidade com mísseis e outros equipamentos aeroespaciais.

É claro que era preciso prestar mais atenção ao desempenho do produto ao longo do tempo. O primeiro passo foi definir com maior precisão a confiabilidade - como “a probabilidade de um produto desempenhar uma função especificada sem falhas, durante um certo tempo e sob condições preestabelecidas”. Associada aos recursos da moderna teoria da probabilidade, esta definição levou a métodos formais de previsão do desempenho de equipamentos ao longo do tempo. Também resultou em técnicas de redução dos índices de falhas enquanto os produtos ainda estavam no estágio de projeto.

Grande parte da análise baseava-se no conceito de distribuição de probabilidades. Isso não passava de uma relação matemática que especificava a confiabilidade de um produto (ou, inversamente, sua taxa de falhas) como função do tempo. Os engenheiros logo verificaram que diferentes condições de operação e diferentes produtos

aproximavam-se melhor por meio de formas matemáticas diferentes. As mais conhecidas eram a função exponencial, que partia da premissa de que o índice de falhas de um produto permanecia relativamente inalterado durante toda a sua vida útil; a distribuição de Weibull, que permitia que as taxas de falhas aumentassem ou diminuíssem com o tempo, se os produtos melhorassem ou se deteriorassem com a idade; e a “curva da banheira” - assim chamada devido à sua forma característica - que afastava a premissa de que as taxas fossem constantes ou variassem regularmente no tempo, argumentando, ao invés disso, que o que havia era um período de adaptação (quando as taxas de falhas eram altas), um período de operação normal (quando as taxas de falhas eram constantes e relativamente baixas) e uma fase de desgaste (quando as falhas aumentavam sempre e o produto se deteriorava). Estas relações eram, então, associadas a programas de testes meticulosos que visavam a simular condições extremas de operação, para estimar níveis de confiabilidade mesmo antes de os produtos atingirem uma produção a plena escala.

Mas, a previsão foi só o primeiro passo. O verdadeiro objetivo da disciplina era melhorar a confiabilidade e reduzir as taxas de falhas ao longo do tempo. Para atingir esses objetivos, aplicavam-se diversas técnicas: a análise de modo e efeito de falhas (FMEA, de “failure mode and effect analysis”), que examinava sistematicamente como um produto poderia falhar e, com base nisso, propunha projetos alternativos; a análise de componentes individuais, que calculava a probabilidade de falha de componentes chaves e, feito isso, procurava eliminar e reforçar os elos mais fracos; a reavaliação, que exigia que as peças fossem usadas abaixo de seus níveis de tensão especificados; e a redundância, que exigia o uso de sistemas paralelos para assegurar a exigência de “backups” sempre que um componente ou um subsistema importante falhasse. Um programa de confiabilidade eficaz também exigia o acompanhamento de perto das falhas em campo. Se assim não fosse, os engenheiros ficariam privados de informações fundamentais - a verdadeira experiência do produto em operação - úteis para o planejamento de novos projetos. As informações de falhas dos produtos em uso normalmente exigiam amplos sistemas de coletas de dados, bem como esforços para se assegurar que as peças que falhavam retornassem ao laboratório para mais testes e análises.

Como o controle total da qualidade, a engenharia de confiabilidade visava, antes de mais nada, prevenir a ocorrência de defeitos. Também ela enfatizava as habilidades de engenharia e atenção para a qualidade durante todo o processo de projeto. Zero defeito, a última inovação significativa da era da garantia da qualidade, seguiu uma trilha diferente: concentrava-se nas expectativas de gerenciamento e nas relações humanas.

Zero Defeito

Propôs-se desenvolver um programa cujo objetivo preponderante fosse “promover uma vontade constante, consciente, de fazer o trabalho (qualquer trabalho) certo da primeira vez”. O programa resultante chamou-se zero defeito.

Apesar das mudanças, as orientações em relação com a qualidade continuaram em grande parte defensivas. O principal objetivo de qualidade ainda era a prevenção de defeitos. Muito embora se estivesse seguindo agora uma orientação preventiva, a qualidade ainda era vista negativamente - como algo que podia prejudicar uma empresa se deixada de lado - e não como uma possível base de concorrência. Esta visão acabou se modificando na década de 70 e 80, quando os aspectos estratégicos da qualidade foram reconhecidos e incorporados.

Gestão Estratégica da Qualidade

Estão associando a qualidade à lucratividade, definindo-a de acordo com o ponto de vista do cliente e exigindo sua inclusão no processo de planejamento estratégico. No mais radical de todos os avanços, insistem em que a qualidade seja vista como uma arma agressiva de concorrência.

A Gestão Estratégica da Qualidade (GEQ) caracteriza-se por:

- a) estabelecer uma ligação forte entre qualidade e lucratividade;
- b) definir qualidade pelo ponto de vista do consumidor, e
- c) comprometer a alta gerência com a qualidade.

A GEQ não é a negação das fases anteriores, ao contrário, incorpora elementos destas. As idéias bem sucedidas de coordenação interdepartamental, do programa zero defeitos, a preocupação com os custos da qualidade e as ferramentas de controle estatístico (visando produção uniforme) continuam válidas. Só que o foco não é mais a detecção de defeitos, o controle do nível de não-conformidades ou a coordenação da empresa; o objetivo fundamental da gestão estratégica é fazer com que a organização faça exatamente o que vai trazer satisfação ao seu público consumidor.

A força motriz da GEQ é a dificuldade em sobreviver em mercados cada vez mais competitivos, e deste modo a qualidade tornou-se uma função gerencial, tanto como finanças ou “marketing”.

Abordagem Estratégica

Um número cada vez maior de empresas chegou à uma conclusão: a qualidade era uma poderosa arma na concorrência. Tanto do lado do mercado quanto do lado do custo, oferecia uma grande alavancagem. Os gerentes mais dinâmicos deram mais um passo. Se a qualidade estava associada tão de perto à rentabilidade, eles não viam razão alguma em se equipar aos níveis de qualidade dos concorrentes.

Isso exigia uma reformulação das abordagens tradicionais da qualidade, já que se desejava uma melhoria marcante - e continuada. Não seria de se esperar que os concorrentes ficassem de braços cruzados ao perceberem que sua qualidade tinha sido superada; eles também procurariam melhorar. As metas de qualidade tornar-se-iam, então, alvos móveis, que seriam sempre reformulados em níveis cada vez mais altos. O objetivo passaria a ser melhoria contínua. Isso exigia uma dedicação ao processo de melhoria, bem como o compromisso de toda a companhia. Um importante pré-requisito ficou logo claro: a alta gerência teria que ter uma participação ativa no processo. Este compromisso de alto nível era considerado essencial para estabelecer seriedade de propósito e dedicação a longo prazo à qualidade. De fato, muitas empresas constataram

que só depois de seus mais altos executivos terem destinados algum tempo à qualidade é que os empregados perceberam sua importância.

O objetivo é a obtenção de um compromisso de toda a organização para com a qualidade. A participação da alta gerência tem sido uma abordagem utilizada; outra tem sido o treinamento generalizado e a formação de equipes. A abordagem estratégica da qualidade também faz novas exigências aos profissionais da área de qualidade. A especialização técnica continua sendo desejável, mas passa a ser mais importante uma compreensão dos objetivos estratégicos da empresa. A educação e treinamento tornam-se responsabilidades vitais, assim como a avaliação de programas, o estabelecimento de objetivos e o trabalho de consulta a outros departamentos. De modo geral, há um claro afastamento de um papel de policiamento estreito e uma aproximação de um papel que enfatize mais uma perspectiva de gerência.

Para apoiar esta perspectiva, a qualidade é muitas vezes incluída explicitamente no processo de planejamento estratégico. Estabelecem-se metas anuais específicas e viáveis para a melhoria da qualidade. As metas normalmente levam em conta a perspectiva dos clientes e também são comparadas com o desempenho esperado dos concorrentes.

Na gestão estratégica da qualidade pode-se ver aspectos tanto de garantia de qualidade quanto controle estatístico da qualidade. Mas não se deve confundir os três movimentos. A abordagem estratégica da qualidade é mais ampla que suas antecessoras, mais intimamente ligada à lucratividade e aos objetivos básicos, mais sensível às necessidades da concorrência e ao ponto de vista do consumidor e, mais firmemente associada à melhoria contínua.

Melhoria da qualidade acaba levando a menores custos, maior produtividade e a uma vantagem sustentável a longo prazo.

I.2 - Conceito de Qualidade

Através da história, integrantes de toda a sociedade tem feito uso de materiais e energia para criar o produto trabalhado para uso e benefício de outros integrantes da mesma sociedade, sempre com a intenção de satisfazer as expectativas e necessidades - real ou percebida - dos usuários. Especialmente desde a revolução industrial, a noção de qualidade tem sido associada com produtos manufaturados e serviços apresentados. Com a diversificação das necessidades na sociedade moderna e o aparecimento dos serviços industriais, o conceito de qualidade tem sido naturalmente estendido em todos os aspectos.

Produto trabalhado é fornecido para satisfazer as necessidades e expectativas de seus clientes. Este conceito é universal e é aplicado para todo produto trabalhado. A representação formal das expectativas (necessidades e benefícios) são incorporadas nas exigências que fornece o básico para futuras medidas, avaliações e gerenciamento de ações. Quando se assume que qualidade não pode ser realizada a menos que ela possa ser medida e ela não pode ser medida a menos que seja definida anteriormente, então qualidade significa a conformidade com as exigências. Assim, a definição da qualidade do produto trabalhado é obtida quando o produto encontrar as exigências estabelecidas

Para procurar definir o que é qualidade, é preciso descobrir sob que referencial se deseja focá-la. Qualidade depende, antes de tudo, do referencial pela qual ela é empregada. Para o usuário final, qualidade significa “atendimento às necessidades a um preço razoável”. Para o produtor, qualidade significa “conformidade com as especificações”. Para um funcionário responsável pela assistência técnica, qualidade significa “baixa necessidade de consertos”.

Qualidade, como pesquisa e desenvolvimento, é um investimento a longo prazo, com retorno incerto, pois vai depender de inúmeras variáveis, e possui a necessidade de altos investimentos iniciais. É necessário ter paciência para obter os resultados.

David Garvin [referência 20] identificou cinco abordagens principais para definir qualidade:

1) Definição Transcendental

Qualidade é algo universalmente conhecido, uma propriedade que não se pode analisar, que se reconhece unicamente pela experiência.

Segundo esta abordagem qualidade só pode ser percebida após a exposição de uma sucessão de objetos com esta característica. A limitação desta abordagem é que ela oferece pouca ou nenhuma utilidade prática.

2) Definição Baseada no Produto

Esta classe de definição traz a idéia de que qualidade é uma variável precisa e mensurável. Diferenças na qualidade são vistas como diferenças na quantidade de algum ingrediente ou atributo possuído pelos produtos.

3) Definição Baseada no Usuário

Este tipo de definição parte do princípio de que qualidade está nos olhos do observador. Os consumidores possuem diferentes necessidades ou preferências, e os bens que melhor satisfizerem suas preferências serão os que possuírem maior qualidade. Estes produtos atingiram os chamados “pontos ideais”, que são a combinação precisa de produtos ou atributos que fornecem a maior satisfação a um consumidor específico. Deste tipo de enfoque surgiu a idéia da “adequação ao uso”.

4) Definição Baseada na Produção

As definições baseadas na produção focalizam conceitos de engenharia. Elas identificam qualidade como “conformidade com as especificações”. Uma vez que tenha

sido estabelecido um projeto ou uma especificação, qualquer desvio implica em uma redução de qualidade.

5) Definição Baseada no Valor

Este tipo de definição encara a qualidade em termos de custos e preços. Deste modo, um produto possui qualidade se fornece desempenho ou conformidade a um preço ou custo aceitável.

I.3 - Dimensões Múltiplas da Qualidade

David Garvin [referência 6] propôs oito dimensões de qualidade:

- Desempenho

O desempenho de um produto é a capacidade de responder às solicitações para as quais foi projetado. Por exemplo, a velocidade de um computador em processar informações.

O referencial de desempenho não é único. A idéia de qualidade ser adequação ao uso é válida.

- Características Secundárias

Representa as características que suplementam o funcionamento básico dos produtos. É “algo mais” de cada produto.

- Confiabilidade

A confiabilidade de um produto é a probabilidade deste não falhar em um período específico de tempo, sob determinadas condições de operação. Esta dimensão pode ser medida objetivamente pela taxa de falhas, pelo tempo médio entre falhas ou pelo tempo médio até a primeira falha.

Esta dimensão da qualidade torna-se muito importante quando:

- a utilização do serviço/produto envolve grandes somas de dinheiro, como por exemplo, um sistema bancário informatizado;
- o tempo de reparo é longo e envolve lucros cessantes, como em colheitadeiras na época da colheita e equipamento industrial indispensável para a produção;
- a manutenção é cara;
- ocorre o envolvimento de vidas humanas, como em transporte aéreo (motores de aviões) e equipamentos de mergulho.

- Conformidade

É uma dimensão da qualidade facilmente entendida por técnicos, engenheiros e operários ao nível da fábrica, pois ela está associada à uniformidade da produção e controle da variabilidade.

A conformidade é uma dimensão baseada na produção. Ela mantém uma estreita ligação com a confiabilidade, por serem ambas, muito tecnicistas. Melhoramento em uma destas categorias vai exigir - ou induzir - o da outra.

- Durabilidade

A durabilidade de um produto possui conotações econômicas e tecnológicas. Durabilidade de um bem significa um valor de revenda mais alto. Do ponto de vista tecnológico, significa o quanto é possível utilizar um produto, ou seja, qual é a sua vida útil. Apesar desta dimensão poder ser expressa em termos de uma medida bem definida de tempo (1 ano, etc.), ela depende muito das condições de uso do produto. Existem usuários mais e menos cuidadosos, e isto afeta a durabilidade de seus bens.

- Capacidade de Receber Assistência Técnica

Esta dimensão representa a velocidade e facilidade de reparo. Esta dimensão pode ser medida objetivamente pelo tempo médio gasto para reparar um produto, e a competência do reparo pode ser atestada se o serviço for ou não chamado novamente para resolver o mesmo defeito.

- *Estética*

Esta é uma dimensão mais subjetiva. Ela está relacionada com o padrão de beleza individual do cliente.

- *Qualidade Percebida*

A qualidade percebida -ou induzida- é a influência que o nome do fabricante e a propaganda exercem sobre o cliente, ou seja, a percepção da qualidade que o cliente possui.

A excelência em todas as categorias raramente é uma necessidade para se ter êxito. O excesso de características pode prejudicar um bom equacionamento dos serviços, devendo-se escolher algumas dimensões da qualidade como centros de atenção em vez de procurarem ser o número um em todas as categorias. Desde que as dimensões selecionadas estejam bem ajustadas às necessidades do mercado, a abordagem direcionada pode dar ótimos resultados. Não é necessário desenvolver todas as dimensões de uma só vez.

As estratégias que visam identificar e assegurar um nicho de qualidade podem ser, então, bastantes eficazes. Poucos produtos ocupam uma posição elevada em todas as oito dimensões de qualidade; os que ocupam são quase sempre fruto de artesanatos e são caros. Não é preciso atentar para as oito dimensões ao mesmo tempo para se estabelecer uma reputação da qualidade. A superioridade em uma ou duas dimensões da qualidade - especialmente a *confiabilidade* e a *conformidade*, que ficaram cada vez mais importante para os clientes - muitas vezes basta para se ganhar uma posição dominante.

Pode-se citar como erros estratégicos da qualidade: atacar o líder nas áreas em que este já é reconhecidamente forte, investir em perfeccionismo desnecessário e aprimorar dimensões que não são perceptíveis aos consumidores.

I.4 - Organização da Tese

Nos capítulos seguintes serão mostrados mais detalhes das análises de qualidade relativo aos produtos - sistema de “software” e sistema baseado em conhecimento (SBC). Assim:

- no capítulo II serão apresentadas as principais visões sobre qualidade de “software”, definindo conceitos básicos e considerando alguns aspectos significativos dessa área;

- no capítulo III será descrita uma análise sobre a avaliação da qualidade de sistema baseado em conhecimento, evidenciando uma necessidade complexa que o SBC precisa satisfazer;

- no capítulo IV será apresentado um estudo de caso, procurando utilizar as técnicas de qualidade de um SBC;

- no capítulo V serão indicadas algumas conclusões.

CAPÍTULO II

QUALIDADE DE “SOFTWARE”

II.1 - Introdução

Atualmente existe uma grande preocupação com a qualidade de um produto de “software”. Portanto, tem sido intensa a pesquisa em busca de melhores conhecimentos com relação a “qualidade de “software”.

A palavra *qualidade* evoca a imagem de *excelência*. Existem diversas visões sobre o termo “qualidade de software”. Do ponto de vista transcendente o produto de “software” não pode ser quantificado, mas pode somente ser sentido e apreciado pelos especialistas da área. Uma outra visão é voltada para os que procuram direções objetivas para melhorar a qualidade de seus produtos, na qual qualidade de “software” é tida como um produto-base. Esta visão olha a qualidade de “software” como um pacote de atributos de produtos que podem ser medidos quantitativamente. Esses atributos podem referenciar a qualidade congênita do produto (complexidade, desempenho) ou qualidade psicológica (usuário satisfeito, fácil para usuário) percebida pelo usuário final do produto.

O objetivo deste capítulo é apresentar alguns aspectos considerados significativos em termos de estudos na área de “qualidade de software”.

II.2 - O Que é um Sistema de Qualidade?

Aplicar os princípios de qualidade ao processo do “software” é o princípio do sucesso. O termo “sistema de qualidade” é usado internacionalmente para descrever um

processo que assegura e demonstra a qualidade dos produtos e serviços por ele produzidos.

O termo “sistema de gerenciamento de qualidade” é usado às vezes, em vez de “sistema de qualidade”. Isto enfatiza a necessidade de que o processo da qualidade seja **ativamente gerenciado**, para assegurar sua contínua eficiência quando da mudança das circunstâncias. Particularmente, os padrões da melhor prática estão evoluindo continuamente, e as ferramentas de hoje podem não ser apropriadas amanhã. Os criadores de “software” devem manter as práticas e ferramentas que usam sob revisão constante, fazendo mudanças de modo controlado.

Tão importante quanto as práticas e ferramentas é a equipe que as utiliza. O sistema de qualidade tem que assegurar que eles sejam corretamente qualificados para suas funções, do ponto de vista profissional. Caso precisem de treinamento, devem recebê-lo. O sistema deve também assegurar que eles compreendam suas responsabilidades e como o seu trabalho se relaciona com o dos demais.

Os sistemas de qualidade bem sucedidos dão grande ênfase à ação corretiva precoce. É muito mais barato para o analista corrigir os erros no início do seu ciclo de vida. Também vale a pena pensar que um erro pode custar muito mais para o cliente do que para o analista. O objetivo deve ser assegurar que o trabalho seja feito corretamente, logo da primeira vez, em todos os estágios do processo do desenvolvimento. As atividades de controle de qualidade, tais como inspeção e testes são, deste modo, embutidas em todos os estágios, para detectar erros tão cedo quanto possível.

Evitar cometer os primeiros erros é ainda melhor. Os sistemas de qualidade bem sucedidos incluem modos de análise de registros de erros, para determinar suas causas originais, agindo na prevenção dos erros pela eliminação das causas.

O sistema de qualidade deve assegurar a clientes e criadores que os produtos de “software” ali desenvolvidos serão de boa qualidade. O sistema de qualidade deve ser à prova de auditoria. Isto significa que o processo de desenvolvimento deve ser

documentado, e que os registros de qualidade, incluindo medidas adequadas, devem ser gerados para demonstrar a aquisição da qualidade e a efetiva operação do sistema de qualidade.

Os encargos de manutenção de tais registros não precisam ser excessivos, se o sistema de qualidade for cuidadosamente projetado. A melhoria na qualidade compensa amplamente os custos envolvidos.

A filosofia que apóia os sistemas de qualidade bem sucedidos é a da melhoria contínua de todos os aspectos do processo do “software”. Os registros e medidas de qualidade são analisados e usados com este propósito. Sob a luz desta análise, um gerenciamento superior deve rever regularmente a eficiência e a efetividade do sistema de qualidade, e assegurar que serão tomadas medidas para sua melhoria.

Em resumo, um sistema de qualidade é tudo o que o gerenciamento organiza, para assegurar e demonstrar a qualidade dos produtos de “software” e dos serviços associados de suporte. O sistema de qualidade é o processo de trabalho completo, incluindo políticas, procedimentos, ferramentas e recursos, tanto humanos quanto tecnológicos.

II.3 - Perfil das Análises de Qualidade de “Software”

II.3.1 - Intenções da Filosofia de Qualidade em “Software”

Qualidade de “software” é algo que qualquer pessoa que trabalha em desenvolvimento de “software” deseja.

Do ponto de vista mais abrangente, qualidade de “software” significa que o “software”:

- (1) cumpre seus objetivos;
- (2) é gerenciável;

(3) é passível de manutenção e tem longa duração;

(4) é passível de aprendizagem por uma pessoa de desenvolvimento que não faça parte do grupo original de projeto.

Cumprindo Objetivo

Sistemas não se comportam sempre como esperado. Isto não é surpresa para qualquer um que tenha desenvolvido “software”. Algumas vezes, os sistemas não cumprem um objetivo e não satisfazem as necessidades de seus usuários.

Podemos listar três maneiras de um sistema falhar no cumprimento de seus objetivos:

1. O sistema é implementado a partir de uma especificação incorreta. Se um analista cria uma especificação que não modele ou represente as necessidades do “software”, os analistas e programadores implementarão um sistema desnecessário, tendo como resultado um “software” que não faz o que o usuário deseja.

Há quatro critérios para se aplicar a uma especificação. Quando uma especificação preenche todos eles, ela é uma boa (ou seja, correta) especificação. Esses critérios são:

- *Clareza*. Uma especificação que não se comunica bem não tem utilidade. Uma especificação é clara, quando a audiência à qual ela se dirige a entende.
- *Precisão*. Uma especificação imprecisa é pior que a inútil, pois a equipe de desenvolvimento receberá uma especificação vaga (apesar de clara) e projetará um sistema sem utilidade.
- *Integridade*. Há duas medidas de integridade: primeira, uma especificação necessita conter somente a informação de seu escopo. Segundo, uma especificação necessita

conter toda a informação dentro do escopo. Óbvio, mas a maioria das especificações falha em ambas as medidas.

