JÚLIO HENRIQUE DOS NOGUEIRA E OLIVEIRA GUIMARÃES

MÉTODO PARA MANUTENÇÃO DE SISTEMA DE SOFTWARE UTILIZANDO TÉCNICAS ARQUITETURAIS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia

Área de Concentração:
Engenharia de Software

Orientador:
Prof. Dr. Reginaldo Arakaki

SÃO PAULO
2008

JÚLIO HENRIQUE DOS NOGUEIRA E OLIVEIRA GUIMARÃES

MÉTODO PARA MANUTENÇÃO DE SISTEMA DE SOFTWARE UTILIZANDO TÉCNICAS ARQUITETURAIS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia

Área de Concentração:
Engenharia de Software

Orientador:
Prof. Dr. Reginaldo Arakaki

SÃO PAULO
2008

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 24 de outubro de 2008.

Assinatura do autor _____

Assinatura do orientador _____

FICHA CATALOGRÁFICA

Guimarães, Júlio Henrique dos Nogueira e Oliveira
Método para manutenção de sistema de software utilizando técnicas arquiteturais / J.H.N.O. Guimarães. – ed.rev. -- São Paulo, 2008.
101 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Manutenção de software 2.Arquitetura de software 3.Qualidade de produto de software I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

DEDICATÓRIA

Dedico este trabalho a Deus, meus amigos e minha família. Se cheguei aqui foi só porque vocês estiveram tão perto SEMPRE!

AGRADECIMENTOS

Ao amigo e orientador Prof. Dr. Reginaldo Arakaki pela orientação no caminho da pesquisa, pelas discussões dos diversos conceitos, pelas críticas visando meu empenho, pelas tantas oportunidades de aplicação prática da pesquisa e pelo incentivo acima de tudo.

Ao amigo e colega Renato Manzan de Andrade pela verdadeira parceria no andamento da pesquisa e pelo incentivo constante.

À amiga Profa. Dra. Selma Shin Shimizu Melnikoff pelos ensinamentos, conversas e direcionamentos.

Aos professores Prof. Dr. Denis Gabos e Prof. Dr. Jorge Luis Risco Becerra pelas importantes considerações que ajudaram a amoldar este trabalho.

Aos amigos do grupo de pesquisas do IPT com quem dividi conhecimentos e experiências tão importantes para este trabalho.

Aos amigos do LARC, bem como os da Scopus que me auxiliaram tantas vezes e permitiram que me mantivesse neste trabalho.

A todas as pessoas do Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da USP que contribuíram diretamente ou indiretamente para a realização deste trabalho.

À minha família e tantos amigos que, cada um à sua maneira, “quebraram mais do que galhos” – uma floresta inteira, isso sim – e me desculparam pela ausência temporária.

Por fim, mas não por último, a Deus que me permitiu mais esta realização.

RESUMO

Diversos negócios hoje são suportados por sistemas de software. Acredita-se que o uso de Arquitetura de Software é fundamental para atingir alcançar as metas de negócio e qualidade. Visto que o conjunto de requisitos que levou à construção de uma determinada arquitetura pode mudar, tal arquitetura pode tornar-se inadequada. Em diversas situações é preciso conviver com os sistemas existentes, portanto é preciso alterá-los para as novas necessidades. Manutenção de sistemas usando técnicas de evolução arquiteturais tem se mostrado um eficaz caminho para alterar um sistema à nova situação. O objetivo deste trabalho é apresentar um método para manutenção de sistemas de software usando técnicas arquiteturais de forma a convergir mais rapidamente à adequação da arquitetura destes sistemas. Algumas técnicas do método incluem avaliação de arquitetura de software, levantamento de riscos, provas de conceito construtivas e destrutivas e métricas estáticas e dinâmicas de software. O método proposto foi aplicado em contextos de laboratório e da indústria, permitindo a verificação de pontos fortes e críticos para realizar seu refinamento e tais aplicações são também relatadas neste trabalho. Embora as aplicações do método proposto tenham sido diferentes, tanto no objetivo principal quanto no desenrolar das atividades, seus resultados foram considerados bastante satisfatórios, tanto no contexto de ensino quanto no contexto de indústria.

Palavras-chave: Manutenção de Arquitetura de Software. Técnicas arquiteturais. Evolução de Arquitetura de Software. Qualidade de Produto de Software.

ABSTRACT

Several business today are supported by software systems. It is believed that the use of software architecture is fundamental to achieve the business goals and quality. Since the set of requirements that led to the construction of a given architecture may change, such an architecture may become inadequate. In many situations we must live with existing systems, so we must change them to meet the new needs. Systems maintenance using architectural evolution techniques has proven to be an effective path to take a system to the new situation. This work's goal is to provide a method for maintenance of software systems using architectural techniques in order to converge more rapidly to match the architecture of these systems. Some techniques used in the method include software architecture evaluation, risks survey, constructive and destructive proofs of concept, static and dynamic software metrics. The proposed method was applied in contexts of laboratory and industry, allowing the verification of strengths and critical points to guide its refinement and such applications are also reported in this work. Although the applications of the proposed method has been different in both the main goal and the conduct of activities, their results were considered very satisfactory, both in the education and the industry context.

