UNIVERSIDADE SALVADOR – UNIFACS
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO
MESTRADO EM SISTEMAS E COMPUTAÇÃO

MARIA DAS GRAÇAS DA SILVA VASCONCELOS

REJUVENESCIMENTO DE SISTEMAS LEGADOS - UMA FERRAMENTA PARA APOIAR A EVOLUÇÃO DO SOFTWARE

Salvador
2011

MARIA DAS GRAÇAS DA SILVA VASCONCELOS

REJUVENESCIMENTO DE SISTEMAS LEGADOS - UMA FERRAMENTA PARA APOIAR A EVOLUÇÃO DO SOFTWARE

Dissertação apresentada ao Mestrado em Sistemas e Computação, Universidade Salvador – UNIFACS, como requisito parcial para obtenção do grau de Mestre.
Orientador: Prof. José Maria Nazar David

Salvador
2011

MARIA DAS GRAÇAS DA SILVA VASCONCELOS

REJUVENESCIMENTO DE SISTEMAS LEGADOS - UMA FERRAMENTA PARA APOIAR A EVOLUÇÃO DO SOFTWARE

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Sistemas e Computação, Universidade Salvador - UNIFACS, pela seguinte banca examinadora:

José Maria Nazar David - Orientador_____
Doutor em Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro - COPPE/UFRJ
Universidade Federal de Juiz de Fora - UFJF

Sidney da Silva Viana – Convidado _____
Doutor em Engenharia de Computação, Universidade de São Paulo - USP
Universidade do Salvador - UNIFACS

Rita Suzana Pitangueira Maciel - Convidada_____
Doutora em Ciências da Computação, Universidade Federal de Pernambuco – UFPE
Universidade Federal da Bahia - UFBA

Salvador, 30 de maio de 2011.

Dedico este trabalho aos maiores mestres da minha vida, minha maior Mestra, minha mãe Ester da Silva Vasconcelos (In memoriam) que me ensinou que o **conhecimento** é o maior e único bem nesta vida e ao meu filho Vitor Silva Vasconcelos Sousa que me ensinou que o **amor** capacita em tudo ou quase tudo.

AGRADECIMENTOS

Agradecer é a etapa mais fácil deste nosso trabalho. Falar com o coração carregado de emoção esquecendo momentaneamente regras e normas que muitas vezes bloqueiam o criar. No momento em que estou chegando ao término deste trabalho, reflito sobre a necessidade constante de sonhar, buscar novos conhecimentos, novos caminhos e para tal precisamos de força e incentivo, por este motivo, quero agradecer a todos que de alguma forma participaram da realização de mais este desafio.

A minha querida colega Professora Diane pela contribuição na produção do Abstract. À Professora Catarina Silveira pela grande ajuda com seus comentários metodológicos.

A todos os colegas que colaboraram fornecendo informações, pacientemente coletando dados no meio das suas tarefas de manutenções nada simples, contribuindo de forma significativa para a realização da pesquisa. Em especial a Rosynara Farias e Cássia Danuza pela disponibilidade em prestar informações com riqueza de detalhes apesar da distancia física.

A todos os amigos e familiares que pacientemente e ansiosos me cobravam pela conclusão do trabalho, para puder desfrutar das suas companhias.

A todos os colegas e professores do mestrado pelas suas sugestões, orientações e incentivos.

E finalmente ao meu Orientador Professor José Maria Nazar David, que com sua habitual atenção, me conduziu ao conhecimento e a busca do aperfeiçoamento, contribuindo de forma significativa e decisiva, no desenvolvimento e conclusão do tema.

Segundo o cristianismo, Deus criou o mundo em seis dias, descansou no sétimo. Tudo mais ocorreu durante **a evolução** (GENESE, 1° livro da Bíblia).

RESUMO

A engenharia de software como geradora de solução dedica pouco dos seus recursos para criação de ferramentas, que possam auxiliar no aumento da produtividade dos projetos de Tecnologia de Informação, voltadas para a área de manutenção evolutiva de software. Os projetos de evolução do software necessários ao rejuvenescimento das aplicações apresentam características de grandes projetos, com nível crescente de complexidade pela idade das aplicações e pelo número de usuários e clientes envolvidos. Estes projetos costumam ser adiados para posterior planejamento, pela ausência de informações necessárias ao planejamento. Essas informações se relacionam, sobretudo, ao contexto no qual as atividades de manutenção ocorrem. Esta dissertação apresenta uma ferramenta para apoiar as atividades relacionadas à evolução de sistemas legados, considerando os elementos de contexto dessas atividades. A pesquisa de campo realizada possibilitou a definição dos requisitos funcionais da ferramenta cujo foco é a sua utilização durante a fase de elicitação de requisitos. Estudos de caso foram conduzidos visando à avaliação da solução.

Palavras-chave: Evolução de sistemas legado. Rejuvenescimento de software.

Elementos de Contexto.

ABSTRACT

Software engineering as a generating solution devotes little of its resources to create tools that can help to increase productivity in software project management, mainly concerning in software evolution. The evolution of software projects aiming application rejuvenation has characteristics of large projects, with an increasing degree of complexity of applications considering the age and the number of users and involved customers. These projects are often postponed because the information absence necessary for planning. This information relates primarily to the context in which maintenance activities occur. This dissertation presents a tool to support activities related to the evolution of legacy systems, considering the contextual elements of these activities. The field study allowed the functional requirements elicitation of the tool which is focused on requirements elicitation phase support. Case studies were also carried out to evaluate the solution.

Keywords: Legacy systems evolution. Software rejuvenation. Context.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

BOKS Biblioteca de códigos fontes

CASE *Computer Aided Software Engineering*

COBOL *COmmon Business Oriented Language*

GRAILS Um *framework* open-source, originario da linguagem Ruby on Rails.

O&M Organização e métodos

PMBOK *Project Management Book of Knowledge* , guia para as melhores práticas de gerenciamento de projeto,

PPF´S Pontos por função

SUMÁRIO

1 INTRODUÇÃO

Rejuvenescer, segundo Ferreira (2009) no dicionário da língua portuguesa, significa fazer parecer mais jovem. Os produtos, serviços, pessoas todos precisam rejuvenescer inclusive os softwares. A engenharia de software, ciência nova em relação a tantas outras, apesar da pouca idade, a qual sempre investiu no aprimoramento, em produtividade, precisão, qualidade de processos e de produtos, precisa preocupar-se com técnicas e ferramentas de renovação dos seus produtos.

Os sistemas legados respondem pela maior parte do processamento de dados mundial, ou, dependendo de entendimentos, pelo todo parque instalado de processamento. Isto porque, qualquer sistema em produção (60% a 70% foram desenvolvidos em linguagem Cobol) pode ser considerado como sistema legado (SEACORD et al, 2003; ULRICH, 2002).

As dificuldades do processo da manutenção evolutiva acontecem principalmente em virtude da indisponibilidade da parada do software. As aplicações têm seu uso contínuo e ininterrupto. No entanto, necessitam da renovação tanto para a equipe de desenvolvimento e manutenção, quanto para os usuários. É praticamente impossível interromper o andamento dos trabalhos; seja o de manutenção corretiva, bem como o de manutenções adaptativas, as quais possuem muitas vezes a natureza obrigatória de caráter legal que por sua vez pode ser ainda evolutivas.

No final dos anos 90, a proposta da refatoração de software de Fowler (1999) aparece como uma solução para evolução dos sistemas legados, pois os mesmos possuem características em que é possível aplicar os conceitos definidos pelo autor como *bad smells.* Assim, cria-se a possibilidade de combinar melhorias evolutivas com manutenções corretivas.

A reengenharia também possibilita em muitos casos criar novas bases tecnológicas, novos bancos de dados, e agilizar os processos de migração de banco de dados. Porém, os sistemas legados precisam mais que refatoração e reengenharia, precisam incorporar novas tecnologias, novas adaptações, novos conceitos e ainda novos requisitos funcionais e não funcionais.

Os projetos de rejuvenescimento, denominado muitas vezes de projetos de migração possuem características e complexidade de grandes projetos, devido à

multidisciplinaridade envolvida nos processos destes projetos. A dificuldade do conhecimento dos sistemas legados tem sua origem no contexto no qual foram desenvolvidos, não existem documentos, informações, registros que consigam recuperar as informações contextuais, as diversas situações nas quais as manutenções adaptativas foram realizadas e até mesmo outros projetos de evolução. O planejamento para a evolução do software que foi concebido há 20 anos, por exemplo, com estrutura de dados (arquivos seqüenciais indexados), sem documentação atualizada, para implementação de requisitos legais com prazos curtos e multas elevadas pelos setores fiscais representam um projeto de elevado risco. Portanto, elementos de contexto são fundamentais para apoiar o processo de rejuvenescimento.

No contexto deste trabalho, a terminologia “contexto” está relacionada ao ambiente onde ocorre o trabalho das manutenções dos *softwares,* o qual consiste, segundo Brézillon (1999) da descrição complexa do conhecimento compartilhado, físico, social, histórico e circunstâncias onde ações ou eventos acontecem.

O objetivo deste trabalho é avaliar como os elementos do contexto podem potencializar as atividades de manutenções evolutivas dos sistemas legados, através de uma ferramenta que possibilite maior conhecimento, registro e recuperação das informações contextuais. Para tanto, buscou-se conceituar os ciclos de desenvolvimento da engenharia de software e os elementos do contexto na área de manutenção evolutiva do software. Foram realizados ainda estudos na busca pelos trabalhos relacionados na área de evolução dos *softwares*, rejuvenescimento de *softwares,* as dificuldades para evoluir os sistemas legados e os conhecimentos contextuais nos cenários de elicitação de requisitos.

Nesta abordagem foi realizado um trabalho de investigação através da observação de três cenários organizacionais distintos. Neles, o único ponto em comum se trata da necessidade de rejuvenescer seus sistemas legados. A partir da pesquisa de campo, foi possível definir os requisitos de uma ferramenta, com a finalidade de apoiar a rastreabilidade dos conhecimentos contextuais durante a manutenção evolutiva dos sistemas legados.

A solução criada durante a pesquisa, denominada MAGIC EYES consiste em uma ferramenta que possibilita apoiar a gestão dos projetos de evolução de software dos sistemas legados utilizando as informações de contexto dos requisitos de

desenvolvimento permitindo manter um histórico dos conhecimentos contextuais, fatores determinantes para compreensão das soluções adotadas.

Os principais requisitos da ferramenta são: mapear os contextos dos requisitos das aplicações em evolução, manter um histórico das decisões das soluções adotadas, criar e manter um cadastro dos elementos do contexto, relacionar os vários elementos, os atores com seus perfis, cenários e suas decisões, ações e suas justificativas dentre outros.

Quando da utilização da ferramenta foi evidenciada a necessidade de estender os requisitos para auxiliar na compreensão dos códigos fontes através de análise sintática da linguagem dos códigos fontes do contexto da aplicação, apoiar o diagnóstico dos pontos críticos para manutenção e registrar os processos de tomadas de decisão. A ferramenta busca apoiar o profissional gestor da área de manutenções evolutivas e adaptativas, que tem muitas vezes acesso apenas aos códigos fontes, aliado a escassez de mão de obra na linguagem dos sistemas legados e a pressão nos prazos.

Este trabalho esta organizado da seguinte forma: o Capítulo 2 discute os aspectos da evolução do software, os conceitos relacionados ao rejuvenescimento de software, o papel da engenharia de software, sistemas legados. Os conceitos do contexto aplicados ao desenvolvimento de sistemas na fase de elicitação de requisitos são abordados no Capítulo 3. No Capítulo 4 é apresentada a ferramenta MAGIC EYES. No Capítulo 5 é explorada a avaliação da ferramenta através do caso realizado. O Capítulo 6 apresenta as conclusões e as sugestões para trabalhos futuros.

2 EVOLUÇÃO DO SOFTWARE

Para estudar o rejuvenescimento do *software* se faz necessário compreender o ciclo de vida do mesmo e o papel da engenharia de software no processo de sua evolução. Este capítulo contempla os conceitos do ciclo de vida da engenharia de *software,* as várias técnicas para evolução do *software*, e os estudos relacionados com as dificuldades no processo decisório para manter, evoluir *(*rejuvenescer) ou descartá-lo.

Para Leite (2010), a engenharia de software é a disciplina envolvida com a produção e manutenção sistemática de *software* que são desenvolvidos com custos e prazos estimados. Esta definição enfatiza que a engenharia visa não apenas o desenvolvimento, mas também a manutenção do produto considerando o processo da gerência do projeto.

O ciclo de vida de um *software* descreve as fases pelas quais o produto passa desde a sua concepção até a sua inutilidade. O conceito de ciclo de vida de um *software* é muitas vezes confundido com o de modelo de processo. Gustafson (2003) define ciclo de vida do *software* como uma sequência de diferentes atividades executadas durante o desenvolvimento do *software* o que reforça a ideal de modelo de processo. Para Pressman (2006, p.32) o ciclo de vida de um *software* se inicia no nível de sistema e avança ao longo da análise, projeto, codificação, teste e manutenção.

A proposta apresentada por Leite (2010) identifica o ciclo de vida de sistemas contendo 4 fases que são delimitadas por eventos típicos em diversos ciclos de vida. São elas: Definição, Desenvolvimento, Operação e Retirada do *software* da operação.

A fase da retirada é o grande desafio da área da engenharia de software. Nos tempos atuais diversos *softwares* que estão em funcionamento em empresas possuem excelentes níveis de confiabilidade e de correção. No entanto, eles precisam evoluir para novas plataformas operacionais ou ainda para incorporar novos requisitos sejam eles funcionais ou não funcionais.

A evolução dos softwares, normalmente é realizada através da atividade de manutenção que pode ocorrer de duas formas: corretiva e evolutiva. A manutenção corretiva visa à resolução de problemas referentes à qualidade do *software* (falhas,

baixo desempenho, baixa usabilidade, falta de confiabilidade, etc.). A manutenção evolutiva ou adaptativa visa à produção de novas versões do *software* de forma a atender a novos requisitos dos clientes, ou adaptar-se às novas tecnologias que surgem (*hardware*, plataformas operacionais, linguagens etc.).

A retirada de sistemas grandes e complexos com um grande número de usuários numa empresa é sempre uma decisão difícil, pois requer descartar uma ferramenta que é confiável, a qual os funcionários estão acostumados, após anos de treinamento e utilização.

Arthur (1994), no estudo da melhoria contínua de um *software,* defende a idéia da melhoria antes da sua retirada exemplificando que é possível atuar reprojetando, reestruturando, reescrevendo parcial ou completamente. Afirma ainda que a aposentadoria do sistema é a melhor solução quando os custos de manutenção de *software* e *hardware* excedem o custo do desenvolvimento.

Segundo Sommerville (2007), os *softwares* como ativos das empresas devem permanecer úteis, por isso a maior parte do orçamento das empresas é destinado à manutenção de sistemas existentes, Erlikh 2000 apud Sommerville (2007) afirma que 90% dos custos do software estão na evolução. As variações neste percentual pertencem aos custos de evolução ou de manutenção.

2.1 PROCESSO DE EVOLUÇÃO DO *SOFTWARE*

Os sistemas grandes e complexos possuem ciclo de vida longo. Os *hardwares* são substituídos ou atualizados, os modelos de gestões são modernizados, enquanto o ambiente externo determina mudanças sistemáticas. Para que os aplicativos permaneçam úteis, eles devem acompanhar as mudanças impostas pelos sistemas sociais (LAUDON; LAUDON, 2007).

A dinâmica organizacional impõe aos sistemas alterações para permanecerem úteis. O desenvolvimento de novas funcionalidades para atender a novos requisitos ou até mesmo para correções e aprimoramentos se tornam constantes.

A necessidade da contínua evolução sugere um modelo em espiral de desenvolvimento onde é possível ter várias versões compreendendo fases de

requisitos, projeto, implementação e testes durante todo o ciclo de vida da construção do software (SOMMERVILLE, 2007)

Não se pode prever os novos desafios para as grandes empresas, no entanto, “os engenheiros de software podem se preparar para instanciar um processo que seja suficientemente ágil e adaptável para acomodar grandes modificações em tecnologias e regras de negócios que certamente virão na próxima década” (PRESSMAN, 2007, P.8).

Lehman e Belady (1974 apud PFLEEGER, 2004) nas décadas de 1970 e 1980 iniciaram o trabalho de investigação da dinâmica da evolução dos programas, das mudanças nos sistemas, o qual resultou na proposta de conjunto de leis denominada de Leis de Lehman. Este conjunto consiste de hipóteses, no qual as cinco primeiras foram propostas por Lehman e as demais por outros estudos. Abaixo, se apresenta a síntese destes estudos, base teórica fundamental às decisões das manutenções dos sistemas legados:

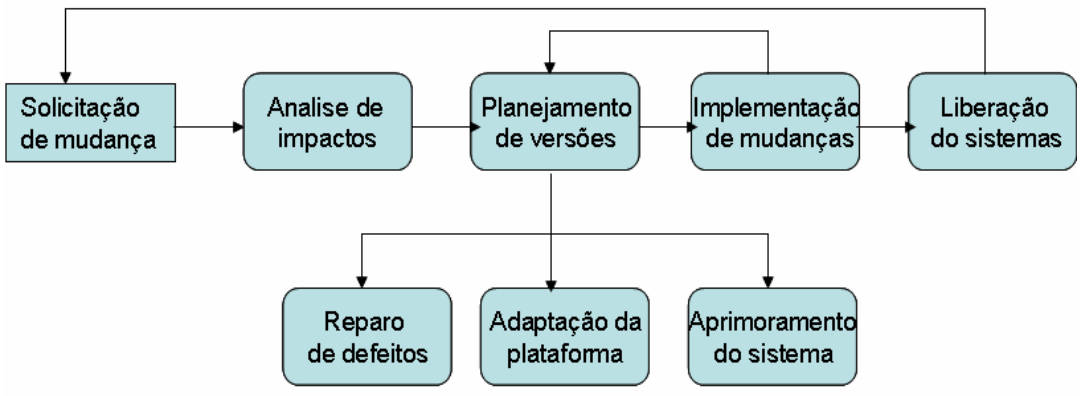
- A primeira Lei estabelece que a manutenção do sistema é inevitável e continua para permanecer útil;
- A segunda Lei sugere que para evitar a degradação decorrente das manutenções deve-se investir em manutenções preventivas, aprimorando a estrutura do software sem adicionar novas funcionalidades.
- A terceira Lei “mais interessante e polêmica” sugere que os sistemas de grande porte possuem uma dinâmica própria, definida nos estágios iniciais do seu desenvolvimento o que limita ao número de mudanças. As questões relacionadas a fatores estruturais, bem como ao tamanho e à complexidade tornam os grandes sistemas difíceis de alterar. A probabilidade dos programadores introduzirem novos erros é elevada, enquanto a confiabilidade diminui.
- A quarta Lei consistente com a terceira sugere que a evolução do programa é independente das decisões gerenciais. Lehman (2007) afirma ainda que os grandes projetos de programação ocorrem no estado “saturado”, isto é, as mudanças são imperceptíveis para a evolução do sistema.
- A quinta Lei relaciona o número de alterações com os números dos defeitos. Assim, a cada nova versão do sistema novos erros são corrigidos e outros adicionados.

- A sexta e a sétima Lei são similares preocupando-se quanto ao nível de satisfação dos usuários: caso o sistema não sofra alterações constantes e seja contemplado com novas funcionalidades, os usuários ficarão cada vez mais insatisfeitos.
- A última Lei relacionada ao sistema de *feedback*, onde os processos de evolução incorporam agentes e *loops* como fonte de aprimoramento significativos de produto.

O objetivo da engenharia de software moderna é conceber uma metodologia que seja fundamentada na noção de evolução, ou seja, a noção de que os sistemas de software modificam-se continuamente. Novos sistemas de software são construídos a partir dos antigos e todos precisam interoperar e cooperar uns com os outros (DAYANI-FARD apud PRESSMAN, 2007)

O processo de evolução de um *software* depende das metodologias utilizadas nas estruturas organizacionais. Sommerville (2007) descreve um fluxo de trabalho para o processo de evolução de sistemas (Figura 1) que envolve atividades desde a solicitação efetuada pelo cliente ou usuários, até a liberação de versões dos sistemas, contemplando as atividades de análise de impactos, planejamento e implementações.

Figura 1 - Processo de evolução de sistema



Fonte: Adaptação de Sommerville (2007, p.330)

A fase inicial do processo, a solicitação de mudança pode estar relacionada a problemas, reparos, novas funcionalidades e ainda questões mandatórias (leis) que muitas vezes impedem de ser utilizado o fluxo do processo formal na organização. O processo é iniciado com a etapa de análise de viabilidade, estimativas de custos e

prazos para o atendimento da solicitação. Logo após, definidos a necessidade e a viabilidade da manutenção evolutiva, será o momento de proceder com a análise de impacto. Nesta fase o analista tem por objeto da análise a abrangência das alterações requisitadas, bem como as interferências das novas funcionalidades. As tarefas de avaliação das requisições de alterações em sistemas legados são realizadas com base na documentação existente, na avaliação das estruturas de dados e nas investigações das pessoas envolvidas no processo (usuários, mantenedores, operadores e outros).