- *Concisão*. Uma especificação deveria conter somente a informação suficiente para se comunicar de maneira apropriada. Mais que isto é esforço desperdiçado e pode resultar em confusão.
2. O sistema é implementado de maneira inapropriada a partir de uma boa especificação. Uma vez que o analista tenha produzido uma boa especificação, os analistas e programadores a implementam. Há várias maneiras pelas quais os analistas e programadores podem criar um sistema ruim a partir do que era originalmente uma boa especificação:

Má escolha de tecnologia. A equipe de desenvolvimento deve assegurar que a escolha em tecnologia [“hardware”, sistema operacional, “software”, interface de “software”, “software” de terceiros, 4GL (linguagens de quarta geração), “software” de banco de dados etc.] seja apropriada.

- *Otimização prematura*. O desejo de otimizar é muitas vezes perigoso. Análises e projetos freqüentemente excluem a necessidade de otimização. Por sua natureza, a otimização destrói a estrutura, de modo que ela deveria ser feita com grande prudência. Se feita muito cedo, a otimização freqüentemente reduz o poder de processamento, obrigando a programar adições que superem a redução na estrutura. Em todos os casos, a otimização aumenta o tempo e a probabilidade de erros de manutenção.
3. A tecnologia é insuficiente para permitir mapeamento (“mapping”) sem maiores distorções. Algumas vezes, dadas as restrições de orçamento, pessoal e tempo, é simplesmente impossível implementar o que foi requisitado. Para atingir qualidade em “software”, as restrições devem ser relaxadas ou o escopo deve ser reduzido. É claro, à medida que a tecnologia progride, este problema torna-se cada vez mais

obsoleto. Apesar de os requisitos de “software” estarem se tornando cada vez mais complexos, a tecnologia está mais à frente, em vez de acompanhar seu passo.

Gerenciabilidade do Sistema

É preciso que diversas pessoas possam trabalhar em diferentes partes do sistema ao mesmo tempo. O projeto deve ser passível de delegação.

As necessidades de mão-de-obra no sistema variam com o tempo. No princípio há mais trabalho de análise. Próximo ao final do ciclo de vida do desenvolvimento do sistema, há mais programação. No meio, há mais projeto. Um projeto bem-sucedido deve ter ferramentas e técnicas para decidir quem faz o que e quando.

Possibilidade de Alterações e Vida Longa

Uma vez o sistema em funcionamento, foi atingida a primeira meta. Mas tente fazer uma alteração e um problema pode se instalar, pois o sistema não é passível de manutenção. Tais sistemas têm ciclos de vida útil muito curtos, são onerosos e não reagem em caso de alteração.

A ênfase dada para se ter um produto funcionando, e que não seja possível de se ter alterações, isto implica que o “software” não será passível de manutenção.

Entretanto, todos começaram a perceber que muitos sistemas tentam acertar um alvo em movimento. Um sistema que não pode facilmente incorporar alterações é caro de se manter e propenso a erro, além de usualmente ter um ciclo de vida curto. A filosofia da qualidade em “software” busca maximizar a vida do sistema, fornecendo caminhos diretos para implementar e documentar alterações. Um benefício adicional desta abordagem é a redução do trabalho para implementar cada unidade de alteração.

Possibilidade de Aprendizagem

Há uma grande probabilidade de novas pessoas trabalharem com o sistema muito tempo após seu desenvolvimento. Estas novas pessoas necessitarão de uma curva de aprendizagem, a mais curta possível, para se tornarem imediatamente produtivas; um sistema aprendido rapidamente é menos caro de ser mantido.

Uma redução no tempo necessário para treinar um novo integrante da equipe de projeto é alcançada, insistindo-se para que cada um deles, use ferramentas e técnicas padrões. Padrões tornam possível, para o projeto como um todo, ter acesso ao conhecimento do sistemas. Resumidamente, a filosofia de excelência em “software” torna os sistemas possíveis de aprendizado utilizando ferramentas padrões.

Para que o desenvolvimento de “software” seja um sucesso, os problemas de “software” devem ser evitados.

Sem uma metodologia, nenhum grupo pode alcançar sua meta de maneira bem-sucedida e segura. Mesmo um programador solitário usa um método para produzir “software”.

Assim como o processo de gerenciamento, o processo de criar, escrever e manter “software” necessita ser entendido; ele necessita ser descrito por uma metodologia. Atingir qualidade em “software” demanda uma metodologia. Mais ainda, envolve uma filosofia da qual segue naturalmente uma metodologia.

A adesão muito rígida a uma metodologia é algo prejudicial. O fato é que nenhuma metodologia é perfeita. Quando se descobre que seguir uma metodologia escolhida leva direto para o fracasso de um projeto, é hora de modificar a metodologia. deve-se usar sempre o bom senso para decidir o que necessita ser feito no sistema, mesmo que não esteja na metodologia.

O uso errado ou inadequado de ferramentas de modelagem pode causar certos danos. Dado uma metodologia, procedimentos concretos e bons modelos, ainda é possível se confundir as coisas. Primeiro, a escolha de uma ferramenta de modelagem errada. Segundo, uma vez escolhida a ferramenta de modelagem certa, ela pode ser utilizada de forma incorreta.

Um modelo apresenta uma grande quantidade de informações. Para um modelo de "software" ser entendido, ele deve ser particionado ou quebrado. A habilidade em dividir claramente é necessária para construir um sistema de "software" eficiente. Pode-se razoavelmente afirmar que cada avanço em análise e projeto estruturado na década passada teve como meta uma divisão mais "limpa" do sistema. A qualidade em "software" preenche os detalhes de como dividir de forma limpa, resultando sistemas fáceis de manter, com máxima coesão e mínimo acoplamento.

II.3.2 - Visão da "Gerência de Qualidade Total" em Desenvolvimento de "Software"

Organizações de desenvolvimento de "software" enfrentam esforços de executar gerência de qualidade total numa área em que programação de computador é ainda notado como uma arte e não uma ciência mensurável. Como em qualquer atividade, liderança é essencial para o trabalho de gerência de qualidade total. Qualidade não pode ser delegada, ela requer um compromisso de gerência para definir um conjunto de objetivos para a organização. A gerência não pode mais ser envolvida unicamente com o programa do produto. Agora ela tem que ser envolvida com a qualidade do produto produzido.

A gerência de qualidade total aplica o desenvolvimento de "software" utilizando uma metodologia com seis fases, também conhecido como "modelo cascata". Será mostrado a seguir um perfil das fases de desenvolvimento de "software" [referência 19]:

1ª Fase: Requisição

- Executar uma análise e documentar as requisições;
- Revisar as análises e requisições dos clientes e fornecedores;
- Certificar fornecedores;
- Executar um análise final e segura.

2ª Fase: Projeto

- Especificar as requisições;
- Aprovar clientes e fornecedores;
- Identificar componentes de re-uso de “software”;
- Desenvolver um plano preliminar de unidade e integração testada;
- Desenvolver a unidade e teste de integração dos exemplos;
- Desenvolver um plano preliminar de teste para o sistema e aprovação do usuário;
- Desenvolver sistema e aprovação do usuário para exemplo;
- Revisar com os clientes e fornecedores a aprovação do usuário para os exemplos;
- Desenvolver um plano preliminar para divulgação do produto.
- Desenvolver o primeiro “draft” de manual do usuário.

3ª Fase: Implementação

- Desenvolver o código;
- Revisar o código;
- Finalizar a unidade e planos de teste integração e exemplos de teste;
- Executar unidade e integração dos exemplos e resultados;
- Revisar os resultados de teste.

4ª Fase: Teste

- Finalizar o sistema e aprovação do usuário para planos de teste e teste de exemplos;
- Executar sistema e aprovação do usuário testada e registrar os resultados;
- Clientes e fornecedores devem revisar os resultados;

- Finalizar o plano de divulgação e manuais;
- Clientes e fornecedores devem revisar o plano de divulgação e manuais.

5ª Fase: Divulgação

- Instalar o produto e conduzir o treinamento.

6ª Fase: Revisão após a implementação

- Conduzir uma revisão de uma pós-implementação.

Nesta metodologia não há uma fase de manutenção. Acredita-se aqui que se o desenvolvimento do “software” (produto) for feito corretamente na primeira vez, então não haverá dificuldades.

Quando ocorre uma detecção de erros após a divulgação do produto, a equipe de desenvolvimento deveria neste caso começar desde o início da metodologia para determinar quais alterações poderiam ser feitas para corrigir os erros e que efeito essas alterações causariam no sistema como um todo. Eles também executam uma completa reanálise na definição das novas requisições, atualizações do projeto, desenvolvimento ou alteração de código, e execução da unidade e integração para teste.

Durante todo o processo de desenvolvimento, o projeto de gerenciamento de tarefa deveria ser executado para medir o tempo investido no projeto, para assegurar que o código e a documentação estão controlados, e para assegurar que qualquer alteração no projeto são analisadas para determinar todo efeito que acarretaria no projeto.

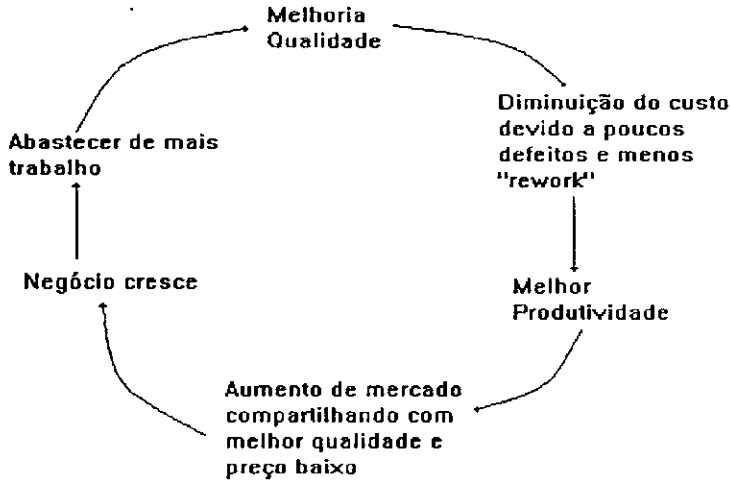
Se ocorrem erros, a segurança e confiabilidade do “software” são os maiores envolvidos e comprometidos. Uma completa e segura análise ajudaria a assegurar a qualidade do “software”, servindo como uma documentação legal, se necessária.

II.3.3 - Melhoria da Qualidade de “Software”

O que é melhoria da qualidade? Apagar “incêndios” e remover defeitos específicos do “software” não é melhoria de qualidade. Os clientes querem qualidade melhor, mais rápida e a um menor preço. A maioria das organizações de sistemas de informações (SI) insiste na entrega do “software” em uma data-limite específica. Infelizmente, isto geralmente significa apenas que o cliente recebe *no prazo* um produto *defeituoso*. No clima atual, enfoca-se o corte nos custos, devido a uma crença induzida de que apenas sobreviverá aquele que oferecer o menor custo. Cortar os custos freqüentemente baixa a qualidade, levando à insatisfação por parte do cliente, com um produto de baixa qualidade por um preço menor, *em princípio*, porém com um custo final maior. É possível oferecer “software” de alta qualidade, dentro do prazo e a um custo mais baixo. Deve-se começar fazendo produtos de “software” que satisfaçam os clientes em primeiro lugar e, depois, reduzir continuamente o tempo e o custo de fazer aqueles produtos de alta qualidade, através de um melhor gerenciamento, melhores processos e materiais.

Qualidade demanda que se solucione menos problemas, mas que sejam todos problemas de alta abrangência. Na maioria das organizações, quando ocorre um problema, ocorre uma reação para solucioná-lo, seja ele importante ou não. Qualidade demanda que toda a organização aprenda como resolver problemas de forma mais rápida e efetiva. Qualidade demanda também que se solucione não só os sintomas aparentes do problema - uma falha no programa - mas que se transfira a maior parte do problema, voltando os esforços para as causas que estão na sua raiz - defeitos no processo que produziu ou aperfeiçoou o programa, em primeiro lugar. Os SI devem parar de enfrentar incêndios, ignorar a crise imediata e começar a prevenir os incêndios corrigindo o modo como o “software” é criado e desenvolvido. Um organização saudável [referência 2] gasta 80% dos seus esforços para solucionar problemas na prevenção dos mesmos, enquanto uma organização reacionária, com uma performance pobre, aplica 90% do seu tempo corrigindo os sintomas, em vez de corrigir as causas (Sirkin 1990). Em uma organização que aprenda com seus próprios erros, a prevenção está na ordem do dia. Prevenir

defeitos, problemas e repetições do trabalho, libera a todos para que se concentrem no aumento da valorização para o cliente.



(figura II.1)

A melhoria da qualidade é uma descoberta contínua das causas que estão na raiz das falhas de produtos e serviços - prevenir o incêndio, ao invés de apenas lutar contra ele. A razão pela qual pretende-se melhorar continuamente a qualidade do “software” está expressa na Cadeia Deming (figura II.1) [referência 2]: a melhoria da qualidade leva a custos menores, com menos erros a remover e menos trabalho a refazer, o que faz com que a produtividade aumente, aumentando a demanda pela capacidade de desenvolvimento e manutenção do seu “software”, o que leva, por sua vez, a companhia ou o departamento a crescer em tamanho e lucratividade. O lado ruim disto é que a qualidade estática ou baixa, levará à deserção em massa dos clientes, à perda de recursos, cortes no pessoal e redução da fatia do mercado.

Satisfação do Cliente

A melhoria da qualidade tem um foco principal: *a satisfação do cliente*. A qualidade alinha os negócios com as necessidades presentes e futuras e com as expectativas do cliente. Onde a velha definição de qualidade sugeria que era suficiente suprir as necessidades dos clientes, a atual definição exige que não apenas supram as suas necessidades, mas que os surpreendam e que os satisfaçam acima e além das suas

expectativas. A satisfação do cliente tem muitos aspectos; aqui estão apenas alguns aspectos-chave:

- Qualidade - conformidade com as exigências válidas do cliente;
- Custo - tão baixo quanto possível, em conformidade com as exigências do cliente;
- Liberação - disponibilidade e confiabilidade dos sistemas de informação onde e quando exigidos;
- Prazo - dentro do previsto;

Um cliente insatisfeito é um terrorista. Um cliente insatisfeito comentará seus problemas com 16 pessoas, em média. Um cliente satisfeito os comentará com oito (Deming 1986).

Melhoria Constante

A melhoria contínua não é algo novo. Virtualmente todos os avanços na evolução humana beneficiaram-se de melhorias contínuas. Apenas, nos últimos 40 anos, W. Edwards Deming e J. M. Juran juntaram alguma disciplina a este processo evolutivo.

As melhorias contínuas vêm de duas fontes principais: a inovação (revolução) e a melhoria contínua (evolução, também conhecida como *kaisen*). A revolução é a metáfora original da América, e não surpreende que os EUA tenham apostado nas inovações para sustentar a sua vantagem competitiva em todo o mundo. De modo similar, a primeira metáfora da revolução industrial foi a *máquina*. Chegou-se ao ponto de depender das máquinas para a extensão das habilidades. Em muitos casos, ficou-se tão apaixonado pela máquina que foi esquecido o papel que processos efetivos e eficientes desempenham na criação e evolução rápidas de produtos e serviços.

Onde a inovação foi, um dia, o suficiente para sustentar a vantagem competitiva, já não é assim. Para ganhar a competição, deve-se ser capaz de inovar e manter aquela inovação inicial, melhorando-a. Deve-se também gerar níveis revolucionários de

melhoria, através do uso de processos graduais, evolutivos. O modo atualmente predominante de se conseguir isto é chamado melhoria contínua da qualidade, e ele é tão aplicável ao “software” quanto o é à manufatura.

Há alguns elementos-chave a respeito de qualidade, que precisam ser compreendidos:

- Qualidade é uma estratégia baseada no mercado. A qualidade cria e mantém a fatia do mercado, mesmo dentro de uma empresa.

- A qualidade depende do processo usado para criar o produto, seja este produto “hardware”, “software” ou gerenciamento.

- A qualidade é dirigida aos clientes. Eles exigem alta qualidade e, sem isto, não continuarão a usar o serviço ou produto. Qualidade mais alta resulta em produtividade, lucros e fatia de mercado aumentados. A qualidade mais alta resulta na diminuição de custos unitários e daqueles decorrentes de falhas.

- A qualidade tem que ser “tecida” juntamente com o sistema, quando da criação deste. Ela não pode ser inserida depois.

Com a chegada da economia da informação, os clientes começaram a exigir a qualidade do “software”. A demanda do mercado por qualidade levará a mudanças radicais na indústria do “software”, e isto é bom. Infelizmente, a maior parte das idéias sobre o que constitui a qualidade do “software” é tristemente induzida. Há muitos mitos a respeito de qualidade:

- As falhas de “software” são inevitáveis
- Testar assegura a qualidade
- Qualidade custa dinheiro

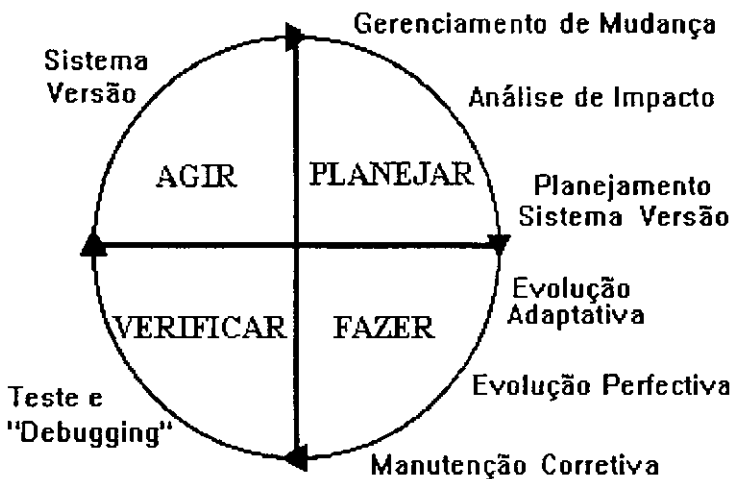
Os atuais processos de desenvolvimento de “software” geram dois produtos: “*software*” (código, documentação, etc.) e *defeitos* (causando trabalhos refeitos, desperdício, perda de produtividade, custos mais altos).

O custo de encontrar e remover estes defeitos é 50% de todos os custos do “software” (Brooks 1975). O método atual de encontrar estes defeitos enfoca apenas o controle dos defeitos, e não a sua prevenção.

A qualidade encontrou várias metáforas: a qualidade é gratuita, a qualidade é a conformidade com as exigências válidas do cliente, qualidade é o mesmo que “zero” defeitos e a qualidade é um caminho, não uma destinação. Técnicas conhecidas de “software” tornam possível o “software” isento de defeitos.

A maioria das empresas de “software”, tolamente, concentra seus esforços, no setor de defeitos, no “encontrar problemas” quando eles já estão no “software”. O número de defeitos que passa pelos testes depende do número de defeitos no “software”, quando de sua liberação para o teste. O número de defeitos no “software” quando de sua liberação para o teste está na razão direta da qualidade do processo usado para criar o “software”. Os testes podem revelar apenas 70 % dos defeitos latentes no código. As verificações podem remover de 80 a 90% dos defeitos antes do teste, mas um bom processo fará com que os defeitos jamais entrem no produto.

II.3.4 - Evolução do “Software”



(figura II.2)

O “software” é, na essência, um reflexo do processo mental dos nossos clientes e programadores. A evolução do “software” [referência 2] envolve ambos os desenvolvimentos, físico e mental, do sistema de aplicação.

A manufatura tradicional através de processos produziu um tempo médio de vida entre cinco e oito anos para o “software”, mas a maioria das grandes empresas estabelecidas possui milhões de linhas de códigos com 15 anos ou mais. Podendo-se deduzir que, através do crescimento e da evolução, os sistemas de “software” asseguram o seu nicho no ambiente da corporação.

Mais do que o custo da criação do “software”, devemos aumentar o investimento na criação do “software”, para obter diminuições significativas na evolução dos custos. Criar um “software” durável, reutilizável, é mais dispendioso a princípio, mas diminui os custos a longo prazo.

Manutenção significa preservar do erro e da queda. Evoluir significa uma mudança contínua de um estado menor, mais simples ou pior, para um maior ou melhor.

Os departamentos de SI gastam dinheiro na criação e evolução do “software”, nos seguintes montantes, aproximadamente [referência 2]:

30 % na Criação

70 % na Evolução

Assim, para reduzir os custos, deve-se inicialmente focar o ponto com o maior potencial de ganho - a Evolução. Pensa-se erradamente que, uma vez que se substitua um sistema, isto significa que se eliminará o fardo de manter o sistema atualizado com o seu ambiente. O que se precisa é uma estratégia para melhorar, fazer a re-engenharia, ou substituir sistemas, para que otimizem o retorno em investimento da companhia. O objetivo deve ser levar todos os sistemas de um patamar menor, mais simples ou pior, para um outro, maior ou melhor.

Gerenciamento de Mudança

Talvez a ferramenta-chave, tanto para a criação quanto para a evolução, seja o sistema de gerenciamento de mudança. É o sistema de ordem de aquisição que dirige todo o trabalho. O contexto está sempre mudando; deve-se ter um mecanismo que apóie tal evolução. Projetos-protótipo freqüentemente enfrentam mudanças contínuas de exigência e projeto, que ocorrem após o início do projeto. O Desenvolvimento Evolutivo rápido ajuda a administrar esta mudança, e o gerenciamento de mudança pode auxiliar no processo.

O gerenciamento de mudança é também uma ferramenta estratégica para a satisfação do cliente. Abra o sistema aos clientes, de modo que eles possam solicitar mudanças diretamente, verificar a situação das suas solicitações.

A maior preocupação atual da equipe de "software" é com a manutenção de programas já existente. A maioria dos programas de computadores é de manutenção difícil e cara.

As modificações do "software" são projetadas e implementadas precariamente. Os documentos de projeto raramente são examinados e atualizados para refletir as mudanças no sistema. Um sistema planejado de modo descuidado toma três vezes o tempo previsto para ser completado; Um sistema planejado cuidadosamente, consome duas vezes este tempo. Sistemas de difícil manutenção são, por fim, refeitos, com uma grande despesa. Os dois anos que se seguem ao lançamento de um novo produto são gastos na implementação de aperfeiçoamentos que tragam o sistema ao encontro das expectativas do usuário. E também, a maioria dos aperfeiçoamentos importantes é tão mal entendida e implementada que várias versões adicionais são necessárias para depurar o aperfeiçoamento. O reparo e o aperfeiçoamento do "software" sempre injeta novos problemas, que têm que ser corrigidos mais tarde.

