Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Werner Tavares Heilbrun

Roteiro de avaliação da degradação do *software* baseado na análise de requisitos não funcionais

São Paulo
2012

Werner Tavares Heilbrun

Roteiro de avaliação da degradação do *software* baseado na análise de requisitos não funcionais

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, como parte dos requisitos para obtenção do título do título de Mestre em Engenharia da Computação.

Data de aprovação ____/____/_____

_____

Prof. Dr. Reginaldo Arakaki (Orientador)
IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Membros da Banca Examinadora:

Prof. Dr. Reginaldo Arakaki (Orientador)
IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Prof. Dra. Gabriela María Cabel Barbarán (Membro)
UNIFIEO – Centro Universitário FIEO

Prof. Dr. Jorge Luis Risco Becerra (Membro)
Escola Politécnica da Universidade de São Paulo

Werner Tavares Heilbrun

Roteiro de avaliação da degradação do *software* baseado na análise de requisitos não funcionais

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, como parte dos requisitos para obtenção do título do título de Mestre em Engenharia da Computação.

Área de Concentração: Engenharia de *Software*.

Orientador: Prof. Dr. Reginaldo Arakaki

São Paulo
2012

Ficha Catalográfica
Elaborada pelo Departamento de Acervo e Informação Tecnológica – DAIT
do Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT

H466r **Heilbrun, Werner Tavares**

Roteiro de avaliação da degradação do software baseado na análise de requisitos não funcionais. / Werner Tavares Heilbrun. São Paulo, 2012.
82p.

Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Área de concentração: Engenharia de Software Software.

Orientador: Prof. Dr. Reginaldo Arakaki

1. Degradação de software 2. Arquitetura de software 3. Ciclo de vida de software 4. Manutenção de software 5. Tese I. Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Coordenadoria de Ensino Tecnológico II. Título

12-61 CDU 004.416(043)

RESUMO

A manutenção de um *software*, tanto corretiva quanto evolutiva, tem impacto em sua arquitetura, o que pode levar à sua degradação, resultando em não atendimento de requisitos funcionais e não funcionais previstos em sua concepção. Essa degradação tem sido investigada de diversas formas, porém, muitas vezes são desconsiderados aspectos de qualidade e funcionalidade descritos no documento de arquitetura dessa ferramenta, seja inicial ou de posterior revisão. Nesta perspectiva, este trabalho apresenta um roteiro para diagnosticar a degradação da arquitetura de um *software* a partir da definição de indicadores e das necessidades dos interessados, permitindo o acompanhamento da fase de manutenção e a detecção de comportamentos que possam vir a comprometer a capacidade de satisfazê-las.

Palavras-chave: Requisitos não funcionais, arquitetura de *software*, degradação de *software*, manutenção de *software*, indicadores para ciclo de vida

ABSTRACT

Method to evaluate the degradation of the software based non-functional requirements

Software maintenance, to correct bugs or to evolve, impacts in its architecture. Usually the process leads to software architecture degradation which results in not achiving functional and non functional requirements as planned. Software degradation is studied in many ways that put quality and functional aspects aside. These are addressed in the coneception or in any review of its achitecture. This work presents an approach to diagnose the software architecture degradation using measures that reflects the needs of all stakeholders. The use of this kind of measure allow the detection of behaviours that compromise the ability of the software to match the expected objective.

Keywords: Non-functional requirements, *Software* architecture, *software* degradation, Arquitetura de *software*, degradação de *software*, *software* maintenance, life cycle indicators

Lista de ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

ATAM	*Architecture Tradeoff Analysis Method*
ABNT	Associação Brasileira de Normas Técnicas
HTML	*HiperText Markup Language*
GQM	*Goal / Question / Metric*
ISO	*International Organization for Standardization*
IEC	*International Electrotechnical Commission*
kLoC	*kilo Line of Code*
NBR	Norma Brasileira
RM-ODP	*Reference Model - Open Distributed Processing*
RNF	Requisito Não Funcional
UP	*Unified Process*
XML	*eXtensible Markup Language*

Sumário

1 INTRODUÇÃO

1.1 Motivação

Decidir sobre a continuidade de um *software* ou sua substituição tem se tornado cada vez mais importante nas organizações (QUEIROZ, 2009). Isso porquem algumas características como capacidade de mudança, capacidade de extensão e capacidade de teste podem ser comprometidas durante a fase de manutenção (BRCINA, 2009). Os engenheiros de *software* e sua equipe devem, pois, acompanhar essas manutenções observando o custo e o impacto na qualidade do *software*.

As freqüentes mudanças durante o período de utilização de um *software* levam comumente à perda de qualidade da arquitetura (BRCINA, 2009). Com isso tende-se a observar a diminuição de sua qualidade geral e consequente deterioração (MEY, 2006; PRESSMAN, 2009). Os processos usuais de reengenharia, substituição ou re-implementação do produto são normalmente utilizados quando já se possui a percepção de que alguma ação deve ser executada para a modernização de um sistema (QUEIROZ 2009).

Outra abordagem para a definição do momento em que se deve substituir um *software* é a realização de análises de custo beneficio. As análises quantitativas tipicamente associam os critérios a valores e utilizam técnicas financeiras, como retorno sobre o investimento, custo de oportunidade e valor presente para determinar a melhor alternativa (RICHMOND, 2006).

Para analisar o tempo de vida do *software*, Richmond (2006) apresenta um modelo matemático empírico associado à percepção de que se deve substituí-lo, uma vez que a comparação dos custos atuais é feita com o valor de renovação do *software*. Tal percepção normalmente ocorre quando o prazo para o desenvolvimento de uma nova solução compromete o negócio.

Brcina, Bode e Riebisch (2009) propõem um processo para manter a capacidade de evolução do *software* durante a fase de manutenção, baseado no

desenvolvimento orientado à funcionalidade e no compartilhamento de conhecimento entre os desenvolvedores (BRCINA, 2009), sendo esta uma restrição a sua aplicação.

O fenômeno conhecido como *software aging* concentra-se prioritariamente na análise do aumento de falhas ao longo do tempo (GROTTKE, 2008). As falhas observadas são: o número de erros inesperados e a exaustão dos recursos do sistema operacional. Porém, isoladamente, essa análise isolada não permite afirmar que a troca do *software* é a solução, às vezes a atualização do servidor (necessidade comum com o aumento de usuários em sistemas web, por exemplo) ou a operação de reinício do sistema operacional é suficiente para resolver o problema.

Já Queiroz et. al. (2009) propõem um processo para acompanhar a deterioração de sistemas de informação baseado na análise de métricas que respondem a três questões (QUEIROZ et al., 2009, p.50):

- “Q1- A manutenabilidade do sistema está sendo comprometida?”
- “Q2 - Os custos do sistema são elementos ofensores à continuidade de sua operação?”
- “Q3 - Qual o nível de satisfação dos usuários com relação à utilização do sistema?”

As questões foram definidas com base na pesquisa feita pelo autor e validadas por cinco gestores ligados à área de tecnologia da informação com mais de dez anos de experiência. No trabalho são propostas medidas para respondê-las, porém, as questões e as medidas são genéricas e podem não corresponder às necessidades de todos os tipos de *software*.

1.2 Objetivo

O objetivo deste trabalho é apresentar um roteiro para determinar indicadores relevantes ao *software* estudado, de modo a auxiliar no diagnóstico de sua degradação. A meta é prover uma visão da evolução da arquitetura desse produto, desde sua implantação inicial, seu período de atendimento pleno às necessidades e sua degradação. Trata-se de um procedimento que deve ser acessível aos diversos níveis da organização, permitindo o acompanhamento de um *software* ao longo do tempo.

1.3 Contribuições

A principal contribuição deste trabalho em relação ao estado da arte é a utilização de indicadores que tenham relevância para o negócio e sejam de fácil compreensão para os envolvidos com o *software*. Serviram de referência os trabalhos de Grottke (2008) e Queiroz (2009), que abordam a deterioração do *software* de diferentes formas.

Outra contribuição se refere à combinação das visões sobre o *software*, com a proposta de escolha dos indicadores baseada na sua relevância no documento de arquitetura.

1.4 Método de trabalho

No processo de elaboração deste estudo, foram realizadas as seguintes atividades:

- Análise de referências

Levantamento e estudo de referências sobre a fase de manutenção do *software* e os processos para acompanhar o ciclo de vida. Foram analisados o processo proposto por Queiroz (2009), os indicadores sugeridos por Grottke (2008) no

fenômeno de *software aging*, o modelo de Richmond (2006) e o processo de Brcina, Bode e Riebisch (2009). O modelo de referência para arquitetura RM-ODP (*Reference Model - Open Distributed Processing*) foi abordado por auxiliar na análise do documento de arquitetura. A ferramenta de avaliação da arquitetura ATAM (*Architecture Tradeoff Analysis Method*) foi utilizada para validar a seleção dos indicadores para o diagnóstico.

- Levantamento das características do *software* em estudo

Com o documento de arquitetura foi realizado um levantamento dos objetivos do *software*. A partir dos objetivos e apoiando-se na norma de qualidade ISO/IEC 9126-1, foram então selecionadas as características a serem acompanhadas ao longo da vida dessa ferramenta.

- Definição das medidas, procedimento de coleta e forma de análise

A partir das características, forma definidas as medidas, que podem ser de monitoração por período, no momento da conclusão de uma determinada tarefa ou contínua, sendo:

- Por período: valor médio do tempo de renderização da página; em caso de sistemas web, no período de um mês.
- Na conclusão: modificabilidade – ao final da tarefa o tempo utilizado e o número de métodos alterados.
- Continua: estabilidade – acompanha-se o número de erros após a entrega em relação a todas as modificações feitas.

O procedimento de coleta também foi definido levando em consideração o menor impacto no *software* e nas equipes envolvidas.

Ao longo do tempo, foram feitas a proposta de documentação e análise das medidas, de modo a garantir que os indicadores tivessem um significado uniforme independente de eventuais mudanças aos envolvidos.

- Demonstração da aplicação do processo utilizando um caso real

O processo proposto foi aplicado em no *software* Cangooroo da empresa T4W Soluções Empresariais e Web Ltda. destinado à pesquisa e comparação de tarifas de hotéis, aluguel de carros, translados e atividades em diversos fornecedores por meio de serviços acessados via Internet. Trata-se um *software* utilizado pelas maiores operadoras de turismo do Brasil, sendo que seus responsáveis devem justificar a decisão de continuar sua manutenção ou desenvolver uma nova versão.

- Análise dos resultados

As medidas e os indicadores resultantes foram analisados para permitir diagnosticar a tendência de degradação da arquitetura do *software* e identificar, com justificativas, a necessidade de substituí-lo em parte ou totalmente.

1.5 Organização do trabalho

No capitulo 2, Contexto da Pesquisa, é apresentado o processo de avaliação de sistemas de informação por meio de medidas de gestão, conceito de *software aging*, processo para manter a evolução do *software* baseado no desenvolvimento orientado à funcionalidade.

O capitulo 3, Roteiro de avaliação da degradação do *software*, é dedicado a explicar o processo, são descritos todos os passos para sua aplicação e apresentando-se os modelos de artefatos resultantes: um guia que define como tomar e interpretar as medidas e o modelo de tabela de registro de resultados.

O capitulo 4, Experimento de uso, apresenta a aplicação do roteiro em um caso real. São apresentados: o cenário, os requisitos colhidos do documento de arquitetura, a definição das características, medidas e interpretação. Como resultado é produzido um guia que define como tomar e interpretar as medidas e uma tabela de resultados.

No capítulo 5, Conclusão, são abordadas: análise dos principais resultados, contribuição deste estudo e sugestões para trabalhos futuros.

2 CONTEXTO DA PESQUISA

Esta seção apresenta a revisão da bibliografia e os trabalhos relacionados ao estudo. São abordados: o ciclo de vida de um *software*, o conceito de arquitetura e a importância de seu documento, bem como a pesquisa relacionada diretamente à degradação do *software*, que é também conceitualizada.

2.1 Ciclo de vida de um *software*

São cinco as áreas de conhecimento no ciclo de vida de um *software*: Requisitos, Projeto, Construção, Teste e Manutenção (SWEBOK, 2004).

A definição de manutenção do *software* envolve as atividades necessárias para correção de possíveis erros e evolução do produto. Essas atividades têm o objetivo de prolongar a vida útil, principalmente a evolução, que inclui a adição de novas funcionalidades (BRCINA, 2009; SWEBOK, 2004), garantindo assim o atendimento ao negócio para o qual ele foi desenvolvido.

Sendo assim, ao longo de sua vida, o *software* passará por manutenções (corretivas ou evolutivas). Porém, é provável que a correção de um defeito ou adição de nova funcionalidade resulte em novos defeitos e cause aumento das falhas, que deverão ser detectadas e corrigidas ao longo do tempo (PRESSMAN, 2009). A fase de manutenção corresponde, portanto, à maior parte do tempo de vida de um *software* (SWEBOK, 2004; ABNT 12207, 2009).

A figura 1 mostra o gráfico de taxa de falhas ao longo do tempo de um *software* que sofre manutenções regulares. Verifica-se na curva real um aumento constante da taxa de falhas.

A norma ABNT 12207 trata do ciclo de vida de um *software*, trazendo, entre seus processos, a manutenção dessa ferramenta. Um dos assuntos abordados diz respeito à atividade de migração de um antigo *software* para outro novo, porém, não é indicado como detectar o momento que a substituição se faz necessária.

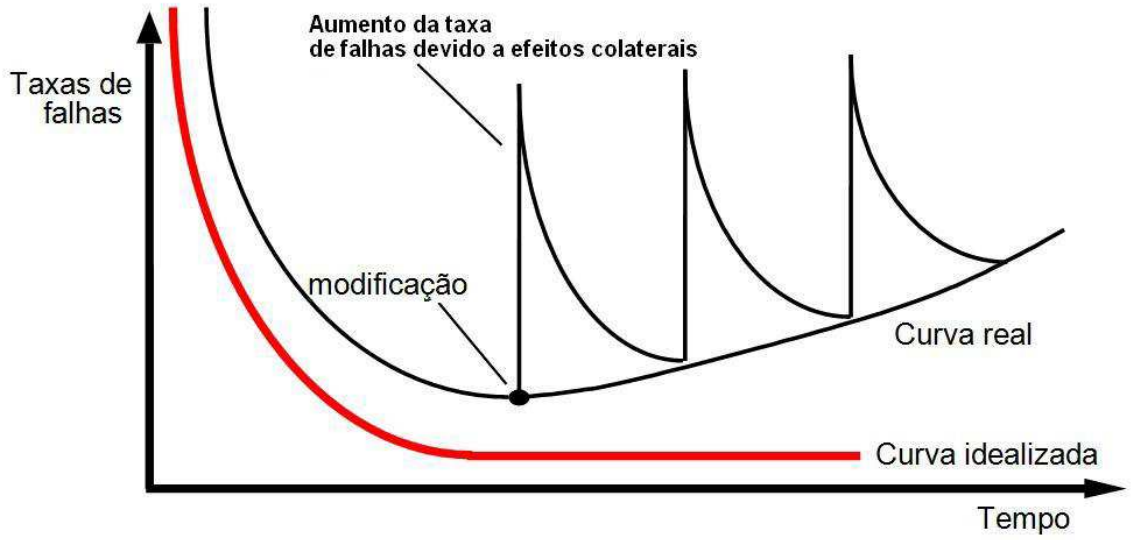

Figura 1: Taxa de falhas ao longo do tempo.