O processo de implementação obedece ao ciclo de desenvolvimento usado pela metodologia. A grande diferença da atividade de implementação da manutenção em relação à atividade de implementação do desenvolvimento é que esta atividade se inicia com a tarefa da compreensão do programa ou rotinas a serem alteradas. A tarefa de compreensão deve assegurar que as novas funcionalidades não irão prejudicar as demais (SOMMERVILLE, 2007).

Um fator importante no processo de manutenção é o tempo da sua realização, as alterações devem ser concluídas o mais breve possível. Às vezes, se pode alterar rapidamente, procedendo com uma solução de contorno, libera-se uma versão do *software* e logo após retorna-se para a análise mais detalhada da questão. Se necessária nova solicitação poderá ser realizada (MAZOLLA, 2008).

2.2 EVOLUÇÃO DE SISTEMAS LEGADOS

Segundo Laudon e Laudon (2007, p.103), sistemas legados são, em geral, antigos sistemas de processamento de transações criados para computadores *mainframe*, que continuam a ser utilizados devido ao alto custo de substituí-los ou reprojetá-los.

Sommerville (2007) afirma que o tempo de duração de sistemas de *software* é muito variável e muitos sistemas de grande porte permanecem em uso por mais de 10 anos. Alguns sistemas legados possuem mais de 20 anos de existência, o que torna as organizações dependentes dos mesmos.

Os sistemas legados foram concebidos com uma estrutura que envolve alguns componentes e relacionamentos, descritos a seguir:

a) *Hardware* de sistema – alguns sistemas foram escritos para *hardware* de *mainframe* que não estão mais disponíveis.

b) *Software* de apoio – os sistemas legados contam com *softwares* de apoio, de sistemas operacionais e utilitários fornecidos pelos fabricantes dos *hardwares*, muitos já obsoletos e sem o suporte dos fornecedores.

c) *Software* de aplicação – os sistemas são compostos de diversos programas separados desenvolvidos em situações e momentos diferentes.

d) Dados da aplicação – os sistemas legados normalmente processam grandes volumes de dados dispersos em vários arquivos, o que tornam inconsistentes (muitas vezes) e duplicados.

e) Processos de negócios – os sistemas podem dar suporte aos negócios da organização e ainda restringir os processos de negócios por algum tipo de limitação operacional (*hardware* ou *software*).

f) Políticas e regras de negócios - o sistema legado contém nos seus diversos programas políticas e regras de negócios os quais determinam a condução dos processos organizacionais.

Ainda segundo Sommerville (2007), outra forma de compreender a estrutura dos sistemas legados é através de um sistema de camadas, no qual cada camada depende da outra imediatamente inferior e ainda das interfaces de cada camada (Figura 2). Uma manutenção na camada de processos de negócios requer alterações e testes apenas nesta camada. Uma alteração na camada de *hardware*, a exemplo de uma adição de um novo modelo de impressora requer alteração, adaptações e testes nas várias camadas; na camada da aplicação (no código fonte, para a inclusão da impressora), nas camadas de softwares de apoio, (testes dos *softwares* de apoio) e finalmente os testes integrados no processo de negócio que irá utilizar o novo *hardware*.

Figura 2 – Sistemas de camadas do sistema legado

Fonte: Adaptação Sommerville (2007)

No cotidiano das manutenções, este encapsulamento, ou seja, o sistema de camadas, raramente funciona e as mudanças em cada uma destas camadas podem provocar uma necessidade de adaptações em outras camadas. As alterações nos sistemas legados podem ser das seguintes ordens:

- *hardwares* - novas implementações de *hardwares*, logo as demais camadas precisam se adaptar;

- novas tecnologias - migração para banco de dados, por exemplo. Como resultado, as demais camadas precisam usufruir destes novos benefícios;

- novos requisitos de desempenhos, com novos *hardwares* podem exigir mudanças na interface.

Além disso, outras mudanças podem ser necessárias, como novas versões de sistemas operacionais, novos mecanismos de segurança, novos *hardwares* de entrada de dados etc.

Diante destas frequentes mudanças e das dificuldades para atendê-las, Rezende (2005) elenca as questões sempre presentes, e as opções de estratégias para evolução dos sistemas. São elas:

a) descartar o sistema completamente – essa alternativa deve ser usada quando o sistema não atende mais aos requisitos do usuário;

b) deixar o sistema sem alterações e continuar com a manutenção regular – essa alternativa é utilizada quando o sistema é bastante estável com poucas solicitações de mudanças;

c) reengenharia do sistema para aprimorar sua facilidade de manutenção – essa alternativa deve ser usada quando a qualidade do sistema foi degradada pelas frequentes mudanças e o sistema ainda é muito necessário. Através do

processo de melhorias será possível integrar com outros sistemas e realizar novas funcionalidades;

d) substituir todo ou parte do sistema por um novo sistema – essa opção deve ser usada quando parte do sistema pode ser substituído por novos *hardwares*, novas tecnologias, ou partes são substituídas por novos sistemas comerciais.

Para quaisquer das opções a serem usadas uma criteriosa avaliação deve ser realizada, o que pode resultar na utilização de uma das alternativas ou combinações das mesmas.

Outra questão a ser analisada, segundo Mazolla (2008), em relação aos sistemas legados é a importância para os negócios da empresa. Após muito tempo de uso estes aplicativos tornaram-se fundamentais para o acompanhamento operacional. Em contrapartida as frequentes manutenções realizadas no decorrer deste período resultaram em códigos incompreensíveis e programas ineficientes, com alta taxa de falhas.

A fim de avaliar a qualidade técnica de uma aplicação, Sommerville (2007) sugere estudar os fatores relacionados à confiabilidade do sistema, às dificuldades de manutenção do sistema e à documentação existente. Ou ainda, realizar análises qualitativas através da coleta dos seguintes dados: número de solicitações de mudanças, número de interfaces com o usuário e o volume de dados usados pelo sistema.

Todos estes dados, apesar da dificuldade da obtenção, serão fundamentais para auxiliar na tomada de decisão quanto a descartar, ou manter e evoluir o sistema. Entre os vários fatores para auxiliar na tomada de decisão, alguns deles são determinantes, tais como: conjuntura organizacional (aquisição de outra empresa que possua *softwares* com funcionalidades semelhantes), pressão de clientes, ou ainda custo elevado da manutenção.

O quadro 1 apresenta os fatores com as respectivas questões, as quais podem ser utilizadas na tentativa de auxiliar o esclarecimento e apoiar a elucidação dos motivos que influenciam a decisão da evolução do software. Qualquer resposta positiva para umas destas questões significa que este é o momento de proceder à análise para troca por um novo aplicativo. Ainda é possível a elaboração de uma comparação entre o custo efetivo de manutenção e o custo da construção da nova solução.

Quadro 1 - Fatores usados na avaliação de ambientes

FATOR	QUESTÕES
Facilidade de compreensão	Qual é a dificuldade para compreender o código-fonte do sistema atual? Qual a complexidade das estruturas de controle usadas? As variáveis têm nomes significativos que refletem suas funções?
Documentação	Qual documentação de sistema está disponível? A documentação é completa, consistente e atualizada?
Dados	Existe um modelo de dados explicito para o sistema? Até que ponto os dados estão duplicados nos arquivos? Os dados usados pelo sistema estão atualizados e consistentes?
Desempenho	O desempenho do sistema é adequado? Os problemas de desempenho têm efeito significativo em relação aos usuários do sistema?
Linguagem de programação	Os compiladores modernos estão disponíveis para a linguagem de programação usada para desenvolver o sistema? A linguagem de programação ainda é usada no desenvolvimento do sistema?
Gerenciamento de configuração	Todas as versões de todas as partes do sistema são gerenciadas por um sistema de gerenciamento de configurações? Existe uma descrição explicita das versões de componentes usadas no sistema atual?
Dados de teste	Existem dados de teste para o sistema? Existe um registro de testes de regressão realizados quando novas características forem acrescentadas ao sistema?
Habilidades de pessoal	Existem pessoas disponíveis que tenham as habilidades para manter a aplicação? Existe somente um número limitado de pessoas que compreendem o sistema?

Fonte: Sommerville (2007, p.336)

Alguns outros fatores não são apresentados no quadro 1, tais como: novas oportunidades de parceiros fornecedores de novas tecnologias, instabilidade da plataforma atual, baixa taxa de produtividade de pessoal e custos de projetos mais elevados.

Pfleeger (2004) cita que estudos publicados apontam que os custos relacionados ao desenvolvimento e à manutenção de sistemas giram em torno de 20% dos esforços para desenvolvimento e 80% para manutenção. O autor também utiliza uma lista de questões que podem elucidar sobre a decisão de continuar a manter os sistemas ou construir um novo sistema, como segue:

a) O custo de manutenção é muito elevado?

b) O sistema aceita melhorias?

c) O sistema aceita adaptações dentro de intervalos de tempos razoáveis?

d) O sistema continua atendendo aos requisitos de *performance*?

e) O sistema funciona no limite esperado?

f) Outros sistemas podem realizar o mesmo, ou melhor, mais rápido ou mais barato?

g) O custo de manutenção de hardware é elevado que justifique a troca por outro mais barato?

As ferramentas relacionadas ao processo de qualidade podem dar suporte à análise para a tomada de decisão. Arthur (1994) sugere o uso do diagrama genérico de Ishikawa com as seis principais categorias de causas sendo elas: pessoas, processos, máquinas, materiais, medidas e ambiente. Deming (1982 apud ARTHUR,1994) afirma que 85% dos problemas originam-se nos processos, enquanto os engenheiros de *softwares* insistem em procurar as causas nos projetos de códigos, arrumando alguns defeitos e criando outros. A idéia do uso do diagrama, mapeando as causas e efeitos permite a análise por todo o ciclo de vida da construção ou manutenção do sistema. Com os resultados da diagramação é possível traçar um plano de ação para implementação da evolução dos *softwares*.

Nos anos 1990 surgem os processos de reengenharia como suporte, ao processo evolutivo do *software*. As fases de transição ou migração de um *software* legado para um novo *software* de tecnologias correntes requerem um planejamento, estudo e conhecimento do legado, um plano de ação de forma a proporcionar uma

retirada mais suave. Neste cenário onde 47% a 62% do tempo da manutenção são consumidos na compreensão de programas e documentos, a reengenharia desempenha o papel de impor rapidez na transformação dos códigos fontes em artefatos de maior clareza para outro nível de profissional (PIGOSRKI, 1997).

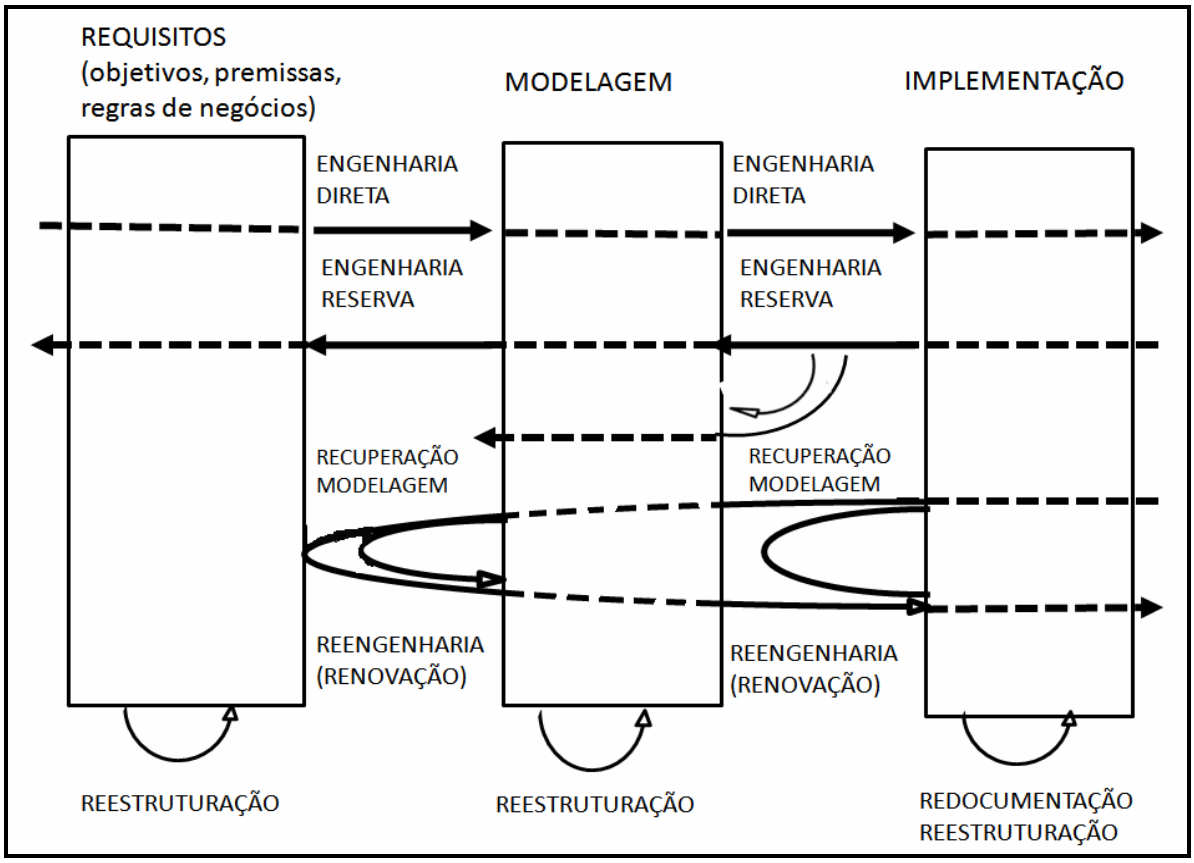
2.3 SOLUÇÕES PARA A EVOLUÇÃO DO SOFTWARE.

A reengenharia apareceu como solução para o problema da evolução do *software* bem como, uma ferramenta para a melhoria da sua compreensão. Diversos termos e formas da engenharia aplicada a engenharia de *software* trazem a necessidade de esclarecer o uso destes termos:

Chikofsky e Cross (1990) definem reengenharia na área de *software* como “o exame e alteração do *software* com o objetivo de reconstituí-lo e implementá-lo em um novo formato.” Os autores exploram os conceitos de engenharia direta (*forward engineering*), a qual trata da migração de visão de alto nível dos requisitos e modelos para a implementação concreta dos mesmos, atuando desde a fase dos requisitos do *software* (Figura 3). Enquanto a engenharia reversa (*reserve engineering*) se utiliza dos códigos fontes para criar modelos em nível de abstração mais alto de uma aplicação, com o objetivo de compreender o *software* e gerar documentação para eliminação de possíveis problemas existentes.

A reengenharia consiste em combinar as engenharias: reversa e direta, através da transformação de determinadas implementações concretas e desenvolvimento de outras soluções. Outros autores exploram e define a reengenharia de várias formas, sempre atribuindo a obtenção de um novo produto. O glossário da engenharia de software define a reengenharia como: a facilidade com que um sistema ou componente de *software* pode ser modificado visando à remoção de falhas, a melhoria de desempenho (ou de outros atributos), ou ainda a adaptação a novas plataformas (IEEE, 1990).

Figura 3 – Fronteiras da aplicação das reengenharias



Fonte: Chikofsky e Cross (1990)

Flower (1999) define refatoração como o processo de modificar um sistema de *software* para melhorar a estrutura interna do código sem alterar seu comportamento externo. O uso desta técnica aprimora a concepção (*design*) de um *software* e evita a deterioração tão comum durante o ciclo de vida de um código. Esta deterioração é geralmente causada por mudanças com objetivos de curto prazo ou por alterações realizadas sem a clara compreensão da concepção do sistema.

Alguns aspectos devem ser observados para apoiar os procedimentos do processo de reengenharia do *software* como uma alternativa de baixo custo para a manutenção do *software* (MAZOLLA, 2008). São eles:

- selecionar os programas que estejam sendo bastante utilizados no momento e que continuarão a ser utilizados nos próximos 5 a 10 anos;
- estimar o custo anual de manutenção dos programas selecionados, incluindo correção de erros, adaptação e melhorias funcionais;
- organizar, por ordem de prioridade, os programas selecionados, registrando os custos levantados no procedimento anterior;

- estimar os custos para realizar a reengenharia dos programas selecionados e estimar ainda os custos de manutenção do programa após realizada a reengenharia;
- realizar uma análise comparativa de custos para cada programa;
- calcular o tempo necessário para o retorno de investimento da reengenharia;
- levar em consideração algumas questões importantes que resultam da reengenharia, tais como a melhor confiabilidade do sistema, melhor desempenho e melhores interfaces;
- obter a aprovação da empresa para realizar a reengenharia de um programa;
- com base nos resultados obtidos desta experiência, desenvolver uma estratégia de reengenharia dos demais programas.

Pelo número de questões relacionadas acima se compreende que a reengenharia auxilia no processo de melhoria das manutenções dos sistemas, porém no caso dos sistemas legados, algumas variáveis tais como: tamanho dos programas, idade dos programas, quantidade de “lixo” (*bad-smell*) existente no código podem dificultar o processo de evolução sem contar com a grande abrangência dos programas que seriam selecionados para aplicação da reengenharia.

Neste cenário complexo os elementos do contexto podem contribuir na compreensão e na melhoria da qualidade da decisão.

3 REJUVENESCIMENTO DE SOFTWARE

Este capítulo aborda os estudos relacionados com as leis da evolução dos *softwares* e as complexidades dos projetos de evolução dos sistemas legados.

As investigações sobre o envelhecimento e rejuvenescimento de software começaram em 1995 e os resultados foram apresentados em diversas conferências em especial a ISSRE - International Symposium on Software Reliability Engineering. Após quase 15 anos de pesquisa, em 2008 quando ocorreu o *First International Workshop on Software Aging and Rejuvenation* (WoSAR) foram reunidos diversos trabalhos de investigação sobre o envelhecimento dos *softwares*. O estado da arte destes estudos chama atenção para os métodos experimentais utilizados na busca dos esclarecimentos quanto ao momento do envelhecimento do *software*. Vaidyanathan e Trivedi (2005) no seu estudo definem que o *software* envelhece com o tempo de uso. Este fenômeno pode ser observado pela degradação de desempenho do sistema ou pela ocorrência dos acidentes com resultados de esgotamento dos recursos do sistema operacional, pela corrupção de dados, ou ainda pelo acúmulo de erros numéricos.

Como solução para o envelhecimento, estudos para rejuvenescimento são apresentados, a exemplo do trabalho de Dohi e Eto (2006) autor dos modelos de rejuvenescimento de *software* com semi-processos de Markov, baseado em algoritmos estatísticos não paramétricos para estimar os horários ideais para melhor aplicar os métodos de rejuvenescimento. Além disso, o modelo busca maximizar a disponibilidade e minimizar custos, os quais são derivados analiticamente dos modelos. O uso do modelo do rejuvenescimento foi estendido aos sistemas de *cluster* e modelos analíticos do programa de implementação de *software* e como resultado do rejuvenescimento um significativo aumento da disponibilidade do sistema e diminuição do tempo de inatividade foi apresentado.

O envelhecimento do *software* não está apenas ligado com a carga e sobrecarga de trabalho, o sistema operacional e a degradação de *performance*. Muitas vezes está relacionado ao grau de satisfação dos usuários com os serviços fornecidos pela aplicação.

Liu (apud PRESSMAN, 2007) observam que muitos sistemas legados permanecem dando suporte às funções importantes do negócio das organizações e

são indispensáveis para os mesmos. Assim, sistemas legados tornam-se caracterizados pela longevidade e criticidade para os negócios. A necessidade de manter aplicações críticas de forma atual e moderna, passa pela dificuldade da tomada de decisão sobre o melhor momento, o modo de fazer a mudança da aplicação e a forma para gerir os recursos necessários, bem como a maneira de medir os impactos para a Organização.

As tarefas de análise de impacto em conjunto com avaliação dos novos requisitos da evolução do software têm um aliado nas técnicas e ferramentas que suportam o processo da tomada de decisão. Outro fator para compreender a estrutura do problema é considerar o contexto (anterior) aonde foi criada a aplicação e ainda comparar os elementos deste contexto com o cenário atual.

O elemento essencial para se produzir uma decisão é a existência de alternativas, ou seja, a necessidade de se realizar uma “escolha” deriva da posse de dois ou mais diferentes objetos ou caminhos, dos quais apenas um pode ser selecionado. Se não existirem alternativas poderá existir um problema, mas não haverá um problema de decisão (PESCADA, 2009).