Para resolver estes problemas e gerenciar o “software” em desenvolvimento, são necessárias melhorias na perícia e na produtividade dos mantenedores, bem como na qualidade e na eficácia do seu trabalho.

Oferecer ao pessoal de manutenção os mais modernos conhecimentos, capacitações e técnicas para que cumpram sua missão executando um pouco melhor as atividades-chave da evolução do “software”, resultará em melhorias significativas de produtividade e qualidade.

A evolução do “software” consiste nas atividades necessárias a que se mantenha um sistema de “software” operacional e com resposta, após ter ele sido aceito e colocado em produção. Estas atividades incluem:

- Corrigir defeitos (manutenção)
- Aperfeiçoar a funcionalidade do “software” (evolução)
- Melhorar a qualidade do “software” já existente (evolução)

Em geral, estas atividades mantêm o sistema em sincronia com um usuário e com um ambiente operacional em constante evolução e expansão. Funcionalmente, a evolução do “software” pode ser dividida nestas três categorias: corretiva, adaptativa e preventiva.

Ciclo de Vida do “Software”

O ciclo de vida do “software” cobre o período da concepção à desativação de um dado produto. Há muitas definições do ciclo de vida do “software”. Elas diferem principalmente nas classificações das fases e atividades.

A evolução do “software” tem características próprias, únicas, que incluem:

- Restrições de um sistema já existente - A evolução do “software” é feita em um sistema de produção já existente. Quaisquer mudanças devem estar em conformidade ou ser compatíveis com uma arquitetura, projeto e restrições de código já existentes.

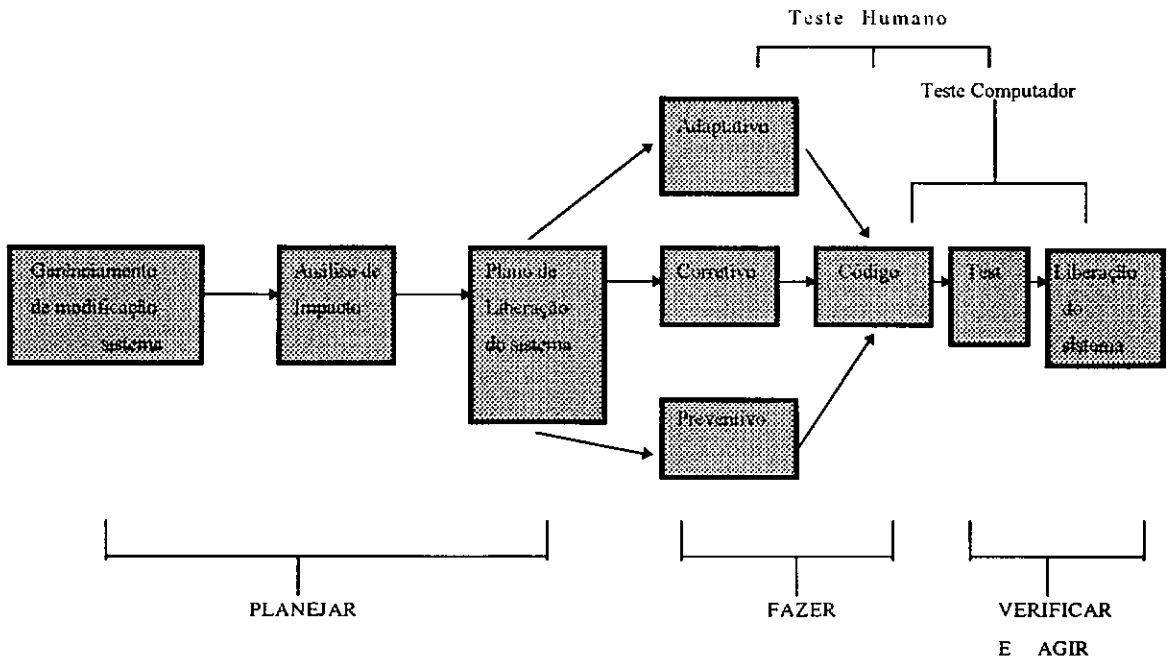
- Estruturas mais rápidas - A criação do “software” pode durar um ou mais anos, enquanto a evolução pode levar algumas horas, para ciclos de um a seis meses.

- Dados de teste disponíveis - A criação do “software” cria todos os dados de teste a partir do esboço. A evolução do “software” pode usar estes dados de teste existentes e realizar testes regressivos. Assim, o desafio é criar novos dados para testar adequadamente as mudanças e o seu impacto sobre o resto do sistema.

A evolução do “software” pode e deve ser um processo estruturado. A figura 2 ilustra o processo evolutivo do software e sua relação com o modelo PDCA (Plan,Do,Check and Act).

Processo Evolutivo

O processo evolutivo começa quando um usuário solicita uma mudança. Ele se encerra quando o sistema passa nos testes, é aceito pelo usuário e liberado para operação. Nesse ínterim, várias atividades envolvendo o pessoal de manutenção, garantia de qualidade, gerenciamento de configuração e pessoal de teste devem ser planejadas, coordenadas e implementadas. Estas atividades serão descritas a seguir [referência 2].



(figura II.3)

Gerenciamento de Mudanças

O objetivo básico do gerenciamento de mudanças é unicamente identificar, descrever e acompanhar a situação de cada mudança solicitada. Uma solicitação de mudança proporciona o veículo para registrar informações sobre um defeito, aperfeiçoamento solicitado ou melhoria de qualidade do sistema.

Análise de Impacto

Uma vez iniciada a modificação, um analista deve avaliar seu impacto sobre o sistema já existente e fazer a estimativa dos recursos necessários para completar a modificação. O objetivo geral da análise de impacto é determinar o escopo da modificação solicitada, como base para o seu planejamento e implementação.

Planejamento da Liberação do Sistema

Uma vez analisadas estas modificações, elas podem ser agrupadas como uma versão de evolução programado. Isto requer planejamento. O objetivo básico do

planejamento da liberação do sistema é determinar os conteúdos e o prazo da liberação do sistema.

Execução das Modificações

A manutenção *corretiva* enfatiza o saneamento do sistema - reparando os defeitos. A evolução corretiva é geralmente um processo reativo. Os defeitos geralmente necessitam atenção imediata

Como nos sistemas ativos, os defeitos do “software” indicam que o sistema não está funcionando como se pretendeu originalmente, ou conforme o especificado nas solicitações. Há várias situações que podem ser descritas como manutenção corretiva. Algumas delas incluem:

- Corrigir um programa que falhe no campo ou durante a testagem (falhas).
- Corrigir um programa que produza resultados incorretos (erros).

A evolução *adaptativa* inclui todos os trabalhos relativos à mudança do modo de funcionamento do “software” para que ele atenda às exigências do seu ambiente. A evolução adaptativa inclui modificações nos sistemas, inclusões, inserções, deleções, modificações, extensões e aperfeiçoamentos. A evolução adaptativa é feita geralmente como resultado de exigências novas ou modificadas. O Desenvolvimento Evolutivo Rápido pode ser usado durante a fase de adaptação da Evolução do “Software”, como um protótipo ágil para novos aperfeiçoamentos importantes do sistema.

A evolução *perfectiva* inclui todos os esforços para aumentar a qualidade do “software”. Estas atividades podem incluir reestruturação de código, criar e atualizar documentação, melhorar a confiabilidade ou eficiência, ou quaisquer outras qualidades.

Embora estes três tipos de trabalho sejam discutidos separadamente, grande parte do trabalho é realizada em conjunto. Por exemplo, aperfeiçoamentos e melhorias na qualidade são freqüentemente trabalhados e testados juntos. O projeto das mudanças de

um programa envolverá a codificação de outro. Todas essas atividades ocorrem durante o ciclo de vida da Evolução do “Software”.

Codificação

O objetivo da codificação é modificar o “software” para refletir as mudanças aprovadas representadas nos projetos de programa (físicas) do sistema (lógicas).

Teste

O passo seguinte no processo evolutivo põe em teste os projetos e códigos revisados. O objetivo principal do teste é assegurar o cumprimento das solicitações originais e das modificações aprovadas. Uma estratégia de teste evolutiva, gradual, funcionará da melhor maneira. Este tipo de processo de teste elimina defeitos ao longo do trajeto, e não quando a maior parte do trabalho está feita.

As principais atividades de teste são:

1 - Teste Humano: inspecionar as exigências, projetos, código e dados.

2 - Teste de Computador:

- Teste de Unidade - modificações por módulo

- Teste de Integração - as interfaces entre cada módulo do programa e o programa como um todo.

- Teste de Sistema - as interfaces entre programas, para assegurar que todos os sistemas atendam a todas as solicitações originais, mais as modificações acrescentadas.

- Teste de Aceitação - onde o usuário aprova o sistema revisado.

Liberação do Sistema

Uma vez que o sistema tenha sido totalmente testado e aceito, ele pode ser liberado para o uso. O objetivo da liberação do sistema é entregá-lo aos usuários, juntamente com a documentação atualizada, para instalação e operação.

II.3.5 - Importância da Padronização da Qualidade de “Software”

Considera-se que padronização é o processo de desenvolver guias (manuais), regras e convenções do aspecto de qualidade, e assegurar a qualidade do “software”, com o propósito de simplificar e determinar exatamente quais atividades são associadas com esses aspectos.

Um objetivo básico de padronização é que o acompanhamento durante todo o ciclo de vida da produção do “software” produza uma razoável segurança que a demanda e serviços proporcionam ao produto correspondente, competitivamente, para os requisitos dos consumidores.

ISO (International Standards Organizations)

Durante vários anos, organizações profissionais, nacionais e internacionais tem produzido padrões para segurança de qualidade de “software” e gerência de qualidade de “software” com diferentes abordagens, escopos, campo de aplicação, conteúdo e estruturas.

O processo de desenvolvimento e manutenção de “software” é diferente dos outros tipos de produtos industriais, sendo que houve a necessidade de uma direção adicional para sistema de qualidade onde produtos de “software” são envolvidos, levando-se em conta o atual status dessa tecnologia.

O comitê técnico ISO TC/176 da ISO, após vários anos de trabalho, publicou a série ISO 9000 em março de 1987. A família de padrões internacionais ISO 9000 define e descreve o que é exigido de um sistema de qualidade satisfatório. A ISO 9000 é o padrão internacional para sistemas de qualidade que contenham um componente projeto/desenvolvimento. A ISO 9001, 9002 e 9003 são especificamente modelos de três níveis de sistema de qualidade variando pelo projeto, fabricação, instalação e sistemas de serviços além de inspeção e teste. A ISO 9000 e 9004 são guias para a seleção e uso da série ISO 9001-9003. ISO 9001 fornece os requisitos para um sistema de qualidade em situações contratuais. Embora a ISO 9001 não é especificamente para características de produto, ou serviços que os padrões aplicariam, ela pode ser melhor aplicada para sistema de qualidade de produtos de máquina. A ISO 9003 dá diretrizes para a aplicação da ISO 9001 ao desenvolvimento, suprimento e manutenção do “software”. A ISO 9004-9002 indica diretrizes para serviços gerais, que são aplicáveis aos serviços de suporte de “software”.

Um sistema de qualidade de “software” fornece uma estrutura que facilita a sua criação e monitoração. Todo o aspecto de projeto de “software”, desenvolvimento e manutenção pode ser endereçado pelo sistema de qualidade de “software”, incluindo procedimentos, controle de qualidade e segurança de qualidade para assegurar que o “software” encontrará seus requisitos.

Definições :

Para um melhor entendimento das atividades e aplicações na área de qualidade de “software”, apresenta-se a seguir conceitos da ISO [referência 13]:

“Software”: Criação Intelectual compreendendo os programas, procedimentos, regras e algumas documentações associadas relativo as operações dos dados processados no sistema. [ISO 2382 - 1: 1984, 01.04.04]

“Software” Product: Conjunto completo de programas, procedimentos e documentação associadas e dados designados para distribuição para um usuário.

“Software” Item: Qualquer parte de um produto de “software” com passos intermediários ou com o passo final do desenvolvimento.

Desenvolvimento: Toda atividade será realizada para criar um produto de “software”.

Fase: Definindo um segmento de trabalho.

Verificação (do “software”): O processo de avaliação de produtos de uma dada fase para assegurar correção e consistência com respeito aos produtos e padrões fornecidos como (input) entrada para a fase.

Validação: (do “software”): O processo de avaliação de “software” para assegurar concordância com exigência especificada.

Características segundo o padrão internacional ISO/IEC 9126 “Information Technology “Software” Product Evaluation Quality Characteristics and Guidelines” para uso internacional:

1 - Funcionalidade

Um grupo de atributos relativos a um grupo de funções e suas propriedades específicas. As funções são aquelas que satisfazem necessidades declaradas ou implícitas.

2 - Confiabilidade

Um grupo de atributos relativos a capacidade do “software” de manter seu nível de performance sob condições determinadas, por um dado período de tempo.

3 - Usabilidade

Grupo de atributos relativos ao esforço necessário ao uso e à avaliação individual de tal uso por um grupo de usuários, declarados ou implícitos.

4 - Eficiência

Grupo de atributos ligados à relação entre o nível de performance do “software” e o volume de recursos usados sob condições dadas.

5 - Manutenção

Grupo de atributos relativos aos esforços necessários para fazer modificações específicas.

6 - Portabilidade

Grupo de atributos relativos à capacidade do “software” de ser transferido de um ambiente para outro.

II.3.6- Análise das Características de Qualidade de “Software”

Qualidade de “software” pode ser definido como uma coleção de características (atributos), tais como, eficiência, manutenção, testabilidade. O grau de importância de cada característica (atributo) varia de projeto para projeto. Entretanto, as características de qualidade são muito difíceis de serem medidas.

Para obter uma alta qualidade de “software”, em um sistema, é necessário que se leve a sério os atributos de qualidade. Os vícios adquiridos ao longo do tempo, devem ser evitados. Desde o princípio de concepção do “software”, deve-se procurar a maneira correta de desenvolvê-lo, reavaliando-o, continua e criteriosamente, cada etapa desse processo, com o objetivo final de incremento da confiabilidade do produto.

Ao examinar um sistema, é necessário avaliá-lo em termos de características (atributos), esperando encontrar dentro dos seguintes objetivos de qualidade: utilizabilidade, confiabilidade conceitual e confiabilidade da representação.

Utilizabilidade refere-se às características de utilização em um programa, sob as mais diversas formas, tanto durante a implementação como na operação ou na manutenção. Entretanto, para ser utilizado é necessário que o programa atenda às necessidades e aos requisitos que motivaram a sua construção.

Confiabilidade Conceitual refere-se às características de compreensão e de correção, em relação ao conteúdo da documentação interna e do código fonte do programa, de forma que satisfaça às suas especificações.

Confiabilidade da Representação refere-se às características de compreensão e manipulação com relação à descrição, à organização e à representação da documentação interna e do código fonte, de forma que o programa possa ser manipulado por diferentes pessoas durante sua vida útil.

Segundo o Método de Rocha [referência 22], os objetivos de qualidade são atingidos através de fatores de qualidade, que podem ser compostos por outros fatores. Objetivos e fatores não são, diretamente mensuráveis e só podem ser avaliados através de critérios. Um critério é um atributo primitivo. Nenhum fator isolado define completamente um objetivo.

A tabela II.1(Qualidade de Programas - Fontes: [referência 22]) a seguir apresenta os fatores e suas definições relacionados a cada objetivo de qualidade.

Qualidades de Programa

Objetivos	Fatores	Definições dos Fatores
Confiabilidade da Representação	Legibilidade	O programa deve ser escrito de tal forma que permita a sua compreensão, de maneira fácil, por quem não participou de seu desenvolvimento. É atingido pelos subfatores: clareza, consistência, estilo e modularidade.
	Manipulabilidade	É a característica de um Programa ser facilmente manipulado por diferentes pessoas. É atingido pelos subfatores: disponibilidade, estrutura e rastreabilidade.
Utilizabilidade	Manutenibilidade	É a característica de um programa permitir a introdução de alterações após ter sido inicialmente definido, desenvolvido e aceito como operacional. É atingido pelos subfatores: alterabilidade e consistência.
	Operacionalidade	É a característica de um programa ser oportuno e ameno ao uso, facilitando a comunicação com o usuário durante todo o tempo em que este o utilizar. É atingido pelos subfatores: oportunidade e amenidade de uso.
	Portabilidade	É a característica de um programa ser alterado de maneira fácil e adequada em configurações de equipamentos diferentes da original. É atingido pelos subfatores: independência de ambientes e configurabilidade.
	Reutilizabilidade	É a característica de um programa ter suas funções desenvolvidas, de maneira que o esforço requerido, para sua reutilização parcial ou total em outras aplicações, é menor para criar um novo código. É atingido pelos subfatores: generalidade, aplicabilidade e adaptatividade.
	Eficiência	É a característica de um programa realizar suas funções sem desperdício de recursos. É avaliado pelos subfatores: eficiência de execução e de armazenamento.
	Rentabilidade	É a característica de um programa possuir relação custo/benefício aceitável. É avaliado pelo subfator: lucratividade.
	Avaliabilidade	É a característica de um programa que retrata a facilidade em verificá-lo de modo a assegurar à execução da função, que lhe cabe. É avaliado pelos subfatores: verificabilidade e validabilidade.
Confiabilidade Conceitual	Fidedignidade	É a característica de um programa corresponder ao que foi especificado e projetado. É atingido pelos subfatores: precisão, completude e necessidade.
	Integridade	É a característica de um programa enfrentar situações hostis - dados errados ou agressões. É atingido pelos subfatores: robustez e segurança.

(tabela II.1)

II.3.7 - Análise das Influências de Falhas de “Software”

Sistemas de computação estão presentes em um grande número de aplicações industriais, série de sistema de defesa militar, tráfego aéreo, sistema de controle bancário, etc. O produto de “software” está tornando-se cada vez mais um elemento de enorme interesse, principalmente devido a seus custos, que pode vir a ultrapassar o hardware.

Projetistas de “software” tem entretanto desenvolvido numerosas técnicas para produzir “software” com um máximo de confiabilidade no estágio de elaboração. Apesar de todos os esforços, é muito difícil produzir “software” sem nenhum erro.

O “software” pode ser considerado como um sistema. Seus componentes são, por exemplo, as instruções ou os módulos que o compõem. Um módulo é então denominado “componente de “software”. Uma das principais características de um “software” é que ele é abstrato.

A falha em um sistema de “software” é o término da habilidade do “software” para executar uma função requerida. Se o programa contém erros, eles serão revelados, causando uma falha de “software”; as falhas de “software” ocorrem ocasionalmente.

Um erro de “software” é causado por um erro humano no projeto de “software” afetando uma instrução ou uma ligação entre módulos. Erros podem ser extremamente variados, por exemplo:

- defeitos relativos à manipulação do dado de entrada (estranha definição de dados de entrada, etc);
- defeitos que afetam a interface entre os diferentes módulos de um programa, isto é, número, formato, e tipo de parâmetros de subrotinas;
- defeitos envolvendo sequência de operações;
- defeitos semânticos: erro que modificam o nome da variável;
- defeitos que afetam a estrutura do programa: segmentos que não podem ser encontrados ou que não é possível terminar (final de loops);
- defeitos relativos à linguagem (sintaxe e semântica);
- defeitos que afetam operações de entrada e saída;
- defeitos de performance (tempo-real, tamanho de memória requerida);
- defeitos de projeto devido a erros na interpretação na especificação.

Os defeitos de “software” são geralmente revelados durante a fase de teste e estão apresentados e classificados a seguir (tabela II.2) de acordo com as principais categorias (porcentagem de defeitos por categoria) [referência 28]:

Categorias	Compilador	“software” Tempo-Real
Computação	6	9
Lógica	38	26
Input/Output	2	14
Manipulação Dados	15	18
Interface	13	16
Definição Dados	19	3
Base Dados	1	7
Outros	-	7

(tabela II.2)

Defeitos Lógicos acontecem com 40% de todos os defeitos e defeitos de interface com 20%. Foi observado que “input-output” ocorre mais frequentemente nas aplicações de tempo real sendo que isso envolve muita funções de “input-output”. Além disso, 2/3 dos defeitos originam-se das especificações mal definidas e 1/3 de códigos incorretos.

Quando ocorre uma falha num produto de “software”, ela pode ser reparada usando as seguintes técnicas:

- fazendo o “debug” do programa, onde o erro é detectado e corrigido;
- recomeçando programa com um dado de entrada diferente do anterior, que revelou os erros. Isso nem sempre é possível.

Geralmente, os modos de falhas (efeitos observados de falhas de “software”) são classificados em cinco passos diferentes, começando com o mais fácil de detectar e finalizando com o mais difícil:

- sistema operando parando de funcionar;
- programas param a execução com clara interpretação de “display”;
- interrupção sem nenhum diagnóstico;
- programas executando, mas produzindo resultados inconsistentes;

- programas executando, produzindo resultados aparentemente corretos pela escolha da configuração de entrada, mas que, de fato, estão errados.

Obs: Os dois últimos modos de falhas comprometem a segurança dos resultados a serem obtidos.

A confiabilidade é a habilidade que o “software” possui em executar uma dada função, sob certas condições, por um dado intervalo de tempo. Uma medida de confiabilidade é a probabilidade que o “software” irá operar sem falhas por um dado período de tempo e sob estados de condições.

A manutenção é a habilidade que o “software” possui em ser restaurado para um estado na qual ele possa executar a sua função requerida. Ela é frequentemente utilizada para suportar alterações que aumentariam a sua performance ou sua capacidade.

Principais características do sistema de “software”

- *Ciclo de vida*

O ciclo de vida do “software”, como alguns produtos industriais, pode ser particionado em vários estágios:

- definição da especificação funcional: especificação funcional precisa decrever os propósitos dos programas e suas condições de uso o mais completo possível; eles precisam preocupar-se com todas as ligações de dados de “input-output”. O sistema está consequentemente interrompendo grandes interfaces lógicas. Execução e características de qualidade são também estágios e, em muitos casos, o alto balanço entre o custo e expectativa de execução poderiam ser solicitadas. Geralmente, muitos erros originam de má especificação. Isso é a razão de porque é essencial verificar se as especificações estão de acordo com os objetivos.

- projeto e desenvolvimento: o projeto de “software” é baseado na especificação funcional acima mencionada. É preciso dividir a estrutura do “software” e essa divisão é feita em módulos. A lógica interna desses módulos é então estabelecida.