Fonte: Adaptado de Pressman (2009)

2.2 A degradação de um *software*

Um *software* evolui por várias razões. As evoluções são obtidas tipicamente através da inclusão de uma nova funcionalidade ou correção de erro. Entretanto o processo de evolução ao longo do tempo resulta na degradação do produto (PRESSMAN, 2009), como ilustrado no gráfico da figura 1.

Em seu trabalho, Jermakovics, Scotto e Succi (2007), sustentam que a degradação de um *software* é mais notável quando se aumenta a dificuldade em atender requisitos de modificações. A piora de sua estrutura leva ao declínio da qualidade, desta forma, qualquer *software* que sofre contínua modificação tem como conseqüência inevitável sua degradação (PRESSMAN, 2009; JERMAKOVICS, 2007; EICK, 2001).

Já Eick (2001) define a degradação do *software* através da expressão “decadência do código” tendo como principais sintomas o aumento da dificuldade e de custos em se operar a manutenção.

Durante a evolução do *software* espera-se que sua arquitetura mantenha a capacidade de evolução, sendo esta a única maneira de evitar que o *software* se torne obsoleto (BRCINA, 2009). Uma análise mais aprofundada da degradação pode ser feita separando-se a fase de manutenção em evolução e correção de erros. Enquanto a evolução está relacionada à adição de novas funcionalidades com maior possibilidade de impacto na arquitetura do produto, a correção de erros contribui apenas para a estabilidade do *software*, normalmente sem impacto na arquitetura.

Ao adicionar uma nova funcionalidade, devem ser mantidas as propriedades evolutivas da arquitetura do *software*, que são, segundo Brcina (2009):

- Capacidade de análise: os artefatos a serem analisados durante o desenvolvimento de uma nova funcionalidade devem ser completos, ter suas dependências coesas e serem compreensíveis. Documentação, modelos e código devem ser avaliados para expor inconsistências entre o código e a funcionalidade ou modelos arquiteturais.

- Integridade arquitetural: esta propriedade é importante, mas freqüentemente negligenciada durante a evolução (BRCINA et al., 2009). O documento de arquitetura deve ser colocado a prova durante a fase de manutenção para garantir que seja correto, completo e compreensível.

- Modificabilidade e capacidade de extensão: dentre os riscos de se efetuar uma modificação está o aparecimento de erros em outra funcionalidade relacionada ou não. O aumento da dependência entre as funcionalidades causa a diminuição da capacidade evolutiva do *software*.

- Rastreamento: a capacidade de análise ou de extensão depende do rastreamento dos artefatos. O rastreamento eficiente consiste no

relacionamento entre documentos, nível mais abstrato, até o código, nível concreto, possibilitando a avaliação dos riscos e resultados da evolução.

- Capacidade de teste: esta característica é necessária para garantir que os testes de regressão e testes unitários sejam feitos, certificando-se assim que a evolução não gerou nenhum erro em outra funcionalidade do *software*. Esta capacidade, quando comprometida, afeta diretamente a capacidade de modificação de um *software* por aumentar o seu risco.

- Portabilidade: componentes reutilizáveis, com descrição clara de suas interfaces e casos de testes aumentam a estabilidade do *software*. O aspecto negativo é a existência de muitas configurações, o que aumenta a complexidade de sua utilização no *software* e por conseqüência impacta na complexidade do *software*.

- Características específicas do domínio: as características do negócio podem comprometer a capacidade de evolução do *software*. O documento de arquitetura atua como referência para expor as características do domínio.

Neste trabalho a degradação foi definida pelo conceito geral de não atendimento às necessidades do negócio em que o *software* está inserido. O não atendimento dessas necessidades é caracterizado pela:

- Impossibilidade de se executar uma modificação.

- Eventual melhora do elemento que teve manutenção solicitada, porém piora de outros elementos importantes ao *software* quando se executa a modificação.

2.3 Qualidade de um *software*

O modelo de qualidade apresenta as características necessárias a um *software* (GUIMARÃES, 2008), que pode ou não utilizar todas elas ou eventualmente valer-se de outras, específicas.

Importante ressaltar que a utilização de um modelo confere ao projeto maior precisão e padronização quando observado o conjunto de projetos da organização. Neste trabalho o modelo de qualidade adotado é a norma ABNT 9126 de Qualidade de Produto de *Software*, com seis características externas com sub-características, sendo (ABNT, 2003):

- Funcionalidade: capacidade do *software* de satisfazer necessidades implícitas e explicitas através de suas funcionalidades.
    - Adequação: existência de funções que atendam ao objetivo do *software*.
    - Acurácia: apresentação de resultados com grau de precisão especificado ou efeitos corretos conforme especificado.
    - Interoperabilidade: capacidade de interagir com outros sistemas conforme especificado.
    - Segurança de acesso: proteção de informações a usuários ou sistemas que não estejam autorizados ou permissão de acesso a usuários e sistemas que autorizados.
    - Conformidade: estar de acordo com normas, leis, convenções e regulamentações quando necessário.
- Confiabilidade: capacidade do *software* de manter um determinado desempenho por certo tempo para a condição especificada.
    - Maturidade: baixa frequência de falhas.

  - Tolerância a falhas: manutenção do desempenho mesmo na presença de falhas, dentro das condições especificadas.
  - Recuperabilidade: capacidade de reestabelecer o nível de desempenho especificado e recuperar dados após uma falha.
  - Conformidade relacionada à confiabilidade: estar de acordo com normas, leis, convenções e regulamentações relacionadas à confiabilidade quando necessário.

- Usabilidade: capacidade do *software* de ser entendido, aprendido e atraente aos usuários dentro das condições especificadas.
  - Inteligibilidade: medida da facilidade do usuário em reconhecer a lógica de funcionamento do *software* e suas funções
  - Apreensibilidade: facilidade em aprender a utilizar o *software*.
  - Operacionabilidade: facilidade para operar o produto.
  - Conformidade relacionada à usabilidade: estar de acordo com normas, leis, convenções, guias de estilo e regulamentações relacionadas à usabilidade quando necessário.

- Eficiência: capacidade de o *software* apresentar desempenho apropriado relativo à quantidade de recursos utilizados nas condições especificadas.
  - Comportamento em relação ao tempo: fornecer tempos de resposta e processamento, além de taxas de transferência, apropriados ao longo do tempo sob as condições especificadas.
  - Utilização de recursos: capacidade de utilização de tipos e quantidades apropriados de recursos quando utilizado sob as condições especificadas.

- Conformidade relacionada à eficiência: estar de acordo com normas, leis, convenções e regulamentações relacionadas à eficiência quando necessário.

- Manutenabilidade: capacidade do produto em ser modificado. Modificações podem ser correções, melhorias ou adaptações devido a mudança de ambiente ou requisitos.
  - Analisabilidade: esforço necessário para diagnosticar deficiência, causas de falhas ou partes a serem modificadas.
  - Modificabilidade: capacidade de permitir que uma modificação seja implementada.
  - Estabilidade: capacidade de evitar efeitos inesperados devido a uma modificação feita.
  - Testabilidade: esforço necessário para testar o *software* após a manutenção.
  - Conformidade relacionada à manutenabilidade: estar de acordo com normas, leis, convenções e regulamentações relacionadas à manutenabilidade quando necessário.

- Portabilidade: capacidade do *software* de ser transferido de um ambiente para outro conforme especificado.
  - Adaptabilidade: facilidade para do *software* em se adaptar nos diferentes ambientes especificados sem a necessidade de aplicar recursos além dos fornecidos pelo *software*.
  - Capacidade para ser instalado: facilidade de ser instalado no ambiente especificado.
  - Coexistência: capacidade de coexistir com outros *software* independentes em um ambiente comum, compartilhando recursos.

- Capacidade para substituir: esforço para substituir um *software* por outro com o mesmo propósito e no mesmo ambiente.
- Conformidade relacionada à portabilidade: estar de acordo com normas, leis, convenções e regulamentações relacionadas à portabilidade quando necessário.

A qualidade em uso define quatro características sob a perspectiva do usuário, sendo (ABTN, 2003):

- Eficácia: capacidade do usuário de atingir a meta especificada com acurácia.
- Produtividade: capacidade do usuário em realizar determinada tarefa com quantidade de esforço apropriado nas condições especificadas.
- Segurança: capacidade de apresentar nível de risco aceitável a pessoas, negócios, *software*, propriedades e ambiente dentro do especificado.
- Satisfação: capacidade de satisfazer as necessidades do usuário conforme especificado.

O roteiro proposto neste trabalho endereça os requisitos colhidos no documento de arquitetura a características da norma ISO/IEC 9126 por serem amplamente aceitas, permitindo a substituição de uma medida por outra, de modo a atender a mesma característica, caso esta tenha mais relevância quando da atualização do documento de arquitetura.

2.4 Arquitetura de *software*

A arquitetura se refere aos elementos utilizados pelo *software* e o relacionamento entre eles. Tais elementos podem ser características de velocidade, tratamento e frequência de erros, recursos de processamento, entre outros (BASS, 2003).

A arquitetura de *software* é considerada um dos mais importantes artefatos do processo de desenvolvimento pelas seguintes razões (BASS, 2003):

- Comunicação entre os interessados: a arquitetura de *software* representa uma abstração comum do sistema que pode ser utilizada pela maioria dos interessados para entendimento mútuo, negociação, consenso e comunicação;

- Decisões de projeto iniciais: manifesta as decisões iniciais, sendo conduzidas por todo o desenvolvimento, entrega e manutenção. Trata-se do ponto no qual mais cedo podem ser analisadas as decisões de projeto que guiam o sistema;

- Abstração transferível de um sistema: a arquitetura constitui um modelo relativamente pequeno e compreensível de como um sistema é estruturado e como seus componentes trabalham juntos. Este modelo é transferível para outros sistemas, em particular àqueles que apresentam requisitos funcionais e atributos de qualidade similares.

A arquitetura do *software* é definida pelo relacionamento entre modelos de referência, padrões de arquitetura e uma arquitetura de referência (BASS, 2003).

O modelo de referência divide o *software* em partes que contribuem para o seu entendimento (BASS, 2003). Neste trabalho, foi utilizado o modelo de referência ODP tratado na ISO/IEC 10746.

Uma arquitetura de *software* é o resultado de decisões que devem atingir determinados objetivos. Dentre eles, existem alguns que podem exigir a não realização de outros. O método de análise da arquitetura por troca (em inglês ATAM - *Architecture Tradeoff Analysis Method*) permite o relacionamento entre os objetivos documentados e as consequências possíveis das decisões de trocas escolhidas (KAZMAN, 2000).

2.4.1 Modelo de referência ODP

O modelo de referencia ODP estabelece cinco visões que podem descrever o sistema e seu ambiente (KUDRASS, 2003). Cada visão estabelece o seu conjunto de características que afetam todo o sistema. As visões são:

- Empresa;
- Informação;
- Computacional;
- Engenharia;
- Tecnológica.

A figura 2 mostra a relação entre as visões do RM-ODP.

A visão de empresa descreve as atividades e metas de médio e longo prazo que devem ser atingidas. Essas metas resultam em um conjunto de requisitos (KUDRASS, 2003), dos quais são extraídas políticas e estruturas da organização. Tais políticas são determinadas pelo negócio e não por restrições de tecnologia (VAROTO, 2002). Os requisitos não funcionais, geralmente associados à qualidade do *software*, sendo também extraídas dessa visão, por exemplo: manutenabilidade, adaptabilidade, velocidade e outros (KUDRASS, 2003).

A visão de informação descreve os dados contidos no *software* e suas relações, sendo atualmente representada por diagramas de entidade e relacionamento em diferentes níveis de refinamento (KUDRASS, 2003).

A visão de computação especifica a decomposição das funcionalidades, dados e processamento em um conjunto de elementos que interagem entre si (VAROTO, 2002). O comportamento da interação é definido por pré- e pós-condições que devem estar de acordo com modelo de negócio. É definida também a forma como o *software* irá interagir com outros sistemas (KUDRASS, 2003).

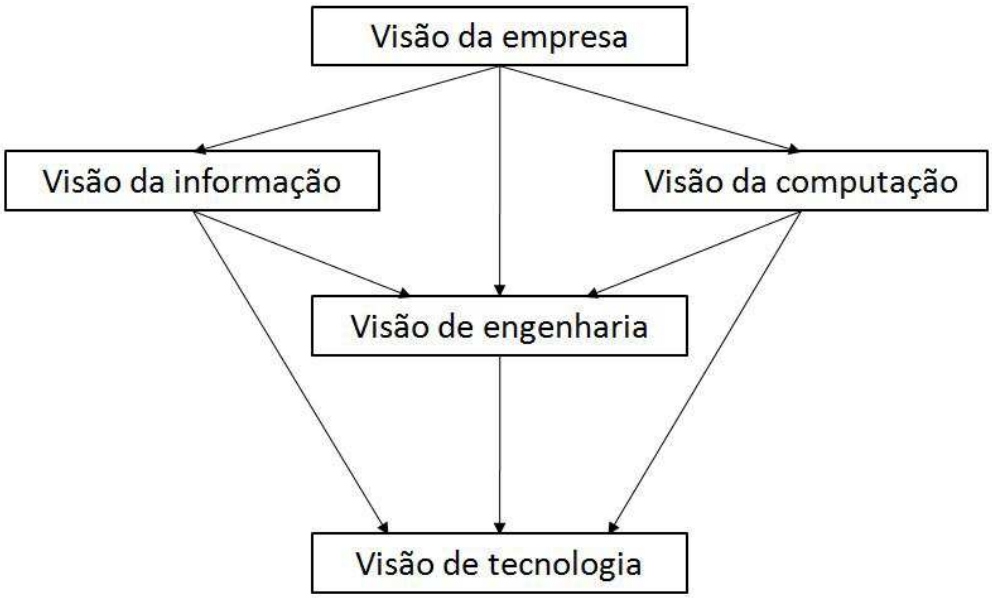

Figura 2: Visões do RM-ODP.