Segundo Bana e Costa (2008), existem três tipos de processos para realizar a tomada de decisão:

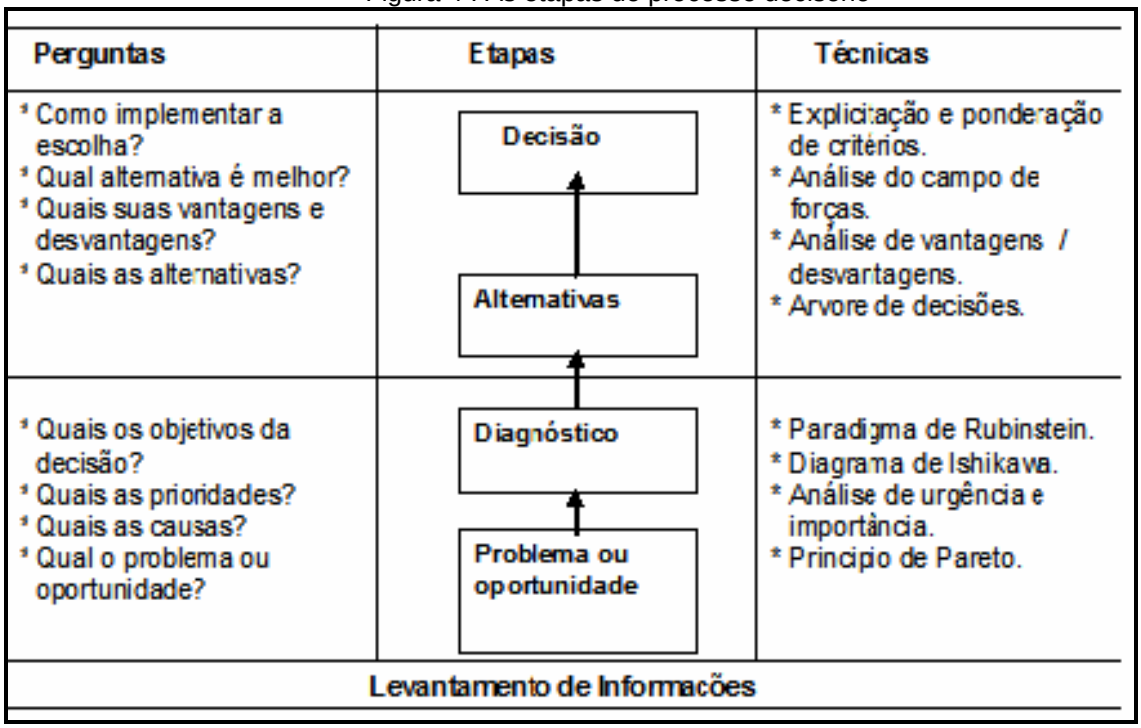
- Processo intuitivo – o qual acontece utilizando-se conhecimentos, intuição, experiência anteriores; por isso de forma menos precisa devido a falhas de memórias, limites mentais, distrações e uso de informações mais recentes, ou mais fáceis de serem avaliadas;
- Processo analítico – o qual adota procedimentos analíticos, com risco de serem inconsistentes;
- Processo consultivo- o qual acontece com a ajuda de um facilitador – analista de decisão; onde a análise do contexto da situação faz parte da estruturação do processo de ajuda à decisão.

Na metodologia de conferências de decisão, Thomaz (2005 apud CUNHA; THOMAZ; MOURA 2005) integra a teoria da decisão no processo técnico de modelagem do contexto e situação de decisão. Como resultado as tecnologias de informação no processo tecnológico de apoio computacional especializado e os processos de grupo criam um ambiente propício para o planejamento e gestão de uma reunião bem sucedida.

As decisões precisam de uma base sólida teórica que considere os diversos aspectos (critérios) que afetam a tomada de decisão. A metodologia baseada na teoria dos multicriterios é composta por três fases: (i) Estruturação, a qual se relaciona à identificação do problema e ao diagnóstico; (ii) Avaliação; e (iii) Elaboração de recomendações, diz respeito ao levantamento de alternativas e decisão (escolha da solução).

Segundo o PMBOK – *Project Management Body of Knowledge* (2008), para o levantamento das informações do processo decisório pode-se combinar várias teorias e técnicas para realizar as diversas etapas. A Figura 4 mostra de forma estruturada que para cada etapa utilizam-se as questões investigatórias (perguntas), combinadas com as técnicas, as quais provocam o esclarecimento e permitem encontrar direção para a solução. Por exemplo, na fase de Diagnóstico, a compreensão do problema é possível com a utilização do diagrama de Ishikawa buscando-se determinar as principais causas da questão discutida.

Figura 4 : As etapas do processo decisório



Fonte : Adaptação PMBOK, 2004.

As várias técnicas do processo decisório remetem à compreensão, ao entendimento da situação a qual pode ser auxiliada pela representação do conhecimento contextual para melhor assertividade durante o processo.

Nas manutenções dos sistemas legados, os volumes de informações a serem analisadas bem como o número de pessoas envolvidas são enormes, o que caracteriza um processo complexo de análise para a tomada de decisão, logo, a necessidade da utilização do modelo do processo decisório

3.1 REJUVENESCIMENTO DE SISTEMAS LEGADOS

As leis da evolução do software estudadas há 30 anos aplicadas aos *softwares* do tipo E[1], sugerem uma avaliação quanto a necessidade de evolução quase permanente, pois estes sistemas, que resolvem os problemas do mundo real , estão sempre em processos de mudança.

Os sistemas legados pertencem à categoria dos sistemas do tipo “E”, os quais atendem às questões usadas pela Lei de Lehman que justificam os problemas ocorridos quando da sua evolução.Esses sistemas, devido ao cenário no qual foi construído com um grande número de usuários, interações entre sistemas, muitos requisitos e diversos recursos, não conseguem atender a toda as demandas. Como resultado o ciclo vicioso encontra-se instalado. A Lei de número um afirma que: “os sistemas grandes nunca estão completos necessitam de constantes mudanças”.

Os sistemas legados sofrem continuas manutenções o que degrada a sua estrutura, incrementando a sua complexidade e a criação do lixo nos códigos fontes (trechos não utilizados). A Lei de número dois trata do aumento da complexidade devido às contínuas manutenções.

A necessidade de novos requisitos com a pouca compreensão das funcionalidades existentes, criam disfunções nestas funcionalidades ou ainda geram redundâncias nas novas implantadas. A Lei de número cinco - explora o aspecto das muitas modificações que alteram as funcionalidades originais.

Diante das questões estudadas pelas leis de Lehman observa-se o cuidado necessário para os projetos de evolução do software. A decisão da evolução do sistema legado para o seu rejuvenescimento, e a gestão das manutenções evolutivas precisam do apoio do processo decisório em ambiente colaborativo para

[1] Os tipos de sistemas considerados pelo estudo de Lehman : (i)Sistemas S: são formalmente definidos e derivados de uma especificação, (ii) Sistemas P: possuem requisitos baseados em uma solução aproximada que seja prática de construir e utilizar, e (iii) Sistemas E: estão embutidos no mundo real e se modificam à medida que o mundo muda.

dar qualidade a decisão. A gestão destes projetos possui um papel fundamental; especificamente como estimar os recursos para a execução, como determinar a produtividade, e como calcular o tempo necessário.

Belady e Lehman (1978) consideram dois fatores para o aumento da complexidade do sistema legado ao longo do tempo: O primeiro está relacionado ao fato de como são executadas as manutenções, como as falhas são corrigidas, bem como a maneira pela qual novas falhas são introduzidas na aplicação. O segundo, está relacionado à qualidade das correções, ou seja, aos novos produtos oriundos das correções, e o seu impacto na estrutura dos códigos. Os efeitos destes fatos são apresentados pela equação (1): PFLEEGER (2004)

$$M = p + k^{c-d} \qquad (1)$$

Onde, M é o total de recursos necessários para a manutenção do sistema,

p representa os recursos produzidos para; análise, avaliação, modelagem, implementação e testes,

k é chamada de constante empírica porque depende da comparação entre o modelo da aplicação e a relação dos recursos do projeto atual.

c é a complexidade causada pelo modelo na estrutura e na documentação do sistema,

d é o nível da familiaridade da equipe de manutenção com o sistema,

Se a aplicação foi construída utilizando-se das técnicas e métodos da engenharia de software, a constante c possui um alto valor. Se a equipe que mantém o *software* é a mesma ou grande parte dos profissionais que participaram da sua construção, a constante d possui um valor baixo. Como resultado desta equação pode-se obter a definição que o custo da manutenção será incrementado exponencialmente em relação às variáveis c e d.

Complementando o estudo das estimativas para a evolução, Belady e Lehman sugerem o cálculo do tamanho da manutenção através da equação (2):

Tamanho manutenção= ASLOC (AA + SU + 0,4DM + 0,3CM + 0,3IM) (2)

Onde :

ASLOC é o número de linhas do código a ser alterado

AA é o valor inteiro do intervalo de (0 a 8) capaz de mensurar o nível de documentação e esforços de assimilação dos componentes .(Quadro 2).

SU é o número inteiro no intervalo de (0 a 5) conceitual ; muito pouco, pouco, normal, alto e muito alto identificando o entendimento e a compreensão do software em relação à clareza da estrutura , dos programas, e da documentação;

DM é o percentual (%) do modelo a ser modificado

CM é o percentual (%) do código a ser modificado

IM é o percentual (%) do código externo (código reutilizado)

Quadro 2 - Classificação do esforço de avaliação e assimilação

Classificação do esforço de avaliação e assimilação, segundo o COCOMO II	
Aumento na avaliação e assimilação	**Nível de esforço de avaliação e assimilação**
0	Nenhum
2	Pesquisa e documentação dos componentes básicos
4	Alguma documentação de teste e avaliação de componentes
6	Consideravel documentação de testes e avaliação de componentes
8	Extensa documentação de testes e avaliação de componentes

Fonte : Adaptação Pfleeger , 2004

Apesar da teoria desenvolvida acerca da manutenção evolutiva do *software*, na prática as ferramentas disponíveis não passam de editores de texto, comparadores de arquivos, compiladores e *linkers*, ferramentas de depuração, geradores de referência cruzadas desenvolvidos pelos fornecedores das linguagens, analisadores estáticos de códigos e ainda *softwares* para gerência de configuração. Estas ferramentas apóiam os implementadores nas suas atividades de alterações de códigos fontes. Entretanto, as atividades de planejamento, tais como: análise de impacto, estimativa de esforços e, finalmente, a tomada de decisão são realizadas utilizando-se das experiências individuais e de conhecimento tácitos, referentes às lições aprendidas dos projetos anteriores.

As empresas do segmento de serviços, *outsourcing* na área de TI, ofertam ferramentas (que atuam no nível operacional) e serviços para facilitar e prover a migração dos sistemas legados, ou ainda desenvolvem um *middleware* capaz de prover as evoluções dos *softwares* para ambiente e soluções mais modernas, ou ainda converter automaticamente suas bases de dados. No entanto estes projetos são muitos grandes e complexos devido a falta de conhecimento dos sistemas legados (TINEWS, 2010).

Para facilitar a compreensão do impacto da evolução dos sistemas legados, pode-se utilizar o estudo dos cenários das elicitações de requisitos como papel fundamental para apoiar a modelagem das soluções apresentadas. Os cenários evidenciam elementos dos contextos dos ambientes organizacionais aonde ocorrem as evoluções dos sistemas legados.

3.2 CONTEXTO DE UMA APLICAÇÃO

Quando um grupo se reúne para elicitar os requisitos de uma evolução da aplicação, conhecimentos são compartilhados. Tais conhecimentos podem ser enriquecidos se forem apoiados por informações históricas das discussões anteriores, bem como, das informações dos cenários quando da criação da aplicação, objeto da discussão.

O conceito de contexto está sendo estudado desde 1960, em muitas áreas da informática. Na computação ubíqua, por exemplo, é utilizado para ampliar as atividades humanas nos novos serviços tecnológicos sensíveis às circunstâncias.

Contexto, segundo Nunes e outros (2006 apud BRÉZILLON, 2002), consiste na relação entre o texto e a situação em que o fato ocorre representado pelo conjunto das informações, tais como as circunstâncias, lugar, tempo, cultura dos envolvidos, relacionamentos, e outras. Estas informações são utilizadas para representar a história de eventos que ocorrem num determinado espaço de tempo, com o foco nos interesses dos envolvidos. Brézillon (2002) afirma que o contexto é um conjunto de informações numa determinada situação onde existe a interação homem-máquina.

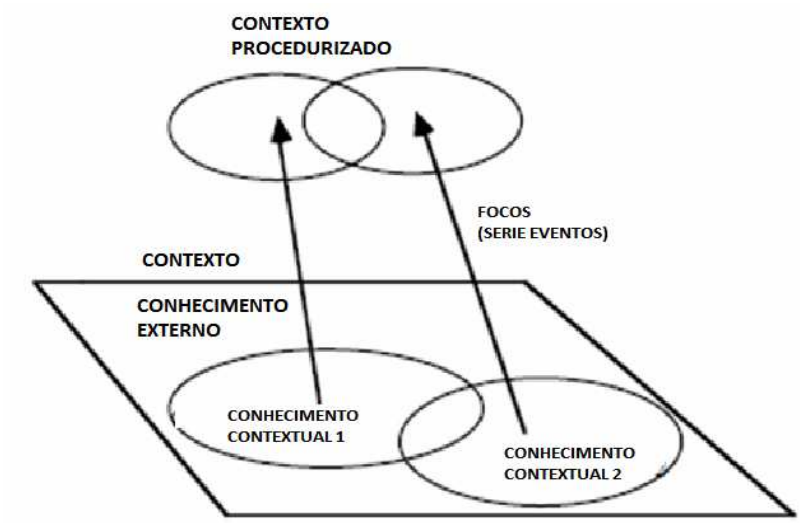
O conceito de contexto não existe de maneira individual. Ele está sempre relacionado a algum objeto, a uma ideia e a um comportamento. Ou ainda, contexto

de um ator, contexto de uma tarefa a ser executada, contexto de interações semelhantes ocorridas anteriormente, etc. Assim sendo, é muito difícil representar e disponibilizar todos os elementos de contextos relacionados à interação de grupos. Tal fato se torna mais complexo ao considerarmos grupos geograficamente dispersos (BRÉZILLON, 2002).

Em termos gerais, o contexto consiste numa descrição complexa de conhecimento compartilhado, físico, social, histórico e circunstâncias onde ações ou eventos acontecem. Todo este conhecimento não está contido nas ações e nos eventos, mas são premissas da execução e interpretação dos mesmos. Para o entendimento total das várias situações ocorridas a representação das informações contextuais impõe uma diferença, no ponto da riqueza de detalhes das informações, todas elas relacionadas (BRÉZILLON, 1999).

No âmbito dos sistemas colaborativos, Brézillon e Pomerol (1999) propõem uma classificação relacionada às atividades. Esta classificação está dividida em três tipos de conhecimentos *de* contexto, representado na figura 5. São eles: a) o conhecimento dos elementos contextuais, necessário para a execução da tarefa, denominado de conhecimento do contexto; b) o conhecimento do contexto, que é compartilhado, porém não é utilizado para a realização da tarefa, denominado de conhecimento externo; e a parte do conhecimento contextual que é realmente utilizado quando da execução da tarefa denominado de contexto procedural. Este sim, é o conhecimento envolvido para o entendimento e a estruturação da resolução do problema bem como para o auxílio nas tomadas de decisões. Neste conhecimento deve-se focar as questões a serem elicitadas durante as discussões.

Figura 5 - Diferentes tipos de contexto



Fonte : Adaptação Brézillon e Pomerol (1999)

Brézillon (2002) explora os conceitos de contexto relacionados ao foco da ocasião criando visões horizontais e verticais exemplificando com a engenharia de software. Há algum tempo, estes estudos buscam o uso do contexto com duas visões: a primeira ligada à engenharia e a segunda com base nas ciências da cognição. Em quaisquer visões a engenharia de software se beneficiará das pesquisas sobre a modelagem da contextualização. No primeiro caso, auxilia no processo de desenvolvimento de *software* e no segundo caso, nos mapas cognitivos, quando das atividades de grupo cujo objetivo seja a compreensão dos modelos e a criação das soluções que deram origem aos códigos fontes de um *software*.

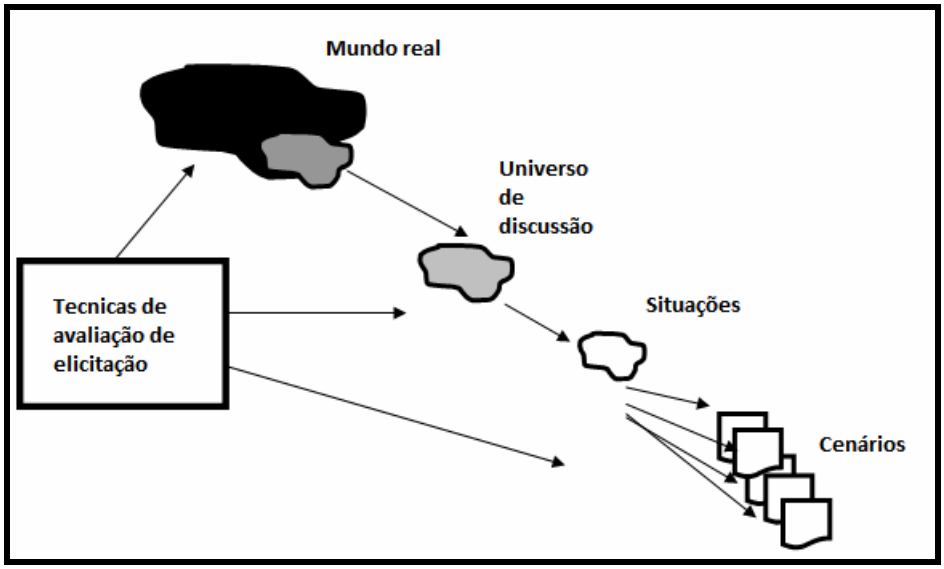
A aplicação do conceito de contexto ao processo de desenvolvimento de *software*, através de triplas de uma ontologia sugerida por Nunes e outros (2006), mostra como compartilhar conhecimento, partindo dos conceitos básicos, explorado pelos autores Brézillon desde 1999, enriquecido com informações dos indivíduos dos grupos participantes. A representação gráfica das triplas permite uma visualização das interações dos muitos elementos envolvidos na definição de requisitos de um software. Neste modelo é explorado um ambiente de elicitação de requisitos e seus elementos: as iterações da solução, os atores envolvidos no processo o perfil e a competência destes atores, as regras de negócios, os

procedimentos e as atividades envolvidos no processo. Com o recurso das triplas é possível determinar os relacionamentos entre os elementos.

A utilização de ontologias representando o conjunto de conhecimento, o qual Schjorring (2002 apud NUNES; SANTORO; BORGES, 2006) denomina de informações ecológicas, demonstra a quantidade de informações relacionadas ao desenvolvimento de software, nas diversas etapas da sua construção, principalmente na etapa de elicitação dos requisitos, quando as atividades devem ser realizadas em grupo (pequenos ou grandes, dependendo da complexidade dos sistemas). Estas informações ecológicas podem ser registradas como o contexto do conhecimento, as quais contêm ainda, artefatos, documentos, as lições aprendidas, técnicas utilizadas e melhores práticas do processo no ambiente organizacional.

No trabalho realizado por Leite (2007) sobre cenários na elicitação de requisitos, o autor ilustra a relação do universo de discussão com parte do mundo real (Figura 6). Ou seja, as técnicas de avaliação de elicitação devem contemplar o contexto onde ocorre as situações e os diversos cenários para representar a situação atual. Nas metodologias de desenvolvimento de *softwares* atuais apenas a solução é documentada através dos casos de uso, toda a informação do contexto é ignorada.

Figura 6: Cenários da etapa de elicitação de requisitos



Fonte : Adaptação Leite (2007)

Leite (2007) apresenta as vantagens de se mapear os cenários com a finalidade de enriquecer a compreensão da situação problema. O autor elenca os

seguintes benefícios para o uso das informações dos cenários, na elicitação de requisitos:

- Permitir o uso da linguagem natural do problema entre o cliente e usuários;
- Apoiar na unificação dos critérios das alternativas das soluções;
- Estimular a discussão e a troca de idéias;
- Organizar as informações disponíveis a cerca do problema;
- Treinar os novos participantes;
- Apoiar o detalhamento dos requisitos, e identificar os requisitos não funcionais.

Para a engenharia de software a documentação do sistema começa com a fase inicial da elicitação dos requisitos representada através dos casos de uso. Para Leite (2007) a representação gráfica dos cenários antecede aos desenhos dos casos de uso. No quadro comparativo (Quadro 3), pode-se verificar as diferenças, entre os cenários, os quais representam a situação em que ocorrem as discussões do problema, e se encontram os elementos contextuais. Os casos de uso representam as soluções possíveis para a resolução dos problemas. Ou seja, os casos de uso representam o final das discussões.