Escolher uma linguagem é uma importante tarefa. Certamente, ele pode afetar a confiabilidade do “software” posteriormente. Portanto, o desenvolvimento possui uma linguagem estruturada como uma tentativa de reduzir erros fontes relacionados com linguagem.

- processo de codificação: isso é o próximo passo. Codificação é a maior fonte de erros, que crescem com o tamanho do programa. O código é revisto antes do programa ser testado na máquina. Isso vem mostrar que “debugg” é um longo processo para os programadores; o uso de compiladores facilitou a detecção de erros antes do programa ser executado.

- verificação e validação: a conformidade do “software” com as especificações de execução e confiabilidade são verificadas. Testes são executados até mesmo porque eles são um fator determinante para a confiabilidade do sistema.

Em um sistema de programação, interfaces “hardware-software” seriam também tratadas durante o estágio de verificação, para ter certeza que “hardware-software” estão perfeitamente integrados.

- operação e manutenção: o termo “operação” refere-se a atividades envolvidas com a disponibilidade e funcionamento do “software”. “Manutenção” geralmente cobre diferentes aspectos, tais como:

- “debugging” de uma sequência de erros detectados;

- atualizando consistência em qualquer das modificações da especificação funcional ou um ajuste para uma modificação do meio;

- melhoria, ou seja, aumentar a performance ou melhoria do sistema qualidade (documentação, modularidade, etc).

“Debugging” é algumas vezes executado somente depois que um certo número de erro for registrado, o “software” corrigido é introduzido em forma de uma nova versão. Seria recomendado que o “software”, uma vez corrigido, se torne outro “software” um novo produto.

- *Testando o “software”*

Existem várias técnicas de teste disponíveis:

- teste estático: as especificações e códigos são lidos antes da execução. O objetivo é detectar(analisar) a estrutura, lógica e sintaxe de erros e verificar a compilação com as aplicações padrões. Esse processo de interpretação pode ser feito por um simples troca de documentação (especificação ou código) entre dois programadores para uma inspeção formal.

- teste simbólico: especificações ou o código são executados simbolicamente pelas variáveis computacionais usando seus nomes simbólicos. O objetivo é analisar o comportamento ao londo do programa.

- teste dinâmico: são executados uma série de testes durante a execução do programa, e a execução e resultados são verificados. Ocorre três tipos:

- teste baseado na estrutura do “software”;
- testes baseados nas definições das funções para serem executadas;
- testes baseados nos erros detectados.

- *Segurança da Qualidade*

Qualidade de “software” é como a qualidade de qualquer outro produto; o produto precisa estar de acordo com a necessidade do usuário. Para assegurar qualidade de um produto, primeiro é preciso definir o nível de qualidade, e então determinar o mecanismo para assegurar que o nível escolhido é satisfatório. Entretanto , é difícil definir o nível de qualidade e seu relevante mecanismo de controle , devido a duas

características específicas do produto de “software”: primeiro, “software” é frequentemente um produto único; segundo, ele é um objeto essencialmente abstrato.

Para evitar critérios subjetivos em contratos industriais, várias associações e departamentos governamentais tem elaborado e preparado padrões em segurança da qualidade de produtos de “software”.

Ao mesmo tempo, numerosas técnicas tem sido desenvolvidas para tornar produto de “software” o mais confiável possível. Por exemplo:

- programação estruturada : o objetivo é quebrar o programa em módulos, cada módulo com sua especificação particular, programação específicas e testes.

- programação top-down: essa técnica é agora recomendada. Originalmente, os “software” eram programados em módulos e os módulos eram integrados de forma a compor o programa. Programas mais seguros estão fazendo o processo reverso; programação top-down. O corpo do programa é escrito primeiro, com módulos considerados como uma caixa preta; então, esses módulos são escritos em paralelo com o corpo.

- linguagem estruturada: essas são recomendadas e fazem o possível para eliminar instruções como GO TO, que universalmente são consideradas as causas de muitos erros. O uso de uma linguagem de alto nível também melhora a confiabilidade do programa.

Dependendo do tamanho e complexidade, qualidade de “software” é produzida para minimizar a quantidade de erros pelo uso dessas diferentes técnicas através de processo de projeto de “software”. Entretanto, tem-se observado que o “software” mesmo projetados de acordo com com essas mais recentes técnicas, ainda possuem erros. Assim sistema de medida de qualidade de “software” e confiabilidade de segurança estão começando a ter uma crescente impotência.

II.4 - Comentários

O capítulo apresentou uma tentativa de se ter uma abordagem na área de qualidade de “software”. Mesmo que este estudo se encontre incompleto, não invalida a sua utilização nos sistemas reais em desenvolvimento. Qualquer que seja a metodologia adotada, o importante é a sua utilização, sendo que isso certamente auxiliará a aumentar a qualidade do sistema produzido.

Como já foi dito, nos últimos anos são inúmeros os esforços que tem sido feito para desenvolver padrões e guias relacionados com a qualidade de “software”, tendo em vista as suas especiais características de processos e produtos de “software”.

O enfoque mostrado em melhoria da qualidade está na satisfação do cliente, a atual definição exige que não apenas supram as suas necessidades, mas que os surpreendam e que os satisfaçam acima e além das suas expectativas. A melhoria da qualidade leva a custos menores, com menos erros a remover e menos trabalho a refazer, o que faz com que a produtividade aumente, aumentando a demanda pela capacidade de desenvolvimento e manutenção do seu “software”, propiciando o crescimento da empresa.

Aqui foi introduzido também o assunto para identificar as principais características de qualidade de “software” (programas) e ainda foi focalizado uma análise das falhas do “software”, de forma que os produtos de “software” após serem colocados em operação, possam ter uma vida útil longa e produtiva.

CAPÍTULO III

QUALIDADE DE SISTEMA ESPECIALISTA

III.1 - Introdução

Sistema especialista é um sistema de “software” que fornece uma ferramenta de apoio para resolver problema de domínio específico. Em um sistema especialista existem basicamente dois componentes chaves, que são: a base de conhecimento e o mecanismo de inferência. A base de conhecimento é derivada do conhecimento do especialista e experiência na área, como resultado da etapa de aquisição de conhecimento. De um modo geral contém dois tipos de conhecimento: o primeiro tipo é o “domínio” - o amplo compartilhamento de conhecimento que é comumente ajustado entre profissionais. O segundo tipo é o “conhecimento heurístico” - um conjunto de regras que direcionam o uso de conhecimento para resolver problemas num domínio particular. Em outras palavras, ele é o conhecimento de boa prática e de bom julgamento na área que um especialista humano adquire sobre anos de experiência de trabalho. O mecanismo de inferência é o “cérebro” do sistema especialista. É uma regra ou um conhecimento interpretado. Essencialmente, é um programa de computador que fornece uma metodologia de raciocínio sobre informações da base de conhecimento para formular as respostas. O raciocínio é usualmente em forma de um encadeamento de regras “se-então”. Se o encadeamento começa de um conjunto de condições (entradas) e move para algumas conclusões (respostas), ele é chamado de método “forward chaining”. No caso em que a resposta é conhecida primeiro e possui o raciocínio retroativo é chamado de método “backward chaining”. São esse métodos que formam o corpo do mecanismo de inferência.

Atualmente essa tecnologia vem produzindo um extraordinário número de sistemas de aplicações. O elevado padrão da qualidade de “software” dos fornecedores e clientes vem apresentando ultimamente a necessidade de que uma maior atenção seja

dada na avaliação da qualidade de sistemas especialistas. Este tópico vem se tornando cada vez mais importante e tem recebido uma grande atenção tanto de forma teórica como prática. Entretanto, toda a literatura da área indica que a avaliação de um sistema especialista é uma tarefa muito difícil. As propostas de validação, verificação e teste de “software” convencional para sistemas especialistas tem tido pouco sucesso, tendo em vista que os sistemas especialistas são substancialmente diferentes dos sistemas de “software” tradicional. Um sistema especialista não é projetado e desenvolvido da mesma forma que um “software” tradicional. Ele não passa da especificação de projeto para a implementação e também não é baseado em conceitos como de modularização e refinamento “top-down”.

Sistemas especialistas contam com os conceitos de prototipagem, projeto interativo, desenvolvimento evolutivo e programação progressiva. Portanto, não compartilham o mesmo conceito de prototipagem proposto em engenharia de “software”, pois a prototipagem de um sistema de “software” tradicional pretende validar as necessidades, assegurar qualidade e uso da resolução da aplicação. A prototipagem de um sistema especialista é principalmente dedicada em experimento, refinamento e decisões sobre técnicas de validações e sistemas de projeto e requer análises e validações parciais. Consequentemente um sistema especialista se baseia numa característica de ciclo de vida muito específica e seu desenvolvimento é baseado em métodos e técnicas também muito específicas, consideravelmente diferente da tradicional engenharia de “software”.

No desenvolvimento de aplicações de sistemas especialistas determinadas características são consideradas importantes e críticas para se obter uma boa avaliação. Durante a fase de projeto e construção, são necessárias metodologias e técnicas para dimensionar o grau de performance e a qualidade obtida pelo sistema, especificando etapas do processo desenvolvimento. Após o término de cada etapa de desenvolvimento ocorre uma avaliação, onde o trabalho é revisto e cuidadosamente testado, e decisões desconhecidas de projeto e todos os erros são corrigidos e ainda novas indicações apropriadas para a continuação do projeto são estabelecidas. A próxima etapa seria a conclusão e teste de toda a aplicação incluindo o resultado da certificação da

performance e qualidade, atestando que o sistema se comporta de acordo com as necessidades do usuário, tornando-se apto para ser divulgado e colocado em operação. Pode-se ainda considerar a possibilidade de se ter diferentes implementações proposta para o mesmo problema, provocando conseqüentemente uma comparação entre eles, sendo que essas comparações normalmente são feitas com o uso da técnica de “benchmarks”.

III.2 - Segurança da Qualidade

Tentando cada vez mais garantir a alta performance dos sistemas de “software”, a segurança da qualidade tem assumido uma grande importância.

Durante vários anos, organizações profissionais internacionais e nacionais vem produzindo padrões para segurança da qualidade de “software” e gerência de qualidade de “software” com diferentes abordagens, escopos, campo de aplicação, conteúdo e estruturas. Considera-se que padronização é o processo de desenvolver manuais, regras e convenções do aspecto de qualidade e assegurar a qualidade do “software”, com o propósito de simplificar e determinar exatamente quais atividades são associadas a esses aspectos. Um objetivo básico de padronização é que o acompanhamento durante todo o ciclo de vida da produção de “software” forneça uma razoável segurança nos requisitos dos consumidores.

Atualmente é grande o número de clientes insistindo na padronização como pré-requisito para sistemas especialistas, sendo que a implementação de padrões precisaria de atenção especial dentro da área de sistemas especialistas.

Em termos de padronização pode-se incluir uma qualidade de sistemas baseada num método evolucionário, onde se tem uma gerência de diferentes versões, que engloba:

1. Gerência de configurações para assegurar que diferentes versões de “software” estão corretamente e consistentemente identificadas e que a versão corrente está claramente apresentada como tal.
2. Documentação de controle para garantir o mesmo do documento que a acompanha.
3. Distribuição e instalação para garantir que somente a produção e versão final do “software” estão continuamente liberadas para os usuários.

Praticamente todo sistema de qualidade assume um modelo de encadeamento de desenvolvimento no seu ciclo de vida. Frequentemente em um sistema especialista, o desenvolvimento de um protótipo vem como um precursor para finalmente depois ter a versão final do sistema.

Na prática, o padrão ISO não é muito consagrado nesta área, especificando meramente que uma correta metodologia sistemática é empregada. Ao mesmo tempo, o argumento é que não é possível usar uma estrutura convencional para desenvolver aplicações de sistemas especialistas.

As equipes de desenvolvimento de sistemas especialistas se tornaram mais criativas e menos restritivas e gerenciar essa cultura é necessária e aceita, caso o sistema formal proposto não venha destruir a criatividade normalmente usada no desenvolvimento de sistemas especialistas. O maior problema visto pelas equipes de desenvolvimento de sistemas especialistas é a validação do conhecimento. O conhecimento não é validado da mesma forma que um sistema com características de soluções algorítmicas. Não existe referência dizendo que uma resposta possa ser testada, existe simplesmente a opinião do especialista. A segurança da qualidade de um produto apresenta duas propostas: a primeira é assegurar que o produto tenha todas as necessidades e expectativas do usuário e a segunda é fornecer melhoria de dados para processo de qualidade.

Segurança da qualidade é um processo para garantir uma necessidade complexa que o sistema especialista precisa satisfazer. Ele envolve duas fases: primeiro, a propriedade estrutural do conhecimento do sistema precisa ser validada, e então a correção da performance do sistema precisa ser demonstrada. Usualmente, a primeira atividade é chamada de verificação e a segunda de validação.

Como em todos os sistemas, um sistema especialista precisa ser validado antes de ser colocado dentro de uma aplicação. A principal proposta da validação é examinar se o modelo reproduz adequadamente o mundo real.

O processo de validação se baseia praticamente em dois estágios:

1. Validação do modelo básico: esse é o processo para assegurar a produção de bons resultados. Isso está completamente separado do sistema. Ele pode ser realizado pelos especialistas da área. Qualquer cliente em potencial pode conferir a confiabilidade do modelo básico.

2. Validação da implementação: esse é o processo para assegurar que o sistema implementado seja uma clara representação do modelo. Esse estágio é feito simplesmente pelo documento baseado no modelo para que o sistema possa ser verificado. Verificação requer pouco conhecimento do domínio do problema, assim o processo verificação pode ser realizado pelos especialistas de "software". Embora isso não seja uma técnica genérica, ela fornece uma estrutura e uma meta simples para verificação do conhecimento.

III.3 - Avaliação de um Sistema Especialista

Durante os últimos 15 anos um grande número de abordagens e métodos tem sido proposto. Embora a avaliação de um sistema baseado em conhecimento seja intuitivamente vista para transmitir um significado específico, vários significados tem sido atribuído a ela em diferentes propostas, objetos, e métodos de avaliação.

A avaliação orientada a uma base de conhecimento de um sistema tem como objetivo identificar possíveis problemas na representação do conhecimento de um nível lógico, tais como consistência e perfeição. Uma base de conhecimento é consistente se não existe forma de contradizer os dados de entrada validados, assim a consistência é verificada para identificar os problemas como conhecimento conflitante, encadeamento do ciclo de inferências, ambiguidades, etc. Na literatura, o conceito de consistência também inclui verificação de outros possíveis problemas da base de conhecimento, como redundância não necessária. Uma base é considerada completa se ela pode enfrentar todas as situações que possam chegar ao seu domínio. Assim perfeição é verificada principalmente para identificar problemas derivados de conhecimentos perdido. A consistência e a perfeição da base de conhecimento verificadas e os algoritmos projetados para isso, dependem muito do formalismo adotado para a representação do conhecimento. Assim, essa abrangência pode ser caracterizada através de dois atributos: primeiro, o formalismo de representação do conhecimento aplicado para, e segundo o conjunto de base de conhecimento verificados. Esta abordagem é derivada da lógica formal. Ela é baseada na semântica de verificação da base de conhecimento através da inspeção sintática e manipulação de regras e “frames”, interpretadas como expressão lógica. Certamente, estas verificações geralmente são capazes de descobrir os problemas da base de conhecimento.

A abordagem baseada no critério de avaliação engloba um método para medir o sistema baseado em conhecimento. Os critérios de avaliação são definidos informalmente, suas medidas são frequentemente vagas. Pode-se citar quatro critérios de avaliação:

- validação: refere-se a correção, qualidade, segurança e perfeição de respostas (soluções, decisões, recomendações, etc);
- usabilidade: refere-se a qualidade da interação homem-máquina, entendimento, facilidades de explanação;
- confiabilidade: inclui confiabilidade de hardware e “software”, resistência;

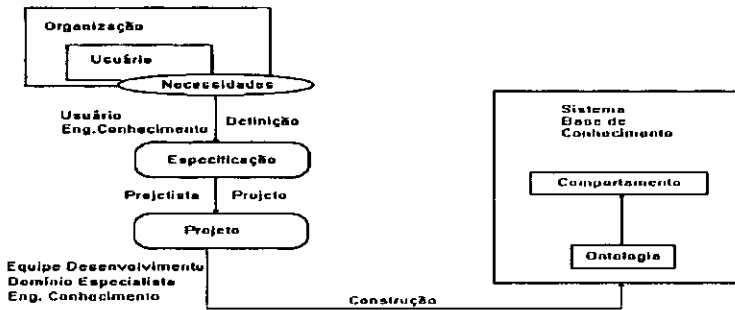
- eficiência: inclui custo-eficiência (custo e benefícios envolvidos na resolução de tarefas, em termos de tempo de resposta e uso do recurso do computador), tempo de desenvolvimento e custo, manutenção e extensibilidade.

Esses critérios são derivados de forma experimental, usados numa base de conhecimento em situações controladas. Isso envolve executar um sistema (um protótipo ou um sistema final) selecionando testes de casos, e acessando os resultados. Essa tarefa é feita diretamente se existe um padrão de referência, na qual a saída do sistema possa ser comparada. Caso contrário, podem ser usados métodos para medir a correspondência entre comportamento de soluções de um sistema baseado em conhecimento e de um especialista humano. Algumas vezes o tempo do especialista não está disponível, e é muito oneroso. Tais testes em casos específicos podem também ser realizados usando um modelo do comportamento esperado do sistema, baseado no conhecimento, e recorrendo ao especialista humano somente para casos mais difíceis; métodos de ajuste também são inspirados no clássico “Turing Test”.

A literatura propõem um conjunto de necessidades para se ter uma avaliação da base de conhecimento ideal, isso inclui:

- uma definição precisa da avaliação do sistema baseado em conhecimento
- uma forte base de conceitos de avaliação do sistema baseado em conhecimento, onde é claramente declarado que características do sistema baseado em conhecimento serão avaliadas e reavaliadas;
- uma metodologia geral, definida de acordo com o conceito da avaliação do sistema baseado em conhecimento adotado, especificando quais medidas e como usá-las;
- um procedimento eficiente para aplicar uma metodologia geral para casos reais.

Os principais passos do projeto e construção de um sistema baseado em conhecimento [referência 11] são mostrados na figura III.1, que define de uma maneira geral o conceito de avaliação do sistema baseado em conhecimento:



(figura III.1)

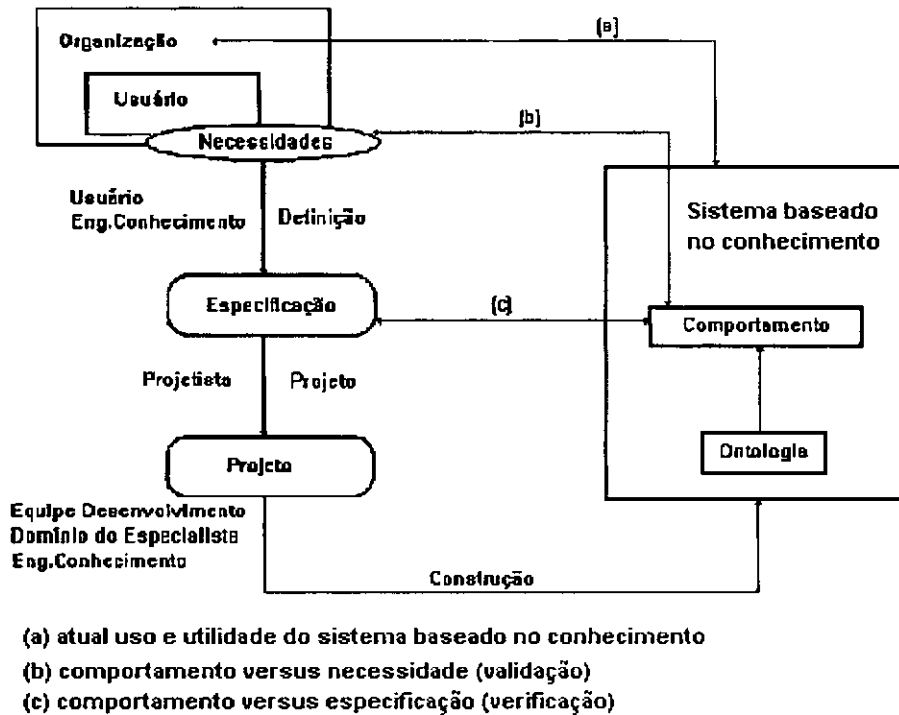
- primeiro, o usuário assistido pelo engenheiro de conhecimento, especifica suas necessidades em termos de um conjunto de características de procedimento que o sistema poderia apresentar quando estivesse operando (especificação de um sistema baseado em conhecimento);

- depois, o projetista planejará a estrutura do sistema baseado em conhecimento (arquitetura, representação da estrutura do conhecimento, raciocínio algorítmico, interfaces e suporte do sistema) de maneira que ele possa produzir o comportamento desejado de acordo com a especificação do sistema (projeto de um sistema baseado em conhecimento);

- finalmente, a equipe de desenvolvimento implementará o projeto da estrutura do sistema (chamado de sistema vazio) enquanto isso o domínio do especialista, assistido pelo engenheiro de conhecimento irá definir o seu conteúdo (terá a atual base de conhecimento). Quando o sistema baseado em conhecimento estiver completo, ele caracterizará uma certa ontologia e quando em operação, produzirá um certo comportamento.

O conceito de avaliação [referência 11] pode ser analisado conforme a figura

III.2.



[figura III.2]

A avaliação é dita como uma medida de um conjunto de interessantes propriedades técnicas de um sistema baseado em conhecimento, independentemente do seu atual uso numa dada organização, ou seja, é uma avaliação das propriedades intrínsecas, como por exemplo, a qualidade do sistema de informação e decisão, a correção da técnica de raciocínio usada, a qualidade da interface homem-máquina, e a eficiência do sistema baseado em conhecimento.

A medida do sistema (figura III.2(a)) baseado em conhecimento envolve a investigação do último critério de sucesso da aplicação do sistema baseado em conhecimento, ou seja, como atualmente ele é usado e se é usado para produzir. A medição não está diretamente envolvida com avaliação do sistema, mais precisamente com a apreciação e medida das modificações induzidas pelo uso no contexto onde ela é aplicada. Deste modo, ela tem diferentes formas objetivas de avaliação e além do mais, ela envolve métodos específicos de investigações e ferramentas, que estão mais

diretamente ligados com ciências humanas (por exemplo, sociologia, psicologia, economia, etc.) do que com ciência de computação.