Fonte: Adaptado de Varoto (2002).

A visão de engenharia lida com a infraestrutura para distribuir os acessos ao *software*. Uma dimensão dessa visão é a divisão de um *software* em três camadas lógicas: empresa, intermediária e apresentação. A camada empresa lida com a informação; a camada intermediária, com as regras de negócio, especificada na visão de computação; a de apresentação, com a forma como os dados serão apresentados (KUDRASS, 2003).

A visão de tecnologia descreve as justificativas para a escolha das tecnologias de mercado (VAROTO, 2002). São representadas como caixas pretas onde apenas as entradas e saídas são de interesse da análise da arquitetura (KUDRASS, 2003).

As visões do modelo de referência ODP permitem compreender e integrar os conceitos de cada visão que o *software* deve atender. O conjunto de termos usado em cada uma das visões facilita a comunicação entre os envolvidos (KUDRASS, 2003).

Neste trabalho o RM-ODP será usado como base para análise do documento de arquitetura. As características observadas e medidas ao longo do processo são

ligadas às visões, permitindo acompanhar a evolução da arquitetura do *software* de acordo com os objetivos dos envolvidos.

2.4.2 O documento de arquitetura de um *software*

O documento de arquitetura de um *software* serve como base para o desenvolvimento de um *software* (BASS, 2003). Com a representação das diversas visões dos envolvidos, esse documento se torna uma fonte importante de requisitos não funcionais e alguns funcionais. Após o desenvolvimento do *software* o documento pode ser utilizado para avaliá-lo (KUDRASS, 2003).

A avaliação periódica do *software*, embora necessária, é uma prática pouco utilizada (RICHMOND, 2006), e quando realizada, geralmente está associada à percepção de que alguma característica apresenta um desvio do que foi idealizado (QUEIROZ, 2009).

A aplicação do modelo de referência ODP para a confecção do documento de arquitetura garante o registro de todas as visões e expectativas dos envolvidos. O documento pode ser renovado, sempre que necessário, ao longo da vida útil do *software*.

O documento de arquitetura foi o ponto inicial da aplicação do roteiro proposto neste trabalho.

2.4.3 A avaliação da arquitetura por meio do ATAM

O ATAM é um método de análise da arquitetura que considera, no processo de avaliação, as necessidades do negócio relacionadas ao *software*, mapeando-as em relação às decisões arquiteturais (KAZMAN, 2000).

Existem três propósitos principais para a avaliação da arquitetura de um *software* (CLEMENTS, 2001):

- Identificação de requisitos não alcançados: quanto antes for verificado que a arquitetura não atinge determinado requisito mais fácil para se realizar a modificação;

- Comparação de arquitetura: podem existir muitas arquiteturas candidatas a atingir determinados requisitos funcionais porém elas podem endereçar atributos de qualidade de modo diferente;

- Identificação dos riscos aos atributos de qualidade: a avaliação permite identificar os riscos de certos atributos de qualidade, que podem então ser mitigados ou exigir a troca com outro atributo para a sua realização;

O método é baseado em questionamento e não em medição. A estratégia do ATAM é basicamente identificar as abordagens arquiteturais, tais como: táticas arquiteturais, padrões de arquitetura e padrões de projeto, que afetem o atendimento dos atributos de qualidade do *software* (BASS, 2003).

Embora técnicas baseadas em medição possam apresentar resultados mais concretos, a grande vantagem do ATAM é que a avaliação pode ser executada ao longo do processo de desenvolvimento de um sistema; já as medições exigem um artefato pronto para ser testado (BASS, 2003).

O processo do ATAM é composto por quatro fases, apresentadas na figura 3, e nove passos (BASS, 2003).

- Fase 0 – Parceria e Preparação: Planejamento do processo de avaliação incluindo definição dos participantes, papéis de cada um deles e uma agenda para o processo;

- Fase 1 – Avaliação: primeira avaliação da arquitetura;

- Fase 2 – Avaliação continuada: avaliação da arquitetura incluindo a equipe que a desenvolve;

- Fase 3 – Acompanhamento: Nesta fase é produzido um documento com as decisões arquiteturais, riscos, cenários utilizados e as trocas propostas.

Figura 3: Fases do ATAM.

Fonte: Adaptado de Kazman (2000).

Os passos do ATAM são (BASS, 2003):

- Passo 1 – Apresentar o ATAM a todos os envolvidos na avaliação
- Passo 2 – Apresentar o objetivo de negócio que o *software* deve atingir, contemplando as principais funções, restrições, atributos de qualidade e os interessados;
- Passo 3 – Apresentar a arquitetura com foco nas táticas arquiteturais e nas restrições relacionadas à plataforma escolhida. As integrações com outros sistemas e suas restrições também são apresentadas;

- Passo 4 – Identificar abordagens arquiteturais utilizadas;
- Passo 5 – Gerar a árvore de atributos de qualidade através dos cenários organizados por complexidade. Cada cenário deve ser descrito com um nível de detalhe que permita a análise dos atributos de qualidade e possíveis refinamentos deste atributo. A tabela 1 apresenta um exemplo de árvore de atributos;
- Passo 6 – Análise das abordagens arquiteturais por meio do questionamento, em cada cenário, da equipe de avaliação em contraposição à equipe de arquitetura. Ao final deste passo deve ser possível identificar as decisões arquiteturais tomadas e como elas afetam os outros cenários.
- Passo 7 – Nova árvore de atributos e cenários, como feito no passo 5. Este passo está relacionado à fase 2, em que a equipe que desenvolve a arquitetura está presente e cria cenários. O resultado é uma nova priorização dos cenários definidos do passo 5 com os deste passo.
- Passo 8 – Nova análise das abordagens arquiteturais com os cenários definidos no passo 7;
- Passo 9 – Apresentação do resultado da análise. São descritos: abordagem arquitetural, riscos e trocas identificados. Todas as informações são compiladas e fazem parte da documentação da arquitetura.

O processo proposto neste trabalho analisa a fase de manutenção do ciclo de vida do *software*, durante a qual os objetivos do *software* podem ser alterados e o documento de arquitetura pode ganhar uma nova versão. O ATAM pode ser utilizado na avaliação desta nova versão.

Tabela 1: Exemplo de árvore de atributos.

Atributo de qualidade	Sub-atributo de qualidade	Cenário

Desempenho	Latência de dados	Minimizar a latência do banco de dados para 200ms
		Entregar vídeo em tempo real
Modificabilidade	Modificar componentes	Modificar telas web cont apresentação ao usuário em menos de um mês com equipe de quatro pessoas
Disponibilidade	Falha do servidor	Interrupção de energia no servidor 1 dispara redirecionamento para o servidor 2 em menos de três segundos.
Segurança	Confidencialidade dos dados	Autorização de acesso ao usuário tem erro em menos de 0,001% do tempo

Fonte: Adaptado de Kazman (2000).

O ATAM foi utilizado neste trabalho para auxiliar a validação da escolha dos atributos e medidas de acordo com as trocas definidas pelo grupo de pessoas envolvidas com o *software*, que pode ser formado por profissionais ligados à área técnica e por profissionais ligados a questões comerciais e de suporte ao produto. Importante ressaltar que a aplicação do ATAM é opcional.

2.5 Análise empírica para determinar a vida do *software*

As empresas realizam pesados investimentos quando adquirem um *software* (RICHMOND, 2006). O termo aquisição, neste caso, se refere a uma ferramenta desenvolvida pela empresa ou comprado de outras. Em seu trabalho, Richmond (2006), analisou 180 *software* com o objetivo de determinar o tempo de vida para trocá-lo. A análise foi feita em estágios iniciais, sendo que os patrocinadores tinham um conhecimento limitado dos projetos.

A avaliação do *software* nas organizações analisadas por Richmond (2006) levou em conta fatores de negócio, custo da manutenção de *software* atual versus o custo de aquisição de um *software* para substituir o atual e custo de aquisição de um novo *software* para novas necessidades.

A análise de Richmond (2006) começa com a determinação do valor presente do total de investimento. Esta determinação contempla o custo de aquisição e operação do *software*, a remuneração do capital dentro do período, um peso dado pelo beneficio do *software* para o negócio e o tempo esperado de vida do *software*. O custo de operação, neste caso, representa o custo de manutenção esperado pelo tempo de vida estimado para o *software*.

Em teoria, Richmond (2006) afirma que, a vida útil de um *software* termina quando o custo do benefício de troca é igual ao custo de troca a valor presente, conforme apresentado na figura 4. O ponto t1 indica o momento em que a troca deve de um produto por outro.

Na prática, a relação de custo e beneficio da troca do *software* não é linear e sofre o o efeito das mudanças das necessidades do negócio, podendo, ao longo do tempo ser mais ou menos acentuada para um mesmo *software* (RICHMOND, 2006).

Richmond (2006) então analisa como as decisões de projeto, a abordagem tecnológica, a manutenção e as mudanças do negócio influenciam o tempo de vida útil de um *software*. Partindo dessa premissa, o autor elabora as seguintes hipóteses:

- Hipótese 1: *Software* que suporta várias áreas do negócio tem vida mais curta.
- Hipótese 2: *Software* que suporta várias áreas de decisão tem vida mais curta.
- Hipótese 3: *Software* pronto é associado a vida mais longa.
- Hipótese 4: Quanto mais modificações o *software* sofre menor é a vida útil.

- Hipótese 5: Um time de desenvolvimento misto, que utiliza recursos da empresa e de terceiros, produz *software* com vida mais longa.
- Hipótese 6: *Software* que utiliza múltiplas tecnologias tem vida mais curta.
- Hipótese 7: *Software* aderente aos padrões de tecnologia da organização tem vida mais longa.

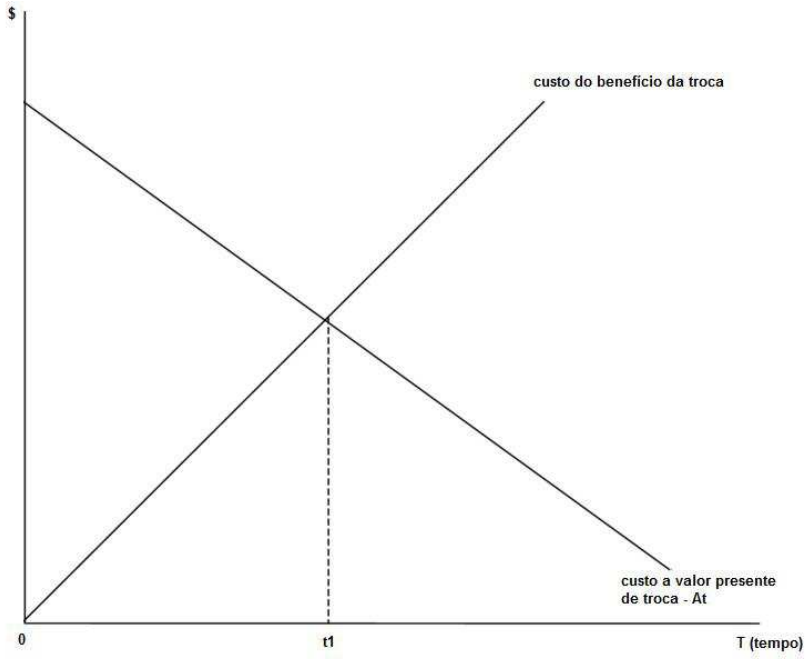

Figura 4: Momento teórico quando a vida útil do *software* termina.

Fonte: Adaptado de Richmond (2006).

As hipóteses foram validadas com questionários enviados aos patrocinadores dos projetos analisados, incluindo uma uma variedade de *software* substituídos e outros que ainda estão em uso.

Fatores relacionados ao ambiente, como capacidade técnica do time, controle de projeto, documentação, velocidade de evolução da tecnologia entre outros têm influencia na vida útil do *software* e embora não façam parte do foco do estudo eles foram levados em conta pelos patrocinadores para validar as hipóteses.

Dos dados analisados no começo dos projetos e com o acompanhamento destes projetos ao longo do tempo, Richmond (2006) propôs um modelo matemático que permite determinar a vida útil de um *software*. Isto significa que, baseado em seus resultados, um *software* desenvolvido internamente, não estratégico para o negócio, com um interessado que não seja executivo e com as outras variáveis com medidas médias tem 75% de chance de ter sua vida útil de 5 anos. Para o mesmo cenário, mas adquirindo um *software* pronto, a chance é de 75% para durar 22 anos. O próprio autor explica que as estimativas de tempo de vida podem variar ao longo do ciclo de manutenção do *software* devido às modificações as quais o negócio está sujeito

O modelo é apresentado por ser oposto ao roteiro proposto neste trabalho. Aqui, o foco foi medir a degradação da arquitetura ao longo do ciclo de manutenção de um *software* criando indicadores que auxiliem na decisão de troca do produto.

O acompanhamento da degradação do *software* do ponto de vista da sua aderência ao documento de arquitetura oferece aos envolvidos a possibilidade de compreender e planejar com antecedência a troca ou não levando em conta aspectos de negócio e técnicos. Ao apresentar uma indicação que signifique aos envolvidos a necessidade de troca é natural que se faça, mesmo que informalmente, uma análise de custo benefício.

2.6 Avaliação da deterioração de um *software* por medidas de gestão

Partindo da premissa de que as características gerais de deterioração de um *software* são baseadas no declínio de sua capacidade de se manter e evoluir ao longo de seu ciclo de vida, o estudo conduzido por Queiroz et al.(2009) utiliza o método GQM para analisar um sistema de informação, com o propósito de monitorar a deterioração do produto, sob o ponto de vista de gestores da aplicação.

Os autores (QUEIROZ et al., 2009, p.50) elaboraram três questões para avaliar a deterioração de um *software*:

- “Q1- A manutenabilidade do sistema está sendo comprometida?”
- “Q2 - Os custos do sistema são elementos ofensores à continuidade de sua operação?”
- “Q3 - Qual o nível de satisfação dos usuários com relação à utilização do sistema?”

As questões foram validades por gestores com mais de dez anos de experiência na área de tecnologia da informação.