Quadro 3 : Comparativo - Cenários da Elicitação de requisitos e Casos de uso

Cenários de elicitação de requisitos	**Casos de Uso**
Visão da situação atual	Visão das funcionalidades que são necessárias a solução
Envolvidos nas questões, atores, ações	Atores, ações do sistema
Baseado no problema	Baseado na solução
Pode ser usado na elicitação dos requisitos	Omite aspectos importante da elicitação
Apresentação Textual	Apresentação Gráfica (sem detalhes)
Relacionamentos dependem da representação	Relacionamentos são padronizados

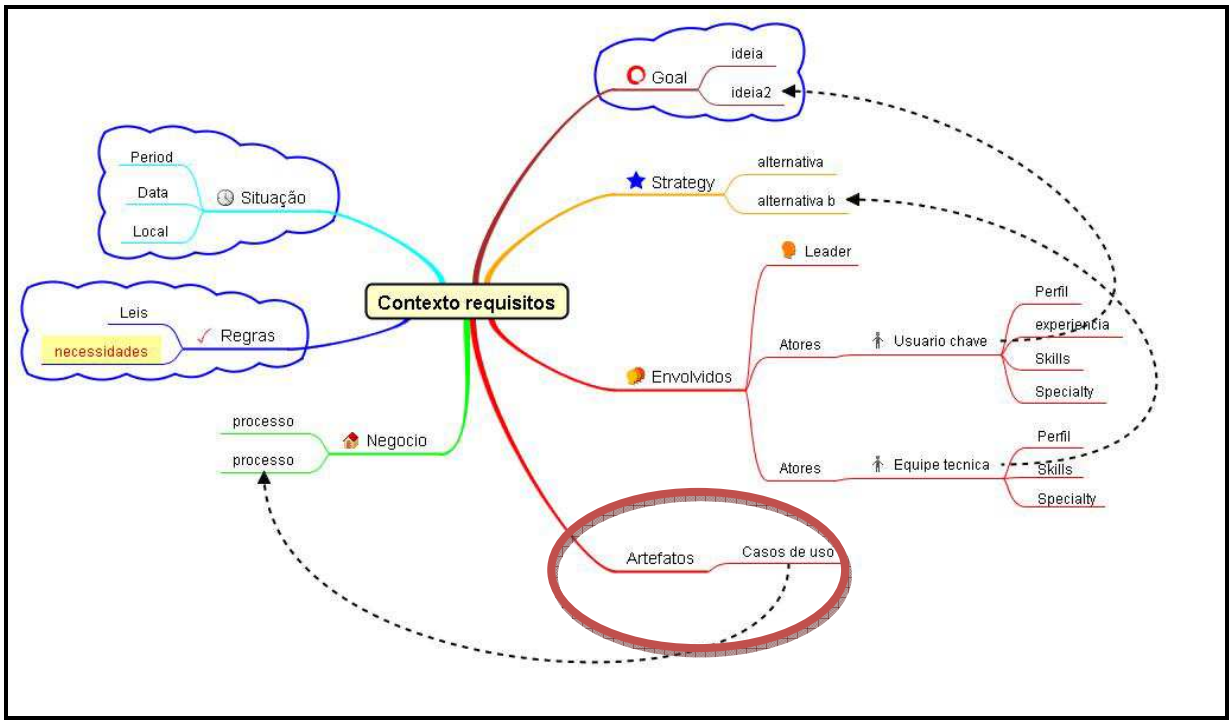
Fonte :Adaptação de Leite (2007)

Utilizando a definição de contexto, pode-se criar um mapa mental, a exemplo do modelo da ontologia da etapa de elicitação de requisitos, para a melhor visualização dos elementos envolvidos, a quantidade de possibilidades de alternativas, e ainda documentar os entendimentos acerca do assunto. A

recuperação das informações dos elementos de contexto de cenários anteriores produz uma maior compreensão acerca dos resultados das discussões anteriores e como resultado, apóia a criação das soluções.

As metodologias de desenvolvimento de sistema publicadas e praticadas frequentemente não consideram as informações contextuais, representando como resultado da etapa de elicitação de requisitos, apenas os casos de usos da solução. A Figura 7 exibe a representação de um mapa mental de um cenário de elicitação de requisitos, onde o círculo vermelho serve para assinalar o universo no qual o caso de uso encontra-se inserido. Todas as demais informações do conhecimento contextual não são registradas.

Figura 7. Mapa mental da etapa de elicitação de requisitos.



Fonte : Elaboração do autor

Rosa, Santoro e Borges (2003) defendem a “clara necessidade de apoio explícito do contexto em *groupware*”, ou seja, a necessidade da inclusão dos elementos contextuais nas ferramentas que suportam as atividades de *groupware*. Os autores realizaram um estudo comparativo entre as ferramentas existentes para apoiar o *groupware*, cujo objetivo permitiu determinar o quanto estas ferramentas utilizam os elementos contextuais nas iterações dos grupos sejam elas assíncronas ou síncronas. As ferramentas avaliadas foram:

a) BSCW - Basic Support for Cooperative Work;

b) FLE3 - Future Ambiente de Aprendizagem; e

c) Quickplace 3.

O resultado da pesquisa relata o nível de aderência de cada ferramenta em relação aos elementos contextuais, a seguir:

a) As três ferramentas prevêem uma cobertura parcial em relação às informações do indivíduo e do grupo, assim como das tarefas e contextos de interação;

b) O contexto ambiental não é contemplado pelas ferramentas;

c) Os elementos contextuais disponíveis em todas as três ferramentas são capazes de identificar as ações realizadas, porém não determinam o porquê da realização da ação;

d) Nas três ferramentas não existe separação entre o histórico do contexto e os contextos atuais.

Os pontos fracos detectados pelos autores, relatados nos itens de “a” a “d” são maximizados na evolução dos sistemas legados, pelas características da falta de conhecimento e documentação das manutenções evolutivas. A tarefa de elicitação de requisitos como atividade de grupo do tipo assíncrono se encontra no universo das tarefas, as quais necessitam de ferramentas para apoiar a sua execução.

3.3 TRABALHOS RELACIONADOS

A ocorrência de envelhecimento de software em sistemas reais tem sido documentada na literatura através de pesquisas concentradas em compreensão dos seus efeitos teoricamente e empiricamente. O resultado destas buscas, deu origem às técnicas chamadas de rejuvenescimento software.

O rejuvenescimento de software é tratado por Pfleeger (2004) como parte do processo de manutenção, considerando possibilidades de inclusão de qualidade nos softwares, a partir dos seus códigos fontes. Todas as técnicas: refatoração, engenharia reversa, reengenharia, documentação e reestruturação são consideradas pelo autor como fator de aceleração para o processo de evolução do *software*. A atenção ao papel da documentação na compreensão dos sistemas do tipo “E” aparece como sugestão para estudos nesta área. Entretanto aspectos como

informações contextuais não fazem parte das diversas metodologias de desenvolvimento de *software* ao se tratar a documentação.

O enfoque dos trabalhos de rejuvenescimento na área de aspectos estratégicos ou econômicos, inclui estudo dos cálculos de estimativas para o processo de evolução do software. Desde os anos 2000, Boehm e outros autores publicam trabalhos como o COCOMO (*COnstructive COst MOdel*), os quais demonstram a importância da estimativa de tempo e custos para o projeto. Porém, não abordam como resolver a dificuldade de realizar estes cálculos sem o conhecimento da aplicação atual pela ausência da documentação e a complexidade dos códigos fontes (SOMMERVILLE, 2007).

Fowler e outros (1999) propõem soluções para modernização de *softwares* , com o destaque do processo de refatoração, no qual a compreensão dos códigos fontes e a localização dos lixos nos códigos têm um papel fundamental no processo. Para tal é importante possuir um mecanismo que facilite a identificação destes “lixos” nos sistemas legados, haja vista que os códigos são enormes (maior que 5.000 linhas).

Segundo Parnas (1994), os *softwares* envelhecem porque as manutenções sem o conhecimento do código fonte produzem uma deterioração do código original. O autor explora mecanismos para retardar o seu envelhecimento. Tais mecanismos são muitos comuns em sistemas legados. Apesar dos trabalhos explorarem os aspectos que envolvem o envelhecimento, as questões sobre o conhecimento contextual são ignoradas. Nas manutenções dos sistemas legados os aspectos contextuais, tais como: o que, como, quando e onde ocorreram tais modificações podem auxiliar na compreensão dos códigos fontes.

Como a elicitação de requisitos é fundamentalmente uma atividade que se desenvolve em ambientes de colaboração assíncrona, desperta um interesse na observação do papel dos elementos contextuais no histórico da documentação da elicitação dos requisitos.

As lacunas evidenciadas pelos autores analisados levam a decisão para a pesquisa por uma ferramenta que permita apoiar a área da gestão no ambiente de manutenção dos sistemas legados, considerando o papel das informações contextuais como diferencial para compreensão do sistema a ser evoluído.

O quadro 4 contém um resumo dos principais trabalhos analisados e discutidos nesta pesquisa.

Quadro 4 – Resumo dos trabalhos relacionados

Período	Autores	Pesquisa realizada	Influência dos temas para a pesquisa
Década 80	Lehman e Belady	Evolução dos softwares - Lei de Lehman	As dificuldades para decidir sobre a evolução, o que leva a sugerir a falta de ferramenta para apoiar a gestão
1994	Parnas et al	Envelhecimento de Software	A falta de histórico das evoluções dos softwares. A importância do registro das informações contextuais.
1999	Fowler et al.	Refatoração de códigos fontes	Melhoria dos códigos fontes a partir do conhecimento dos “bad smell”. A criação da automação para reconhecimento dos lixos nos códigos fontes.
2000	Boehm, et al.	Estudo de métricas para evolução	Estimativas para o processo de evolução, alinhando o conhecimento dos códigos fontes.
2006	David et al	Informações Contextuais como valor agregado nos ambientes colaborativos	A riqueza das informações do ambiente do contexto nas atividades colaborativas do tipo assíncrono

Fonte : Elaboração do autor

4 A SOLUÇÃO PROPOSTA

Neste capitulo são relatadas as diversas etapas da criação da solução proposta, desde a pesquisa de campo para a elicitação dos requisitos, a escolha da arquitetura tecnológica e o ambiente de desenvolvimento até a construção da solução proposta.

O trabalho realizado possui uma natureza de **pesquisa aplicada**, enquanto que, do ponto de vista da forma de abordagem do problema, se caracteriza por uma pesquisa quantitativa e qualitativa descritiva. Quanto ao ponto de vista dos seus objetivos, **a pesquisa exploratória** contribuiu permitindo a compreensão do problema de maneira explícita, após a realização da pesquisa de campo, numa amostragem de diferentes cenários. Do ponto de vista dos procedimentos técnicos a pesquisa tem um **caráter experimental**, pois a exploração das variáveis, descritas abaixo, que melhor representam a formulação da solução para o problema, permitiu a manipulação dos seus valores para contribuir com as considerações do estudo.

Na fase inicial, da realização do projeto de pesquisa, a investigação para coleta dos dados sobre o fenômeno, aconteceu através da observação do contexto das manutenções evolutivas de sistemas legados, nos seguintes cenários:

Cenário A - Uma empresa fornecedora de um único produto - um software aplicativo ERP – especialista em um segmento de mercado, o qual deve estar sempre atualizado para a sobrevivência da Empresa.

Cenário B – Uma empresa onde a quantidade de sistemas legados é significativa, e a própria empresa cria suas próprias soluções.

Cenário C – Uma Instituição financeira, onde os sistemas legados são de vital importância para a continuidade dos seus negócios.

A coleta de dados foi realizada através de entrevistas com pessoas chaves e aplicação de um questionário. As variáveis selecionadas, as quais definem as formas de controle e permitem observar os efeitos produzidos nos objetos em estudo, foram:

a) Variáveis de interesse quanto ao conhecimento da aplicação. São elas: nível de conhecimento da aplicação pelos profissionais que executam a manutenção (qualitativa).

nível de conhecimento do domínio; ambiente, cultura organizacional e usuários (qualitativa).

números de vezes que o usuário trabalhou com a aplicação ou alterou qualquer um dos códigos fontes (quantitativa).

números de erros encontrados durante a fase dos testes (quantitativa).

b) Variáveis de interesse quanto a qualidade da manutenção evolutiva. São elas:

números de ocorrências de erros reportados após a liberação da nova versão (quantitativa).

números de manutenções complementares àquelas realizadas por período (quantitativa).

números de chamados realizados pelos usuários após a nova versão, referentes às novas funcionalidades (funcionalidades antigas que deixaram de funcionar) (qualitativa).

Com base nos resultados da pesquisa foi possível apoiar construção da solução, a qual pode ser utilizada em um dos cenários (Cenário “A”). Este cenário foi escolhido devido a oportunidade da participação em uma evolução do sistema legado para avaliação posterior da proposta.

4. 1 PESQUISA DE CAMPO

A pesquisa de campo foi realizada, em caráter observatório em ambientes organizacionais de Empresas fornecedoras de soluções de TI, com sistemas legados em fase de planejamento de reconstrução (rejuvenescimento da aplicação), muitas vezes chamada de “projeto de migração”. Migrar para uma nova tecnologia requer tanto o conhecimento da aplicação atual, das regras de negócios e do contexto quando da criação da aplicação (contexto anterior), como o entendimento do novo contexto para a nova aplicação (contexto atual).

Os acadêmicos e estudiosos da área de engenharia de software publicam trabalhos sobre metodologias ágeis, produtividades, novos modelos de gestão para motivação das equipes, qualidade de software dentre outros. Para as empresas que possuem sistemas legados as equipes vivem no ambiente bem diverso dos demais grupos desenvolvedores.

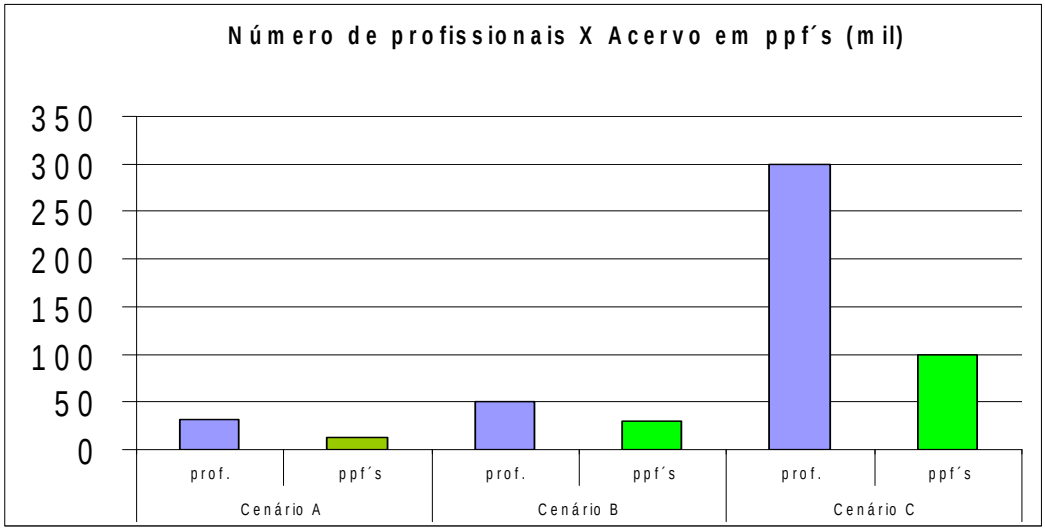
Para estas novas metodologias ágeis haverá uma maior probabilidade de sucesso em um projeto com grupos que possuam uma equipe motivada. SILVA e outros (2007) mostram informações a respeito da motivação, habilidades, treinamento, estilo de gerenciamento, resultados dos projetos e ainda os fatores que influenciaram os resultados. O crescimento dos novos modelos de gestão baseado nas pessoas, valorizando as habilidades dos gestores bem como da equipe, com ênfase aos fatores humanos do comportamento, tais como: companheirismo, talento, habilidades técnicas e comunicação interferem diretamente na produtividade da equipe. Os resultados alcançados demonstram a importância e a influência da motivação; 84% dos gerentes mostraram preocupação com a motivação da equipe. O gestor de TI convive, portanto, com um desafio constante: manter equipes motivadas, sistemas atualizados e clientes satisfeitos.

A constante evolução dos sistemas legados nos cenários de desenvolvimento e manutenção de sistemas cria um ambiente organizacional bem diversificado com equipes que trabalham no mesmo ambiente físico, porém em ambientes tecnológicos diferentes, realidades, prazos e principalmente pressões diferentes. Logo este ambiente possui contextos de evolução de software também diversificados.

A pesquisa foi realizada em 3 cenários, denominados neste estudo de “A”, “B” e “C”. As empresas são bem diferentes no tamanho, na estrutura organizacional, na formação de capital e ainda no segmento em que atuam. São eles: comércio varejista, prestação de serviço de software e setor financeiro, respectivamente. O ponto em comum destas empresas encontra-se no momento em que elas precisam remodelar seus sistemas legados.

A Figura 8 sintetiza a quantidade de profissionais destinados à manutenção dos sistemas legados das empresas pesquisadas. As empresas do cenário B e do cenário C não possuem o acervo dos sistemas quantificado. Os números de pontos por função são estimados pelos portes dos sistemas, enquanto a empresa do cenário A mantém o acervo atualizado em pontos por função (ppf) em torno de 12.000 com um total de aproximadamente 100 profissionais responsáveis pela sua manutenção e suporte.

Figura 8 – Quantitativo de acervo de aplicativos X número de profissionais



Fonte : Elaboração do autor

Através do porte dos sistemas se estima um total de 50.000 ppf para a Empresa “B” e um total de 300.000 ppf para a Empresa cenário “C” . Independente do tamanho do acervo das empresas, seja de pequeno, médio ou grande porte, as dificuldades no processo da manutenção evolutiva são muito semelhantes.

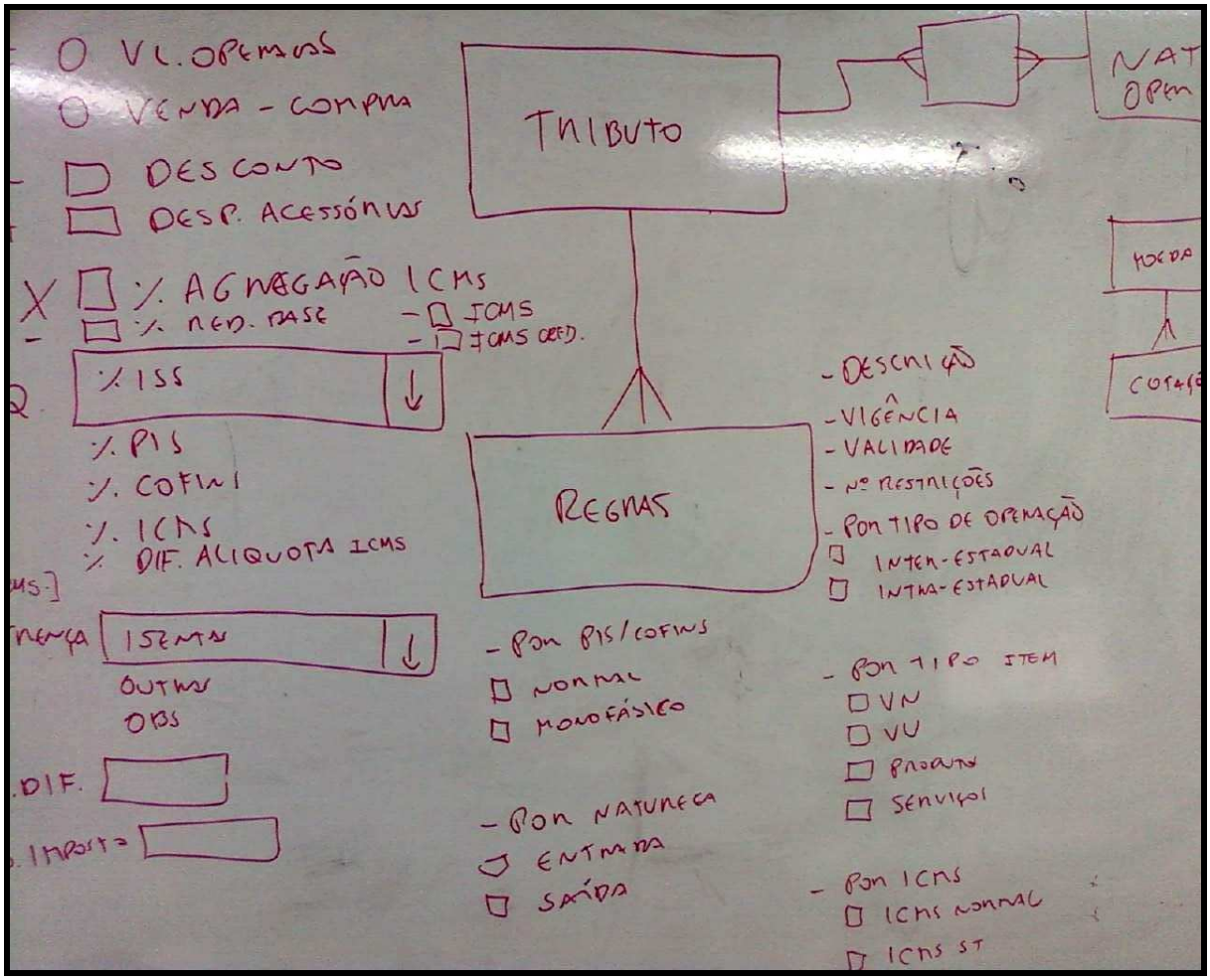
Durante a pesquisa de campo se observou que as equipes de desenvolvimento e manutenção de software utilizavam uma variedade de sistemas; seja para atendimento aos processos, controles de qualidades, acompanhamento de custos e ainda vários aplicativos para apoiar o modelo de gestão. No dia a dia se convive com a construção de novos produtos para atender aos clientes e usuários. Para ajudar nas suas tarefas diárias, para a criação de novos produtos ou ainda na evolução dos aplicativos existentes, os profissionais contam apenas com ferramentas capazes de auxiliar na operacionalização das tarefas e construção de novos aplicativos.

As discussões sobre as possíveis soluções para os novos requisitos são realizadas em grupo de profissionais com perfis heterogêneos tais como: especialistas em negócios da Empresa, representantes de usuários, analistas de sistemas, analistas de O&M (atualmente analista de negócios), utilizando-se na maioria das vezes papeis de rascunhos, quadro branco e pincel atômico. Em algumas ocasiões o acesso aos sistemas atuais é necessário para esclarecimentos das soluções.