Para *definir avaliação*, duas coisas precisam ser especificadas: quais características do sistema baseado em conhecimento serão avaliadas, e quais avaliações serão efetuadas. Até agora a característica mais direta e objetiva de avaliação do sistema baseado em conhecimento é sobre o seu comportamento. Sua ontologia está presente somente na sua operação. É preciso um padrão de referência para que o comportamento do sistema baseado em conhecimento possa ser comparado e acessado. Dependendo da escolha do padrão de referência, diferentes conceitos de avaliação podem ser obtidos.

Assumindo as necessidades como um padrão de referência de avaliação do comportamento do sistema baseado em conhecimento, obtem-se o conceito de validação (figura III.2(b)). Por outro lado, adotando-se a especificação como um padrão de referência, obtem-se o conceito de verificação (figura III.2(c)).

Portanto, a proposta de validação é para determinar se o sistema baseado em conhecimento satisfaz em performance a tarefa para qual ele foi criado, enquanto o objetivo da verificação é determinar se a implementação do sistema baseado em conhecimento satisfaz completamente a sua especificação.

A diferença entre validação e verificação originam do fato que geralmente a especificação não é uma completa e correta descrição de necessidades. Além do mais, em um sistema baseado em conhecimento é frequentemente impossível especificar completamente o sistema antes do começo da implementação, embora seja importante especificar tanto quanto possível o desejado comportamento do sistema baseado em conhecimento no começo do projeto. Desenvolvimento interativo é típico da tecnologia de sistema baseado em conhecimento, sempre alterando as especificações feitas durante o protótipo. Assim a verificação precisa ser entendida como um processo interativo, que muda quando necessário no contexto da corrente especificação.

III.4 - Validação de um Sistema Especialista

No contexto do sistema especialista, o principal objetivo de validação é investigar a segurança, perfeição e consistência de resposta recomendada pelo sistema especialista. Segurança significa produção de uma resposta que é melhor ou ao menos muito próxima da resposta do especialista humano. Perfeição implica que dentro do domínio da aplicação, toda possível resposta pode ser derivada, toda entrada aceitável produzirá uma resposta. Consistência significa que o sistema daria uma resposta similar quando dado uma entrada similar. Validar é a “qualidade” da base de conhecimento e do mecanismo de inferência em termos de perfeição e segurança dos fatos, regras e raciocínio algorítmico armazenados.

Ter uma equipe para testar um sistema de “software” significa um aumento na probabilidade de que os novos “software” terão confiabilidade e toda as necessidades do usuário. Testar um sistema requer que mais dinheiro seja gasto no projeto, mas tem sido provado repetidamente que testes garantem uma economia ao longo da execução, se ele é feito cedo no seu ciclo de vida, quanto mais cedo o erro for corrigido, mais acessível sairá o projeto.

Para sistemas especialistas, duas questões devem ser avaliadas:

- 1 - Estão as necessidades do sistema especialista adequadas, isto é, pode um leigo, ajudado pelo sistema especialista, fazer o trabalho de um especialista?
- 2 - O sistema está amigável para o usuário? Pode um usuário utilizar sem a menor dificuldade?

Há uma variedade de métodos para validar sistemas especialistas, um deles é a combinação do uso de validação qualitativa com teste objetivo, onde a validação qualitativa envolve uso de comparação subjetiva de performance. Isso significa que membros da equipe de projeto, usuários e especialistas da aplicação da qual o sistema especialista foi escrito serão usados para comparar a performance do sistema com a atual

performance do especialista. Testes objetivos significam simplesmente estruturar um objetivo específico para concluir o teste.

Para todo produto de "software", não só sistema especialista, é essencial que a maior ênfase seja dada nos testes das necessidades. Se isso não for feito, toda a proposta de validação do sistema pode ser frustrada.

Uma vez que as necessidades para o teste objetivo foi escolhido para a validação do sistema especialista, a equipe de teste precisa determinar exatamente que partes específicas do sistema serão validadas. As partes do sistema para serem validadas poderiam incluir:

- 1 - Resultados intermediários do sistema.
- 2 - Resultados finais (conclusões).
- 3 - Processo de raciocínio do sistema.

Somente um, uma combinação de, ou todos os três poderiam ser validados. Existe uma grande chance de precisão se todos forem validados. Entretanto, nem sempre é possível validar todas as partes do sistema de uma vez, até o sistema estar completo.

No estágio de desenvolvimento, somente parte de raciocínio pode ser validado. O resto teria que esperar até um estágio posterior de desenvolvimento para validar o resultado intermediário e/ou conclusões do sistema.

Uma vez que a equipe de teste sabe o que deve ser validado, eles necessitarão algumas vezes validar o resultado do sistema especialista novamente. Usando o método de validação qualitativa, eles usam um especialista humano ou executam casos de teste para validar o sistema novamente. O especialista precisa estabelecer um nível de performance aceitável para o sistema baseado na sua experiência.

Há diversos tipos de métodos qualitativos que podem ser usados. Normalmente não é suficiente usar somente um método de teste, de forma que vários testes devam ser usados.

Um tipo de teste que pode ser feito é chamado de “Turing Test”. Num “turing test” o especialista humano e o sistema especialista ficam em salas separadas e com um problema para resolver. Após a resolução do problema por ambas as partes, a conclusão de ambos são comparadas. A razão pela qual as partes ficam em diferentes salas é para que o especialista não veja o que o sistema está fazendo para não ser influenciado por ele. Se ambos, resultado do especialista e resultado do sistema concordarem, então pode-se dizer que o sistema funciona corretamente.

Outro tipo de teste que é feito é chamado de “Predictive Validation Test”. Este método qualitativo de validação conta com um resultado de casos de teste anteriores que envolvem especialistas humanos. Depois o sistema é executado, os resultados são comparados com os resultados dos casos de teste histórico para determinar a validação do sistema.

Ainda outro tipo de teste de validação qualitativa é chamado de “Field Testing”. Não é frequentemente feito, e nunca é utilizado quando envolve “software” “crítico”, isto é, com resultados que podem ser catastróficos (por exemplo, um sistema especialista de diagnóstico de coração). Um “field test” coloca o sistema especialista no campo e tendo os usuários relatando qualquer insuficiências ou falhas que eles encontram como erros que ocorrem no dia a dia normal de uso do sistema. A maior desvantagem deste tipo de teste é que ele pode ser usado somente com “software” não “crítico” , isto é, sem graves consequências.

Um outro tipo é o “Subsystem/System Validation Test”. Este envolve quebrar o sistema especialista dentro de uma estrutura de subsistema e testar cada subsistema separadamente. Esta técnica poderia ser usada com sistemas especialistas grandes e facilita o teste. Entretanto, este método não garante que todos os subsistemas agrupados trabalhem corretamente. Por essa razão, após todos os subsistemas terem sidos

validados, o sistema precisa ser validado como um todo num sistema de validação de teste.

Seis idéias básicas poderiam ser feitas num teste de um sistema especialista:

- 1 - Juntar um objetivo para cada teste.
- 2 - Determinar qual a parte do sistema para teste
- 3 - Decidir o que usar para validar novamente
- 4 - Projetar os testes cuidadosamente
- 5 - Implementar o teste corretamente
- 6 - Repetir o processo até que o sistema especialista seja totalmente testado.

III.5 - Verificação de um Sistema Especialista

Verificação é o conceito que mais se encaixa dentro de nosso significado intuitivo. As necessidades geralmente não possuem uma representação explícita, presente somente sem expressão na mente do usuário, sendo problemático aceitá-las como um padrão de referência para avaliação. Além do mais, validação não está diretamente e unicamente envolvida por avaliação de propriedades intrínsecas de um sistema baseado em conhecimento. Pode ser vista como verificação mais a certificação da especificação do sistema baseado em conhecimento versus necessidades. Finalmente, pode-se considerar a proposta de avaliação do sistema baseado em conhecimento tal como:

- durante o projeto e construção do sistema baseado em conhecimento, comparação do comportamento atual do sistema com a especificação do sistema (ou seja, verificação) é importante para controlar os resultados das atividades desenvolvidas e, se necessário, refinar ou corrigir passos inapropriados;

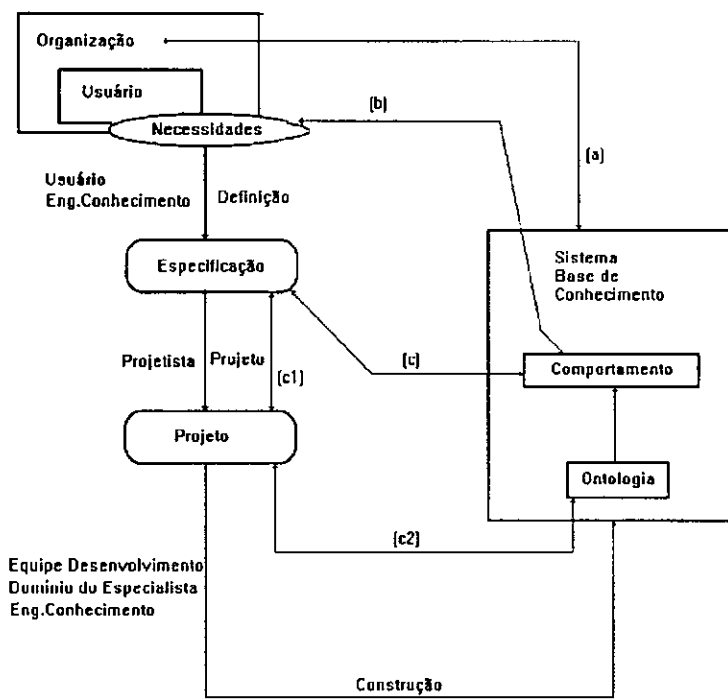
- para o final do projeto, quando a aplicação estiver completa, comparação do comportamento final do sistema com a especificação definitiva do sistema (verificação) é um componente essencial de certificação e aceitação;

- quando diferentes implementações forem propostas para o mesmo problema, comparação e "ranking" podem somente serem feitas levando em consideração uma especificação comum do sistema baseado em conhecimento (ou seja, através de verificação).

A análise e correção de possíveis insuficiências na especificação do sistema com respeito às necessidades que surgirem durante o processo de desenvolvimento é um importante passo na construção do sistema, mas não diz respeito ao resultado da avaliação do sistema baseado em conhecimento.

O termo avaliação do sistema baseado em conhecimento foi discutido como resultado da verificação, projetado como valor do comportamento atual do sistema baseado em conhecimento versus especificação do sistema baseado em conhecimento (figura III.2(c)).

Medir performance (conceito específico de avaliação do sistema baseado em conhecimento) é uma tarefa difícil; de fato, é difícil analisar o atual comportamento do sistema baseado em conhecimento e compará-lo com as especificações. Analisando o comportamento do sistema baseado em conhecimento de forma ontológica através de execução simbólica, obtém-se um resultado especulativo muito difícil. Compará-lo com a especificação do sistema baseado em conhecimento não é uma boa decisão. Além do mais, na prática, medidas de performance podem ser baseadas em técnicas de testes. Entretanto, testes podem fornecer uma avaliação parcial e incerta de performance; de fato, teste pode não ser completo e a necessária limitação para exemplos de casos selecionados implicam que as avaliações obtidas possam somente ser próximas da medida exata. Portanto, medir performance diretamente da forma de definição não parece totalmente satisfatório.



- (a) uso e utilidade do sistema base de conhecimento dentro da organização
 (b) comportamento versus necessidades (validação)
 (c) comportamento versus especificação (verificação)
 (c1) projeto versus especificação de acordo com os princípios do projeto
 (c2) ontologia versus projeto de acordo com o princípio da construção

(figura III.3)

Para medir a performance (figura III.3) [referência 11] pode-se avaliar o comportamento do sistema baseado em conhecimento versus sua especificação (figura III.3(c)). Ou pode-se ter o seguinte: primeiro, avaliação do projeto do sistema baseado em conhecimento versus especificação, de acordo com aceitação do projeto principal, verificando sua adaptabilidade para produzir o comportamento desejado e observando os possíveis erros de projeto (figura III.3(c1)) e depois avaliar a ontologia atual do sistema baseado em conhecimento versus seu projeto, de acordo com a aceitação da construção principal e observando possíveis erros de construção (figura III.3(c2)).

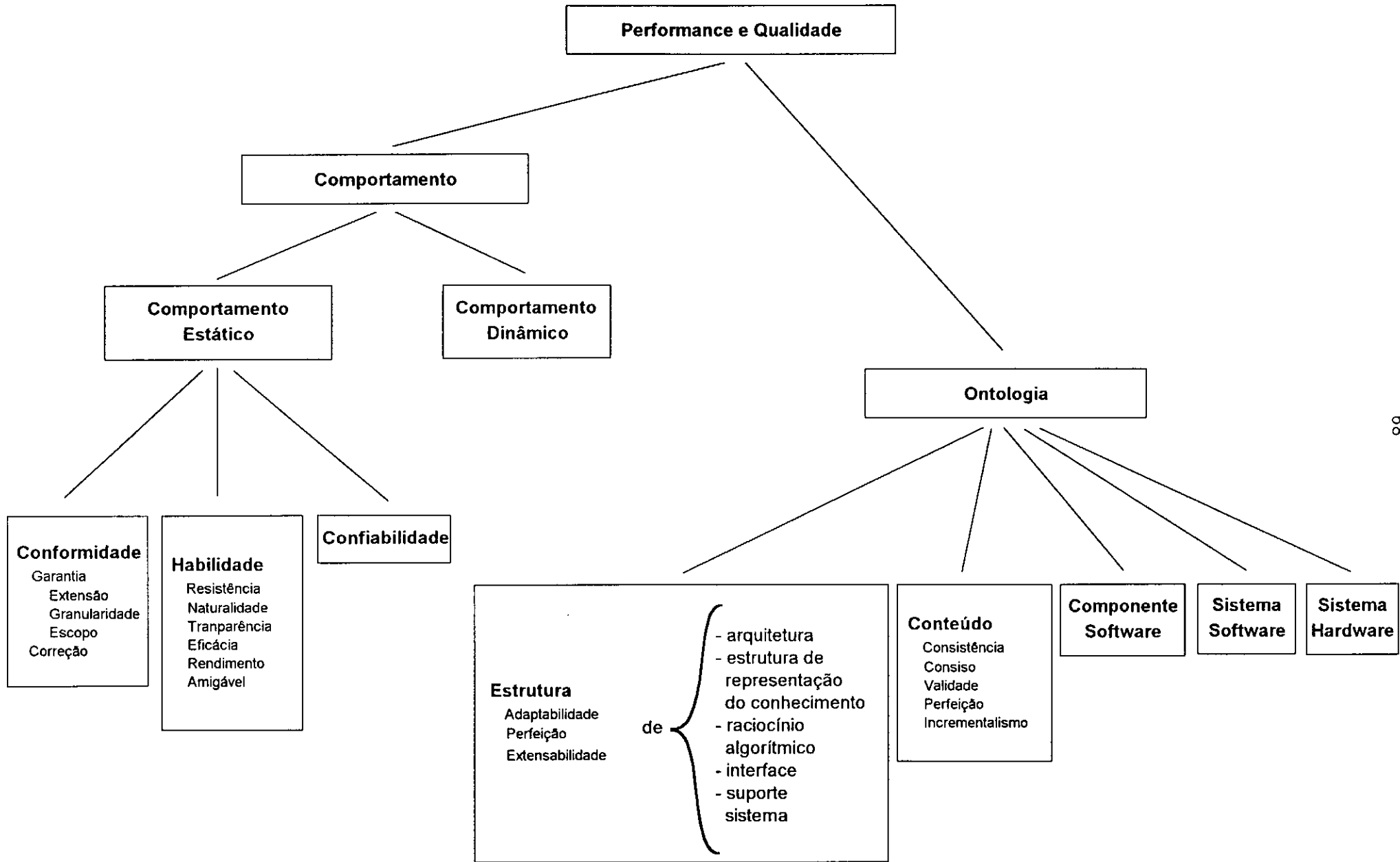
Performance e qualidade consiste de um comportamento e de uma ontologia. Pode-se dizer que o comportamento e a ontologia podem ser decompostos dentro de componentes independentes, usando um método de classificação hierárquica. Do mesmo modo, os componentes obtidos podem ser adicionais e decompostos em mais refinamentos, até que componentes de importância apropriadas sejam obtidos, sendo

considerados componentes elementares. Através deste procedimento, obtem-se uma árvore de componentes, uma taxonomia, onde o significado de cada nó (um componente genérico) é fornecido pelo conjunto de seus sucessores, e o significado desse nível (componente elementar) pode ser definido intuitivamente.

A definição de performance e qualidade através da decomposição “top-down” resulta numa definição organizada em forma de níveis de granularidade decrescente (ou seja, está estruturada) e pode ser estendida quando necessário ou apropriada através de um futuro refinamento dos níveis dentro dos componentes (ou seja, está aberto). O resultado desse procedimento de definição é a taxonomia como mostra a figura III.4 [referência 11].

Na taxonomia novos componentes podem ser eliciados refinando os níveis da taxonomia dentro dos componentes (comportamento dinâmico, correção e resistência são os principais candidatos desse tipo de extensão). A escolha de parada da decomposição é arbitrária, até que o nível de detalhes atingido for suficiente para demonstrar o conceito de performance e qualidade.

A definição de métodos para medir a performance e qualidade pode ser alcançada através da medida independente de seus componentes elementares e depois combinar essas medidas obtidas. Isto enfoca num aspecto específico de performance e qualidade como tempo, reduzindo a complexidade da tarefa de avaliação e melhorando a precisão de medidas. Certamente, o caminho específico para compor as medidas separadas de componentes elementares em ordem para obter uma avaliação global de performance e qualidade precisa ser projetada muito cuidadosamente.



(figura IV.4)

Observando a figura III.4, o comportamento de um SBC (sistema baseado em conhecimento) inclui dois componentes principais, denominados:

- Comportamento dinâmico envolve o comportamento de um SBC sobre um intervalo de tempo quando sua ontologia é alterada. Ele representa uma reação do SBC para as modificações de sua estrutura e base de conhecimento pela equipe de projeto durante seu desenvolvimento ou pela equipe de manutenção durante sua vida operacional.

- Comportamento estático envolve o comportamento de um SBC sobre um intervalo de tempo quando sua ontologia é fixada. Ele representa uma resposta para uma entrada que recebe durante uma operação normal, é o tipo de comportamento observado pelo usuário durante uma seção de trabalho.

O comportamento estático engloba três componentes, denominados:

- conformidade, se refere a que um SBC faz durante sua operação, ou seja, para os resultados de suas soluções dos problemas;

- habilidade, se refere como um SBC comporta durante sua operação, ou seja, como o caminho e forma de solução de seus problemas é atualmente realizado e apresentado para o usuário;

- confiabilidade, se refere a capacidade do SBC em evitar uma variável inexplicável de comportamento, ou seja, produzir a mesma saída, dado a mesma entrada todo o tempo.

Por exemplo, o fato de que uma solução fornecida por um SBC de um dado problema estar correta envolve conformidade, o fato de que o tempo de resposta é muito rápido ou que a solução produzida é facilmente compreendida pelo usuário envolve habilidade, e o fato de que diante de um mesmo problema submetido ao SBC com diferentes tempos, o sistema fornece a mesma solução envolvendo confiabilidade.

Conformidade, habilidade e confiabilidade são independentes e possuem distintas dimensões de comportamento.

A Conformidade compreende dois seguintes componentes:

Garantia representa o conjunto de conceitos de domínios e tipos de problemas que um SBC pode ocupar-se, e possui três componentes:

- extensão, representa o conjunto de entidades, propriedades e relações que um SBC pode ocupar-se.

- granularidade, demonstra o nível de detalhes da representação de entidades, propriedades e relações que um SBC pode ocupar-se.

- escopo, que representa a variedade de diferentes tipos de problemas que um SBC pode ocupar-se (e, conseqüentemente, o conjunto de entradas aceitáveis).

Correção representa a habilidade do SBC gerar soluções certas para o problema submetido dentro de sua atual garantia. Correção também abrange o conceito de otimização, isto é, a propriedade de soluções geradas pelo SBC em ser uma boa solução para o problema considerado, de acordo com o critério dado e com outras possíveis comparações.

Habilidade engloba seis componentes, que são:

- resistência, representa a habilidade do SBC comportar num aceitável e próximo ao limite de sua garantia.

- naturalidade, demonstra a propriedade do comportamento de um SBC ter um fácil entendimento para o usuário, ou seja, a habilidade para fazer um natural e apropriado casamento entre sua representação interna e o modelo mental do usuário.

- transparência, representa a possibilidade do usuário inspecionar o modo interno de operação do SBC, como ter um entendimento de seus problemas resolvidos e uma justificativa de seu comportamento.

- eficiência, representa a capacidade do SBC fornecer uma boa solução de um problema com respeito a execução de seu algoritmo de raciocínio e com o uso do conhecimento disponível.

- rendimento, representa a capacidade do SBC fornecer uma boa resolução do problema com respeito ao uso dos recursos físicos disponíveis (tempo, memória do computador, etc).

- amigável, representa a capacidade do SBC suportar fácil e eficiente interações com o usuário, especialmente em relação a aspectos linguísticos de comunicação homem-máquina.

A ontologia do SBC envolve cinco principais componentes:

- estrutura, que se refere a arquitetura, representação da estrutura do conhecimento, raciocínio algorítmico, interfaces e suporte do sistema (editor da base do conhecimento, sistema de justificativa, etc) que são projetados para implementar uma base. Deve-se observar que não se pode confundir a estrutura do SBC com técnicas gerais (arquitetura principal, técnicas de representação de conhecimento, paradigmas de raciocínio, etc.) adotadas para o projeto. De fato, a estrutura do SBC é uma especialização da técnica geral para o atual domínio da aplicação considerada. Por exemplo, produção de regras organizadas dentro dos módulos de conhecimento e processadas através de uma mecanismo de inferência são técnicas gerais de representação de conhecimento e paradigmas de raciocínio, enquanto suas especializações com tipos específicos de objetos domínios, atributos, módulos conhecimento e critério de resolução de conflito fazem parte da estrutura do SBC.

- conteúdo, demonstra o conhecimento atual representado, sendo armazenado numa base de conhecimento do sistema. Observa-se que a base de conhecimento em termos gerais, inclui qualquer tipo de declarativo ou conhecimento procedural armazenado, ambos num nível de domínio e nível de controle que um SBC possa conter, de acordo com sua estrutura específica.