Para definir medidas que respondessem a essas questões foram avaliados critérios definidos em métodos existentes de apoio à decisão sobre a modernização de *software* definidos na norma ISO 9126. A investigação resultou em um total de 169 medidas, sendo 44 consideradas duplicadas em relação à sua definição ou forma de calculo, resultando em 125 medidas efetivas (QUEIROZ, 2009).

As medidas foram analisadas considerando seis premissas, sendo as quatro primeiras elaboradas no objetivo GQM. Já a quinta e a sexta são voltadas para o âmbito gerencial. As premissas são (QUEIROZ, 2009):

1. Medidas que têm foco na deterioração do *software*: quantidade de falhas por período.
2. Medidas que representam o contexto da organização: para a que foi utilizada pelo autor, o tamanho do *backlog* é uma medida que representa seu contexto.
3. Medidas que representam a visão dos gestores: na organização analisada por Queiroz, o custo da manutenção versus o orçamento total da área é uma medida que representa a visão dos gestores.
4. Necessidade de evolução.

5. Maneira como as decisões são tomadas, sendo excluídas, neste caso, medidas subjetivas.

6. Utilização de medidas simples e factíveis.

Utilizando as premissas 96 medidas das 125 consideradas foram excluídas.

Uma nova análise foi feita com gestores de *software*, os quais foram entrevistados e as informações consolidadas resultando em 12 medidas que respondem às 3 questões iniciais.

O processo foi aplicado em diversos *software* de uma mesma organização e os resultados obtidos foram considerados satisfatórios (QUEIROZ, 2009). A tabela 2 apresenta o resultado consolidado, para um determinado *software*, das medidas apuradas no primeiro trimestre de 2008 (1T/2008), a exceção das medidas M8 e M9 que são consolidadas por ano. Esta tabela também apresenta o sinal de deterioração que é definido da seguinte forma:

- Alto: o valor é igual ou maior do que a média móvel **e** maior ou igual à média acumulada, independente da relação entre as médias;
- Médio: o valor é maior ou igual que a média móvel **ou** maior ou igual que a média acumulada, independente da relação entre as médias;
- Baixo: o valor é menor que a média móvel **e** menor que a média acumulada **e** a média móvel é maior que a média acumulada;
- Nenhum: o valor é menor que a média móvel **e** menor que a média acumulada **e** a média móvel é menor que a média acumulada.

Tabela 2: Resultado consolidado das medidas.

	Medida	Valor 1T/2008	Média Acumulada	Média Móvel	Sinal de deterioração
Q1	M1 – Volume de falhas	1210	984	1135	Alto

	M2 – Tamanho do Backlog	158	106	150	Alto
	M3 – Tamanho do esforço por demanda	236	225	193	Alto
	M4 – Produtividade invertida	102%	149%	139%	Nenhum
	M5 – Novas demandas	133	97	123	
	M6 – Eficiência do ciclo de correções	224	244	271	Baixo
Q2	M7 – Custo médio por demanda	10322	10721	8638	Médio
Q3	M1 – Volume de falhas	1210	984	1135	Alto
	M2 – Tamanho do Backlog	158	106	150	Alto
	M4 – Produtividade invertida	102%	149%	139%	Nenhum
	M10 – Tempo médio entre falhas invertido	0,56	0,45	0,52	Alto
	M11 – Tempo de resposta	Não apurado			
	M12 – Nível de utilização do processador	Não apurado			

Fonte: Adaptado de Queiroz (2009).

2009) é permitir a indicação da ‘saúde’ do *software* durante sua vida. As medidas são colhidas manualmente e alimentadas em uma planilha. O autor, porém, reconhece a necessidade de refiná-las para que, dentro do possível, elas possam ser tomadas de maneira automatizada.

A abordagem do autor é semelhante à proposta neste trabalho. A diferença é que aqui foram utilizadas medidas relacionadas a características diretamente endereçadas no documento de arquitetura, conferindo a possibilidade de indicadores específicos e relevantes ao negócio em que o *software* analisado está inserido.

2.7 *Software aging*

Software aging é o nome dado ao fenômeno observado empiricamente que em geral é resultado do aumento de falhas ao longo do tempo em que um *software* está em execução (GROTTKE, 2008).

Os computadores podem apresentar indicações de desgaste quando chegam próximo ao fim de sua vida útil, geralmente relacionado ao tempo de uso. Já o *software* pode apresentar desgaste a qualquer momento por diversas influências sem relação direta com seu tempo de existência. Em alguns casos as falhas são consequência do acumulo de outras pequenas falhas (GROTTKE, 2008).

A propagação dos erros requer um estado particular de todo o sistema. A maioria deles que ainda não causaram falhas tem seu estado mantido pelo sistema, acumulando-se. O acumulo de diversos erros leva o *software* a apresentar uma ou mais falhas (GROTTKE, 2008).

O principal indicador deste fenômeno é a exaustão de recursos do sistema operacional (GROTTKE, 2008), ou seja, nível de uso da memória e processador.

O fenômeno de *aging* é dividido em dois tipos. O primeiro trata da falha dos interessados ao conduzir mudanças para atender a novas necessidades. O segundo é o resultado das mudanças feitas. Por outro lado, *software* que não é atualizado, dependendo do negócio em que está inserido, se torna obsoleto e perde seu valor, mesmo que siga executando as suas tarefas corretamente. Um exemplo dessa situação é o incremento cada vez maior dos recursos de interação com o usuário (PARNAS, 1994).

A partir do inicio de sua operação um *software* fica exposto a uma falha relacionada ao *aging*. O período de tempo até essa falha ocorrer é influenciado pela intensidade do uso ou exposição a fatores que podem levar a sua ocorrência, como exaustão da memória do computador (GROTTKE, 2008).

Os efeitos do *aging* só podem ser observados com o *software* em operação analisando medidas relacionadas ao fenômeno. Tais medidas podem ser feitas em diversos níveis do sistema como: sistema operacional, processo da aplicação, banco de dados ou maquina virtual (GROTTKE, 2008).

De acordo com Grottke (2008), dependendo de onde as medidas são tomadas, os indicadores resultantes podem ser classificados em dois tipos:

- Indicador de sistema: medidas de que podem sofrer influencia de outras aplicações que compartilham seus recursos, exemplo: sistema operacional e máquina virtual.

- Indicador de aplicação: medidas da aplicação em análise, por exemplo: tempo de resposta de uma rotina. A aplicação pode sofrer influência indireta de outras, pode compartilhar recursos de um mesmo computador, mas as medidas tomadas são unicamente relacionadas a esta.

As causas de falhas atribuídas à dinâmica natural do sistema são denominadas *natural aging*. O *natural aging* é observado por falhas decorrentes da fragmentação dos dados em disco, arquivos de índices de banco de dados corrompidos e exaustão da memória do computador (GROTTKE, 2008).

Nem todas as falhas podem ser atribuídas ao fenômeno de *aging*. Grottke (2008) propõe a observação da seguinte característica: o efeito do *aging* deve ser resolvido com intervenção externa. O aumento de erros causados pela fila de tarefas a serem executadas, sendo esta fila resultado do aumento de uso da aplicação irá desaparecer quando as tarefas forem executadas e a aplicação voltar ao seu uso normal. Neste caso as falhas acabam sem intervenção externa. Um exemplo comum dessa intervenção é a interrupção e inicio de um processo ou mesmo de todo o computador. O resultado dessa intervenção recebe o nome de “rejuvenescimento do *software*”.

Grottke (2008) destaca a discussão em torno das técnicas de rejuvenescimento pois acredita-se que elas não tratam dos sintomas do problema em busca de uma

solução definitiva. Entretanto, elas tendem a ser usadas pelas limitações técnicas, econômicas ou de prazo de outras soluções (GROTTKE, 2008).

Este trabalho se vale do conceito de *software aging*, endereçando a exaustão dos recursos de *hardware* à característica “eficiência”, sub-característica “utilização de recursos”. A análise das medidas, diferente do fenômeno exposto, é feita em conjunto com os outros indicadores uma vez que a analise isolada do fenômeno não permite afirmar que a troca do *software* é necessária.

2.8 O método de medição por objetivo, questão e métrica

Medir é representar atributos de entidades do mundo real com símbolos ou números como forma de descrevê-los de maneira clara e padronizada. A medida pode ser usada no processo de desenvolvimento e no artefato resultante (VAN SOLINGEN, 1999). Em organizações a medida ajuda a somar dados para a memória corporativa e responde a diversas questões ligadas ao modelo de processo adotado e aos produtos entregues (BASILI, 2009).

Como toda a disciplina de engenharia, a engenharia de *software* demanda um mecanismo de avaliação e análise (BASILI, 2009). Basili (2009) aponta três aspectos que devem ser observados na escolha de mecanismo de avaliação e analise para projetos de *software*:

- Foco em objetivos ou metas específicas;
- Aplicação em todo o ciclo de vida dos produtos, processos e recursos;
- Interpretação baseada na caracterização e entendimento do contexto da organização, ambiente e metas.

A adoção do método GQM resulta em regras para medir e interpretar um conjunto particular de itens do projeto. O método tem três níveis de abstração (BASILI, 2009):

- Nível conceitual - objetivo: uma meta é definida para um objeto do projeto, por razões aderentes ao ambiente ou organização desse projeto. Normalmente objetos são:
    - Artefatos, entregas e documentos que são produzidos durante o ciclo de vida do *software*;
    - Atividades relacionadas ao desenvolvimento do *software* (processos);
    - Itens usados no desenvolvimento de *software* (recursos);
- Nível operacional – questões: conjuntos de questões relacionadas ao objetivo. A questão deve caracterizar o objeto de medida (artefatos, processos, recursos e outros) endereçando para uma determinada qualidade e que possa determinar essa qualidade sob o ponto de vista escolhido;
- Nível quantitativo – medida: informações e dados associados a cada questão e que a responda. Esses dados podem ser objetivos ou subjetivos, sendo que as medidas objetivas dependem apenas do objeto que está sendo medido e não do ponto de vista de quem as toma. Já as subjetivas dependem do objeto e do ponto de vista de quem as toma.

O GQM é um modelo hierárquico e começa com uma meta que é refinada em diversas questões. Para cada uma delas são definidas medidas objetivas e subjetivas. As medidas podem responder a mais de uma questão para um mesmo objetivo. As metas podem ter questões e medidas em comum, o que normalmente ocorre com questões subjetivas, para assegurar que as medidas sejam tomadas levando em conta os diferentes pontos de vista (BASILI, 2009). A figura 5 exibe a representação do modelo.

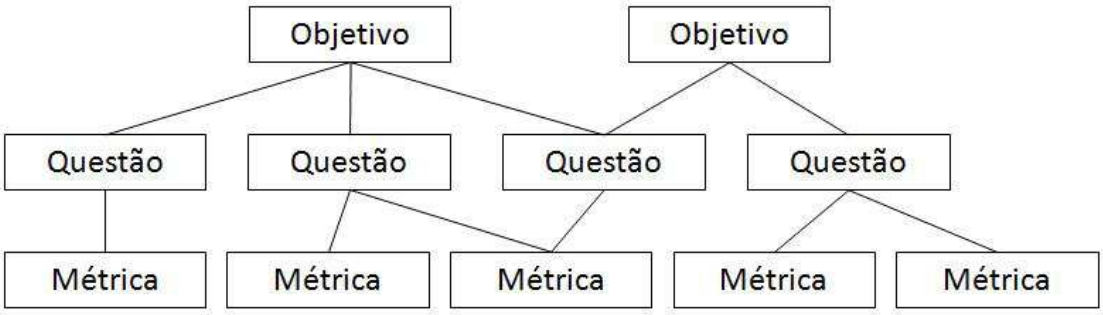

Figura 5: Representação de níveis do GQM.

Fonte: Adaptado de Basili (2009).

O processo de definição de metas é crítico para a utilização do método GQM, sendo suportado por passos específicos (BASILI, 2009) que resultam nos seguintes atributos:

- Objeto: o que será analisado;
- Objetivo: porque analisar o objeto;
- Enfoque: o atributo a ser analisado deve ser claramente identificado;
- Pontos de vista: coleta e utilização dos dados das medidas;
- Contexto: o ambiente em que as medidas serão feitas;

Neste trabalho o método GQM será utilizado para definir métricas alinhadas aos atributos mais relevantes para o *software* analisado.

2.9 Conceito de suficiência

Chung (2000) propõe o conceito de suficiência no atendimento de requisitos não funcionais (RNF). Este conceito trata cada RNF como um objetivo e seu atendimento é representado por um de cinco estados pré-estabelecidos.

Dias et al. (2010) apresenta descrições de cada estado relacionando o atendimento do requisito não funcional à decisão arquitetural:

- Quebra (-2): quando uma decisão ou um componente arquitetural existente impede o atendimento do requisito;
- Afeta negativamente (-1): quando uma decisão dificulta o atendimento de um ou mais objetivos;
- Neutro (0);
- Ajuda (+1): contribui positivamente;
- Implementa (+2): quando a decisão viabiliza o requisito ou objetivo;

Neste trabalho propõe-se uma adaptação do conceito para auxiliar a avaliação da degradação da arquitetura. O principal objetivo é obter uma escala que indique o estado do *software* em relação a sua arquitetura.

Considerando-se os atributos no lugar dos requisitos, e que cada atributo é um pequeno objetivo a ser atingido, os estados propostos podem ser adaptados para variar do não atendimento do atributo ao atendimento completo do atributo, sendo:

- Degrada (-2): quando o atributo indica degradação da arquitetura;
- Afeta (-1): o atributo tem tendência negativa na arquitetura;
- Neutro (0): o atributo não ajuda ou prejudica a arquitetura;
- Ajuda (+1): o atributo tem tendência positiva na arquitetura;
- Melhora (+2): o atributo melhora a arquitetura;

A utilização do conceito de Chung (2000) adaptado é permitir a comparação de diferentes medidas de atributos endereçadas a uma mesma característica da norma ISO 9126.

2.10 Conclusão

Esta seção apresenta as pesquisas recentes sobre a escolha do momento da troca do *software*, a definição do conceito de degradação e sua detecção.

O trabalho mais similar a este é do Queiroz (2009), entretanto suas medidas foram definidas através das características que os gestores acreditam estarem ligadas à deterioração. Já o presente trabalho apresenta um roteiro que defini indicadores em função dos requisitos do documento de arquitetura e em relação à norma de qualidade. O RM-ODP auxilia na definição das características atendendo as necessidades dos envolvidos. A ferramenta utilizada para definir medidas que representem essas características será o método GQM.

A razão pela qual foi definido que os requisitos fossem endereçados a características estabelecidas na norma ISO/IEC 9126 é a oportunidade de se alterar uma medida sem prejudicar o indicador vinculado a ela. O método ATAM evidencia características que são prioritárias em relação a outras validando assim a escolha dos critérios e indicadores.