Após as reuniões são elaboradas “atas” as quais formalizam o resultado das decisões, perdendo-se o conteúdo contextual, o teor real das discussões. As atas raramente são revisadas não sendo alertado quanto a qualquer situação adversa. As informações do contexto não são relatadas, logo, nenhuma informação é armazenada em relação às discussões, apenas as decisões são registradas.

A figura 9 mostra uma imagem (foto do quadro de uma sala de reunião) do resultado de uma discussão a qual envolvia mudanças de requisitos em aplicações tributárias de caráter legal, incluindo novos relacionamentos, (observa-se que é usual a prática do desenho das entidades e relacionamentos) com novas especificações documentadas apenas no quadro, durante o momento da discussão. Estas alterações obrigam a revisão da estrutura de dados da arquitetura do sistema atual, que necessariamente será realizada no contexto da elicitação dos requisitos.

Figura 9: Representação dos resultados da discussão de novos requisitos



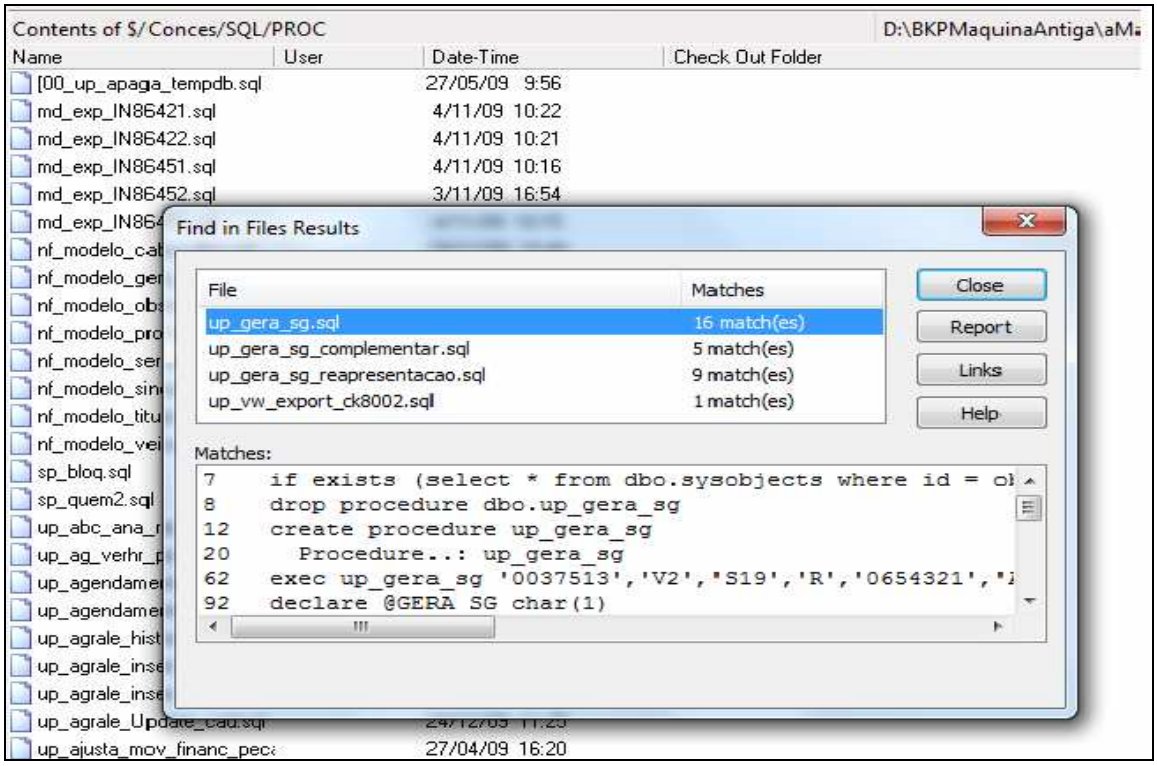
Fonte : Elaboração do autor

Quanto à análise dos impactos das soluções discutidas, os profissionais contam com o conhecimento tácito dos implementadores, usuários chaves e ainda os profissionais de suporte ao atendimento do produto.

Alguns utilitários de pesquisas de conteúdo são usados para conhecer o domínio de determinadas variáveis envolvidas no novo requisito. Na situação observada à discussão trata da criação de uma tabela que deve conter todas as variáveis para cálculo dos tributos legais a qual estará relacionada com uma tabela de regras que caracteriza o tributo. A análise de impacto em relação aos dados (atributos) já existentes em outras tabelas (entidades) não são analisados.

A figura 10 é um exemplo de uma ferramenta proprietária de busca de conteúdo no ambiente tecnológico *power builder,* onde se torna necessário recorrer às informações contidas nos códigos fontes das *procedure*s e *triggers* das bases de dados por total ausência de outro recurso. Estas ferramentas oferecem uma consulta de quais códigos fontes utilizam as variáveis necessárias ao escopo dos novos requisitos . Nestes casos os implementadores ou analistas devem acessar todos os códigos associados aos *triggers* (dezenas ou centenas) para obter a compreensão do tipo de uso das variáveis.

Figura 10. Tela da ferramenta oferecida pelo *Power builder*



Fonte : Elaboração do autor

A figura 11 ilustra outra situação, na qual um trecho do programa COBOL é exibido pela ferramenta dos fornecedores da linguagem. Este programa utiliza os *boks* (recurso usado para criar as bibliotecas das estruturas dos arquivos).

Figura 11. Trecho do programa COBOL

```
SELECT DO ARQUIVO DE TABELA GERAL DOS SISTEMAS.
ro Focus Server Express        V4.0 revision 000 18-May-09 17:32 Page   2
017.cbl
*COPY "/home/desenv/fontes/geral/gersltb.bok".
*----------------------------------------------------------------*
* SELECT DO ARQUIVO DE TABELA GERAL DOS SISTEMAS - GERFDTB.BOK*
*----------------------------------------------------------------*
   SELECT    ARQ-TABGERAL         ASSIGN      TO DISK
                          LOCK MODE IS AUTOMATIC
                             WITH LOCK ON RECORD
                          ORGANIZATION IS INDEXED
                          ACCESS MODE  IS DYNAMIC
                          RECORD KEY   IS TBCHAVE-01
                          FILE STATUS  IS ERCHAVE.


COPY "../estoque/srest07.bok".                *>
          SELECT DO ARQ.DE EMPRESA  SREST07.BOK            *
*----------------------------------------------------------------
   SELECT  EPARQ-EMPRE ASSIGN      TO DISK
                  LOCK MODE    IS AUTOMATIC
                     WITH LOCK ON RECORD
                  ORGANIZATION IS INDEXED
                  ACCESS MODE  IS DYNAMIC
                  RECORD KEY   IS EPCHAVE
                  ALTERNATE RECORD KEY IS EPCHAVE2
                            =  EPCOD-EMPRE
                              EPNOME
                            WITH DUPLICATES
                  ALTERNATE RECORD KEY IS EPCHAVE3
                            =  EPCOD-EMPRE
                              EPCOD-DEPOSITO
                            WITH DUPLICATES
                  FILE STATUS  IS ERCHAVE.
 73*----------------------------------------------------------------*
DATA                DIVISION.
FILE                SECTION.

* ARQ-TABGERAL (Book da tabela arq-tabgeral).


* FD DO ARQUIVO DE TABELA GERAL DOS SISTEMAS.
*COPY "/home/desenv/fontes/geral/gerfdtb.bok".
*----------------------------------------------------------------*
* FD DO ARQUIVO DE TABELA GERAL DOS SISTEMAS   -  GERFDTB.BOK    *
*-----------------------------------------+                *
* REGISTRO    | DESCRICAO                 |                *
```

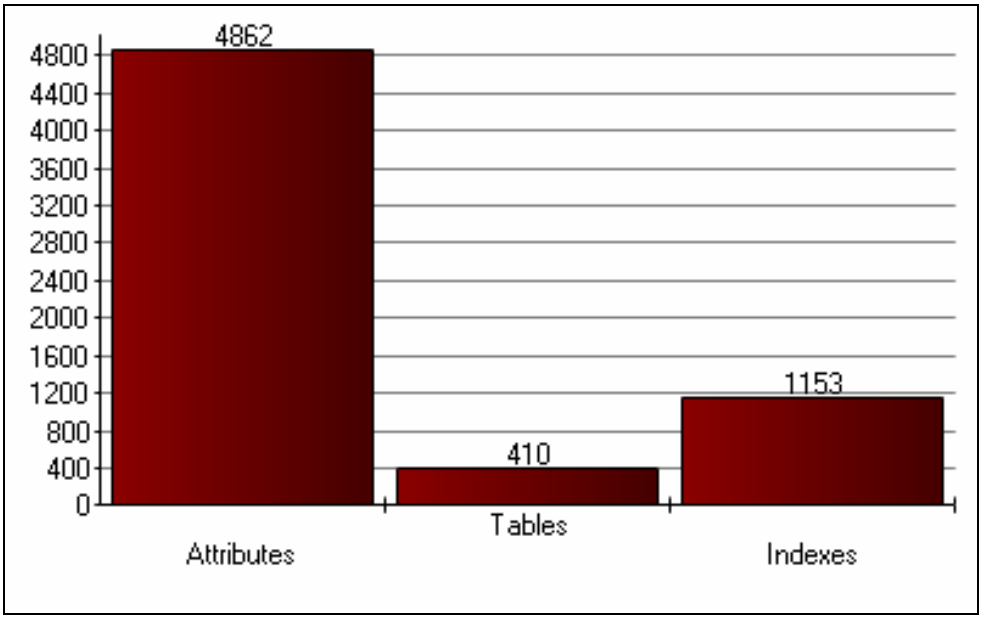
Fonte : Elaboração do autor

Alguns programas chegam a utilizar 50 boks. O que significa 50 módulos (definições de arquivos e de rotinas encapsuladas).

O profissional possui duas alternativas para montar o entendimento do processo: (i) anexar todos os *boks* em um único arquivo (opção de compilação) neste caso ficará com um programa de aproximadamente 5.000 kloc; ou (ii) acessar cada arquivo separadamente então iniciando o processo do entendimento e compreensão de cada funcionalidade. O tempo médio de uma tarefa desta natureza, segundo os profissionais entrevistados é de 5 a 7 dias .

As equipes contam com algumas ferramentas de mercado que acompanham o desenvolvimento das manutenções através do somatório de objetos; tais como: tabelas, atributos e indexadores (Figura 12). Estas quantidades de objetos sempre em unidades absolutas servem para calcular o esforço gasto até o momento. Porém não produzem informações para estimativas e planejamento, apenas acompanham a execução do projeto de evolução das aplicações não sendo capazes de calcular uma estimativa até a sua conclusão, devido a falta de outros elementos, tais como a complexidade de cada objeto, a equipe e experiência dos seus membros e outros.

Figura 12: Base de Conhecimento da Ferramenta geradora de software - GENEXUS



Fonte : Elaboração do autor

4. 2 PROJETO DA SOLUÇÃO

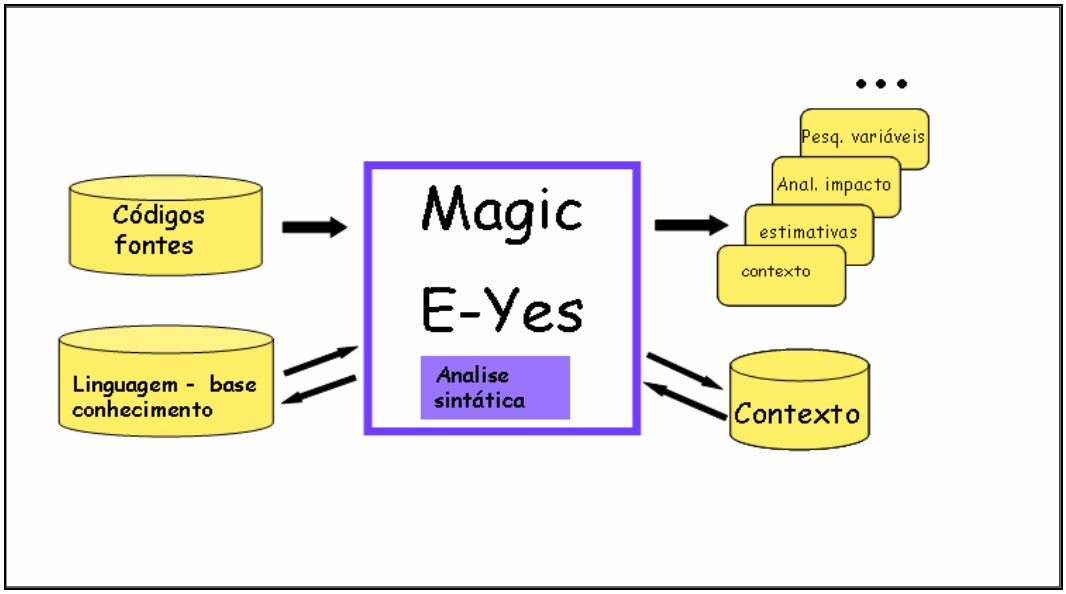
A solução proposta, denominada – Magic-Eyes objetiva apoiar o processo de manutenção evolutiva dos sistemas legados tendo como foco a etapa inicial de elicitação de requisitos. Permite criar um histórico das decisões quanto às implementações a partir dos conhecimentos contextuais da evolução do software.

Pelo fato dos levantamentos de requisitos de evolução de sistemas legados envolverem diversos níveis de usuários atuando em momentos distintos, estas atividades são sempre realizadas em grupo, logo é possível caracterizar a necessidade de suporte por um *sistema colaborativo,* assíncrono. Durante estas discussões são realizadas análises de impactos da criação de novas soluções, as quais podem envolver outros sistemas (integrações de sistemas) sendo necessário incorporar outros usuários com perfis diferentes de conhecimento, de interesses e até mesmo de prioridades distintos.

A ferramenta consiste em registrar o conhecimento procedural das discussões tornando-o disponível para ser utilizado nas discussões posteriores.

Além do suporte no processo decisório e análise de impacto, a solução automatizada deve prover informações dos conhecimentos contextuais e o conhecimento procedural, para auxiliar nas retomadas das discussões e manter uma base de dados atualizada para suportar os futuros projetos de rejuvenescimento do software. A Figura 13 ilustra um diagrama do projeto com as entradas e saídas que serão manipuladas pela ferramenta: os códigos fontes dos sistemas legados utilizados como matéria prima para análise de impacto, a base de conhecimento criada pela ferramenta para suportar as decisões e o histórico dos conhecimentos procedural.

Figura 13: Visão Geral da ferramenta MAGIC EYES



Fonte : Elaboração do autor

Dentre os conhecimentos do contexto da manutenção evolutiva dos sistemas legados os pontos chave residem na plataforma e no ambiente tecnológico no qual a aplicação encontra-se desenvolvida. Um fator diferencial para apoiar a compreensão e assegurar algumas informações necessárias à tomada de decisão é o conhecimento do sistema atual. Este conhecimento na maioria das vezes está contido apenas em códigos fontes. Por este motivo, para que a ferramenta possa apoiar totalmente o processo deve contemplar a realização de uma análise sintática, a fim de produzir informações em linguagem natural.

Como resultado da pesquisa de campo tornou-se possível elencar os seguintes requisitos funcionais, subdivididos por níveis hierárquicos:

a) Nível Estratégico

- Realizar comparativo entre cenários, utilizando-se cenários históricos para análise de impacto;
- Manter históricos das produtividades das manutenções evolutivas;
- Manter Indicadores de produtividade por projetos de evolução;
- Prover uma matriz de decisão para apoiar o processo decisório, baseado na teoria de multicritérios.

b) Nível Tático

- Analisar as rotinas e programas através das métricas de pontos por função;
- Calcular as estimativas de recursos para manutenção;
- Analisar o nível de complexidade da(s) rotina(s) a partir dos códigos fontes (os programas), quantificando e qualificando os mesmos;
- Analisar o nível de qualidade dos programas: medindo o percentual de (%) de lixo de código e a clareza das regras de negócios;
- Determinar a partir de um requisito de mudança (atributos ou entidades, classes a serem modificados) o tamanho do projeto, especificando o esforço em pontos por função para realização do mesmo.

c) Nível operacional

- Manter os cenários das discussões das tomadas de decisão, através da apresentação dos elementos do contexto;
- Criar uma definição da linguagem para prover uma análise sintática a fim de permitir um suporte na compreensão dos códigos fontes;
- Prover a análise da medição dos objetos não referenciados ("lixo tecnológico"), objetos ausentes, objetos reaproveitados, fronteiras entre sistemas;
- Manter um histórico dos contextos das elicitações de requisitos até a decisão final da evolução.

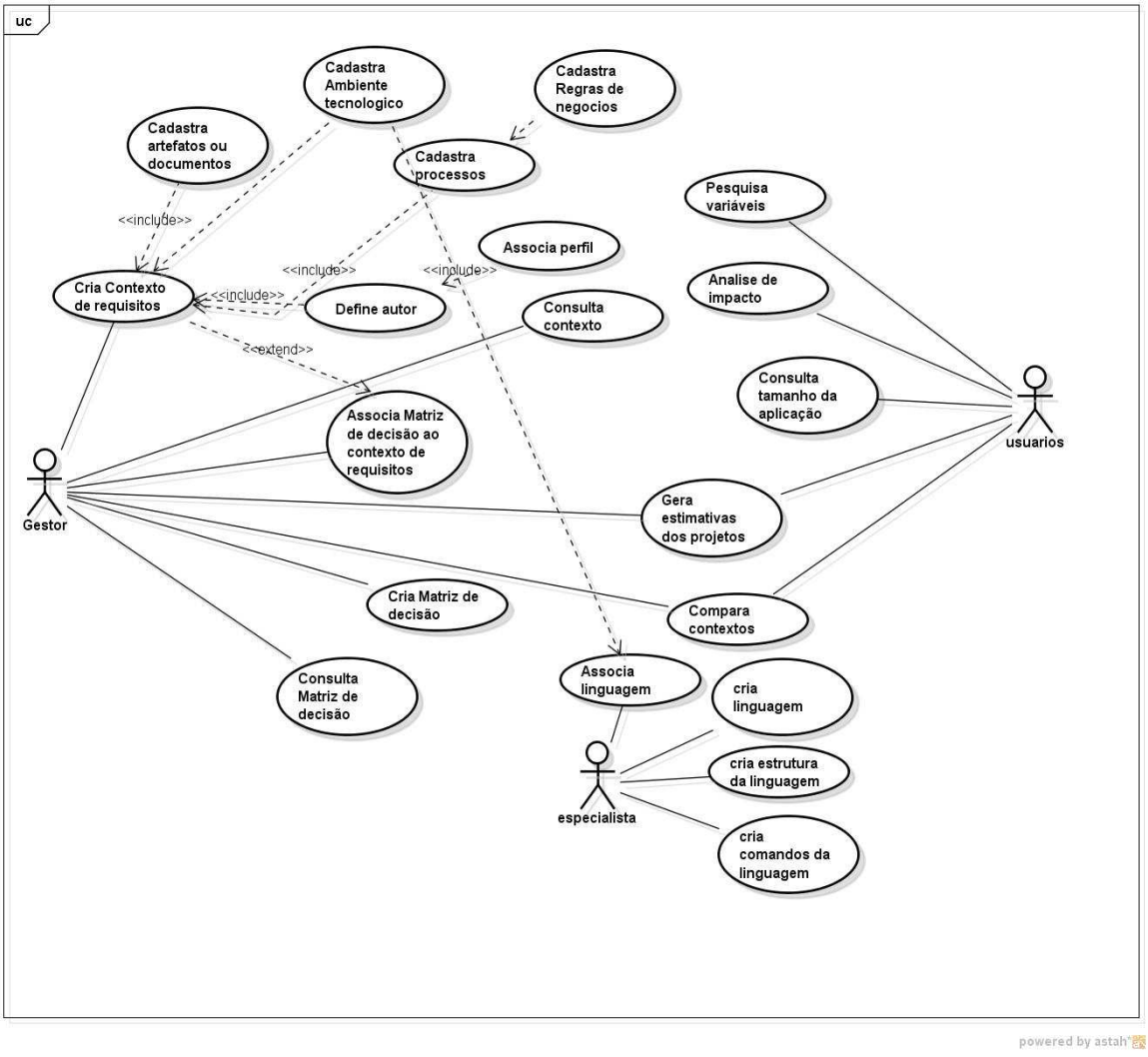
Quantos aos requisitos não funcionais, foram analisados :

- Portabilidade - a aplicação deve ser incorporada ao ambiente de desenvolvimento a fim de apoiar o processo de evolução dos *softwares*. Seguindo a tendência atual das arquiteturas das aplicações a solução deve ser desenvolvida no ambiente WEB.
- Segurança – a solução deve atender aos aspectos de segurança de acesso considerando os vários perfis de usuários: especialista, administrador e usuários gestores.
- Escalabilidade – a solução deverá atender à gestores e grupos de usuários com aplicações em manutenção.

- Desempenho – a solução deve observar os cuidados na implementação de uma aplicação WEB garantindo sua *performance*.

A partir da elicitação dos requisitos foi possível modelar os casos de usos (Figura 14) e elaborar o diagrama de classes (Figura 15).