Por exemplo, a adoção de regras de produção de um dado tipo de representação de conhecimento, a identificação de objetos específicos, atributos e valores e a escolha de uma mecanismo de inferência são partes da estrutura, enquanto que a base de regra contém certas regras faz parte do conteúdo.

- componente “software” do SBC são os programas que implementam sua estrutura, incluindo arquitetura, estrutura de representação do conhecimento, raciocínio algorítmico, interface externa e suporte de sistema.

- sistema de “software” na qual o SBC está implementado.

- sistema de hardware na qual o sistema baseado no conhecimento está instalado.

Estrutura engloba três componentes, que são definidos a seguir:

- adaptabilidade, representa como a estrutura do SBC casa as características conceituais do domínio de aplicação considerado, sendo definida através de dois conceitos, denominados de modelo conceitual e modelo lógico. O modelo conceitual é uma representação formal das características estáticas e dinâmicas de um domínio de aplicação, relevante para uma classe de problemas resolvendo tarefas. Um modelo conceitual descreve ambos os tipos e organização do conhecimento num domínio (conceitos, atributos, relações, restrições, operações, eventos, etc.) e o caminho usado para resolver problemas (tipo de problemas considerados, raciocínios de paradigmas usados, etc.). Certamente, a construção do modelo conceitual de um domínio de aplicação requer disponibilidade de um estudo formal e uma linguagem para ser usada na identificação dos elementos que constituem o modelo e na sua expressão linguística. O modelo lógico é a representação da arquitetura, representação da estrutura do conhecimento, raciocínio algorítmico, interfaces e suporte do sistema adotado para implementar o modelo conceitual de um domínio de aplicação num SBC real. O termo modelo lógico é um sinônimo de estrutura. O modelo conceitual, dito como nível de conhecimento, serve como a especificação para que o modelo lógico, dito como nível símbolo possa realizar.

Adaptabilidade é forte para reconhecer que ele primariamente envolve a relação entre o modelo conceitual do domínio e o modelo lógico do SBC: alta adaptabilidade significa um natural e eficiente casamento entre esses dois modelos.

Se o modelo conceitual pode refletir o modelo da mente humana e processos, adaptabilidade não necessariamente compreende a solução de uma simulação conhecida. O estudo de organização de memória e processamento de informação em humanos pode ajudar no projeto de máquinas artificiais que pode fornecer uma performance similar num domínio limitado, mas a arquitetura interna e o modo de operação de tais máquinas não são obrigadas a reproduzir fielmente a organização interna e operação da mente humana.

- perfeição, representa a propriedade da estrutura do sistema baseado no conhecimento de ser projetado e construído de acordo com o projeto principal do SBC, que pode ser uma herança correta e coerente do ponto de vista técnico e lógico.

- extensibilidade representa a facilidade de modificação dos elementos estruturais do SBC em frente a inadequância que pode surgir durante a fase de desenvolvimento do SBC ou frente a novas necessidades que possam surgir na fase operacional.

Conteúdo envolve cinco seguintes componentes:

- consistência envolve um conjunto geral e formal de atributos da base de conhecimento do sistema, que representa suas propriedades para ser livre de inconsistência lógica. O conceito de consistência depende amplamente do formalismo adotado para representar o conhecimento, consistência pode ser comprometida como:

- conflito de conhecimento, ou seja, duas regras que possuem a mesma, premissa e produzem conclusões contraditórias:

$$r(x) \text{ ----> } s(x)$$

$$r(x) \text{ ----> } \text{não}(s(x))$$

- conhecimento circular, ou seja, um conjunto de regras que pode ser encadeada em forma de círculo:

$$r(x) \text{ ----> } s(x)$$

$$s(x) \text{ ----> } t(x)$$

$$t(x) \text{ ----> } r(x)$$

- conciso, envolve um conjunto geral e formal de atributos da base de conhecimento do sistema, que representa sua propriedade para ser livre de objetos inúteis. A concisão não implica em nenhuma redundância. Sabe-se que um certo grau de redundância na representação de conhecimento pode ser natural e usada. Observamos que nos casos de violação de concisão considerada, notamos realmente conhecimento inútil, ou seja, tipo de redundância que não contribui para a naturalidade ou eficiência. O conceito de conciso depende amplamente do formalismo adotado para representação do conhecimento.

Por exemplo, para o caso de produção de regras:

- redundância de conhecimento, ou seja, duas regras possuem premissas equivalentes (aplicadas para a mesma situação) e produzem a mesma conclusão:

$$r(x) \text{ e } s(x) \text{ ----> } t(x)$$

$$s(x) \text{ e } r(x) \text{ ----> } t(x)$$

- conhecimento subordinado, ou seja, duas regras que produzem a mesma conclusão e tem uma premissa que implica uma da outra:

$$r(x) \text{ e } t(x) \text{ ---> } s(x)$$

$$r(x) \text{ ----> } s(x)$$

- conhecimento desnecessário, ou seja, duas regras que produzem a mesma conclusão e tem premissas que podem ser compostas em uma única forma:

$$r(x) \text{ e } t(x) \text{ ----> } s(x)$$

$$r(x) \text{ e } \text{não}(t(x)) \text{ ---> } s(x)$$

$$\text{tal regra poderia ser substituída por: } r(x) \text{ ---> } s(x)$$

- conhecimento fechado, ou seja, uma regra que nunca pode ser usada desde que sua premissa nunca possa ser satisfeita:

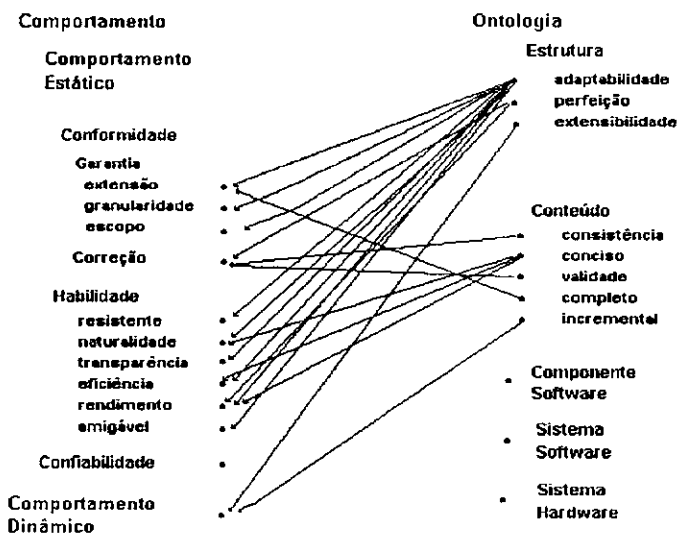
$r(x)$ e $t(x) \dashrightarrow s(x)$ e $t(x)$, tal regra pode ser desconsiderada

- validade, envolve a exatidão do conhecimento armazenado na base de conhecimento do sistema que diz respeito ao conhecimento real especificado no domínio da aplicação considerada.

- completo, representa a propriedade da base de conhecimento do sistema contendo conhecimento suficiente para produzir o comportamento definido na especificação do SBC.

- incremental, representa a facilidade de desenvolvimento da base de conhecimento, debugging, teste e refinamento durante a fase de desenvolvimento do sistema ou na necessidade de novos requerimentos na fase operacional do sistema.

A relação entre componentes ontológicos e comportamentais pode ser de muitos para muitos, como mostra a figura III.5 [referência 11]. Ontologia é de certo modo uma condição necessária, ou seja, é a eficiente causa do comportamento. Ela é propositalmente projetada para que possa produzir o comportamento desejado e conseqüentemente vem de acordo com as especificações. Os componentes elementares da performance e qualidade podem ser particionados em duas classes, denominadas de componentes comportamentais e componentes ontológicos. Pode-se dizer que os componentes das duas classes são praticamente independentes, sendo que existe uma dependência causal entre os componentes comportamentais e ontológicos. Cada componente ontológico pode ser causa de um ou mais componentes comportamentais.



(figura III.5)

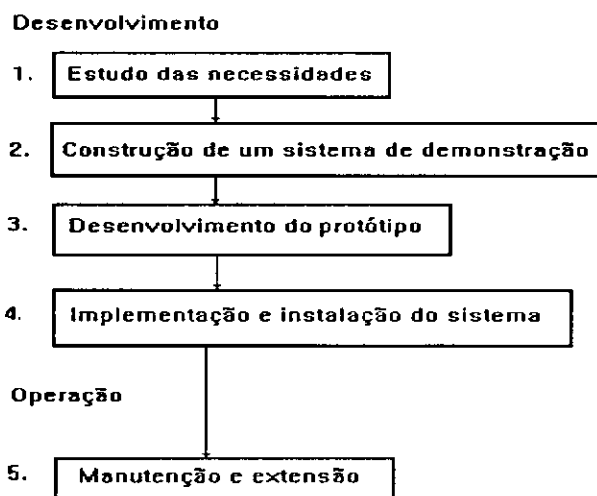
Como já foi mencionado anteriormente, a avaliação de um sistema baseado no conhecimento possui três principais objetivos:

1) manter o projeto do sistema e o processo de construção, isso requer uma avaliação individual dos componentes para serem integrados dentro do ciclo de vida do SBC. Sabemos que a complexidade e o custo para detectar possíveis erros num projeto e construção de um SBC pode ser amplamente reduzida se a falha for descoberta o mais cedo possível. Erros ocorrem no estágio de desenvolvimento e se permanecerem podem afetar a performance do sistema.

2) certificar a performance e qualidade do sistema final, isso sugere que uma avaliação seja feita após o término do desenvolvimento e uso do sistema, sendo que essa avaliação não se concentre somente no final do desenvolvimento do SBC, mas também se concentre em todo o seu ciclo de vida.

3) comparação de diferentes implementações do sistema para um mesmo problema.

Pode-se dizer que o ciclo de vida compreende cinco fases, como mostra a figura III.6 [referência 11] a seguir.



(figura III.6)

Fase 1, *estudo das necessidades*, são feitas as investigações necessárias para a aplicação do sistema baseado em conhecimento em relação a técnica, impacto organizacional, implementação prática, aspectos econômicos (análise custo-benefício) e ambiente. Resume-se em três objetivos principais:

- 1) a análise de um dada área de aplicação, identificando um domínio promissor e selecionando um problema específico;
- 2) a identificação da principal especificação técnica e funcional do sistema baseado em conhecimento e o valor da necessidade da aplicação;
- 3) executando as primeiras decisões técnicas, desenvolvendo um projeto de sistema “draft” e preparando um plano de projeto.
- 4) estudo da viabilidade técnica-econômica

Fase 2, *construção de um sistema de demonstração*, é desenvolvido um sistema de demonstração com a finalidade de antecipar o tipo de performance que o sistema será capaz de fornecer, sendo que a sua construção não é um passo necessário para o ciclo de vida do sistema baseado em conhecimento. Ele apenas é conveniente para a gerência,

para atingir seus compromissos, para envolver os especialistas e usuários no projeto e para coletar suas primeiras sugestões.

Fase 3, *desenvolvimento do protótipo*, aqui se encontra toda a especificação funcional que consta no estudo das necessidades, mas ainda não está instalado o ambiente operacional real, ou seja, ainda não está testado com os exemplos reais, não se encontra otimizado. Esta fase faz parte do corpo do ciclo de vida do sistema baseado em conhecimento.

Fase 4, *implementação e instalação do sistema*, é o desenvolvimento completo do sistema baseado em conhecimento, com a mesma performance funcional do protótipo, mas instalado num ambiente operacional real, testado com dados reais e otimizado.

Fase 5, *manutenção e extensão*, começa com o sistema em uso operacional e compreende a correção de possíveis divergências no comportamento do sistema baseado em conhecimento com relação a especificação funcional que são detectados durante a sua utilização, observando constantemente a performance e qualidade do sistema baseado em conhecimento com as alterações necessárias e pedidas pelo usuário.

Esta fase compreende duas atividades principais:

- 1) manutenção, que envolve modificações no conteúdo da base de conhecimento, deixando a estrutura inicial inalterada;
- 2) extensão, que inclui modificações na estrutura do sistema baseado em conhecimento.

De forma geral, a avaliação (verificação) se concentra nas fases 3,4 e 5, sendo que as fase 1 e 2 são fases preliminares do ciclo de vida do sistema baseado em conhecimento e não são envolvidas diretamente com o projeto e construção. Pode-se observar a seguir as principais tarefas internas das fases 3, 4 e 5 proposta segundo Giovanni Guida e Giancarlo Mauri [referência 11]:

Fase 3 - Desenvolvimento do Protótipo

Análise e projeto conceitual

- 1 - análise e modelagem do conhecimento
- 2 - desenvolvimento do modelo conceitual

Projeto técnico

- 3 - escolha da técnica
- 4 - desenvolvimento do modelo lógico do protótipo
- 5 - definição da especificação técnica do sistema
- 6 - escolha do ambiente de desenvolvimento do protótipo

Desenvolvimento do sistema específico

- 7 - projeto detalhado
- 8 - implementação da estrutura de representação do conhecimento
- 9 - implementação do raciocínio algorítmico
- 10 - implementação das interfaces
- 11 - teste do sistema completo

Desenvolvimento da base de conhecimento

faça

- 12 - aquisição de conhecimento
- 13 - codificar o conhecimento
- 14 - testar e refinar o conhecimento

se o sistema se encontra inadequado

então retornar para o passo 7 ou 5 ou 3 e atualizar o projeto e a implementação

até que

a extensão e qualidade da base de conhecimento for satisfatória e a especificação do sistema se encontra completa

Teste e avaliação

15 - teste do protótipo

16 - avaliação do protótipo

Fase 4 - Implementação e Instalação do Sistema Final

Especificação e planejamento

1 - definição do ambiente final

2 - definição da especificação operacional

3 - definição da especificação técnica do sistema final

4 - escolha da estratégia de produção do sistema final

5 - definição do plano de projeto para o sistema final

Projeto e construção

6 - projeto e construção do sistema final

Instalação, teste e validação

7 - instalação do sistema final

8- teste

9 - certificação

Documentação

10 - documentação (manual do usuário, de referência e de manutenção)

Treinamento

11 - treinamento do usuário e da equipe de manutenção

Fase 5 - Manutenção e Extensão

faça

Monitorando

faça

1 - coletar informação do usuário até um número significativo de divergência da especificação inicial ou de novas necessidades que tenham sido coletadas

Análise e intervenção da definição

2 - análise da informação coletada

3 - análise da divergência da especificação inicial

4 - análise de novas necessidades

5 - definição de nova especificação

6 - identificação do tipo apropriado de intervenção

se manutenção então ir para o passo 7

se extensão então ir para o passo 12

Execução da manutenção

7 - planejamento da atividade de manutenção

faça

8 - aquisição de conhecimento

9 - codificar o conhecimento

10 - testar e refinar o conhecimento

se o sistema se encontra inadequado

então ir para o passo 12

11 - teste final

Execução da extensão

12 - planejamento

13 - revisão da escolha da técnica

14 - revisão do modelo lógico

15 - detalhes da modificação de projeto

16 - implementação da modificação

17 - teste do sistema revisado

Revisão e manutenção da base de conhecimento

faça

18 - aquisição de conhecimento

19 - codificação do conhecimento

20 - teste e refinamento

se o sistema se encontra inadequado

então retornar para o passo 13

até que a extensão e qualidade da base de conhecimento for satisfatória e a especificação se encontrar completa

21 - instalação de novo “release” do sistema

22 - teste do novo “release” do sistema

23 - Certificação do novo “release” do sistema

Documentação

24 - documentação do sistema final (manual do usuário, de referência e de manutenção)

Treinamento

25 - treinamento do usuário e da equipe de manutenção até que o sistema baseado em conhecimento se encontre operacional

III.6 - Comentários

Com o intuito de garantir uma abordagem transparente da área de qualidade de um sistema especialista, procurou-se neste capítulo apresentar as seguintes informações:

- segurança de qualidade, visando garantir a alta performance dos sistemas baseados no conhecimento;

- avaliação de um sistema especialista, baseado na identificação de possíveis problemas que possam ocorrer na base de conhecimento. Os benefícios da avaliação de um sistema baseado em conhecimento durante o desenvolvimento do sistema são extremamente importantes:

- 1 - o desenvolvimento do sistema, que é tipicamente interativo e incremental, é feito de forma mais estruturada

- 2 - decisões desconhecidas de projeto e construção mal elaborada são identificadas mais cedo e a sua correção se torna mais fácil, propiciando um custo baixo

- 3 - a performance profissional do projetista, da equipe de desenvolvimento e do engenheiro de conhecimento é portanto excelente

- 4 - a qualidade do produto final obtém um ótimo resultado

- validação de um sistema especialista, que tem como principal objetivo investigar a segurança, perfeição e consistência de resposta recomendada pelo sistema. Na validação é feito teste para verificar se a tarefa foi executada corretamente;

- verificação de um sistema especialista, onde testes são realizados para observar se a tarefa acompanhada pelo “software” foi executada corretamente.

CAPÍTULO IV

ESTUDO DE CASO

IV.1 - Introdução

Este capítulo apresenta uma análise da qualidade do desenvolvimento de um projeto de sistema baseado em conhecimento - Sistema Programador - como um estudo de caso, utilizando métodos e ferramentas anteriormente apresentadas, possibilitando melhorar o comportamento do sistema em termos dos critérios de qualidade e performance usados. Este caso possibilita a visualização da solução de produção de um sistema com as características necessárias de qualidade.

Atualmente o sistema se encontra em fase final - término da fase de extensão. O Sistema Programador está sendo desenvolvido para a Petrobrás, projeto Coppetec ET-150585 e tem como principal finalidade dar apoio a decisão de programação de barcos especiais para manutenção de plataformas.

Avaliou-se a utilização de um SBC para a solução do problema através dos seguintes pontos:

- a decisão da programação deve ser tomada rapidamente, pois normalmente envolve uma soma de recursos muito elevada. Manutenção e investimento de plataformas "offshore" que demandam investimentos elevados e o risco de danos é muito alto.

- existem poucos especialistas na área, na Empresa propriamente, mas contam com uma equipe com condições de tomar as decisões, se auxiliados por um SBC.

- a tomada de decisão envolve muitas variáveis com um alto grau de incerteza e uma longa experiência na área. São vários os fatores que influenciam na decisão, o que torna muito complexa este tipo de solução.

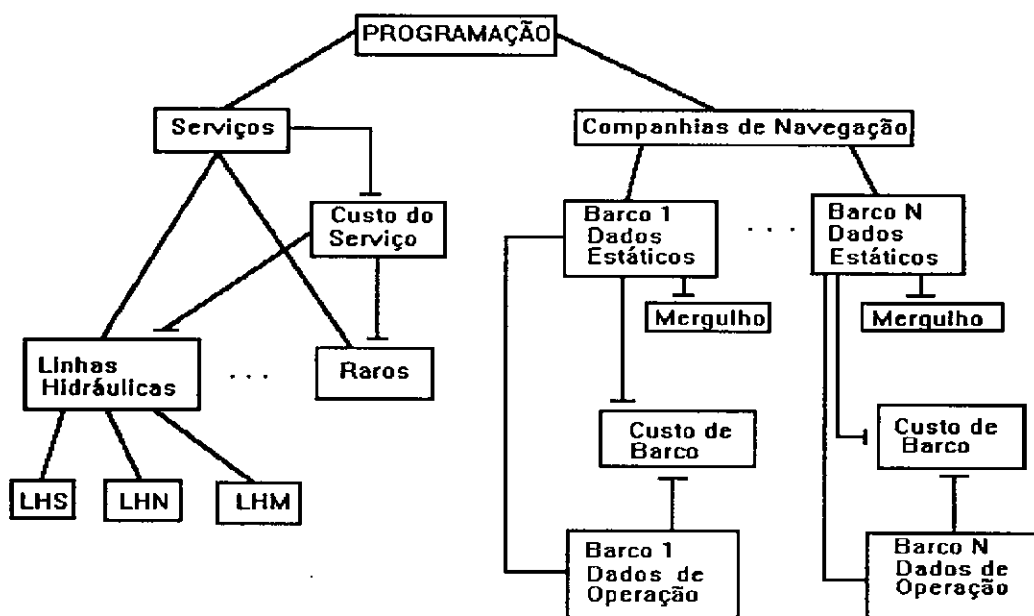
Dessa forma, foi estabelecida uma expectativa relativa ao SBC: que ele seja uma ferramenta efetiva para auxiliar os programadores da Empresa, não só na tomada de decisão, como também na formalização do domínio do conhecimento. O sistema também deverá ser integrado com outras ferramentas computacionais já existentes e também ainda em desenvolvimento. Assim, o SBC deve interagir e gerar informações para de forma indireta permitir atualizar os dados relativos às solicitações de serviços em questão no banco de dados da Petrobrás.

IV.2 - Descrição do Sistema

O sistema programador tem como principal objetivo fornecer subsídios para o auxílio à programação de barcos especiais para manutenção de plataformas, classificando e priorizando os serviços a serem executados de acordo com a estratégia utilizada pelos especialistas da área. Todo o serviço de programação é feito em função de priorização.

O sistema abrange dois aspectos fortemente interrelacionados: *disponibilidade e custo*. Essas relações envolvem barco, mergulho, serviço e tempo. Na verdade todas as “entidades fortes” envolvidas nesse processo (exemplo: disponibilidade, custo, tempo, barco, mergulho, serviço, etc.) estão muito interrelacionadas, havendo fortes relações de dependência uma das outras e em alguns pontos até mesmo similaridade.

Vindo da aquisição do conhecimento feita, mas visando possibilitar a implementação, elas estão de certa forma “discretizadas” como descrito na estruturação (figura IV.1), sem no entanto perder o seu papel na dinâmica desse complexo serviço.



(figura IV.1)

Pode-se definir a programação da seguinte forma:

- solicitações de serviços;
- serviços da programação anterior que foram executadas por cada barco;
- situação de cada barco:
 - localização dos barcos;
 - situação de mergulho;
 - situação de equipamentos
- lista de inspeções programadas pendentes

Em função dos dados mencionados, foi gerado uma lista de serviços (pendentes) já solicitados, solicitações com serviços ainda com um tempo de indisponibilidade para sua execução e serviços de inspeções programadas que normalmente possui uma baixa prioridade. O barco só faz uma inspeção programada quando não possui nenhuma intervenção para ser feita.