Já o trabalho de Richmond (2006) foi apresentado por ser uma forma de determinar a vida útil, porém levando a decisão para aspectos financeiros. O trabalho de Grottke (2008) sobre o fenômeno de *software aging* indica a maneira de se detectar o desgaste do *software* baseado na exaustão de recursos da máquina, mas esta análise isolada não permite indicar que ele atende ao negócio e deve ser substituído.

O capitulo seguinte apresenta o método e a forma de aplicação, bem como os artefatos que devem ser gerados.

3 ROTEIRO DE AVALIAÇÃO DA DEGRADAÇÃO DE UM *SOFTWARE*

O roteiro aqui apresentado é uma proposta estruturada para atender à necessidade de avaliação da degradação de um *software*, que é normalmente baseada na experiência da equipe, o que torna os critérios e argumentos subjetivos.

Embora um *software* faça parte de um sistema[1] mais amplo e complexo, o roteiro proposto neste estudo permite sua avaliação isolada, possibilitando assim identificar a que aspectos dedicar esforços para evitar a degradação do produto.

Importante ressaltar que a solução proposta não depende do processo escolhido para gestão do *software*. Projetos baseados em métodos planificados ou ágeis têm, em sua fase de manutenção, a possibilidade de aplicar o roteiro.

O roteiro pressupõe que o *software* esteja em produção e sofra manutenções corretivas ou evolutivas, as quais não devem, porém, impactar as características que se pretende manter. De fato, como já apontado anteriormente, a manutenção é a fase em que o *software* passa a maior parte do tempo de seu ciclo de vida.

As manutenções também devem acompanhar a evolução do negócio. No roteiro aqui apresentado, esta evolução é caracterizada pela atualização e avaliação do documento de arquitetura do *software*.

3.1 Requisitos para o roteiro

Os atributos de qualidade de um *software* são inúmeros, alguns são dependentes, outros são excludentes. Para avaliá-lo no que se refere à degradação, pode ser necessário utilizar um método de avaliação de arquitetura que exponha o conjunto de atributos relevantes em análise e suas dependências, quando existem. Desta maneira o primeiro requisito é a definição dos atributos relevantes para o *software* em questão.

1 Para este trabalho o termo sistema é utilizado para caracterizar um conjunto de *software* que atendem a um determinado negócio. Estes *software* podem ser aplicações que representam as camadas de uma determinada arquitetura.

Dentre esses atributos, alguns podem ter limites (mínimo e máximo) definidos em função das necessidades do negócio em que o *software* está inserido, podendo inclusive sofrer alteração ao longo do tempo, por mudanças nesse negócio ou ambiente.

Outro requisito fundamental é definir medidas para os atributos, as quais servem para apresentar, de forma acessível a todos os envolvidos, a melhora ou piora na degradação do *software* em função dos limites estabelecidos. Este roteiro propõe um modo estruturado para a definição dessas medidas.

O procedimento de coleta e análise das medidas deve ser padronizado para permitir a análise ao longo do tempo e eventuais mudanças na equipe. O registro destas medidas resulta nos artefatos do roteiro.

Para aplicar o roteiro, sugere-se estabelecer uma equipe com as seguintes responsabilidades: arquiteto de *software*, analista de negócio e analista de qualidade. O número de profissionais e o envolvimento de outros especialistas devem ser definidos em função do tamanho do *software*, sua abrangência e importância dentro da organização. Esta equipe deve se preparar e preparar os envolvidos para aplicação do roteiro.

A apresentação das medidas comparadas entre si ao longo do tempo e com os limites estabelecidos no documento de arquitetura e em suas revisões é o resultado do roteiro, que permite avaliar, então, o nível de degradação da arquitetura do *software*.

Os próximos itens caracterizam os pontos importantes do roteiro elaborado no presente estudo.

3.1.1 Identificação do ciclo de vida

Um *software* está na fase de manutenção, quando os seguintes requisitos são observados:

- O produto está em uso, atende a uma ou mais necessidades do negócio em que está inserido.
- A evolução não deve impactar as características do *software*, as quais são representadas por requisitos funcionais e não funcionais.
- A evolução mantém a capacidade competitiva do negócio.

3.1.2 Preparação da equipe

Para ser responsável pela aplicação do roteiro, destaca-se uma equipe treinada na estrutura desse instrumento, seus artefatos e suas atividades. Os participantes ficam responsáveis por treinar os envolvidos, acompanhar suas atividades, coletar os dados e avaliar a arquitetura do *software*.

Esta preparação compreende também a formação da base de conhecimento e sua política de atualização.

3.1.3 Identificação ou confirmação dos atributos

Trata-se da etapa inicial de aplicação do roteiro, consistindo no levantamento dos atributos de qualidade relevantes ao negocio em que o *software* está inserido.

A identificação dos atributos de qualidade é dividida em quatro etapas:

- Analise / atualização do documento de arquitetura. Neste trabalho optou-se pela utilização do modelo de referência ODP, descrito no capitulo2, para a análise ou atualização do documento de arquitetura. Esta análise foca nos requisitos não funcionais, sem entretanto desprezar outros atributos relevantes.
- Identificação ou confirmação dos atributos de qualidade relevantes ao *software*, podendo-se utilizar uma técnica como o ATAM. A função

principal da ATAM neste roteiro é expor os atributos mais relevantes e a dependência destes com outros quando existir;

- O ciclo de aplicação do roteiro é feito em função do plano estratégico da empresa.
- Mapeamento dos atributos escolhidos com as características da norma 9126, sendo desta forma possível alterá-los e manter certa compatibilidade principalmente para comparação, mesmo que qualitativa, de medidas vinculadas a atributos extintos;
- Registro dos atributos de forma padronizada.

3.1.4 Definição ou confirmação das medidas

A partir dos atributos definidos, aplica-se o GQM para definir as medidas que melhor os representem. Deve-se observar o impacto na equipe no *software* para a obtenção das medidas, de modo a torná-lo mínimo. As medidas definidas podem ser:

- Por período: por exemplo, o valor médio do tempo de renderização da página, em caso de sistemas web, no período de um mês;
- Na conclusão: pode-se verificar ao final da tarefa o tempo utilizado e o número de métodos alterados;
- Contínua: acompanha-se o número de erros após a entrega em relação às modificações feitas;

O registro das medidas deve ser feito de modo padronizado permitindo comparações.

3.1.5 Avaliação da arquitetura quanto a sua degradação

Neste trabalho, a degradação é definida como a impossibilidade de se executar uma modificação ou melhora de um elemento do *software* sem que ocorra piora de outros elementos significativos. Dentro deste conceito a avaliação da arquitetura é feita através da análise das medidas em confronto com os valores obtidos em aplicações passadas do roteiro e com os limites estabelecidos para o *software* e presentes no documento de arquitetura.

O registro dos atributos com a atribuição destas notas pode sofrer várias analises, que devem ser definidas para cada *software* podendo variar da média simples das notas até a definição de pesos para cada atributo que resulta em média ponderada.

Observando-se o resultado, e comparando-se com os outros obtidos da aplicação recorrente do roteiro, pode-se apontar quais atributos estão sofrendo degradação ao longo do tempo, quais estão melhorando e ainda se existe relação entre eles.

3.2 Proposta de roteiro

O roteiro concentra-se na análise das características de negócio registradas no documento de arquitetura. A avaliação e atualização deste documento garantem a inclusão de novas necessidades geradas pelas mudanças no negócio .

O roteiro de avaliação da degradação do *software* está dividido em quatro etapas visando atender cada um dos requisitos expostos no item 3.1. A aplicação é recorrente durante a etapa de manutenção, permitindo a comparação entre as medidas ao longo do tempo. A recorrência deve ser definida pelos envolvidos no projeto. A figura 6 representa a seqüência de atividades do roteiro ao longo do ciclo de manutenção. A modelagem IDEFØ foi utilizada para representá-lo.

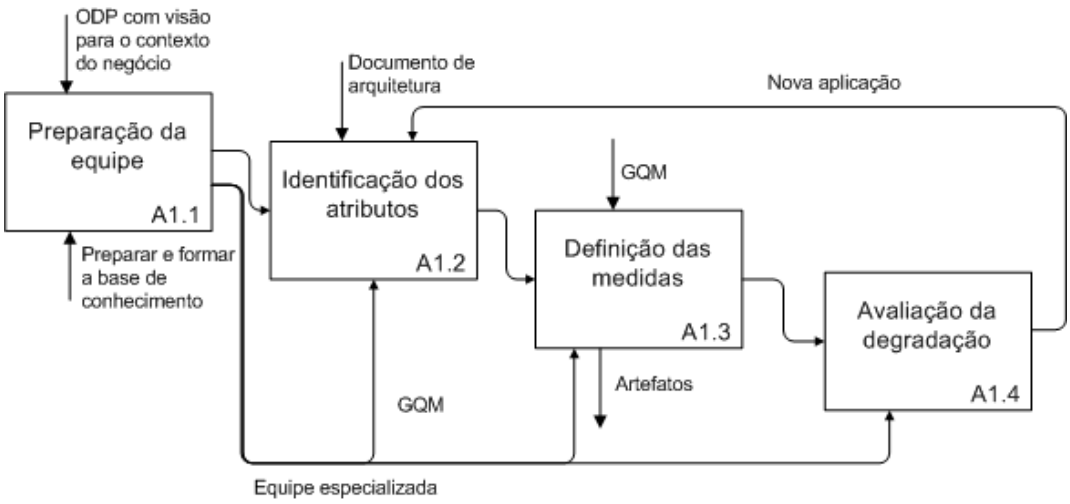

Figura 6: Seqüência de atividades do roteiro.

Fonte: Elaborado pelo autor.

Para avaliar a degradação da arquitetura o roteiro proposto analisa o documento a ele referente e as necessidades atuais do negócio. Algumas dessas necessidades são atendidas ao longo da manutenção, seja pela introdução ou melhoria de uma nova funcionalidade ou pela melhoria de algum aspecto relacionado à utilização do *software*. Essas manutenções podem gerar degradação em outro aspecto do produto, sendo esta, justamente, uma das situações que o roteiro pretende identificar. Outro aspecto que o roteiro pode identificar é algum desalinhamento entre as necessidades do negócio e o *software* existente.

As atividades do roteiro utilizam os conceitos de modelo de referência de arquitetura (RM-ODP) e o método para avaliá-la (ATAM) ambos descritos no capitulo 2.. Outras técnicas, por exemplo, para condução de reuniões ou extração de requisitos, podem ser agregadas à aplicação do roteiro como facilitadoras, mas não serão tratadas neste trabalho.

O experimento de uso do capitulo 4 apresenta uma aplicação prática do roteiro e seus resultados. Os itens seguintes tratam de aspectos da aplicação de cada uma das etapas do roteiro.

3.2.1 Preparação da equipe

Após definida, a preparação da equipe é feita com a apresentação do diagrama de sequência de atividades, conforme a figura 6. A equipe decide se devem ser agregadas outras ferramentas ao roteiro para atender às atividades propostas e também identifica os participantes da atividade de identificação ou confirmação dos atributos relevantes ao *software*.

3.2.2 Identificação ou confirmação dos atributos

A equipe que aplica o roteiro reúne os interessados para verificar se o documento de arquitetura reflete as necessidades mais atuais que o *software* deve atender. Se existir alguma alteração no documento, esta é feita imediatamente e submetida à aprovação dos presentes.

Com o documento atualizado é feito levantamento dos atributos relevantes. Neste momento a técnica exposta no ATAM pode ser utilizada para auxiliar os envolvidos. Definem-se os principais requisitos do *software* e aqueles que devem ser medidos e acompanhados. Com os requisitos escolhidos, faz-se seu registro em uma tabela conforme o exemplo da tabela 3.

A etapa seguinte é a análise da equipe para verificar se os requisitos escolhidos são de responsabilidade do *software*. Caso o *software* tenha uma arquitetura em camadas e opte-se por fazer a análise de cada uma delas isoladamente, recomenda-se separar os requisitos por camadas.

Tabela 3: Tabela de requisitos escolhidos.

#	**Requisito**

R1	**Todo o registro da comunicação entre o sistema e os fornecedores deve ser armazenado. Este registro é o conjunto de requisição e resposta e é um XML (*eXtensible Markup Language*).**
R2	**O tempo de resposta da pesquisa não pode aumentar, mas deve-se aumentar a quantidade de recursos de tratamento da resposta (cálculos de *markups* diferenciados, bloqueio de produto por fornecedor ou cliente, avaliação de tipos de quartos, informações personalizadas por produto).**

Fonte: Elaborado pelo autor.

Identifica-se o atributo de qualidade a ser observado para cada requisito. Com estes atributos definidos e aprovados, monta-se a tabela de registro deles, conforme o modelo apresentado na tabela 4. Esta tabela se relaciona com os requisitos e a características de qualidade, conforme a norma ISO 9126, o que permite realizar a análise macro independente da modificação, ao longo do tempo, de um ou outro atributo.

Tabela 4: Registro das características da ISO 9126 em relação aos atributos.

Requisito	Atributo de qualidade	Característica 9126
R1	Rastreamento da comunicação	Funcionalidade
R2	Tempo de tratamento dos dados de resposta de pesquisa	Eficiência

Fonte: Elaborado pelo autor.

As atividades de definição de requisitos, de atributos e seu mapeamento com a norma 9126 devem ser feitos a cada aplicação do roteiro, de modo a permitir que novas necessidades do negócio tenham seu acompanhamento e seu efeito no *software* comparadas a outras definidas e medidas ao longo do tempo.

3.2.3 Definição ou confirmação das medidas

Partindo dos atributos definidos, utiliza-se o método GQM para determinar medidas que possam atendê-los. O método é um modelo para definir medidas e modos de interpretação (BASILI, 2009) e pode resultar mais de uma medida para um mesmo atributo.

O método GQM é um modelo hierárquico com os seguintes níveis:

- Objetivo (Goal): são os atributos definidos
- Questão (Question): é o que se pretende verificar
- Metrica (Metric): é a medida propriamente dita, representa a resposta para a questão. As medidas devem ser acompanhadas do procedimento de coleta e interpretação.

As medidas, procedimento de coleta e interpretação são registrados conforme modelo da tabela 5.

Tabela 5: Exemplo de tabela para registro de medidas.