Figura 14 : Casos de uso da ferramenta MAGIC EYES



Fonte : Elaboração do autor

Logo após a etapa de modelagem foram escolhidas as funcionalidades para implementação da primeira iteração da ferramenta, a fim de validar a proposta junto ao grupo usuário. São elas:

- Manter os cenários das discussões das tomadas de decisão através do registro dos elementos do contexto;
- Manter um histórico dos contextos das elicitações de requisitos até a decisão final da evolução;

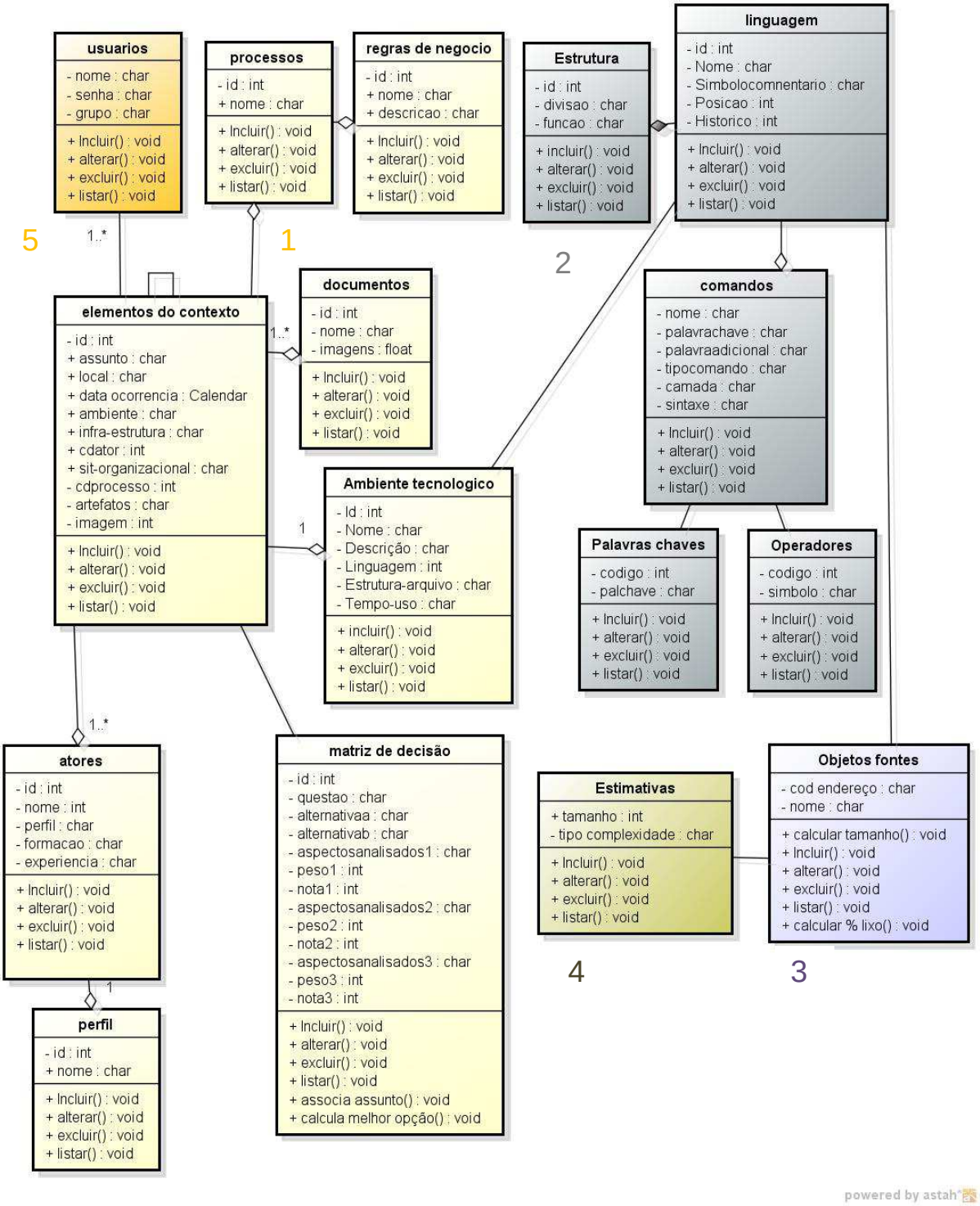
- Prover uma matriz de decisão para suportar o processo decisório, baseado na teoria de multicritérios;
- Criar uma definição de linguagem para prover uma analise sintática para apoiar a compreensão dos códigos fontes.

As demais funcionalidades para atender aos requisitos devem fazer parte da segunda iteração.

A figura 15 ilustra o diagrama de classe da solução modelada para apoiar a primeira iteração da solução. A modelagem atende aos requisitos:

1. Cadastro dos elementos do contexto. A classe denominada de “elementos do contexto” possui os atributos que caracterizam as ocorrências das discussões, das reuniões, tem como indexador o “assunto” que esta sendo tratado. Desta forma, é possível também criar relacionamentos recursivos e obter o histórico das discussões do assunto em vários momentos (cenários).
2. Definição da linguagem com seus comandos, palavras chaves e operadores.
3. Criação da interface para leitura de um objeto (código fonte) em determinada linguagem para compreensão do código.
4. Calculo das estimativas para cada solicitação dos objetos fontes incluindo os objetos.
5. A classe Usuários deverá atender aos requisitos de segurança que deverão ser tratados na segunda iteração.

Figura 15 : Diagrama de Classe da ferramenta MAGIC EYES



Fonte : Elaboração do autor

4. 3 AMBIENTE PARA CONSTRUÇÃO DA SOLUÇÃO

O protótipo da ferramenta foi construído considerando que a solução seria instalada como *plugin* do Eclipse. As características deste ambiente garantem a facilidade do desenvolvimento de aplicações multiplataformas e multilinguagens. A figura 16 mostra o desenho do protótipo no ambiente Eclipse.

Logo após a etapa da prototipação, no momento da construção realizou-se uma pesquisa sobre outros ambientes que garantissem uma maior produtividade da curva de aprendizado para a implementação no ambiente WEB.

Figura 16 – Tela inicial da aplicação MAGIC EYES(protótipo)

Fonte : Elaboração do autor

A seguir são apresentados os resultados dos estudos realizados entre as possibilidades de ambientes para desenvolvimento: Plataforma Eclipse, o framework Demoiselle e a novo ambiente Grails (Groovy e rails):

a) A plataforma Eclipse é escrita em linguagem Java e possui muitos kits de construção de plugins e exemplos de códigos. No ambiente Eclipse existe o conceito de *workspaces*, que são diretórios nos quais todas as suas configurações e preferências ficam armazenadas. O processo de atualização e evolução com adição de novos *plugins* pode ser realizado através das pesquisas em *sites* que possuem informações sobre as

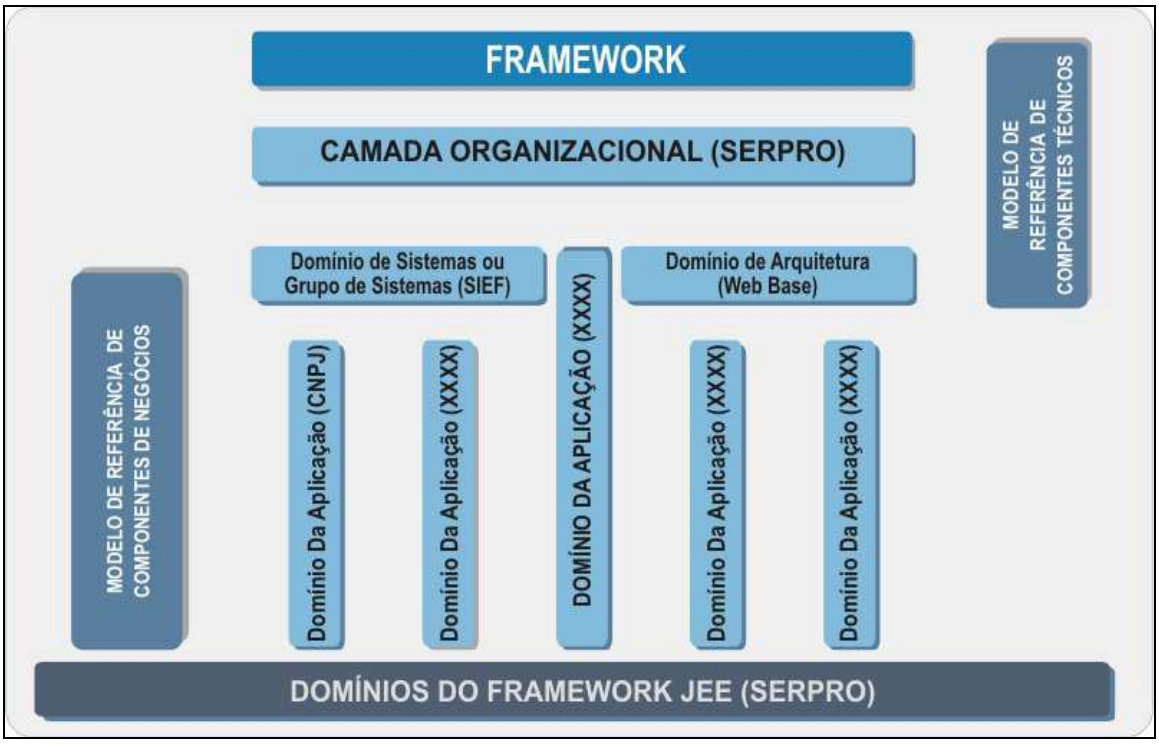
funcionalidades com os respectivos arquivos a serem baixados e atualizados. Esses sites são também chamados de ***Update Sites*** e fazem parte do ambiente onde existe uma lista que é exibida em uma janela, facilitando assim este processo (GAIGALAS, 2010).

A instalação do *plugin* no ambiente Eclipse conta com vários tutoriais que orientam personalizar este ambiente de acordo com a sua necessidade. Apesar de toda a documentação disponível o Eclipse para *web* não tem documentação satisfatória a produção de soluções. A documentação mais consistente disponível cria uma aplicação em ambiente para *Windows*. Logo a formação e o aprendizado seria mais lento.

b) Demoiselle *framework* – A ferramenta denominada Demoiselle[2] trata-se de um projeto de código livre explorado pelo governo federal do Brasil, através da Instituição Serpro - Serviço Federal de Processamento de Dados, responsável pelo programa de divulgação e treinamento do *framework* cuja finalidade é a padronização das soluções do governo além de produzir *softwares* com rapidez para o ambiente web. A ferramenta permite reduzir esforços de desenvolvimento, padronizando e reutilizando componentes relacionados aos aspectos de segurança, acesso a dados e comunicação com outro ambientes, tudo isso no ambiente colaborativo. Baseado no princípio de camadas o *framework*, utiliza a metodologia de Orientação a Objetos, a Plataforma Java Enterprise Edition (JEE), os recursos de Hibernate/JPA e o Java Server Faces (JSF). Além disso, encontra-se estruturado em domínios, característica que possibilita ao desenvolvedor focar nas camadas de visão (interface) negócios e persistência. (Figura 17).

[2] Demoiselle em homenagem a Santos Dumont que presenteou a todos, sem nada cobrar, todas as suas dezenas de invenções, cujo primeiro avião chama-o de Demoiselle

Figura 17 : Estrutura de domínio da ferramenta



Fonte : Botello (2010)

c) GRAILS - Um *framework* open-source, inspirado na linguagem Ruby on Rails. É voltado para aplicações web, e segue o paradigma de "codificação por convenção", proporcionando um ambiente ágil de desenvolvimento e eliminando muitos detalhes de configuração para o desenvolvedor. Utilizam as tecnologias Hibernate, Spring, Log4J, HSQL engine, Quartz Scheduling Framework, Sitemesh, Apache Ant. e implementa um conjunto de ferramentas de construção das camadas de visão e controle na metodologia de orientação a objeto. Os aplicativos são escritos na linguagem Groovy; uma linguagem dinâmica para máquina virtual muito semelhante à linguagem Java (compatível sintaticamente).

A vantagem em se utilizar o Grails reside na padronização da codificação, reutilização dos códigos, redução de trabalho na troca de contexto e ainda pelo fato de possui *templates* do tipo CRUD (ações de criar, ler, atualizar e consultar as tabelas do banco de dados) o que garante a agilidade e produtividade nas funcionalidades básicas da aplicação.

A comunidade GRAILS possui fórum de discussões, disponibiliza códigos fontes das aplicações criadas, mantém documentação atualizada para *download*, criando um ambiente colaborativo de desenvolvimento.

Após o comparativo entre as ferramentas foi escolhido o *framework* GRAILS por apresentar a maior produtividade de aprendizado para a WEB.

4. 4 A CONSTRUÇÃO DA SOLUÇÃO MAGIC-EYES

Com a finalidade de avaliar a solução foi desenvolvida a primeira iteração, baseada no protótipo construído. A Figura 18 mostra as funcionalidades selecionadas. Este desenvolvimento priorizou as funcionalidades relacionadas aos requisitos dos registros do conhecimento contextuais e a compreensão das decisões das soluções. Para acompanhar as discussões é necessário o registro dos elementos do contexto das manutenções evolutivas dos sistemas legados (o conhecimento procedural). São eles:

- Cadastrar os atores responsáveis pela discussão;
- Cadastrar o ambiente tecnológico;
- Cadastrar o processo e as regras de negócios relacionadas com a discussão, e os documentos disponíveis;
- Cadastrar os assuntos, referenciados entre si, criando um histórico das discussões, eliminando as interrupções comuns em reuniões descontinuas.

Figura 18 : Protótipo da ferramenta MAGIC-EYES (tela com menu principal)

Fonte : Elaboração do autor

Para a melhoria da qualidade nas discussões a ferramenta possibilita não apenas criar uma matriz de decisão, baseada nos conceitos do método decisório de multicritérios, mas também associá-la aos assuntos da discussão. A criação não é apenas passiva, mas interativa. Para cada nova opinião, é possível uma nova reavaliação dos pesos (prioridade) e das notas (grau de importância do tema). O aplicativo calcula e aponta os melhores resultados, e a alternativa sugerida. O registro de cada simulação poderá ser armazenado para futuras consultas e comparação com as soluções adotadas. A figura 20, mostra o menu principal da ferramenta MAGIC-EYES, no ambiente WEB, criado no GRAILS com customizações.

A ferramenta permite apoiar a fase de elicitação de requisitos através do acompanhamento das discussões, registrar o contexto no qual elas ocorrem e

reproduzir o cenário das discussões anteriores. Assim é possível acompanhar as evoluções com registros dos elementos do contexto; os assuntos, o ambiente tecnológico, a situação que provocou a necessidade, os atores com seus perfis; analistas, programadores, usuários, projetistas, arquitetos de *softwares*; e ainda associar as regras de negócios cadastradas, aos processos, registrar os artefatos (os documentos) utilizados durante a discussão, as opiniões dos atores relacioná-las com possíveis ações e ainda registrar as imagens (opcionais).

Figura 19. Menu principal da ferramenta MAGIC-EYES

Fonte : Elaboração do autor

5 A AVALIAÇÃO DA FERRAMENTA MAGIC-EYES

Com o objetivo de avaliar a solução desenvolvida durante o trabalho de pesquisa, foi realizada uma investigação, na empresa denominada para esta pesquisa como “A”. A oportunidade de utilização da ferramenta de gestão ocorreu durante o processo de evolução de um sistema legado do tipo SIG – Sistema Integrado de Gestão para um segmento específico da área do comércio. Os encontros(4) foram realizados durante as reuniões de elicitação de requisitos, com duração média de 3 horas. Não aconteceu treinamento na ferramenta devido à simplicidade do ambiente WEB.

O grupo constituído por desenvolvedores com formação de análise de sistemas, com experiência mínima de cinco anos, um gestor com experiência de 10 anos em manutenção do sistema legado, conhecedor da modelagem do sistema. A equipe que executa a manutenção atualmente no sistema é totalmente diferente da equipe do início do desenvolvimento, e aos poucos os desenvolvedores aprendem as regras de negócios, os padrões definidos, a estrutura dos objetos e quais objetos podem ser reutilizados.

O ambiente de manutenção do sistema legado consiste de uma plataforma cliente-servidor, com um gerenciador de banco de dados da empresa Microsoft o Sql-server. A camada de negócios foi construída ao longo de 16 anos, utiliza uma linguagem visual - Powerbuilder para a criação da camada de interface. Existe uma documentação dos padrões de nomenclatura, mas não há documentação dos padrões de programação. Segundo os analistas, periodicamente são feitas melhorias de desempenho principalmente de *tunning* na base de dados. Isso se deve ao fato de que além dos dados dos clientes crescerem em grande escala, a inclusão de novas funcionalidades pode comprometer o desempenho global do sistema por uma análise de impacto deficiente. O nível de defeitos é considerado aceitável pelos clientes, na medida em que novas funcionalidades são adicionadas a cada nova versão do SIG.

A disponibilidade da ferramenta MAGIC-EYES no ambiente WEB facilitou a utilização da mesma, pelo grupo durante a realização das reuniões de forma interativa. O grupo utilizou a solução priorizando as funcionalidades relacionadas aos registros dos conhecimentos contextuais e a compreensão das decisões das

soluções.

Primeiro realizou-se os cadastros dos elementos necessários para o registro do contexto no cenário da manutenção. São eles: cadastro dos atores responsáveis pela discussão, cadastro do ambiente tecnológico, cadastro dos processos e as regras de negócios relacionadas com a discussão, bem como os documentos disponíveis. A Figura 20 mostra o cadastro dos atores. Os demais cadastros seguem o padrão de navegação.

Figura 20 – Cadastro dos atores

Fonte : Elaboração do autor

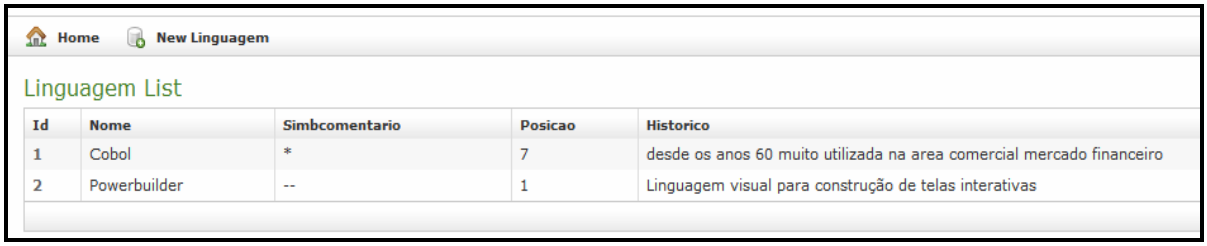
A reunião foi iniciada com a informação dos elementos do contexto, equivalente ao tema da reunião. A partir deste registro o sistema mantém um histórico das discussões, com o registro dos conhecimentos compartilhados de forma estruturada. As informações do assunto, ambiente tecnológico, atores, processos, regras de negócios associadas ao processo, artefatos utilizados durante a reunião, local e data compõem o registro dos elementos do contexto. A figura 21 mostra uma consulta dos elementos do contexto do tema da reunião.

Figura 21 – Elementos do contexto

Fonte : Elaboração do autor

Durante o primeiro dia do uso da ferramenta observa-se que a função da pesquisa de variáveis, apoiou a definição do escopo da manutenção, o que permitiu aos gestores estimativas mais confiáveis. A pesquisa foi realizada tendo como argumento o nome da variável não sendo necessário nenhum conhecimento técnico sobre a linguagem dos códigos fontes. Para definir a linguagem existe um cadastro da mesma linguagem (figura 22) bem como dos seus comandos que devem ser realizado por um conhecedor da linguagem antes do uso das reuniões de elicitação dos requisitos. O papel destas funções é permitir uma maior abstração dos códigos fontes para a realização do planejamento das evoluções, independente do conhecimento das linguagens dos sistemas legados.

Figura 22 – Cadastro das linguagens



Fonte : Elaboração do autor

Para realizar a pesquisa são necessárias as informações sobre a infraestrutura do sistema local (diretório) onde se encontram os códigos fontes. A

figura 23 mostra a tela da funcionalidade. Caso exista algum código crítico pode-se realizar a pesquisa de forma mais detalhada pelo programa ou *trigger* .