A priorização do sistema é basicamente definida em quatro níveis:

- a) pelo tipo de serviço, ou seja, o serviço pode ser classificado como estratégico, neste caso ele deve ter um atendimento imediato, mesmo sacrificando a execução de outros serviços. O serviço pode ser classificado como de manutenção e de investimento. Dentro dessa classificação, os serviços podem mudar de categoria em função dos parâmetros, tempo ou algum estado que leve a alterar a decisão de ser estratégico ou não, por exemplo, vazamento de óleo muito intenso;
- b) serviços propriamente ditos, ou seja, cada serviço possui uma prioridade. Por exemplo, serviço de manutenção do tipo “vazamento de óleo e gás” tem uma prioridade e um tempo estimado de execução do serviço, ocorrendo uma comparação com os demais serviços da lista gerada para a programação. A cada nova programação, esse valor deve ser reavaliado pelo programador. Pode ser alterado diariamente;
- c) disponibilidade é uma variável, ou seja, é instanciada em função do estado corrente do sistema. Cada consulta fornece dados de entrada para o sistema e pré-requisitos para cada serviço, estas informações dependem dos serviços que estão sendo executado. Por exemplo, verificar como pré-requisito se o serviço pode ser executado somente durante o dia ou durante as 24 horas, verificar se precisa de material e se tem material disponível, verificar se o barco está docado ou se possui mergulho e equipamento, etc.
- d) custo do barco , ou seja, consumo de combustível, diária do barco e custo de mergulho.

A programação dos serviços para cada um dos barcos, isto é, a lista inicial de serviços é distribuída pelos barcos em função da priorização feita.

A cada alteração feita, ou seja, a cada comunicação de serviço realizado ou interrompido ou cancelado ou de novas solicitações, a programação é refeita.

A estratégia básica para uma boa programação é tentar fazer que as estimativas feitas para a programação, que podem ser encapsuladas em estimativas de tempo e custo sejam o mais próximo possível do encapsulamento de tempo e custo reais que foram resultados da execução dos serviços, sempre buscando otimizar. Procurar manter sempre um barco de mergulho em cada área, para chegar o mais rápido possível ao serviço (em função do nível de vida e disponibilidade dos mergulhadores e características dos barcos).

Uma marcante característica do domínio do sistema envolvido foi a identificação de um conhecimento com entidades fortes e complexas com um caráter impreciso. A técnica de implementação “modelagem orientada a objeto” utilizada possibilita o encapsulamento das características das entidades viabilizando a hierarquia de regras e métodos, tornando-as identificáveis e completas.

No que se refere à implementação computacional do sistema, pode-se dizer que o conhecimento conta com um número considerável de aproximadamente de 270 regras, 390 métodos, 90 classes, 140 objetos (sistema possui geração dinâmica de objetos), 400 propriedades, e 77 “meta-slots” para atingir o objetivo proposto. E em relação ao tempo de processamento do sistema até alcançar o resultado final, foi observado que o mesmo não ultrapassa dez minutos para realizar uma programação com aproximadamente vinte solicitações de serviços; permitindo também que o usuário participe mais ativamente na elaboração e análise da programação de embarcação.

A maioria das variáveis que entram na solução do problema possuem um componente bastante alto de incerteza, como a performance de cada barco ou de cada

equipe de um barco e as condições ambientais (mar, vento, etc.), e o tratamento para a incerteza foi obtido através da modelagem “fuzzy sets”.

O sistema requer a manipulação de uma considerável gama de dados, o que tornou necessário interagir com um sistema de banco de dados, dispondo de todas as facilidades operacionais, de recuperação, de armazenamento e de fácil crescimento, mantendo uma integração com o conhecimento eliciado.

IV.3 - Ciclo de Vida do Sistema Programador

O sistema foi desenvolvido de modo a atender às expectativas do usuário, minimizando a necessidade de manutenção futura. O desenvolvimento do sistema buscou atender as expectativas dos usuários em duas etapas, correspondentes à implementação do protótipo e ao seu refinamento (implementação final).

Com a prototipagem, os usuários e o próprio especialista possuem condições de entrar em contato com a nova ferramenta, com os benefícios que poderão dispor, validando inclusive todo o processo e o SBC em si - apontando direções para sua apropriação ao contexto - como ferramenta para a solução de seus problemas.

Também para a gerência, um protótipo é de grande importância, pois assim ocorre um retorno das expectativas relativas ao sistema.

IV.3.1 - Fase I - Estudo das Necessidades

A idéia inicial era que mais de um especialista da Empresa trabalhasse na aquisição de conhecimento junto com o engenheiro de conhecimento (EC), para ajudar na aquisição do domínio específico do problema. Assim, foram selecionados possíveis especialistas e escolhido um, levando-se em consideração a maior experiência na área. Os demais especialistas contribuíram em alguns pontos da aquisição de conhecimento.

A definição dos usuários, assim como os possíveis especialistas e as expectativas que envolviam o SBC foi feita através de conversas mais ou menos formais - na verdade entrevistas não estruturadas - com a gerência que contactou para desenvolver o SBC.

A familiarização da gerência e do especialista com o processo de desenvolvimento do SBC apresentou alguns problemas. Em primeiro lugar, apesar de interessados, a gerência e o especialista não tinham conhecimento sobre SBC e então demonstraram um certo receio com relação a resultados efetivos em comparação com os sistemas convencionais, principalmente e especificamente para a área de engenharia. Foi feito um trabalho de explanação sobre SBC, com exemplificação na área de engenharia. Houve ainda uma reunião com a gerência para combinar como seria feito o acompanhamento dos trabalhos.

Inicialmente o especialista apresentou dificuldades de se expressar verbalmente e um certo receio com o uso do gravador - todas as sessões foram gravadas. Com o transcorrer do tempo esse problema foi amenizado. Certas vezes o gravador foi desligado para avaliar o comportamento do especialista; percebeu-se nitidamente uma preocupação em formalizar o que dizia, quando o gravador estava ligado.

As duas primeiras entrevistas com o especialista foram entrevistas não estruturadas, visando complementar a primeira fase do processo da aquisição do conhecimento - identificação do problema.

A familiarização do EC com o domínio do problema transcorreu sem maiores problemas, principalmente porque este já possuía trabalho na área de plataformas “offshore”. Houve apenas a introdução de termos e conceitos específicos da área de embarcação.

De um modo geral, todo sistema possui seus objetivos que, para serem alcançados, necessitam ser levantados claramente. Uma afirmação de objetivos é normalmente um dos principais requisitos para se cumprir as especificações dos usuários. Ela inclui desejos, necessidades, pretensões, tempo e restrições, escolha de “software” e “hardware” e necessidades técnicas adicionais.

Dessa forma, foram a princípio feitas todas as investigações necessárias em relação a técnica, impacto organizacional, implementação prática, aspectos econômicos e ambiente. O domínio do problema específico foi analisado de acordo com as expectativas e necessidades do usuário, que incluíam basicamente:

- Expectativas relativas ao mergulho:

- profundidade

- custo de mergulho de cada barco

- mergulhador extra

- Possibilidade de atualização do sistema, incluindo mudanças na frota e na carteira de trabalho.

- O sistema deve permitir a utilização pelos fiscais de barcos também.

- Deve ser possível o sistema apontar qual, dentre dois serviços, é mais barato para um dado barco.

- Se o sistema conseguir chegar a apontar que, porque vai ser atendida a programação estratégica, o prejuízo será tal, será excepcional.

- Caso dados incorretos fornecidos pelo usuário tenham resultado certa programação, os dados corretos, e apenas estes, ao serem entrados, irão gerar a nova programação.

- O sistema deverá dar condição de tirar justificativa da programação feita, para poder ser apresentada aos clientes.

- O sistema deverá apresentar passo a passo a programação que está montando, dando ao usuário a chance de mudar aquilo com que não concordar.
- Detecção e informação sobre pedidos de serviços similares.

IV.3.2 - Fase 2 - Construção de um Sistema de Demonstração

Foi desenvolvido um sistema de demonstração com o objetivo básico de antecipar o tipo de performance que o sistema é capaz de fornecer e também com a finalidade de envolver o especialista e demais usuários no projeto tornando-os mais familiarizados com o sistema e ainda para coletar as suas primeiras sugestões.

IV.3.3 - Fase 3 - Desenvolvimento do Protótipo

Nesta fase foi desenvolvido um protótipo com a intenção de absorver toda a especificação que consta no estudo das necessidades.

Dada a complexidade do problema , as *necessidades e expectativas* do usuário foram analisadas e organizadas dentro do seguinte escopo:

- Inicialmente o sistema apresenta para o usuário a lista de serviços com os status obtidas de um banco de dados existente. A partir desse ponto o usuário deve definir para o sistema a lista de serviços que farão parte da programação a ser efetuada.
- O sistema avalia a situação atual de cada serviço em função das informações obtidas dos bancos de dados disponíveis e das informações fornecidas pelo usuário, que uma vez confirmadas pelo usuário, serão processadas, fornecendo a lista priorizada de serviços por embarcação
- O sistema abrange dois aspectos fortemente interrelacionados: Disponibilidade e Custo. Essas relações envolvem barco, mergulho, serviço e tempo.
- O sistema fornece os dados de entrada necessários a análise de tempo como também os custos estimados com a programação, possibilitando futuros trabalhos de estatística e/ou simulação, incluindo a programação de serviços estratégicos.

- O sistema permite que o usuário faça alterações após a sugestão de programação apresentada, no que se refere à variação de parâmetros envolvidos, inclusão e exclusão de serviços. Serão considerados para efeito de programação todos os serviços pendente e indisponível. O sistema apresenta as seguintes facilidades para alteração de uma programação efetuada:

- Correção de parâmetros de interesse do usuário sem alterar os demais;
- Reprodução de trechos da programação passo a passo, permitindo que o usuário modifique as decisões associadas com o conhecimento adquirido.

- A cada ponto de decisão ao longo da programação será apresentado ao usuário a sugestão do sistema acompanhado da justificativa correspondente, possibilitando que o usuário aceite ou altere a decisão do sistema. Se o usuário optar por alterar a decisão proposta pelo sistema, a modificação, a justificativa correspondente e algum esclarecimento do usuário por ter seguido um outro caminho (texto opcional) serão armazenados antes que a programação prossiga.

- O sistema mapeia a lista de serviços de mesmo tipo para a mesma localidade no início da programação para detecção de possíveis solicitações similares, a serem confirmadas pelo usuário.

Os sistemas de *banco de dados* fornecem facilidades operacionais, de recuperação, de arquivamento e permitem um fácil crescimento. Possuem interfaces compreensivas para permitir aos usuários o compartilhamento das informações de dados e consultas e, como principal característica, aumentam consideravelmente a independência dos dados. Devido a esses aspectos, este sistema identificou a necessidade da utilização da técnica de banco de dados, tendo como principal característica a capacidade de armazenar e recuperar os dados, mantendo uma integração com o conhecimento eliciado. Foram a princípio levantados todos os possíveis documentos contratuais, normas (NR-15), tabelas de mergulho, enfim, todos os documentos utilizados na metodologia de trabalho da programação fornecidos pela empresa (Petrobrás). O escopo do banco de dados foi identificado após o levantamento e análise dessas informações, que são:

- dados de mergulho
- dados das coordenadas UTM (localização X e Y - unidades em metro)

- dados dos barcos (contratuais)
- dados de custo
- dados da programação
- dados de serviços

Na *aquisição do conhecimento*, optou-se pela técnica de entrevistas estruturadas com o especialista. Em determinado ponto do processo de aquisição do conhecimento, já havia muito conhecimento eliciado, de vários tipos e sobre as mais distintas fases do processo de solução de problema, fornecendo uma visão geral do mesmo.

Se por um lado ter todo o conhecimento eliciado é bom, por outro, muitas vezes não há como distinguir claramente a importância das informações e como utilizá-las. Para poder cobrir cada ponto citado pelo especialista e fornecer subsídio para o desenvolvimento do projeto e implementação da base de conhecimento, foi gerado um documento, contendo partes das transcrições das entrevistas estruturadas para a implementação, a organização deste documento é descrito a seguir:

I) Sistemas Existentes

1) Simulador

- sistema desenvolvido em FORTRAN - grande porte, tendo como objetivo a previsão de forta.

2) Baretta 2

- sistema desenvolvido em CLIPPER - microcomputador
- consiste em fazer todas as combinações possíveis, distribuindo cinco serviços para cada três barcos
- para cada configuração válida, é calculado o custo de atendimento dos serviços, levando-se em conta o tempo de término do serviço, tempo de deslocamento do barco entre os serviços, e tempo de disponibilidade dos serviços
- os custos após serem calculados, são então comparados

3) Baretta 3

- desenvolvido em CLIPPER - microcomputador

- nova versão do Baretta 2, contendo acréscimo das informações de profundidade dos serviços e ainda inclusão de duas tabelas de mergulho, com tempo de compressão e descompressão

4) SASBE

- desenvolvido em CLIPPER - microcomputador
- é basicamente um cadastro contendo informações das características dos serviços, das solicitações dos serviços, histórico do “status” de cada serviço, combustível gasto, e dados contratuais dos barcos

5) CRB

- desenvolvido em CLIPPER - microcomputador
- cadastramento de todas as instalações marítimas, com resumo das atividades desenvolvidas nesses equipamentos

II) Sistema Especialista

- 1) Expectativas
- 2) Objetivo
- 3) Escopo
- 4) Interface
- 5) Plataforma Implementacional

III) Banco de Dados

- Dados dos poços
- Dados dos Barcos Especiais
- Dados dos Equipamentos
- Dados Modalidade
- Dados Custo de Mergulho
- Dados Última Programação
- Dados Mergulho

IV) Estruturação do Conhecimento

- Estrutura
- Programação

Priorização

Resultado da Priorização

Estratégia Básica para uma boa Programação

Companhias de Navegação

Barco 1 a n - Dados Estáticos

Barco 1 a n - Dados Operação

Mergulho

Serviços

Para cada Serviço

V) Descrição da Estruturação

Programação

Priorização

Companhias de Navegação

Barco 1 a n - Dados Estáticos

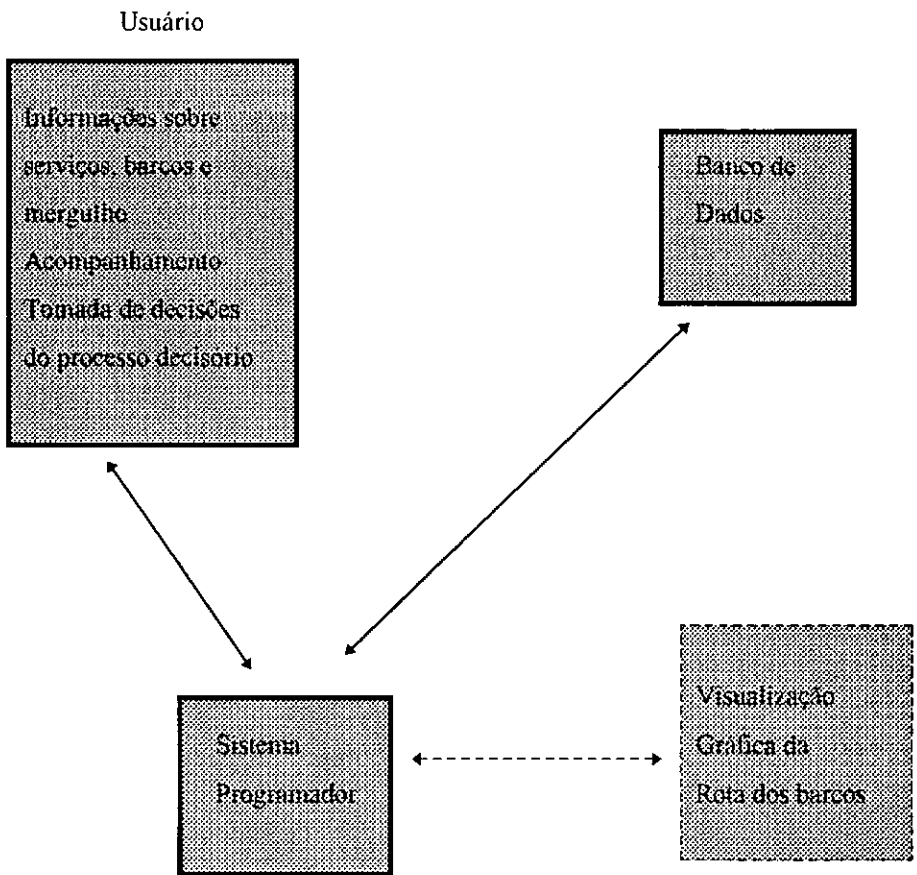
Barco 1 a n - Dados Operação

Mergulho

Serviços

VI) Exemplos

A *interface* que o SBC necessita já se encontra definida. A interface do sistema procurou garantir ao usuário um tempo de resposta rápida, incluindo ainda uma parte gráfica, apresentando uma visão espacial (geográfica) do percurso do navio, ou seja, uma visão global da localização de cada navio. Para que o protótipo cumpra o seu papel, é necessário que a interface já se encontre implementada quando a base de conhecimento estiver completa (figura IV.2), a parte tracejada somente foi implementada na *fase 4*. Era preciso, então, definir o ambiente na qual o protótipo seria implementado para que o desenvolvimento da interface tivesse início.



(figura IV.2)

O *ambiente de desenvolvimento* do protótipo como também do sistema final foi estabelecido com as seguintes características:

- é uma ferramenta híbrida, que suporta a integração do desenvolvimento de orientação-objeto com aplicações de regras
- é um "software" que fornece ferramenta para desenvolvimento de aplicações baseado em conhecimento;
- possui uma interface gráfica amigável para ajudar a criar uma aplicação baseada em conhecimento;
- o domínio é modelado em forma de objetos, classes e propriedades com o seu comportamento;
- apresenta um forte mecanismo de inferência para realizar as tarefas do domínio.

A *plataforma implementacional* foi então cuidadosamente avaliada, chegando-se a seguinte conclusão:

- Ambiente de Desenvolvimento e softwares para interface e rotinas de cálculo: definidos com linguagem Borland Pascal versão 7.0 e SMART ELEMENTS versão.1.0.
- Banco de Dados: formato DBF e NXPDB(do Smart Element's)
- Microcomputador PC 486 DX2-66Mz, 16Mb, VLB, monitor SVGA

Sendo as regras e parâmetros unidades independentes entre si, a sua participação efetiva dentro do domínio pode ser testada independentemente logo após a sua definição, facilitando a correção de erros e indicando novas definições. É importante testar a base desde o início da sua implementação, evitando o acúmulo de erros , cuja detecção pode tornar-se complexa.

Antes de liberar a sua utilização, o produto deve ser testado primeiro pelos especialistas, observando e analisando caso a caso as respostas intermediárias e os resultados finais, e depois também por todos os usuários para garantir que o sistema desperte a confiança nos mesmos. O *teste objetivo na base de conhecimento* foi feito com alguns exemplos reais, pois este já oferecia condição de entrar em operação. Foram escolhidas as necessidades para o teste objetivo; a equipe de projeto determinou o teste da seguinte forma:

- Inicialmente foram executados testes intermediários com a parte estática do sistema, ou seja, realizou-se testes relacionados à priorização dos serviços, classificando -os em função dos parâmetros existentes, como tipo de serviço (estratégico, manutenção ou investimento), tempo gasto para executá-lo, custo do serviço, etc.

- Segundo, foi validado a parte dinâmica do sistema, isto é, verifica-se o estado corrente de cada serviço, se está sendo executado, se ainda será executado, fornecendo assim dados de entrada para o sistema e os pré-requisitos de cada serviço.

- Terceiro, foi avaliado a parte relacionada com as embarcações, isto é, disponibilidade do barco, se ele está em operação, se está em manutenção, se possui equipamento (por exemplo: veículo de observação e/ou intervenção), etc.

- Em seguida foram feitos testes relacionados com mergulho, que envolve profundidade, compressão e descompressão dos mergulhadores, continuidade operacional, excursão, custo do mergulho, etc.

Esta *avaliação* permitiu uma revisão de todo o conhecimento eliciado, de forma que cada etapa foi minuciosamente testada, onde os erros foram corrigidos e as novas indicações foram analisadas e estabelecidas para a continuação do projeto.

IV.3.4 - Fase 4 - Implementação e Instalação do Sistema

Embora esta fase possa incluir novos aspectos ou excluir alguns já levantados, serão aqui listados, numa priorização inicial, resultados das expectativas dos usuários que foram cobertos apenas no sistema final:

- O sistema permite o cadastramento de novos barcos ou exclusão dos existentes pelo usuário.

- O sistema permite ao usuário realizar duas programações (uma em seguida à outra), com tempos de execução dos serviços diferentes, mostrando na tela duas janelas com o resultado dessas programações.

- Após o sistema verificar quais serviços podem ser realizados pelo barco, apresentará ao usuário os custos relativos a cada um desses serviços, para a análise do usuário.

- O sistema permite redefinir uma dada programação para que um serviço estratégico possa ser realizado imediatamente, apresentando os custos decorrentes de sua inserção relativamente à programação original.

A *avaliação* foi executada com um especialista que acompanhou todo o desenvolvimento do projeto e com um outro especialista não envolvido com o projeto e desenvolvimento do sistema. Essa estratégia de validação do comportamento do sistema

foi utilizada porque não existia um padrão de referência do sistema programador, a não ser as necessidades e expectativas dos especialistas. O objetivo maior da proposta de validação foi determinar se o sistema está satisfazendo em performance a tarefa para qual ele foi criado.

A medida da *performance e qualidade* do Sistema Programador foi feita em cima da definição de comportamento e ontologia citadas no capítulo anterior. Desse modo, a análise foi realizada através de alguns componentes elementares da taxonomia já apresentada.

Dentro do comportamento estático, a avaliação foi feita sobre três componentes básicos:

- *conformidade*, devido a metodologia orientada-objeto utilizada, onde a extensão é um componente chave, permitindo generalização, que é simplesmente uma organização de entidades dentro de uma hierarquia, com herança de atributos; classificação, que é a organização de entidades, relacionamentos ou atributos em classe, que enfatizam suas propriedades comuns e revelam as particularidades; e agregação, que é uma organização de entidades, relacionamentos ou atributo como partes de uma nova classe ou agrupamento. Desse modo, é possível observar a facilidade que a metodologia orientada-objeto tem como característica elementar a capacidade de fazer extensão das suas entidades, propriedade e relações, aumentando com isso de maneira rápida e precisa o seu escopo; podendo ainda ter uma ótima representação dos níveis de detalhes das mesmas.