Requisito	Questão	Medida / Indicador	Procedimento
R1	Q1: Número de erros de registro de comunicação entre o fornecedor e o *software*.	M1: Número absoluto de erros por mês.	Indicação dos erros de registro das informações de comunicação entre fornecedor e *software*.
		I1: Relação entre os erros e quantidade de mensagens trocadas.	Relação entre os erros de registro e o total de comunicações feitas.
R2	Q2: Injeção de tempo para tratar a resposta da pesquisa do *software*.	M3: Tempo total do processo.	Contar o tempo entre o início e fim do processo de tratamento dos dados.

Fonte: Elaborado pelo autor.

Para uma comparação mais ampla, utiliza-se o conceito de suficiência de Chung (2000) aplicado às características da norma de qualidade, que é registrado conforme exemplo da tabela 6. Neste trabalho foi feita uma adaptação do conceito, que originalmente trata de requisitos, para tratar atributos. O conceito de suficiência adaptado é assim classificado:

- Degrada (-2): quando o atributo indica degradação da arquitetura;
- Afeta (-1): o atributo tem tendência negativa na arquitetura;
- Neutro (0): o atributo não ajuda ou prejudica a arquitetura;
- Ajuda (+1): o atributo tem tendência positiva na arquitetura;
- Melhora (+2): o atributo melhora a arquitetura;

Tabela 6: Registro de suficiência por característica da norma ISO 9126.

Característica	Suficiência
Funcionalidade	+1
Eficiência	-1

Fonte: Elaborado pelo autor.

Este artefato representa o nível mais alto na visão de degradação do *software* em relação a características da norma.

3.2.4 Avaliação da arquitetura quanto a sua degradação

Com as medidas definidas e tomadas, faz-se a análise ao longo do tempo e avalia-se a tendência de degradação do *software* em relação a sua arquitetura, consistindo na indicação da necessidade de se iniciar um ciclo de trabalho para alterá-lo. Caso a avaliação tenha aplicação em um sistema, pode-se direcionar o

esforço para os *software* que estão mais degradados ou que se degradam mais rapidamente.

A análise da degradação da arquitetura é feita confrontando os valores obtidos com os limites estabelecidos no documento de arquitetura e analisando a série histórica deles, quando disponível.

O confronto direto de valores, o medido e o limite estabelecido, indicam o estado atual do *software*. Já a série histórica indica como o seu comportamento está mudando ao longo do tempo.

Apesar de as medidas apresentarem uma tendência, a análise é feita observando o conjunto: medidas e suficiência.

Para ilustrar o conceito, a figura 7 apresenta a relação entre o tempo dedicado a testes e a quantidade de funções cobertas por estes. Neste caso, quanto maior o valor, mais funções são testadas por hora. A diminuição do valor da medida indica que a capacidade de teste do *software* está piorando e exigindo mais tempo da equipe, aumentando seu custo. Neste exemplo, observam-se melhoras nas medidas ao longo do tempo, as quais estão relacionadas a intervenções no *software* com o objetivo de melhorar o indicador. A tendência geral, entretanto, é de piora.

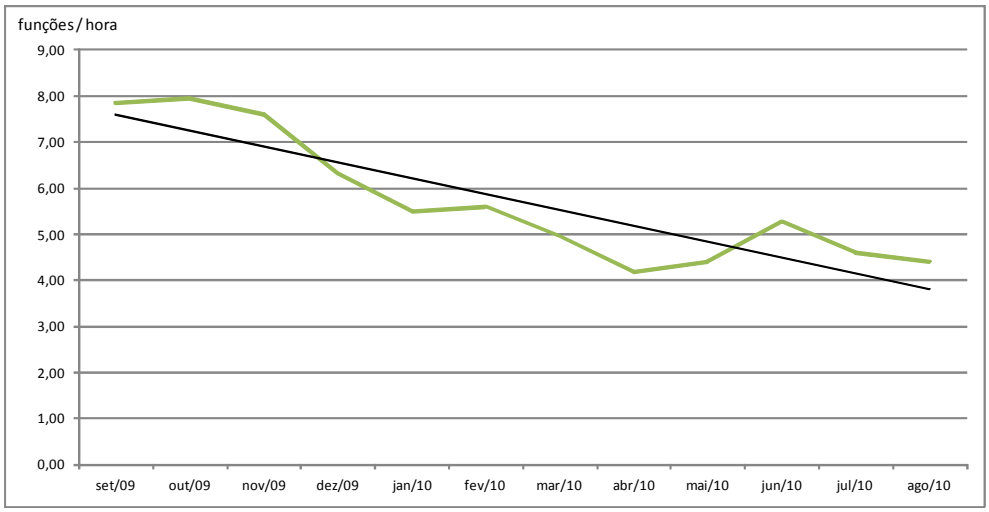

Figura 7: Gráfico da medida de eficiência de testes ao longo do tempo.

Fonte: Elaborado pelo autor.

Apesar de melhoras pontuais, a medida mostra que as intervenções nos *software* tiveram impacto direto na capacidade de teste.

Complementando a análise das medidas é feita a verificação dos valores de suficiência. Em conjunto, medidas e suficiência permitem enxergar a parte do *software* que deve sofrer intervenção ou a necessidade de substituí-lo.

4 EXPERIMENTO DE USO

A aplicação do roteiro foi feita em um *software* voltado ao mercado de turismo chamado Cangooroo, desenvolvido pela T4W. A aplicação do roteiro levou em conta a análise de dados colhidos a partir de 2009.

Este capitulo apresenta o contexto da aplicação e do negócio e sua aplicação. A aplicação está dividida em preparação da equipe, identificação dos atributos de qualidade, definição de medidas e avaliação da degradação da arquitetura.

4.1 O contexto da aplicação e do negócio

O Cangooroo teve seu desenvolvimento iniciado no final de 2008 e lançamento em março de 2009. Desde então a base de clientes cresce consistentemente e tem incluído as maiores operadoras de turismo do Brasil, sendo utilizado tanto por aquelas que vendem para as agências (suas clientes diretas) quanto para o publico final. Com versões em inglês e espanhol atende o mercado externo ainda timidamente.

O desenvolvimento está baseado na plataforma *Microsoft .NET* com a utilização de poucos componentes de terceiros.

O sistema executa a pesquisa de disponibilidade de hotéis, traslados, atividades, aluguel de carros, trens, circuitos e seguros – estes serviços em conjunto são chamados genericamente de serviços terrestres – em diversos fornecedores e apresenta para o cliente a melhor tarifa disponível[2] e todas as encontradas, contando atualmente com as seguintes integrações feitas através de *WebServices*:

- 40 integrações com fornecedores de hotéis;

2 A definição de melhor tarifa disponível para este sistema é aquela com o menor preço dentre as que pode ser confirmadas imediatamente.

- 10 integrações com fornecedores de traslados;
- 10 integrações com fornecedores atividades;
- 5 integrações com fornecedores de aluguel de carros;
- 3 integrações com fornecedores de trem;
- 2 integração com fornecedores de circuitos;
- 2 integrações com fornecedores de seguros de viagem.

O sistema consulta os fornecedores, torna suas regras homogêneas e transmite a resposta ao usuário. São feitas em média 20 requisições por segundo[3], número que pode dobrar quando uma operadora anuncia novas promoções. Cada requisição resulta, em média, em dez requisições para fornecedores, ou seja, 200 requisições por segundo em média. O volume de vendas total em 2010 ultrapassou os US$ 30 milhões.

O sistema está dividido em camadas, cada qual com uma velocidade esperada de desenvolvimento e um foco. As camadas são:

- Interface com o usuário: o objetivo é apresentar uma funcionalidade nova por semana, com facilidade de uso e rapidez na transferência de dados entre o servidor e navegador do usuário.
    - Todo o código HTML e *Java Script* são otimizados ao máximo para serem transferidos ao navegador do cliente no menor tempo possível.
    - A resposta enviada ao cliente deve ser a mais completa, evitando-se assim qualquer requisição ao servidor e permitindo a rápida manipulação das informações;

3 O cálculo do valor médio das requisições é feito entre 8h e 22h.

  - Novas funcionalidades devem ser intuitivas para o usuário.

- Regras de negócio: necessita ser flexível para absorver diversos modos de cálculos de margem e comissão por fornecedor, destino ou produto. Deve ainda permitir bloqueio de produtos (hotéis, passeios, aluguel de carros) em respostas de pesquisa, aplicação de promoções instantâneas e relatórios. O foco é a estabilidade e a precisão.

- Conexão com os fornecedores: pesquisas feitas em paralelo utilizando recursos de compressão na comunicação sempre que possível. As regras de cada fornecedor devem ser adequadas ao padrão de saída e, assim, unificadas. O foco é a facilidade de inclusão de novos fornecedores. A velocidade de desenvolvimento alvo é a seguinte:

  - Fornecedores de hotéis: 5 dias úteis.

  - Fornecedores de aluguel de carros: 10 dias úteis.

  - Fornecedores de traslados: 15 dias úteis.

  - Fornecedores de passeios: 5 dias úteis.

  - Outros fornecedores: 20 dias úteis

A estrutura do projeto é desenhada para permitir que cada uma das três camadas possa ser desenvolvida por equipes diferentes.

O diagrama da figura 8 apresenta uma visão macro da estrutura da aplicação.

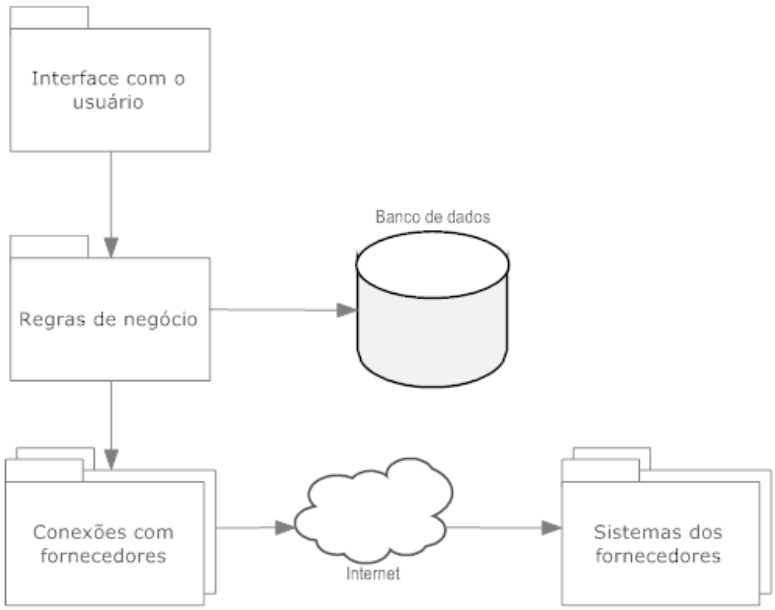

Figura 8: Diagrama de pacotes do sistema analisado.

Fonte: Elaborado pelo autor.

O sistema tem apenas dois concorrentes significativos no Brasil e outros estrangeiros que atuam no mercado brasileiro. Aqui, é líder com maior volume de transações, maior volume de vendas (em valor) e em quantidade de pesquisas simultâneas.

4.2 Preparação da equipe

Para a primeira etapa, preparação da equipe, montou-se o time de aplicação do roteiro, que foi liderado pelo arquiteto de *software* responsável pelo produto, o líder dos analistas de negócio e o analista de qualidade.

Foram feitos treinamentos nos profissionais envolvidos em relação a seus objetivos, sendo:

- Diretores e área comercial: apresentação do formato de análise, impacto em custo para aplicação do roteiro e benefícios do acompanhamento do sistema em relação ao atendimento do negócio quanto a decisões de modificação ou troca de cada camada.

- Equipe de atendimento: apresentação do roteiro, da estrutura do documento de arquitetura e a padronização da entrada de informações colhidas junto aos clientes.

- Equipe técnica: formada por representantes da área de desenvolvimento e infraestrutura, tinha como objetivo, além de apresentar necessidades técnicas, é avaliar e estimar as necessidades das outras equipes em aspectos de técnica, tempo e esforço.

Foram feitas algumas reuniões após os treinamentos para praticar a aplicação do roteiro. Os participantes atribuíram a essa iniciativa a diminuição do tempo de reunião para identificação de atributos.

A preparação da equipe teve como principal consequência o envolvimento dos participantes e a diminuição da resistência aos resultados obtidos.

4.3 Identificação dos atributos de qualidade

O material preparatório e as discussões preliminares com as equipes resultaram em requisitos funcionais e não funcionais com justificativa em sua visão. Este material ajudou a nivelar o conhecimento de todos e os termos utilizados, sendo inclusive criado um glossário que passou a fazer parte do projeto.

4.3.1 Principais demandas do negócio

Os participantes avaliaram o documento de arquitetura para identificar se existiam novas demandas de qualidade ou funcionalidade e se demandas existentes

eram mais relevantes para o atual contexto de negócio. Foram selecionados dez requisitos para acompanhamento, funcionais e não funcionais, como apresentado na tabela 7.

Tabela 7: Tabela de requisitos priorizados.

#	Requisito
R1	**Todo o registro da comunicação entre o sistema e os fornecedores deve ser armazenado. Este registro é o conjunto de requisição e resposta e é um XML (*eXtensible Markup Language*).**
R2	**O tempo de resposta da pesquisa não pode aumentar, mas deve-se aumentar a quantidade de recursos de tratamento da resposta (cálculos de *markups* diferenciados, bloqueio de produto por fornecedor ou cliente, avaliação de tipos de quartos, informações personalizadas por produto).**
R3	**Integração de novos fornecedores nos prazos alvo.**
R4	**Compatibilidade entre os principais navegadores do mercado, incluindo suas versões para dispositivos móveis.**
R5	**Inclusão de novos serviços, tais como: conexões com meios de pagamento, integrações com *software* de gestão entre outros.**
R6	**Facilidade de utilização do grande conjunto de parametrizações.**
R7	**Controle instantâneo do resultado financeiro das vendas, ajustes *online* de percentuais de ganho e pagamento de comissão para ações de incentivo de venda.**
R8	**Diminuição do tempo de interrupção do sistema devido à realização da cópia de segurança do banco de dados. O tempo atual nos maiores**

	cliente é de 15 minutos.
R9	**Tempo médio de apresentação da tela de uma resposta de pesquisa de 20 segundos.**
R10	**Controle do risco de manutenção do sistema em relação a novos erros, principalmente os decorrentes da manutenção. A equipe decidiu estabelecer a relação de um erro a cada quatro mil linhas de código.**

Fonte: Elaborado pelo autor.

Como o *software* está dividido em camadas desenvolvidas por equipes diferentes e com objetivos distintos, decidiu-se organizar os requisitos por camada. Também foi feita a classificação por requisito funcional e não funcional. Embora não prevista no roteiro, esta classificação foi realizada com o intuito de facilitar a comunicação e direcionamento das análises e o resultado registrado na tabela 8.