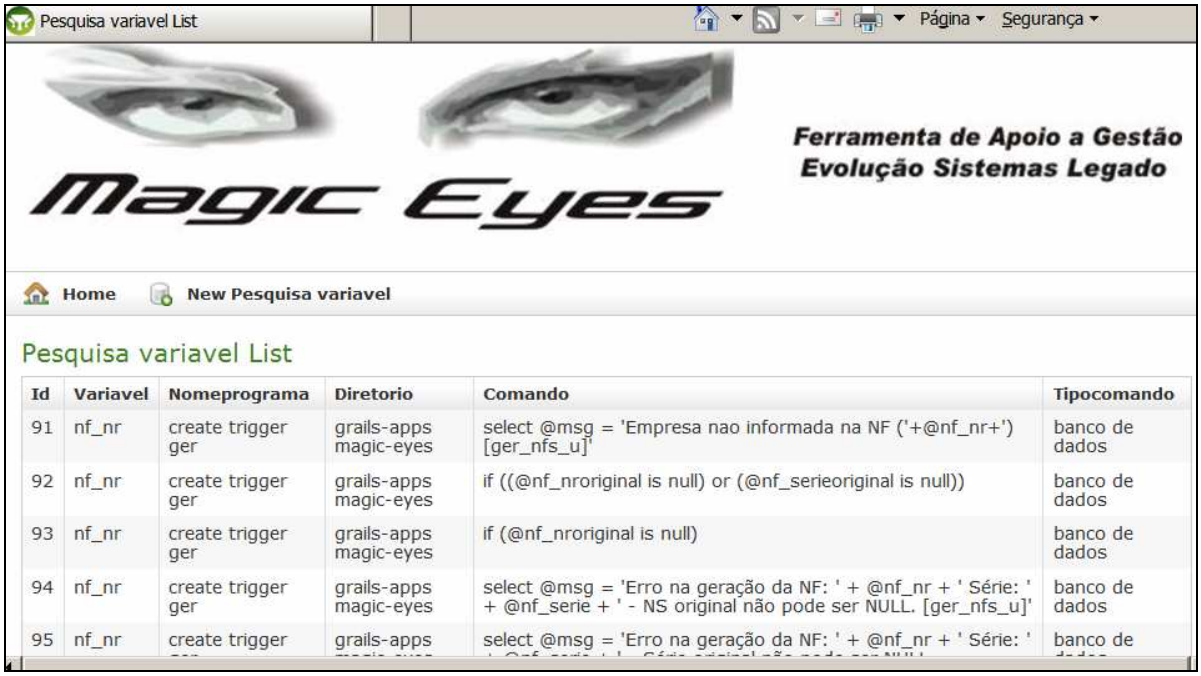
Figura 23. Pesquisa de variáveis

Fonte : Elaboração do autor

Como resultado da função de pesquisa variável a ferramenta mostra todas as linhas do código fonte onde se utiliza a variável. Neste momento é possível avaliar o impacto da alteração, compartilhar as preocupações e discutir com os usuários os possíveis prazos para a alteração (Figura 24).

Além da pesquisa das variáveis é possível realizar uma busca geral pelo diretório para averiguar o impacto das alterações em determinada tabela ou até mesmo as variáveis. Como resultado desta pesquisa o sistema mostra a quantidade de objetos que serão afetados pela mudança, o tamanho do objeto e o percentual de linhas de comentários no código fonte. Muitas vezes este percentual chega a 40 % do tamanho do código fonte, onde é possível acreditar que o código possui muitas manutenções corretivas ou evolutivas.

Figura 24. Resultado da pesquisa de variáveis



Fonte : Elaboração do autor

A figura 25 mostra o resultado da análise de impacto para a discussão da evolução da alteração. Neste ponto da avaliação foi descoberto que o escopo da evolução envolve outros diretórios, logo serão necessárias outras análises.

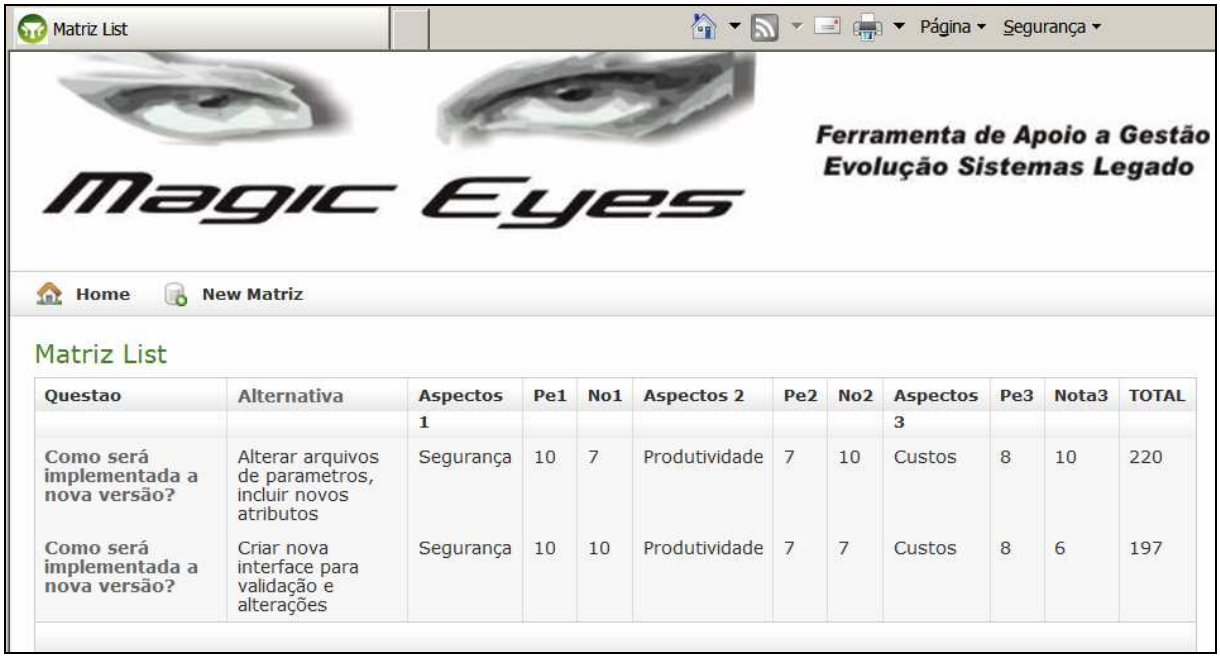
Figura 25 – Análise de Impacto

Fonte : Elaboração do autor

A partir das discussões da solução para a evolução da ferramenta é possível registrar uma matriz de decisão, baseada nos conceitos do método decisório de multicritérios e associá-la aos assuntos tratados. Neste caso, mais um elemento do contexto é adicionado ao conhecimento procedural. A cada nova alternativa de

solução a matriz é atualizada produzindo os resultados que serão analisados e, interativamente para cada nova opinião, é possível uma nova reavaliação. Para cada nova análise das alternativas de solução, os pesos (prioridade) e as notas (grau de importância do tema) podem ser redefinidos. A figura 26 mostra uma matriz decisória no primeiro momento da discussão.

Figura 26 Matriz decisória inicial

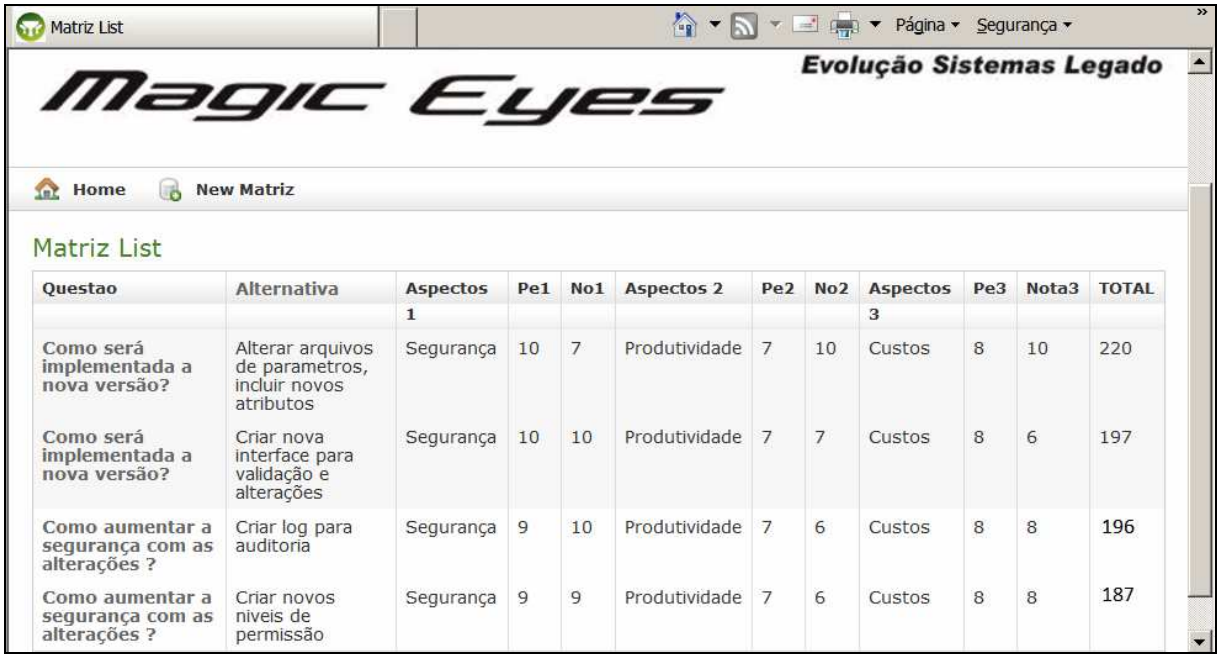


Questao	Alternativa	Aspectos 1	Pe1	No1	Aspectos 2	Pe2	No2	Aspectos 3	Pe3	Nota3	TOTAL
Como será implementada a nova versão?	Alterar arquivos de parametros, incluir novos atributos	Segurança	10	7	Produtividade	7	10	Custos	8	10	220
Como será implementada a nova versão?	Criar nova interface para validação e alterações	Segurança	10	10	Produtividade	7	7	Custos	8	6	197

Fonte : Elaboração do autor

Após cada discussão é possível visualizar uma matriz de decisão contendo de forma clara e objetiva o resumo das várias alternativas e assuntos comentados, associados aos seus atores, os quais são responsáveis pelas sugestões das soluções (Figura 27). Nesta abordagem os conhecimentos além de documentados são compartilhados por todos os envolvidos, respeitando-se as diferentes opiniões e estratégias para resolver as questões.

Figura 27 Matriz decisória - segundo momento



Questao	Alternativa	Aspectos 1	Pe1	No1	Aspectos 2	Pe2	No2	Aspectos 3	Pe3	Nota3	TOTAL
Como será implementada a nova versão?	Alterar arquivos de parametros, incluir novos atributos	Segurança	10	7	Produtividade	7	10	Custos	8	10	220
Como será implementada a nova versão?	Criar nova interface para validação e alterações	Segurança	10	10	Produtividade	7	7	Custos	8	6	197
Como aumentar a segurança com as alterações ?	Criar log para auditoria	Segurança	9	10	Produtividade	7	6	Custos	8	8	196
Como aumentar a segurança com as alterações ?	Criar novos niveis de permissão	Segurança	9	9	Produtividade	7	6	Custos	8	8	187

Fonte : Elaboração do autor

Através dos mecanismos de navegabilidade, a localização dos contextos anteriores (estudo das discussões), classificação, busca dos assuntos anteriores, participantes da discussão e os atores, observa-se o benefício das informações contextuais: as discussões são retomadas quase de imediato quando localizadas as informações, principalmente quando visualizadas as imagens que representam as últimas anotações.

O registro das informações de forma estruturada é outro ponto que agrega valor ao resultado das discussões. Através dele, cria-se um processo de condução das reuniões sem deixar de registrar informações importantes.

Durante a avaliação da solução, fica evidente (através de questões) a necessidade de maiores informações sobre estimativas, análise de impacto quanto aos prazos e até mesmo aos riscos em relação às decisões de manutenções.

Outro ponto é a compreensão dos códigos fontes. Apesar da função “pesquisa variável” apoiar a compreensão e o escopo da evolução, seria interessante aumentar as funções de analise sintáticas, como por exemplo, os comandos de entradas e saídas para saber se os repositórios de dados são temporários ou não, se as Views, ainda são usadas ou se já existe nos relacionamentos. O lixo nos códigos fontes dificultou muito a compreensão. É preciso criar uma funcionalidade para identificação destes trechos.

Os pontos fracos da ferramenta são: a necessidade da ter uma base de dados histórica, e a necessidade de ter uma forma rápida para recuperação dos dados anteriores. Outro ponto que foi evidenciado e merece modificações é a rotina de adicionar documentos. Seria interessante, incluir possibilidades de incluir fotos, rascunhos de discussão ou ainda interface com tablets.

6 CONCLUSÕES

A proposta do estudo da evolução dos sistemas legados aparece como uma oportunidade de atender à falta de metodologias, de publicações e de ferramentas dedicadas a este segmento da engenharia de *software*.

Desde as décadas de 70 e 80, quando surgiram as leis de Lehman, a academia não criou oportunidades para a utilização de espaços suficientemente adequados para uma abordagem mais profunda de contextualização de cenários de ocorrências das manutenções evolutivas dos sistemas legados. Muitos estudos abordaram metodologias ágeis, produtividades de desenvolvimento, novas plataformas e muitas opções de *framework*. No entanto, a etapa de manutenção do ciclo de desenvolvimento de *software*, continua com pouca atenção.

Esta pesquisa explorou os conceitos do ciclo de vida da engenharia de software, com a finalidade da busca do conhecimento pela fase da evolução, especificamente dos sistemas legados. A pesquisa de campo realizada permitiu um maior conhecimento dos cenários de ocorrências das manutenções evolutivas dos sistemas legados o que possibilitou a definição dos requisitos funcionais da ferramenta.

Ao longo do desenvolvimento do estudo, foram adicionados aspectos sobre o conhecimento contextual no ambiente de manutenção de sistemas legados.Estes elementos apoiaram e enriqueceram o entendimento da importância das informações contextuais como valor agregado à análise de impacto da cadeia do processo decisório destas manutenções.

O caráter exploratório da pesquisa permitiu a construção da ferramenta em ambiente WEB, e uma posterior avaliação em um dos cenários de manutenção de sistemas legados com foco na etapa inicial da evolução: análise de impacto e decisão dos novos requisitos, como estabelecido inicialmente.

Durante a realização da avaliação da ferramenta MAGIC-EYES foi possível vivenciar as dificuldades da equipe de manutenção dos sistemas legados quando da elicitação dos requisitos. São muitas variáveis a serem discutidas, ambientes de negócios bastante modificados em relação à concepção inicial da solução, falta de documentação dos sistemas, principalmente destes históricos. O conhecimento tácito envolvido nas discussões é o diferencial.

O uso de uma ferramenta automatizada durante as discussões para definição dos requisitos das evoluções sistêmicas possui mais que um papel de registrar e guardar as informações. Atua também, como fator disciplinador, pois estrutura o desenrolar das discussões, torna obrigatório alguns aspectos importantes no processo decisório, quando da busca de alternativas para a solução.

Outro aspecto relevante construído ao longo da implementação, foi a exibição dos históricos das reuniões anteriores para permitir uma retomada aos assuntos com uma maior brevidade. Nas discussões com o uso da ferramenta a retomada aos assuntos era quase imediato, o que garantia uma melhor produtividade na compreensão dos temas abordados. Com a breve retomada é possível diminuir o tempo e as redundâncias das discussões bastante comuns em tais atividades, bem como uma maior precisão nas soluções adotadas.

A disponibilidade dos históricos contextuais contendo informações sobre as ações e os atores respectivamente, permitiram uma maior segurança em questões e decisões semelhantes, afinal os atores no contexto são, em geral, os especialistas nos assuntos.

Vale considerar que o nível de abstração da compreensão dos códigos fontes envolvidos nos cenários das discussões permitiu decisões mais rápidas, sem interrupções para estudo do escopo do projeto a ser implementado.

A pesquisa evidenciou também a necessidade de um maior investimento em mecanismos que atuem nas práticas para produção das evoluções dos sistemas legados, ou até mesmo nos novos sistemas que venham a substituí-los. Não basta possuir um sistema no qual se registra informações mesmo de forma interativa. O contexto da elicitação de requisitos é algo muito dinâmico, com inúmeras variáveis e informações para produção do conhecimento e da compreensão dos sistemas, o que requer outros estudos relacionados ao comportamento dos indivíduos em atividades de grupos.

A maior dificuldade encontrada no decorrer da pesquisa foi em relação ao grande volume de assuntos relacionados à evolução de sistemas, e o esforço necessário para manter o foco no objeto de estudo. Quando se trata de atividades em grupo, existe um número de variáveis dos fatores humanos, como compreensão, comunicação, representação da informação, conhecimento tácito dentre outros, os quais requerem atenção e modificam a cada momento os resultados dos estudos.

Como trabalho para pesquisas futuras cabe citar aqueles relacionados à produtividade da equipe, como por exemplo, o fator de produtividade da equipe quando utilizada a linguagem natural (obtida pela reengenharia dos códigos) em contrapartida quando utilizado apenas o código fonte. Outra pesquisa de fundamental importância está relacionada à qualidade da decisão em função do escopo do projeto e do tamanho do projeto.

A proposta de continuidade da pesquisa tem foco no desenvolvimento de demais funcionalidades relacionadas ao aumento da compreensão dos sistemas legados, calculo de estimativas baseado na métrica de pontos por função. Com a avaliação das novas funcionalidades (segunda iteração) será avaliar a ferramenta em ambiente controlado, e medir o impacto no suporte a gestão dos processos da evolução do software.

Além das propostas anteriores, cabe avaliar em pesquisas futuras a efetividade de utilização de outras formas para representar e explorar as informações relacionadas ao contexto do sistema em evolução.

REFERÊNCIAS

ARTHUR, Lowell Jay. **Melhorando a Qualidade do software:** Um guia Completo para o TQM. Tradução de Flavio Eduardo Frony Morgado. Rio de Janeiro: Infobook S.A., 1994.

BANA E COSTA, Carlos A. **Como melhorar a tomada de decisão nas organizações?:** A Metodologia MACBETH. Fortaleza, 2008. Disponível em: <HTTP://web.ist.util.pt/carlosbana/bin/help/papers/AMetodologiaMACBETHFaculdadeChristus23_3_06.pdf> . Acesso em: 31 jan. 2010.

BORGES, M.R.S., BREZILLON, Patrick, PINO, J.A., POMEROL, Jean Charles**. Groupware System Design and the Context Concept**, Universidade Federal do Rio de Janeiro, Brazil, 2007.

BRÉZILLON, Patrick, ARAUJO, Renata M. **Using Context to Manage Collaborative Software Development Knowledge .** Revue d'Intelligence Artificielle v. X . France, 2002.

_____. **Reinforcing Shared Context to Improve Collaboration**. LIP6, Université Pierre et Marie Curie, Paris, France. 2005.

BRÉZILLON, Patrick., POMEROL, Jean Charles. **Contextual Knowledge and Proceduralized Context.** AAAI Technical report W8-99-14. France, 1999.

CHIKOFSKY Elliot. J. CROSS II James. H. **Reverse engineering and design recovery: A taxonomy.** IEEE Computer Society, vol 7, USA, January 1990, p.13–17.

CUNHA, José Adson, THOMAZ, João Pedro, MOURA, Hermano Perrelli. **As conferencias de decisão na resolução de conflitos de projetos de software.** 2005. Disponível em: <HTTP://www.cin.ufpe.br/hermano/download/> . Acesso em: 31 jan. 2010.

DAVID, José Maria N., ROSA, Marcio G.P., SANTORO, Flávia M. **.** BORGES, M.R.S. Avaliando o Risco da Divergência e da Perda de Contexto em Groupware. In: SIMPÓSIO BRASILEIRO DE SISTEMAS COLABORATIVO, 2008, Vila Velha-ES. **Anais...** Vila Velha-ES, 2008.

DOHI, Tadashi, ETO, Hiroyuhi. **Determining the Optimal Software Rejuvenation Schedule via Semi-Markov Decision Process.** Journal of Computer Science 2 (6): 528-535, Hiroshima University, Higashi-Hiroshima, Japão. 2006.

ECLIPSE. **What's New in Helios**: Eclipse Communication Framework (ECF). Disponível em: <http://www.eclipse.org/>. Acesso em: 20 abr. 2010.

FERREIRA, Aurélio Buarque de Holanda. **Novo Dicionário Aurélio da Língua Portuguesa.** Conforme a nova ortografia. 4. ed. Curitiba: Positivo, 2009.

FOWLER, Martin. UML Distilled is an excellent way to get started with UML. In: **Fact . Refactoring SQL Applications**. Stephane Faroult; Pascal L'Hermite Publisher: O'Reilly, 2008.

FOWLER, M., BECK, K., BRANT, J., OPDYKE, W. F. e don Roberts. **Refactoring: Improving the Design of Existing Code**. 1 ed. Addison-Wesley Pub Co; 1999.

GAIGALAS, Alexandre. **Tutorial de Instalação do Eclipse**. Disponível em: <http://alexandre.gaigalas.net/>. Acesso em: 20 abr. 2010.

IEEE Standard Glossary of Software Engineering Terminology 610.12-1990. In IEEE Standards Software Engineering, 1999 Edition, Volume One: Customer and Terminology Standards. IEEE Press, 1999.

LEITE, Jair C, **Blog do Professor e pesquisador do DIMAp/UFRN, desde 1993**. Disponível em: <HTTP://engenhariadesoftware.blogspot.com/>. Acesso em: jan. 2010.

LEITE, Julio Cesar Sampaio do. **Os vários contextos da engenharia de software.** Disponível em: <http://livrodeengenhariaderequisitos.blogspot.com/>. Acesso em: fev. 2010.

LAUDON, Kenneth C., LAUDON, Jane P. **Sistemas de informações gerenciais**. 7. ed. São Paulo: Pearson Prentice Hall, 2007.

LEHMAN, M.. **Laws of Software Evolution Revisited**. EWSPT 1996, LNCS 1149, Springer Verlag, 1997. p. 108-124.

MAZOLLA, Bruno Vitorio. **Engenharia de software**. Livro eletrônico. [2008].

NUNES, Vanessa T., SANTORO, Flávia M., BORGES, M.R.S. **Context in Knowledge Intensive Collaborative Work.** Núcleo de Computação Eletrônica, UFRJ, Brazil, 2006.

PARNAS, David L. .Software Aging. Proc. 16th Int'l Conference on Software Engineering (ICSE-16), Sorrento, Italy, 1994, pp. 279-287.