- *habilidade*, o sistema possui uma naturalidade e transparência apresentando justificativas na tela com as informações fundamentais para que o usuário possa analisar cuidadosamente os passos feitos pelo sistema. Dessa forma, o usuário tem justificativas para as informações relativas a serviços, barcos e programação conforme exemplos apresentados a seguir:

- A lista de modificações no sistema programador (figura IV.3) possibilita que o usuário visualize todos os serviços solicitados, contendo todas as informações de entrada atualizadas e os pré-requisitos para a priorização dos serviços para a programação.

- Na configuração atual dos barcos disponíveis para a programação (figura IV.4), o usuário dispõem de todos os dados relativos as embarcações disponíveis ou não para executar a programação.

- A programação dos barcos (figura IV.5) permite que o usuário tenha uma visão gráfica de toda a rota dos barcos para uma dada programação.

- A escolha do barco candidato (figura IV.6) mostra como foi efetuada a escolha do barco para executar um determinado serviço, através dos parâmetros analisados pelo Sistema Programador.

- O sistema é inteiramente amigável para o usuário, pois ele possui telas com explicações ('help') de como se proceder na operacionalidade do sistema (exemplo - figura IV.7).

Quanto ao seu rendimento, é incrivelmente perceptível o tempo rápido de resposta, mesmo contendo uma gama imensa de conhecimento para ser processado. A eficiência foi completamente aprovada pelos especialistas e usuários através de programações reais que ocorrem diariamente na Empresa.

File Edit Search Help

23/02/1995 as 14:26

LISTA DE RS MODIFICADOS NO SISTEMA PROGRAMADOR

Observacoes:|

- Atualizar os dados do SASBE para os servicos listados
- A opcao FILE(Print) do editor imprime o texto
- A opcao FILE(SaveAs) do editor, salva o texto com o nome atribuido pelo usuario.
- Verificar a consistencia de dados no Sasbe antes de efetuar a proxima programacao.

00105/95 : MERGULHO PARA PULL IN DO BUNDLE DO CO-08, NA ANM E NO MSP-CO-01.

- Status atual..... : P (18/02/95)
- Codigo atual..... : MPNNPIMP
- Perda de Producao (m3/dia)..... : 760
- Custo da Sonda + Rebocador (\$/dia): 0
- Coordenadas UTM (N/E)..... : 7504300 / 343000
- Tempo estimado de execucao (hs).... : 24
- Classificacao do servico..... : Manutencao
- Servico diurno : N
- Ranking de priorizacao..... : 1
- Nivel de trabalho(m): 200

(figura IV.3)

File Edit Search Help

23/02/1995 as 14:26

CONFIGURACAO ATUAL DOS BARCOS DISPONIVEIS PARA PROGRAMACAO

Observacoes:

- Atualizar sempre a configuracao dos barcos antes de programar
- A opcao FILE(Print) do editor imprime o texto
- A opcao FILE(SaveAs) do editor, salva o texto com o nome atribuido pelo usuario.
- Verificar a consistencia de dados

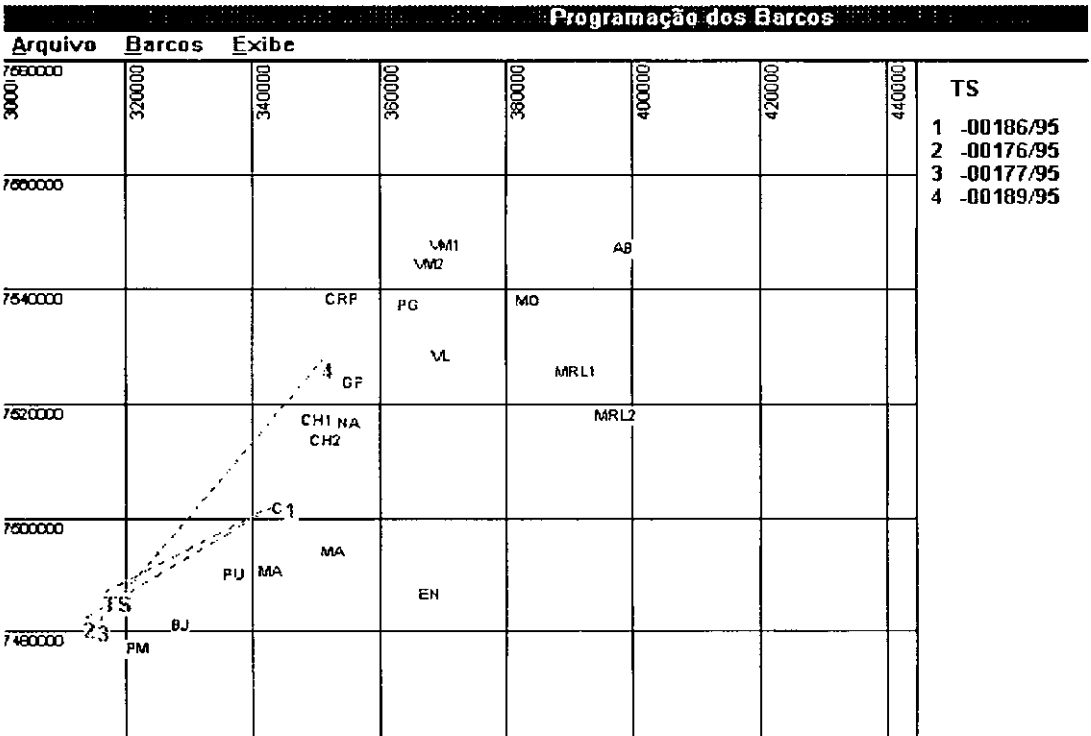
BARCO: TS

```

- Tipo.....: DSU (Marsat)
- Diaria da embarcacao ($).....: 35485.00
- Ultima atualizacao.....: 16/02/95
- Status Atual.....: Operacional
- Tempo de indisponibilidade atual.(hs): 8.00
- UIR (m).....: 1280.00
- UOR (m).....: 600.00
- Profundidade maxima de mergulho (m)...: 300.00
- Posicao atual UTMN/UTME.....: 7486707.00 / 316458.00

```

(figura IV.4)



(figura IV.5)

23/02/1995 as 14:26

ESCOLHA DO BARCO CANDIDATO POR SERVIÇO

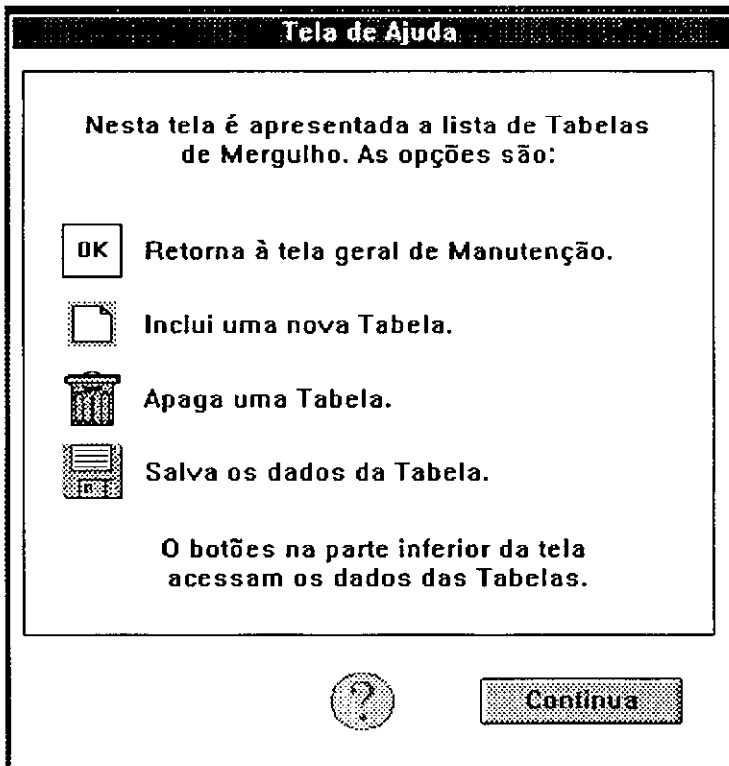
OBSERVAÇÕES:

- Este relatório mostra o mecanismo de inferência utilizado pelo sistema para efetuar os cortes na lista de cada barco.
- A tomada de decisão envolve todos os parâmetros envolvidos: distância, mergulho, custos associados, tempos de indisponibilidade etc.
- O resultado da programação, representado pelos tempos e custos, é parametrizado e analisado com a teoria "fuzzy" (ver manual do sistema).
- Para escolher o barco "ideal" para executar o serviço, o sistema analisa as prioridades associadas relativas aos custos e tempos gastos por cada barco.
- Os fatores de prioridade variam no intervalo [0,1]
- O barco com maior prioridade "associada" será o escolhido pelo sistema.

00185/95 Mergulho a 200.0000 m com o barco SM

- Origem.....: Início (2.2008 Km em 0.0915 hs)
- Chegada ao local.....: 23/02/95 - 22:31 (Qui)
- Tempo Indisponível na chegada.: 0.1667 hs
-: 0.0000 hs

(figura IV.6)



(figura IV.7)

- *confiabilidade*, foram feitos diversos testes para assegurar resultados idênticos, como por exemplo, foram realizados testes com a mesma entrada em períodos diferentes obtendo a mesma solução.

Dentro da ontologia, foram analisados os seguintes itens:

- *estrutura*, o ambiente implementacional usado permite uma perfeita modificação dos elementos estruturais do SBC, pois a sua arquitetura é organizada como uma coleção de objetos independentes, onde cada objeto incorpora a sua estrutura de dados (atributos/propriedades) e o seu comportamento (métodos e regras), possibilitando uma simples manutenção e extensão quando necessário.

- *conteúdo*, a validação sempre envolve numa verificação direta de segurança das regras e métodos armazenados na base de conhecimento, e na verificação do “trace” ou “snapshot” de um particula algoritmo em execução. O “software” de implementação usado possui um modelo interativo gráfico. A força de um modelo visual interativo é permitir que a equipe de implementação trabalhe com o modelo interativamente (tal como alterar um valor de mesma variável na tela durante a sua execução) e examinar o efeito de diferentes decisões em forma de gráfico na tela do computador. Essa característica e a similaridade entre sistema especialista e simulação visual interativa torna a abrangência visual interativa uma ferramenta potencial para ajudar a validar um sistema especialista.

A representação do uso baseado em regras pode ser difícil de compreender, especialmente quando o relacionamento entre as regras não é feito explicitamente. Entretanto, a idéia principal da proposta de abrangência visual interativa para validação de um sistema especialista baseado em regras é para aumentar essa facilidade de explicação. Essencialmente, ele requer a facilidade de explicação para mostrar toda a regra da base de conhecimento em forma gráfica. Em vez de pura leitura do texto dos métodos e regras, a equipe de implementação pode examinar os gráficos (exemplo: figuras IV.8 e IV.9) que mostram não somente as regras, mas também seus inter-relacionamentos e sequências. Podem também interagir com a facilidade, alterar

alguns dados de entrada, e observar como uma resposta é derivada. A representação gráfica permite que a equipe veja os erros muito rapidamente.

Desse modo, permite que a equipe verifique a segurança e a perfeição da base de conhecimento observando o gráfico. A vantagem do seu uso é a apresentação de uma estrutura para essas regras e métodos.

Em validação do algoritmo do mecanismo de inferência pode ser conduzida pela verificação da agenda. Em outras palavras, a equipe pode visualmente examinar o algoritmo pela verificação correta do caminho de decisão.

Uma outra vantagem de mostrar o processo de decisão graficamente é que a equipe pode também ver as regras que não foram usadas. Isso pode algumas vezes ajudá-los a levantar questões de porque algumas regras foram usadas e outras não. Se existe relamente incorreções, a saída seria também alguns erros serem apresentados no algoritmo ou algumas regras faltarem ou erros na base de conhecimento. Portanto, esse tipo de questão também ajuda validar o sistema.

IV.3.5 - Fase 5 - Manutenção e Extensão

O setor onde está sendo desenvolvido o sistema sofreu uma reestruturação, isto é, foi somado com um outro setor, que também realiza programação de embarcação, apenas contém um outro enfoque, ou seja, possui uma forma diferente de trabalhar. Este setor tem um cronograma praticamente estático com uma programação preparada para um período de aproximadamente três anos. Portanto, ocorreu um acréscimo de trabalho com novas informações e maneiras de realizar programações de embarcações. Desse modo, foi necessário re-organizar e re-pensar no sistema como um todo.

Na classe de mergulho ocorreu, independentemente da reestruturação, uma necessidade de manutenção ocasionando uma forte alteração em todo conhecimento relativo a saturação e descanso do mergulhador, surgindo assim uma drástica mudança na programação com relação a esta classe. Atualmente o sistema se encontra na fase de teste (validação), documentação e treinamento.

IV.4 - Comentários

O sistema programador possui muitas variáveis, contendo então um conhecimento complexo e uma série de informações a serem processadas, o que levou a equipe de desenvolvimento a ter uma preocupação elevada com a etapa de implementação e um cuidado particular com a análise do sistema de "hardware" e de "software".

A ausência de um padrão de referência para comparar com o sistema especialista é entretanto uma situação comum. Portanto, a estratégia usada para validar o sistema programador foi comparar o resultado da programação do sistema com o resultado da programação de um especialista envolvido com o sistema e também com um não envolvido com o projeto e desenvolvimento do sistema. A validação do sistema

especialista programador tornou-se fundamental para o término da implementação. A verificação foi feita minuciosamente em cima de alguns componentes, tais como: conformidade, habilidade, confiabilidade, estrutura e conteúdo.

CAPÍTULO V

CONCLUSÕES

Qualidade é um conceito complexo e de múltiplas facetas. O principal enfoque neste trabalho foi dado para qualidade como medida da quantidade de alguns ingredientes ou atributos possuídos pelo produto e também como o reflexo das necessidades e expectativas dos clientes e ainda como a conformidade com as especificações.

No que se refere a análise da qualidade de “software” aqui apresentada, ainda que incompleta, procurou-se ter uma abordagem da área e também demonstrar que a qualidade dá lugar, natural e inevitável a um aumento de produtividade. Qualquer que seja a metodologia adotada, o importante é a sua utilização no desenvolvimento de “software”, de forma que os produtos de “software”, após serem colocados em operação possam ter uma vida útil longa e produtiva.

Existe uma grande variedade de métodos e técnicas da qualidade de “software” citadas na literatura, observou-se também que a sua utilização ainda é feita de maneira experimental, até mesmo devido a busca de uma adequação das técnicas e métodos para se obter uma qualidade de “software”. Esse caráter experimental favorece diversas linhas de análises, variações nos processos e o aparecimento de novas técnicas. Além do mais, outros pontos básicos são definidos e utilizados de maneira distinta e até mesmo, em alguns casos, confusas, como o uso de Gerência de Qualidade Total no desenvolvimento de “software” ou Melhoria da Qualidade “Software”, dificultando o estabelecimento de critérios de utilização dos métodos e técnicas.

Este trabalho preocupou-se principalmente em focalizar o papel da qualidade no processo de construção de um SBC, sua importância e as principais linhas de pesquisa

que se encontram envolvidas na busca da efetivação desse processo, com ênfase na performance e qualidade e na metodologia e técnicas aplicadas nessa área.

O Sistema Programador apresentado como estudo de caso foi abordado e analisado sob duas principais diretrizes: validação e verificação. Onde a validação tem como principal função examinar se o modelo reproduz adequadamente o mundo real, e verificação é o processo para assegurar que o sistema implementado seja uma clara representação do modelo. Portanto, verificação requer pouco conhecimento do domínio do problema. Assim o processo verificação pode ser realizado pelos especialistas de “software”.

Além disso, foi estabelecido uma metodologia adequada para a condução do processo de avaliação, que é considerado indispensável, mas que ainda está em aberto, assim como o estabelecimento de métodos mais potentes, que possam ser usados efetivamente para o processo de avaliação.

Portanto, além de fornecer à equipe de desenvolvimento do SBC condições de escolher ou combinar as diferentes diretrizes, fica reforçada a necessidade do sistema se envolver nas atividades de performance e qualidade, para que na prática possam evidenciar determinantes da satisfação do usuário e ainda obter ótimos resultados de acordo com as dimensões de qualidade.

Talvez a mais forte conclusão que surge deste trabalho é a diferença na proposta de validação, verificação e teste entre um “software” tradicional e um SBC. De fato, isto é uma verdade largamente visível, pois um SBC não é projetado e desenvolvido da mesma forma que um “software” convencional. Ele se baseia em conceitos como de prototipagem, projeto interativo, desenvolvimento evolutivo e programação progressiva. Portanto, a prototipagem de um SBC é aplicado em experimento, refinamento e decisões sobre técnicas de validações e sistemas de projeto e requer análises e validações parciais. Enquanto que a prototipagem de um sistema de “software” convencional pretende validar as necessidades, assegurar qualidade e usabilidade da resolução da aplicação. Dessa forma, um SBC se baseia numa característica de ciclo de vida muito específica e

seu desenvolvimento é determinado em métodos e técnicas também muito específicas, claramente diferente da engenharia de “software” tradicional.

Acredita-se que foi possível deprever sob uma estruturação coerente, apresentando diversos enfoques envolvidos nas principais linhas de pesquisa desta área que é altamente evolutiva, contribuindo para um melhor direcionamento e, tentando mostrar uma visão mais detalhada do desempenho de qualidade e o que ela pode acrescentar no que se refere ao aumento de produtividade.

Acima de tudo, a qualidade permanece sendo uma área fértil para pesquisa. Suas ferramentas técnicas podem ser mais bem desenvolvidas, mas sua teoria e prática estão muito defasadas. O conceito de qualidade é somente vagamente compreendido. Conexões com mercado, custo, e lucratividade ainda não são muito claras. A medição é complexa e irrisória, mas permanece sendo a fonte da maioria das recomendações. Portanto, o desafio diante dos pesquisadores é imenso.

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] Aggarwal, K.K. e Yogesh Singh, (1994), "A Pratical Approach for Software Reliability Apportionment" - Proceeding of the ISSAT International Conference Realibility & Quality in Design.
- [2] Arthur, Lowell Jay, (1993), "Improving Software Quality - An Insider's Guide to TQM", Willey Professional Computing
- [3] Belchior, Arnaldo Dias; Rocha, Ana Regina Cavalcanti; (1992), "Características de Qualidade de Programas, Publicações Técnicas do Programa de Engenharia de Sistemas e Computação" - ES-265/92.
- [4] Brauer, Douglas C., Panduit Corp. e Tinley Park, (1993), "Expert Systems and Strategic Testing" - Proceeding Annual Reliability and Maintainability Symposium.
- [5] Chau, Patrick Y.K., (1994), "Proposing A Visual Interactive Approach for Validation of Rule-based Expert Systems" - The Woprld Congress on Expert Systems - Lisboa.
- [6] Garvin David A., (1992), "Gerenciando a Qualidade" - Qualitymark.
- [7] Geymayr, Jose Antonio Bogarin, (1990), "FRAES - Protótipo de sistema Especialista para Análise do Comportamento Mecânico Local de Risers Flexíveis", Dissertação de Tese, COPPE/UFRJ, Rio de Janeiro .
- [8] Gil, Antonio de Loureiro; (1992), "Qualidade Total em Informática" - Atlas S.A, São Paulo.
- [9] Gilleis, Alan, (1994), "Assuring Quality in Expert Systems" - The Woprld Congress on Expert Systems - Lisboa.

- [10] Gottgroy, Márcia de Paiva Basto, (1990), “O Processo de Aquisição do Conhecimento na Construção de Sistemas Especialistas”, Dissertação de Tese, COPPE/UFRJ, Rio de Janeiro .
- [11] Guida, Giovanni e Giancarlo Mauri, (1992), “Evaluating Performance and Quality of Knowledge-Based Systems: Foundation and Methodology”- IEEE Transaction on Knowledge and Data Engineering, Vol 5.
- [12] Hernandez, Christine, Joan J. Sancho, Miquel A. Belmonte, Alex Patak, Ferran Sanz e Carles Sierra, (1994), “Validation of the medical Expert System RENOIR” - The World Congress on Expert Systems - Lisboa.
- [13] ISO 9000-3: Quality Management and Quality Assurance Standards - Part. 3: Guidelines for the Application of ISO 9001 to the development, supply and maintenance of software.
- [14] Kohd, Takehisa, Takaaki Kojima e Koichi Inoue, (1994), “Reliability Maintenance of Rule-Based Failure Diagnosis Systems” - Proceeding of the ISSAT International Conference Reliability & Quality in Design.
- [15] Lee, Matthew K. O.; (1992) “A Quality Software Process Model Embedding Formal Methods”, Second International Conference on Software Quality.
- [16] Mazzoni, Claudia Junqueira; (1981) “Um Modelo Para Especificação e Controle de Qualidade de Software”, Dissertação de Mestrado - PUC/RJ; Rio de Janeiro.
- [17] Munera, José, (1986), “Software Reliability: A Study Case”, Reliability Engineering, pp 417-445.
- [18] Pall, Gabriel A., “Quality Process Management”

- [19] Powers, Jacklyn; (1993), "TQM in Software Development Organizations", Quality Progress.
- [20] Queiroz, Evódio Kaltenecker Retto, "Análise Crítica do An Insider's Guide to TQM Conceito da Qualidade segundo David Garvin", Dissertação de Tese, COPPE/UFRJ- Rio de Janeiro.
- [21] Rocha, Ana Regina Cavalcanti; (1983) "Um Modelo para Avaliação da Qualidade de Especificações", Tese de Doutorado - PUC/RJ; Rio de Janeiro.
- [22] Rocha, Ana Regina Cavalcanti, (1987) "Análise e Projeto Estruturado de Sistemas, Rio de Janeiro
- [23] Rumbaugh, James, Michael Blaha, William Premerlani, Frederick Eddy, William Lorenzen, (1991), "Object-Oriented Modeling and Design".
- [24] Sanders, Joc e Eugene Curran, (1994), "Software Quality - A Framework for Success in Software Development and Support"
- [25] Sanz, Julio G.; (1992) "Software QA/QM Standards for the 90's", Second International Conference on Software Quality.
- [26] Shiller, Larry; (1992), "Excelência em Software" - Makron Books, São Paulo.
- [27] Troy, Jesse, (1994), "Expert Systems: Validation Techniques and Strategies" - The Woprlnd Congress on Expert Systems - Lisboa.
- [28] Villemeur, Alain, "Software Domain", Reliability, Availability, Maintainability and Safety Assessment - Volume 2, Aassessment, Hardware, Software and Human Factors.

- [29] Zlatareva, Neli P., (1994), "A Quality Assurance Framework for Rule-based Expert Systems" - The World Congress on Expert Systems - Lisboa.