Tabela 8: Tabela de requisitos por camada.

#	Camada	Tipo
R1	**Conexão com fornecedores**	**Funcional**
R2	**Regra de negócio**	**Não funcional**
R3	**Conexão com fornecedores**	**Não funcional**
R4	**Interface com o usuário**	**Funcional**
R5	**Conexão com fornecedores**	**Funcional**
R6	**Interface com o usuário**	**Não funcional**
R7	**Regra de negócio**	**Funcional**
R8	**A responsabilidade não é do *software***	**Não é do *software***

R9	**Interface com o usuário**	**Não funcional**
R10	**Regra de negócio**	**Não funcional**

Fonte: Elaborado pelo autor.

A organização proposta na tabela 8 elimina o requisito R8 pelo fato de ele não ter seu atendimento diretamente ligado ao *software* e sim a uma questão de infraestrutura.

4.3.2 Definição dos atributos de qualidade

Os atributos de qualidade, apresentados na tabela 9, foram definidos em função dos requisitos apresentados.

Tabela 9: Tabela de características da ISO 9126 em relação aos atributos.

Requisito	Atributo de qualidade	Norma 9126
R1	**Rastreamento da comunicação.**	**Funcionalidade**
R2	**Tempo de tratamento dos dados da resposta da pesquisa.**	**Eficiência**
R3	**Tempo máximo para integração.**	**Manutenabilidade**
R4	**Compatibilidade entre navegadores.**	**Portabilidade**
R5	**Novas funcionalidades.**	**Funcionalidade**
R6	**Parametrizações existentes.**	**Usabilidade**

R7	**Ajustes de parâmetros de venda.**	**Usabilidade**
R9	**Tempo máximo de transferência entre o servidor e o navegador do cliente.**	**Eficiência**
R10	**Risco da manutenção**	**Manutenabilidade**

Fonte: Elaborado pelo autor.

4.4 Definição das medidas

A equipe passou então a definir medidas para os requisitos, sendo para tanto aplicado o método GQM, que também exige a definição para a forma de interpretação. As medidas definidas foram registradas conforme tabela 10.

Tabela 10: Tabela de registro das medidas.

Requisito	Questão	Medida	Procedimento
R1	Q1: Número de erros de registro de comunicação entre o fornecedor e o *software*.	M1: Número absoluto de erros por mês.	Registro dos erros de inserção das informações de comunicação entre fornecedor e *software*.
		M2: Relação entre os erros e quantidade de mensagens trocadas.	Relação entre os erros de registro e o total de comunicações feitas.
R2	Q2: Injeção de tempo para tratar a resposta da pesquisa com as parametrizações do *software*.	M3: Tempo total do processo.	Contagem do tempo entre o início e fim do processo de tratamento dos dados.

R3	Q3: Tempo para integração de novos fornecedores em serviços existentes	M4: Tempo utilizado pela equipe.	Registro do tempo trabalhado por esta equipe.
R4	Q4: Compatibilidade entre navegadores	M5: Número de chamados a equipe de atendimento sobre erros relacionados à compatibilidade.	Análise dos chamados registrados.
R5	Q5: Integração com novos serviços	M6: Relação entre o tempo investido e o número de rotinas desenvolvidas.	Garantia de que esta elação deve estar abaixo de 2h. Este valor é a referência da empresa em seus projetos.
R6	Q6: Utilização das parametrizações existentes.	M7: Medida qualitativa com opções: Ruim, Neutro e Bom.	Verificação da quantidade de cada opção em relação ao total de respostas obtidas.
R7	Q6: Utilização das parametrizações existentes.	M7: Medida qualitativa com opções: Ruim, Neutro e Bom.	Verificação da quantidade de cada opção em relação ao cont e respostas obtidas.
	Q7: Assertividade dos ajustes de parâmetros de venda.	M8: Número absoluto de erros de cálculo na venda	Análise dos chamados registrados.
R9	Q8: Tempo máximo de transferência entre o servidor e o navegador do cliente.	M9: Tempo entre o início do envio dos dados para o cliente e a conclusão da renderização da página no cliente.	Registro deste tempo utilizando-se um *script* executado no final da página e do lado do cliente que envia para o servidor uma referência. O servidor faz a diferença entre o momento que enviou a página e a execução do *script*.
R10	Q9: Risco de	M10: Relação entre a	Registro de todas as modificações,

	manutenção	quantidade de linhas de código afetadas em manutenções e erros relatados.	evolutivas e corretivas, e o número de erros reportados na equipe de qualidade e por clientes com relação às modificações feitas. A referência é 0,25 erros por kLoC (mil linhas de código ou kilo Line of Code).
		M11: Número de testes em métodos por hora	Relação entre o número de métodos testados e a quantidade de horas para o teste. O objetivo é observar se existe maior dificuldade em efetuar testes ao longo do tempo, o que se traduz em valor menor na relação entre as grandezas observadas.

Fonte: Elaborado pelo autor.

Os dados coletados na primeira aplicação do roteiro e que têm limite estabelecido foram validados. Os outros foram registrados para estabelecer tendências durante o ciclo de manutenção do *software*.

4.5 Avaliação da degradação da arquitetura

A avaliação da arquitetura foi feita com a apresentação dos resultados das medidas a todos os envolvidos. A cada aplicação do roteiro, a apresentação foi acompanhada de um plano de ação sempre que a medida se mostrou fora dos padrões ou com tendência de sair dos limites.

Com a aplicação recorrente do roteiro, os dados acumulados e as decisões para melhoria das medidas tomadas formam um histórico com capacidade de indicar a tendência de degradação da arquitetura do *software*.

A análise do Cangooroo foi feita em duas frentes:

- Interface com o usuário e regras de negócio.
- Conexões com fornecedores.

Para ilustrar a análise, são apresentadas a seguir: o resultado de medidas de alguns dos requisitos definidos e as ações tomadas.

O requisito R9 – Tempo médio de apresentação da tela de uma resposta de pesquisa de 20 segundos – foi medido ao longo de doze meses. Durante este período algumas ações foram tomadas para manter o tempo dentro dos limites. . A figura 9 apresenta o gráfico do tempo de resposta, sendo que a medida é feita no momento em que se inicia o envio das informações para o navegador do cliente e a execução do comando no final do carregamento da página.

No gráfico a linha dos 20 segundos está reforçada como sendo a medida alvo. A reta é a tendência da medida e a linha mais forte indica os valores obtidos ao longo do tempo.

O pico de fevereiro de 2010 desencadeou um ajuste nos *scripts* e nas folhas de estilo e a modificação da configuração de *cache* do servidor de internet. Os *scripts* e as folhas de estilo foram compactados[4] e a configuração de *cache* modificada para manter os arquivos com poucas mudanças por um mês no navegador, como por exemplo, arquivos de imagens e folhas de estilo.

4 A ferramenta utilizada foi a YUI Compressor, mantida pela empresa Yahoo!, disponível gratuitamente no sitío: http://developer.yahoo.com/yui/compressor/.

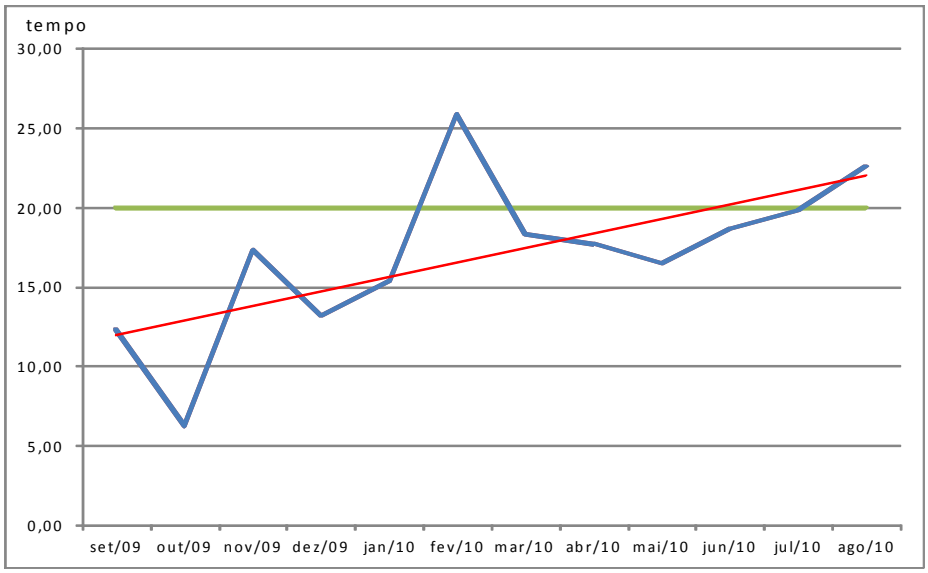

Figura 9: Tempo médio entre o início do envio da página e o carregamento completo no navegador.

Fonte: Elaborado pelo autor.

Tal ajuste resultou na diminuição de tempo nos meses seguintes, mas, ao incluir um novo filtro de dados, o tempo voltou a subir.

O requisito R10 – Controle do risco de manutenção do sistema em relação a novos erros, principalmente os decorrentes da manutenção – também medido por doze meses, representa o risco da manutenção. O gráfico da figura 10 mostra esta relação e a tendência da medida 10.

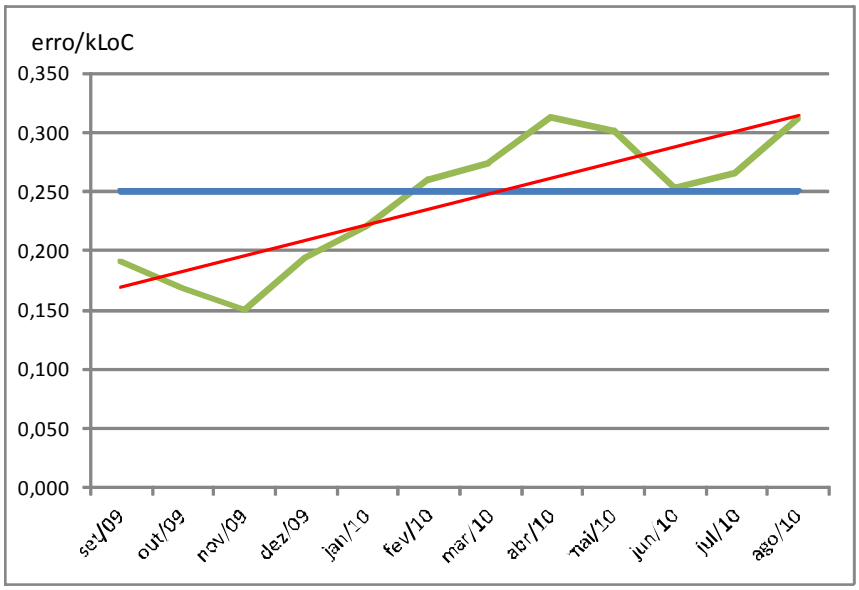

Figura 10: Erros por kLoC.

Fonte: Elaborado pelo autor.

A medida demonstra que o *software* está fora do limite estabelecido desde fevereiro de 2010. A partir de dezembro de 2009 houve um aumentou na equipe de desenvolvedores do produto. O aumento dos erros é atribuído, em grande parte, a entrada dos novos profissionais sem treinamento adequado.

Ainda para o mesmo requisito, a medida 11 representa o número de funções testadas por hora. Estes testes podem ser:

- Testes unitários, executados automaticamente.
- Testes de qualidade, executados por profissionais da equipe de qualidade.
- Testes de funcionalidade, executados por outros profissionais da equipe de desenvolvimento. Este teste, no ambiente da empresa, significa teste de carga, cenários de utilização e conformidade com a especificação.

A medida 11 não tem um limite definido – quanto menor a relação, mais tempo é necessário para colocar uma modificação em produção. A figura 11 mostra o gráfico com a quantidade de métodos testados por hora a cada mês durante doze meses.

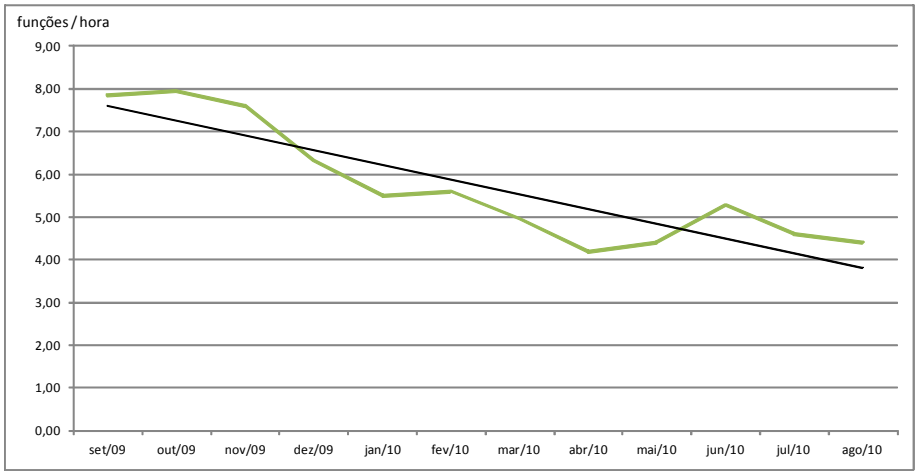

Figura 11: Quantidade de métodos testados por hora.

Fonte: Elaborado pelo autor.

A medida 11 pode ser relacionada com a medida 10 mostrando que a dificuldade em testar resulta em um número maior de erros observados pelo usuário final.

O conceito de suficiência é apresentado para o primeiro e segundo trimestre de aplicação do roteiro. A comparação dos valores indica em que pontos as camadas analisadas perderam mais qualidade de acordo com as medidas obtidas, sua comparação tem valor quando os resultados são ponderados ao longo do tempo. A tabela de suficiência foi utilizada como evidência complementar da degradação da arquitetura.

Com as medidas da aplicação do roteiro foi sugerido, apesar de ainda não existir uma insatisfação dos clientes, o planejamento para a renovação da interface com o usuário e regras de negócio. Esta sugestão foi acompanhada da necessidade do negocio em ampliar os parâmetros de configuração do *software*.

A partir dos dados obtidos, foi apresentado um conjunto de ações para mitigar os problemas identificados como os que mais contribuíram para a degradação da arquitetura.

Tabela 11: Registro de suficiência por característica.