PESCADA, Carlos Alberto. **As fases do Processo de Tomada de Decisão**. Disponível em: <www.administradores.com.br/artigos/as_fases_do_processo_de decisão/20959>. Acesso em: 31 jan. 2009.

PFLEEGER, Shari Lawrence. **Engenharia de software. Teoria e pratica**. Trad. Dino Franklin; Revisão técnica Ana Regina Cavalcanti da Rocha, 2. ed. São Paulo: Prentice Hall, 2004.

PIGOSKI, Thomas M. **Practical Software Maintenance**, Ed. John Wiley & Sons. 1997. (PAPERBACK).

PRESSMAN, ROGER S. **Engenharia de Software**; tradução Rosangela Delloso Penteado, revisão técnica Fernão Stella R. Germano, José Carlos Maldonato, Paulo Cesar Masicro. 6. ed. São Paulo: Mc graw Hill, 2006.

PROJECT MANAGEMENT INSTITUTE, PMI. **PMBOK® Guide - A guide to the Project Management Body of Knowledge.** 4 ed. Local Pennsylvania : Four Campus Boulevard, 2008.

REZENDE, Denis Alcides**. Engenharia de software e sistemas de informação.** 3. ed. ver. e amu. Rio de Janeiro: Brasport, 2005.

ROSA, Marcio G.P., SANTORO, Flávia M. **.** BORGES, M.R.S. **A conceptual framework for analyzing the use of context in groupware.** Rio de Janeiro. 2003.

SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE (21: João Pessoa : 2007) Anais .SILVA,Danielle R.D. SANTANA, André F., TEDESCO, Patricia R., RAMALHO, Hermano P. **Um Retrato da Gestão de Pessoas em Projetos de Software: A Visão do Gerente vs. A do Desenvolvedor.** 2007. Publicado em: 17/04/2007.

SEACORD, Robert C., PLAKOSH, Daniel, LEWIS, Grace A. **Modernizing Legacy Systems**: Software Technologies, Engineering Processes, and Business Practices", Addison-Wesley, 2003. (paperback)

SOMMERVILLE, Ian. **Engenharia de software**.tradução:Selma Shin Shimizu Melnikoff, Reginaldo Arakaki, Edilson de Andrade Barbosa; 8 ed. São Paulo : Pearson Addison_Wesley, 2007.

SECOND WORKSHOP ON REFACTORING TOOLS (WRT'08) IN CONJUNCTION WITH OOPSLA CONFERENCE, October 19-23, 2008 Nashville, Tennessee. Disponível em: http://refactoring.info/WRT08. Acesso em: 12 dez. 2009.

TI NEWS Jornal eletrônico. **3CON e ATERAS apresentam solução automática para modernização de legado.** Disponível em

http://www.tinews.com.br/news/2010/12/06/3con-e-ateras-apresentam-solucao-automatica-para-modernizacao-de-legado/. Acesso em 06 dez 2010.

ULRICH, William M. **Legacy Systems: Transformation Strategies**. Prentice-Hall PTR, 2002.

VAIDYANATHAN, Kalyanaraman, TRIVEDI, Kishor S.. **A comprehensive Model for Software Rejuvenation**. Fellow, IEEE. IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 2, NO. 2, APRIL-JUNE 2005.

APÊNDICE A - QUESTIONÁRIO DA PESQUISA

Questionário elaborado para acompanhar o processo de pesquisa de campo, realizada nas empresas denominadas A, B e C. A coleta foi realizada a partir de entrevistas e observações das reuniões de planejamento do projeto de rejuvenescimento do software.

1. PERFIL DA EQUIPE TECNICA

QUANTIDADE DE PROFISSIONAIS : _____

EXPERIENCIA da EQUIPE:➔	QTDE ☐ EM MANUTENÇÕES ANOS QTDE ☐NO APLICATIVO ANOS QTDE ☐NO SEGMENTO DO NEGÓCIO. ANOS

2. SISTEMAS LEGADO :

a. Qual o papel do(s) sistema(s) para a organização?

b. Qual o tamanho do sistema?

c. Qual a tecnologia utilizada pelo sistema? Há quanto tempo?

d. Quais as demais tecnologias adotadas no mesmo ambiente, para outros sistemas ?

3. MANUTENÇÕES EVOLUTIVAS

a. Qual a periodicidade das manutenções evolutivas?

b. As manutenções evolutivas são realizadas em conjunto com manutenções adaptativas ou corretivas?

c. Existe o histórico das manutenções evolutivas ? Como são registradas?

d. Existe uma metodologia de desenvolvimento? Em caso afirmativo, Nesta metodologia prever artefatos especifico para as manutenções evolutivas, ou são os mesmos ?

e. Os profissionais envolvidos na manutenção evolutiva participam da análise inicial da solução?

4. QUAIS DOS RECURSOS ABAIXOS SÃO UTILIZADOS NO PROCESSO DA REALIZAÇÃO DA MANUTENÇÃO EVOLUTIVA?

☐ FERRAMENTA AUTOMATIZADA PARA DELIMITAÇÃO DO ESCOPO. QUAL?

☐ METODOLOGIA PARA MEDIR O TAMANHO DA MANUTENÇÃO. QUAL?

☐ METODO PARA ANÁLISE DE APOIO A DECISÃO DAS MELHORES ALTERNATIVAS DE IMPLEMENTAÇÃO? QUAL?

☐ PLANEJAMENTO PARA REALIZAÇÃO DAS TAREFAS.

☐ METODO OU FERRAMENTA AUTOMATIZADA PARA ESTIMATIVAS DE TEMPO?

APÊNDICE B - A CONSTRUÇÃO DO PROTÓTIPO DA FERRAMENTA

O protótipo da ferramenta realizado no ambiente ECLIPSE com as suas funcionalidades.

A funcionalidade da análise sintática contida na ferramenta faz todo o diferencial para o sistema legado onde os códigos fontes são muitos extensos (número de linhas de códigos em media - kloc é de 3500 linhas por programas) algumas procedures na camada de banco de dados chegam a 2500 linhas.

Baseados nos princípios das linguagens formais foram criadas funções para montar uma base de conhecimento da linguagem, (Figura 28) a partir da qual é possível realizar as análises sintáticas, produzindo assim informações de forma mais natural .

Figura 28 – Protótipo –Cadastro da linguagem

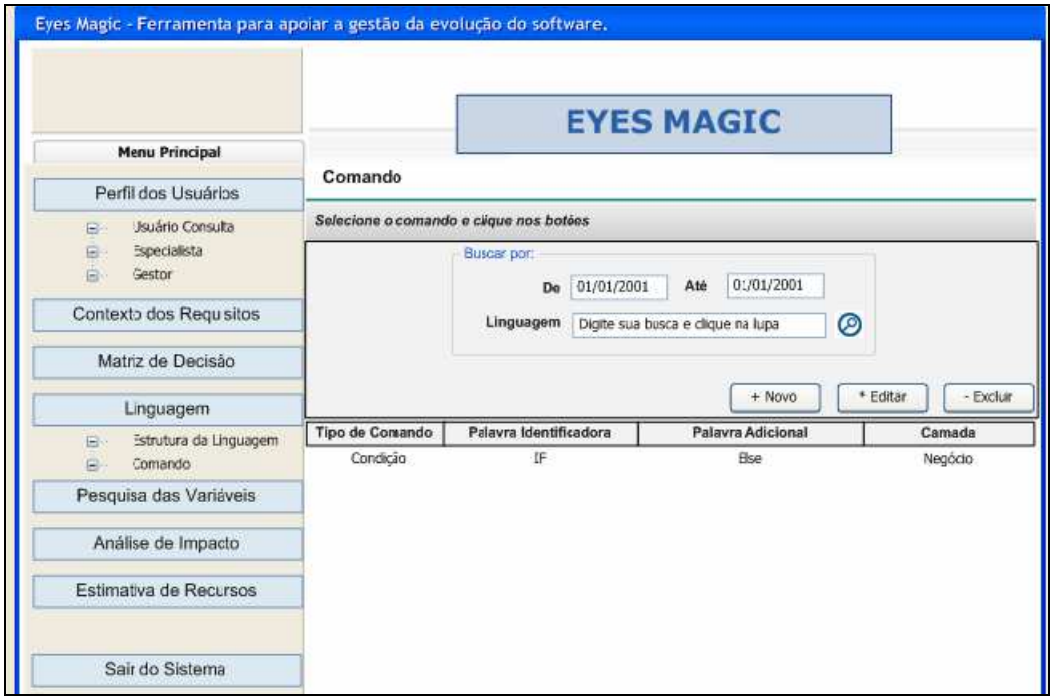
Fonte : Elaboração do autor

Para cada linguagem a ser explorada deve-se informar as estruturas da linguagem, seus comandos, (Figura 29) símbolos que identificam comentários, palavras chaves para identificação dos comandos e demais informações que compõem a sintaxe da linguagem. Através da base de conhecimento é possível desenvolver novas funcionalidades de compreensão, realizar pesquisas de variáveis,

conhecer tamanho de programas, separar nos programas o % de lixo, comandos não utilizados, laços mal construídos etc.

A consulta dos comandos já incluídos mostra a potencialidade da linguagem assim como valida às versões e a completude das informações. Apoia também o nível de abstração.

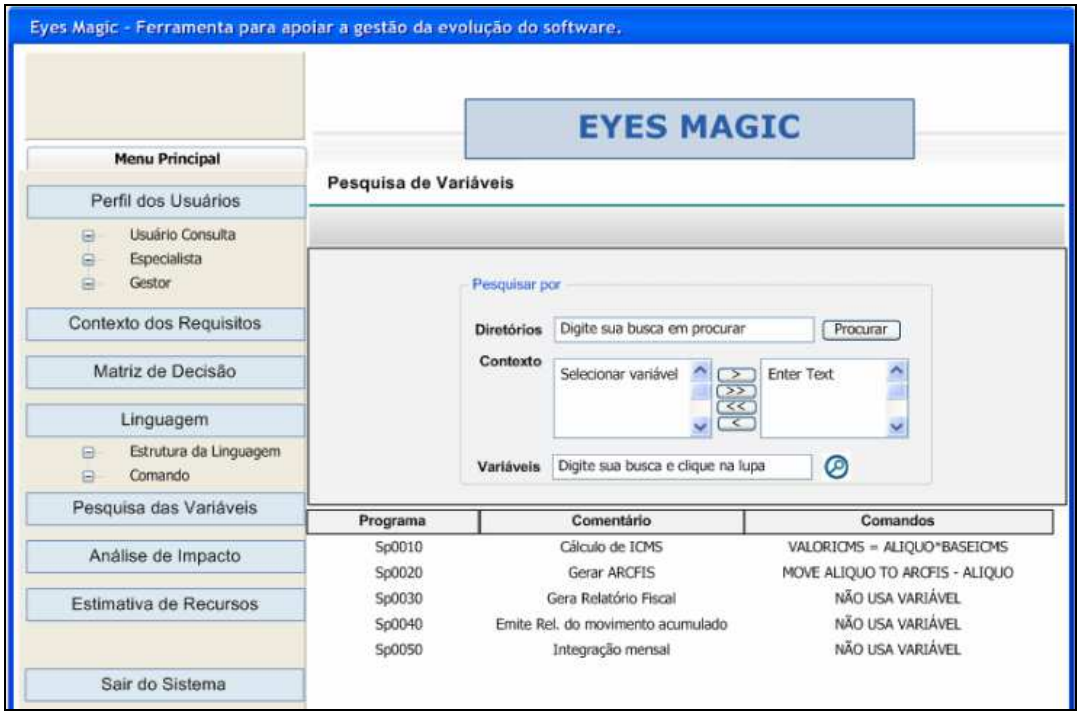
Figura 29 Protótipo Construção da linguagem



Fonte : Elaboração do autor

A funcionalidade da pesquisa de variáveis; permite buscar e selecionar nos diretórios e códigos fontes informados (programas, *procedures*, e outros) aqueles códigos onde as variáveis são manipuladas. De posse destas informações (Figura 30) é possível aumentar o grau de conhecimento e auxiliar nas tomadas decisão**.**

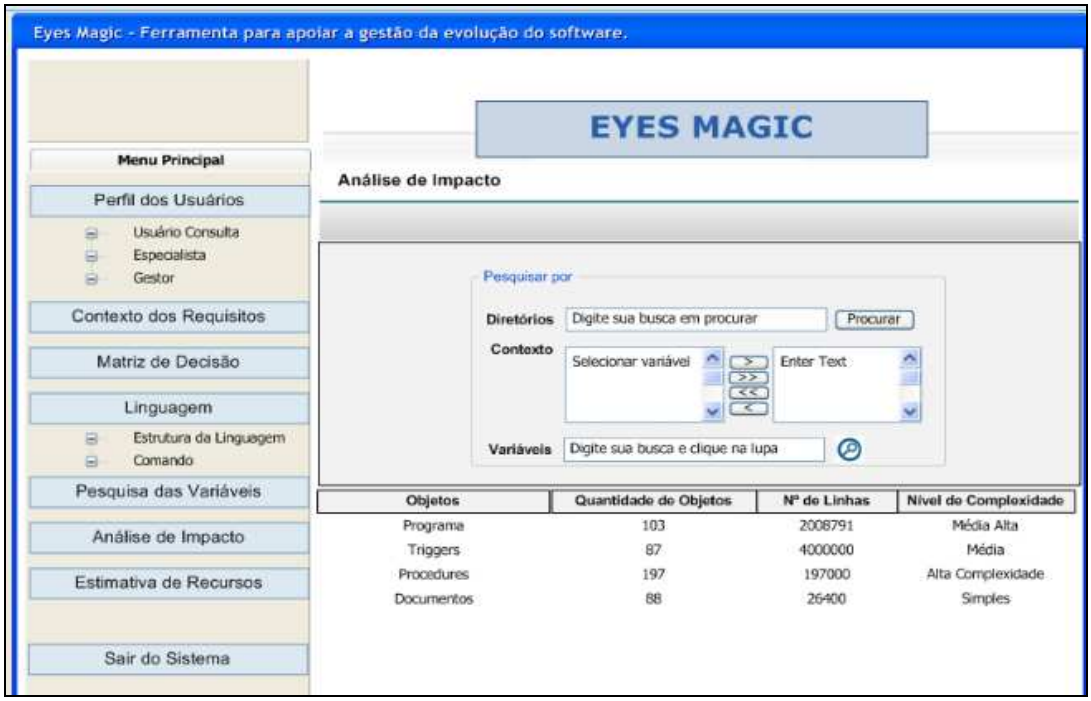
Figura 30 – Protótipo- pesquisa de variáveis



Fonte : Elaboração do autor

A funcionalidade da análise de impacto (Figura 31) alem de detectar "onde" e "como" determinados atributos são utilizados, determina o universo dos códigos fontes a serem alterados.

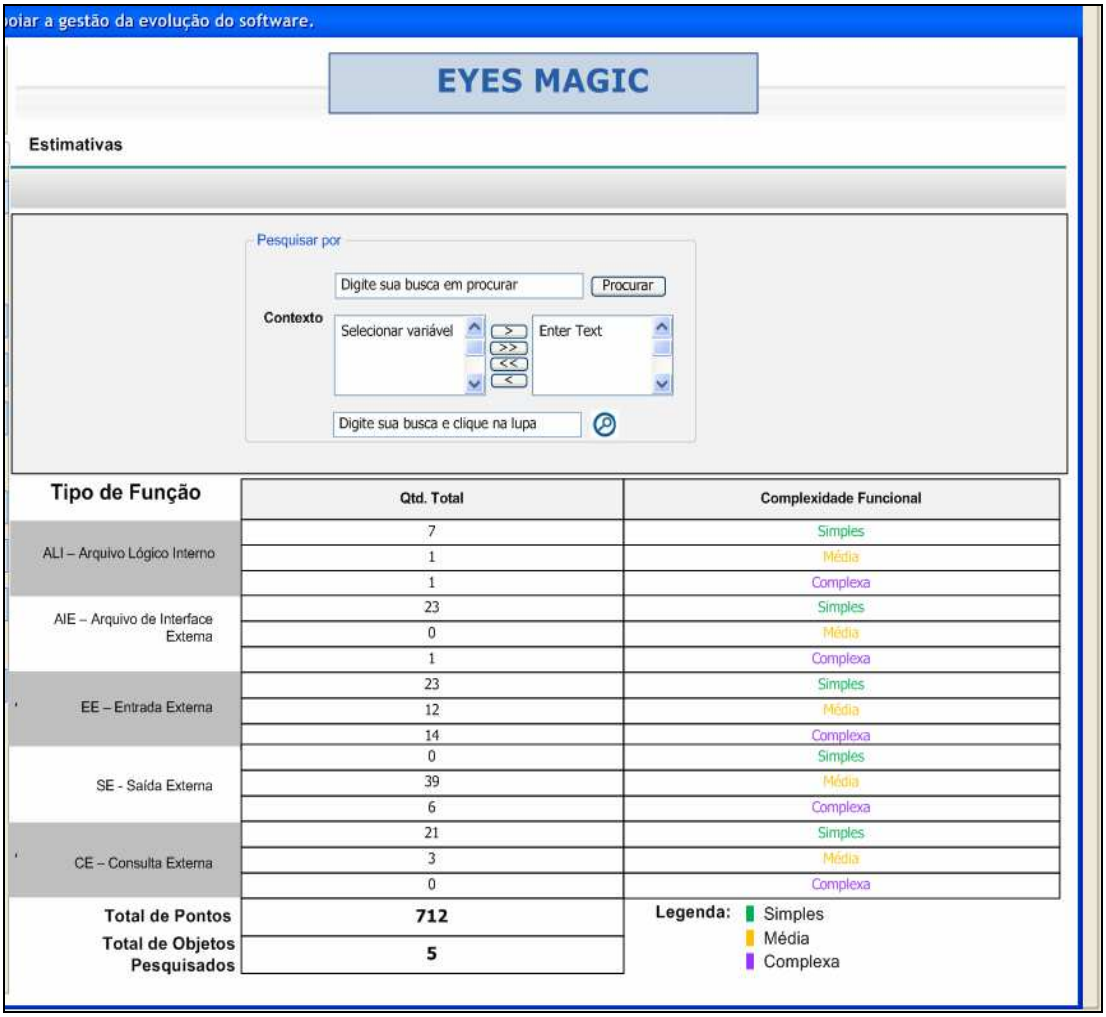
Figura 31 – Protótipo análise de impacto



Fonte : Elaboração do autor

Para estimativa de recursos é possível classificar e agrupar os objetos medindo o nível de complexidade de acordo com a métrica de pontos por função. Com a quantidade de objetos e o grau de complexidade de cada um se obtém o esforço necessário a manutenção (Figura 32).

Figura 32- Protótipo -Estimativa de esforço de manutenção



Tipo de Função	Qtd. Total	Complexidade Funcional
ALI – Arquivo Lógico Interno	7	Simples
	1	Média
	1	Complexa
AIE – Arquivo de Interface Externa	23	Simples
	0	Média
	1	Complexa
EE – Entrada Externa	23	Simples
	12	Média
	14	Complexa
SE - Saída Externa	0	Simples
	39	Média
	6	Complexa
CE – Consulta Externa	21	Simples
	3	Média
	0	Complexa
Total de Pontos	**712**	**Legenda:** Simples, Média, Complexa
Total de Objetos Pesquisados	**5**	

Fonte : Elaboração do autor

GLOSSARIO

Boks – Biblioteca de código fonte, utilizada no ambiente de desenvolvimento COBOL.

Feedback – Processo de informação tem dois objetivos: fazer com que uma ação positiva seja repetida.

Ferramentas CASE – Computer Aided Software Engineering ; ferramenta de automação de processos de construção de software.

Framework - No desenvolvimento do software, um framework ou arcabouço é uma estrutura de suporte definida em que um outro projeto de software pode ser organizado e desenvolvido, utilizando os mesmos padrões já estabelecidos.

Interfaces - Definição que estabelece a fronteira de comunicação entre dois componentes de software. Ou ainda, entre homem-maquina.

Loops - Terminologia na area de informática para definir um laço criado, involuntariamente, nos programas.

Mainframe – Categoria que abarca os computadores de maior porte; usados para processamento empresarial de grande escala.

Middleware - Software que oferece suporte a interação entre sistemas a exemplo de desenvolvimento de conteúdo e aplicativos para TV Digital, entre elas a possibilidade desses conteúdos serem exibidos nos mais diferentes sistemas de recepção, independente do fabricante e tipo de receptor (TV, celular, PDAs etc.).

Outsourcing - Terceirização de serviços que não fazem parte das principais competências de uma organização.

Performance – Desempenho das funcionalidades dos sistemas.

Plugin – Herdado do conceito de plug in do hardware, consiste em programas que podem ser adicionados a outro para estender as funcionalidades do hospedeiro.

Pseudocódigo – Tipo de algoritmo que utiliza uma linguagem flexível, intermediaria, entre a linguagem natural e a linguagem de programação

Protótipo – Versão inicial de um sistema de software usado para demonstrar conceitos, experimentar opções de projeto , permitindo o maior conhecimento sobre o problema e as possíveis soluções.

KLOC - Kylo Lines of code milhão de linhas de código – unidade de medida para calculo de produtividade de desenvolvimento ou manutenção de software.

Sistemas legados – Sistemas que existem ha muito tempo e continuam sendo usados. Muito importante para os negócios da Empresa.