Característica	Suficiência – Trimestre 1	Suficiência – Trimestre 2
Funcionalidade	+2	+1
Eficiência	+1	0
Manutenabilidade	-1	-2
Portabilidade	-1	-1
Funcionalidade	+2	+1
Usabilidade	+2	+1

Fonte: Elaborado pelo autor.

Em abril de 2010 foi montada uma nova equipe que começou a desenvolver a nova interface de usuário e refazer o código de regras de negócio. A aplicação do roteiro continuou a ser feita.

O mesmo roteiro foi aplicado na camada de conexões com fornecedores, sendo que, neste caso, não houve degradação; ao contrário, os indicadores melhoraram ao longo do tempo. Esta camada foi mantida.

4.6 Análise dos resultados obtidos

A aplicação do roteiro gerou uma base de conhecimento para a empresa, permitindo assim a comparação de dois *software* diferentes, sendo que o comportamento observado em um pode subsidiar ações em outro, de modo a se prevenir situações que levem a degradação da arquitetura.

Embora as visões por meio das medidas de suficiência atribuídas às características da norma 9126 não fossem de interesse deste estudo, constituíram-se como um importante resultado observado na aplicação do roteiro.

A aplicação do roteiro por um ano produziu evidências suficientes a todos em relação à degradação da camada de interface com o usuário e regras de negócio. O plano de renovação apresentado, além das inovações técnicas, inclui ações de treinamento e preparação de profissionais para atuar no projeto.

O investimento para renovação foi aprovado rapidamente e com menos atrito do que outros investimentos do mesmo tipo. A diretoria também ficou mais confiante de que a percepção geral de qualidade da nova versão seria maior.

O departamento comercial passou a ter mais confiança na equipe de desenvolvimento, resultando na diminuição de conflitos.

Apesar de o caso apresentado ter dirigido a decisão de renovação, o roteiro proposto não indica se o *software* deve ou não ser substituído. Isto porque uma intervenção pode resultar na melhora do indicador e desta forma a relação custo benefício de manter o *software* influência na decisão de não renovar.

5 CONCLUSÃO

O objetivo deste trabalho foi apresentar um roteiro que pudesse determinar indicadores específicos para o *software* analisado, a fim de diagnosticar a degradação da sua arquitetura durante a fase de manutenção, contando, para isso, com os diversos níveis da organização.

O investimento e a velocidade para responder a mudanças de negócio tornam estratégico o controle do *software*. Uma das ações de controle é decidir sobre sua continuidade ou sua substituição em tempo adequado para não comprometer o negócio. O roteiro apresentado auxilia a decisão com medidas objetivas e acordadas entre os envolvidos.

Pode-se ainda afirmar que a base de conhecimento gerada com a aplicação do roteiro contribui para a maturidade da arquitetura. Os dados de uma arquitetura servem de subsídio para outra e podem ser combinados, quando necessário.

O experimento de uso apresentou evidências da necessidade de atualização de uma camada ante outras analisadas. Esse resultado, baseado em dados acordados entre todos, tornou a percepção da degradação concreta ao invés de baseada em termos técnicos com pouco sentido para a área de negócio. Para os envolvidos, a decisão de troca foi menos conflitante e mais objetiva.

O conceito de suficiência atendeu plenamente a visão abstraída da degradação do *software*. Esta visão deixa a análise menos dependente de um atributo especifico. Este experimento não teve um caso de troca de indicador, situação onde esta abstração seria utilizada para comparar a evolução da arquitetura.

Entre a equipe técnica diminuiu a sensação de que a troca ocorre por conta da decisão de alguns membros da equipe com poder de influência e passou a ser mais pautada em critérios coletivos, ponto que foi apontado como importante para reduzir a resistência de todos os membros em trabalhar na mudança.

Durante a aplicação do roteiro, principalmente nas apresentações de resultados, a percepção da necessidade de mudança e a diferença entre uma manutenção com melhora efetiva e outra com melhoria temporária se tornaram evidentes. Pode-se afirmar que o produto foi impactado positivamente em relação a sua competitividade no mercado, antecipando mudanças dos concorrentes e mantendo entre seus clientes a percepção dos seus diferenciais. O sistema continua crescendo e conquistando novos clientes.

A maturidade dos envolvidos durante a aplicação do roteiro tornou as equipes técnicas, de suporte e comercial mais próximas, este efeito foi ressaltado pelos participantes apesar de não ser o objetivo do roteiro.

5.1 Trabalhos futuros

O roteiro deve ser testado em diferentes cenários e em outros tipos de *software* com o objetivo de refinar e aperfeiçoar suas etapas. Um dos cenários recomendados é utilizar o roteiro para *software* prontos que são implantados e customizados para sua utilização.

A observação da base de conhecimento gerada pode suscitar previsibilidade de comportamento em *software*. O desafio é organizar os dados e sistematizar a sua interpretação.

Outra extensão ao roteiro é a geração de indicadores relacionados à maturidade de processos da empresa, tais como: desenvolvimento e manutenção, analisados à luz dos resultados obtidos durante a aplicação do roteiro apresentado.

Pode-se ainda estender o roteiro para definir o momento de troca de um *software*. Com a aplicação em mais cenários pode-se observar um comportamento que gere indicação de troca.

Por fim, é importante destacar que procedimentos para selecionar atributos e medidas podem ser aprofundados com a utilização de técnicas formais anexadas ao roteiro.

Referências

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR ISO/IEC 9126-1**: Engenharia de *software* - Qualidade de produto Parte 1: Modelo de qualidade. Rio de Janeiro, 2003. 21p.

_____. **NBR ISO/IEC 12207**: Engenharia de sistemas e *software* - Processos de ciclo de vida de *software*. Rio de Janeiro, 2009. 121p.

ABRAN, Alain; MOORE, James W. (Ed.). **SWEBOK**: Guide to the *Software* Engineering Body of Knowledge. Los Alamitos, California: IEEE Computer Society, 2004. 204 p.

BASILI, V.; CALDIERA, G.; ROMBACH, H. D. **Goal Question Metric Approach**. Disponível em: <http://www.cs.umd.edu/users/mvz/handouts/gqm.pdf>. Acesso em 22 set. 2009.

BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. ***Software* Architecture in Practice**. 2. ed. Boston: Addison Wesley Professional, 2003. 560 p.

BRCINA, Robert; BODE, Stephan; RIEBISCH, Matthias. Optimisation Process for Maintaining Evolvability during *Software* Evolution. In: **Proc. 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS2009)**, San Francisco, CA, USA, April 13-16. IEEE CPS, 2009, p. 196-205.

CHUNG, Lawrence. **Non-Functional Requirements in *Software* Engineering**. Massachusetts, EUA: Kluwer Academic Publishers, 2000. 439p.

DIAS, Sérgio A. **Roteiro para atualização tecnológica de sistemas legados baseado na avaliação arquitetural e engenharia guiada por valor**. 2010. 75 f. Dissertação (Mestrado profissional) – Coordenadoria de Ensino Tecnológico, Instituto de Pesquisas Tecnológicas do Estado de São Paulo, São Paulo, 2010.

EICK, Stephen G.; GRAVES, Todd L.; KARR, Alan F.; MARRON, J. S.; MOCKUS, Audris. Does Code Decay? Assessing the Evidence from Change Management Data. **IEEE Transactions on *Software* Engineering**, v.27 n.1, p.1-12, 2001.

GROTTKE, M.; MATIAS JR, R.; TRIVEDI, S. The fundamentals of *software* aging. In: **1st International Workshop of *Software* Aging and Rejuvenation (WoSAR/ISSRE) / 19th IEEE International Symposium on *Software* Reliability Engineering**, Seattle, WA, USA, November 11, 2008.

GUIMARÃES, Júlio. **Método para manutenção de sistema de *software* utilizando técnicas arquiteturais**. 2008. 101 f. Dissertação (Mestrado) - Departamento de Engenharia de Computação e Sistemas Digitais, Escola Politécnica-USP, São Paulo, 2008.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE 1471**: Recommended Practice for Architectural Description of *Software*-Intensive Systems. Nova Iorque, Estados Unidos, 2000. 29p.

JERMAKOVICS, Andrejs; SCOTTO, Marco; SUCCI, Giancarlo. Visual identification of *software* evolution patterns. In: **Ninth international workshop on Principles of *software* evolution in conjunction with the 6th ESEC/FSE joint meeting - IWPSE '07**, Dubrovnik, Croatia, 03 set. 2007.

KAZMAN, Rick; KLEIN, Mark; CLEMENTS, Paul. **ATAM: Method for Architecture Evaluation**, *Software* Engineering Institute. Pittsburgh: 2000. 83p.

KUDRASS, Thomas. **Describing Architectures Using RM-ODP**. In: H. Kilov, K. Baclawski (Eds.) Practical Foundations of Business System Specifications, Kluwer Academic Publishers, 2003. p. 231-245.

MAIER, Mark W.; EMERY, David; HILLIARD, Rich. ANSI/IEEE 1471 and systems engineering. **Systems Engineering**, Malden, p. 257-270. 18 jun. 2004.

MEY, Thomas. Rewrite Patterns for the replacement and maintenance of *software* systems. In: **Workshop of 11th European Conference on Pattern Languages of Programs**, Irsee, Germany, 5-9 jul. 2006.

PARNAS, David. *Software* Aging. In: **Proceedings of the 16th international conference on *Software* engineering**, [S.I.], p. 279-287. 1994.

PRESSMAN, Roger S.. ***Software* Engineering**: A practitioner's approach. 7. ed. New York: Mcgraw-Hill, 2009. 928 p.

QUEIROZ, André Luiz Pimentel; ANQUETIL, Nicolas; OLIVEIRA, Káthia Marçal de. Avaliação Pró-ativa da Deterioração de Sistemas de Informação por meio de Medidas de Gestão. **Revista de Informática Teórica e Aplicada - RITA**, Porto Alegre, v. 16, n. 1, p.45-68, 2009. Quadrimestral. Disponível em: <http://www.seer.ufrgs.br/index.php/rita/issue/view/763>. Acesso em: 23 abr. 2010.

RICHMOND, William; NELSON, Paul; MISRA, Sanjog. An empirical analysis of *software* life spans to determine the planning horizon for new *software*. **Information Technology and Management**, Hingham, Usa, p. 131-149. abr. 2006.

VAN SOLINGEN, R.; BERGHOUT, E. **The Goal/Question/Metric Method - A Practical Guide for Quality Improvement of *Software* Development**, McGraw-Hill Publishing Company, Maidenhead, England, 1999, 216p.

VAROTO, Ane Cristina. **Visões em arquitetura de *software***. 2002. 108 f. Dissertação (Mestrado) - Departamento de Engenharia de *Software*, IME-USP, São Paulo, 2002.

APÊNDICE A – Contexto do *software* e seus concorrentes

O *software* Cangooroo foi desenvolvido para o mercado de turismo, um dos que mais cresce no Brasil. Seu desenvolvimento começou em meados de 2008 e seu objetivo inicial foi preencher um espaço importante no mercado de turismo emissivo[5].

O mercado, para este cenário, tem algumas centenas de empresas multinacionais, chamados comumente de fornecedores, que disponibilizam sistemas para reserva de quartos de hotel e outros serviços para as operadoras de viagem. O agente de viagem, responsável pela venda destes produtos, deve então contatar a operadora e solicitar um orçamento para determinada viagem. A operadora acessa os vários sistemas, colhe os valores e envia o orçamento ao agente.

Com a evolução dos *software*, os fornecedores passaram a disponibilizar suas informações através de *webservices*. Algumas operadoras iniciaram o desenvolvimento de integrações e empresas de desenvolvimento de *software* enxergaram uma oportunidade. Estas integrações eliminam a necessidade do contato do agente de viagem, dado que o orçamento pode ser feito diretamente pelo sistema da operadora.

Na época em que o Cangooroo foi concebido o mercado nacional contava apenas com um concorrente que tinha alto custo de implantação e era considerado pouco eficiente. A eficiência neste caso estava ligada a quantidade de fornecedores conectados ao software em questão. Seu sistema tinha aproximadamente seis anos e pouco tinha evoluído na visão dos clientes.

---

5 O turismo emissivo é definido como aquele gerado pela viagem de pessoas residentes em um país para outra localidade, e que permaneça em seu destino por mais de 24 horas e menos de um ano no local de chegada, não recebendo remuneração no local visitado.

A proposta do Cangooroo ao mercado foi de um *software* confiável, inclusive para sítios de venda direta ao consumidor com custo razoável e constante aperfeiçoamento.

Em pouco tempo grandes operadoras de turismo adotaram a solução em diversas frentes, venda para agentes de viagem e consumidor final. Este crescimento levou o Cangooroo a situações de utilização não planejadas.

Estava claro que para, manter o crescimento da utilização do *software*, era necessário acompanhar sua arquitetura e manter a mesma qualidade com mais conteúdo de venda para seus clientes.

Enquanto o sistema passava a prover a possibilidade de venda de outros serviços terrestres, como atividades, ingressos, passagens de trem e outros, a equipe de desenvolvimento identificava obstáculos ao crescimento.

Neste momento procuraram-se referências sobre como avaliar o impacto do crescimento rápido do *software* em sua arquitetura, uma vez que muitas das funcionalidades eram desenvolvidas em uma semana por pressão do negócio. Ao mesmo tempo a equipe passou a definir aspectos críticos de observação, estes foram definidos em função das reclamações registradas pela equipe de suporte e comercial.

Este trabalho iniciou-se justamente pela falta de estudos adequados para o cenário. O objetivo foi prover uma visão consensual de evolução para a área comercial e diretoria com o objetivo de planejar ações de manutenção e eventual substituição do *software* e assim manter sua capacidade de competir no mercado.

A estruturação do roteiro trouxe o aspecto consensual para os indicadores analisados dado que a decisão passou a ser baseada em diversos pontos de vista. O ganho para a empresa foi muito sensível. O Cangooroo é hoje o maior *software*

para turismo emissivo do país[6] e prepara sua expansão internacional. Sua abrangência de serviços terrestres é a maior entre seus concorrentes, todos estes serviços com pesquisa e reserva *online* de opções, comparação de opções e indicação de melhor compra.

Entre seus clientes estão as maiores operadoras de turismo do Brasil e em 2011 seu crescimento foi de 400%[7], atingindo US$ 125 milhões em vendas.

6 O Cangooroo é líder por volume de venda, quantidade de reservas, quantidade de integrações e quantidade de serviços disponíveis ao cliente.

7 Fonte: Portal Panrotas, http://www.panrotas.com.br/noticia-turismo/servicos/cangooroo-cresceu-400-em-reservas-em-2011_74495.html. Acessado em: 05/01/2012.