Keywords: Software Architecture Maintenance. Architectural Techniques. Software Architecture Evolution. Software Product Quality.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
ISO	*International Organization for Standardization*
SGBD	Sistema Gerenciados de Banco de Dados
UML	*Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

1.1 Objetivo do trabalho

O objetivo deste trabalho é fornecer um método que auxilie a manutenção de sistemas que se tornaram inadequados e cuja arquitetura não fora construída levando-se em conta variabilidades ou as modificações necessárias fogem ao escopo das variabilidades.

O método promove a aliança da Arquitetura de Software com Manutenção na forma de fases iterativas, propondo o uso de técnicas arquiteturais práticas que auxiliem a redução do número de iterações (para convergir mais rápido ao resultado desejado).

1.2 Justificativa

Softwares são construídos para endereçar seus objetivos de negócio e de mercado. Em um trabalho recente procurou-se categorizar os objetivos de negócio, chegando-se à seguinte lista (KAZMAN; BASS, 2005):

- Reduzir o custo total de posse;
- Aumentar a capacidade/qualidade de um sistema;
- Melhorar a posição de mercado;
- Suportar processos de negócio melhorados;
- Melhorar a confiança no sistema e a percepção do sistema.

Pode-se então dizer que um sistema é adequado quando ele atinge seu objetivo de endereçar seus objetivos de negócio e de mercado. Logo, um sistema é inadequado quando a situação é oposta, ou seja, não endereça os objetivos.

Arquitetura de software é a ponte entre os objetivos de negócio e o sistema realizado. Para que o sistema se torne adequado, a arquitetura deve ser moldada de forma a atender uma série de requisitos funcionais e requisitos de qualidade (também conhecidos como “não-funcionais”) que reflitam os objetivos de negócio.

Um sistema pode nascer ou tornar-se inadequado. Um exemplo de nascimento inadequado é quando apenas as funcionalidades são claramente declaradas. Então, embora o sistema tenha seus requisitos de qualidade, estes não foram explicitados e não foram usados para moldar o sistema, fazendo com que o sucesso na sua implementação seja casual.

Já no caso de um sistema tornar-se inadequado, Parnas (1994) descreve dois principais fatores que causam o envelhecimento do software (e sua inadequação): mudanças no domínio em volta do software e alterações no sistema que foram introduzidas descuidadamente e degradaram o sistema.

Para tratar os problemas de inadequação apontam-se como principais tipos de iniciativas:

1) Usar métodos que auxiliem a construção do sistema, auxiliando o controle dos requisitos durante o projeto como são os casos dos métodos apresentados em (HOFMEISTER et al., 2005). Inclusive neste trabalho é proposto um método generalizado de cinco métodos analisados;
2) Dado que uma arquitetura está adequada, utilizar métodos que auxiliem a introdução de modificações de forma controlada, obedecendo a arquitetura proposta, como é o caso de (SADOU; TAMZALIT; OUSSALAH, 2005);
3) Dado que uma arquitetura precise ser modificada dentro das restrições de uma linha de produtos, usar métodos que auxiliem o controle das alterações, respeitando as restrições. Exemplos de trabalhos nessa linha incluem (BACHMANN et al., 2000) e (GRAAF, 2007);
4) Dado que uma arquitetura precise ser alterada e ela não foi construída com pontos de variabilidade ou as alterações fogem ao escopo de tais variabilidades, usar um método que auxilie no processo de manutenção.

Tratando-se do quarto tipo de iniciativa acima, existem diversas propostas que podem ajudar o processo de manutenção, mas não são explicitamente integradas e

encadeadas para esta aplicação. Isto faz com que o processo de manutenção não seja realizado de forma estruturada, reduzindo a previsibilidade de resultados. Portanto, é importante prover uma forma estruturada de tratar o problema de manutenção neste caso, indicando um caminho para a adequação de uma arquitetura.

1.3 Abrangência

O método proposto fornece uma abordagem para realizar manutenção de um sistema de software. O método foi construído para ser usado em casos em que o sistema tenha se tornado inadequado. Uma das metas principais é a convergência rápida. Seu processo é guiado pelas necessidades de negócio e requisitos de qualidade. Na realização das atividades o método propõe o uso de algumas técnicas arquiteturais práticas e medições estáticas e dinâmicas.

1.4 Metodologia de trabalho

A construção deste trabalho seguiu um processo iterativo que envolveram os seguintes passos:

1) Pesquisa bibliográfica: procurou-se neste passo obter o embasamento teórico referente aos trabalhos relacionadas a Manutenção (ou Evolução, em alguns trabalhos) e Arquitetura de Software, verificando a correlação dos conceitos e esforços nestas frentes;
2) Proposição do método: este passo envolveu a elaboração do método (na primeira iteração) e o refinamento do método (em iterações posteriores)
3) Aplicação do método em ambiente de laboratório e de indústria: nesta fase, aplicou-se o método proposto tanto em ambientes controlados quanto em ambientes da indústria;

4) Avaliação do método: os resultados da aplicação do método foram avaliados, identificando pontos críticos.

1.5 Organização do texto

O texto está organizado na seguinte estrutura:

No capítulo 1 apresentam-se o objetivo deste trabalho, suas principais justificativas e abrangência dentro do contexto de manutenção e a metodologia de trabalho adotada para o andamento da pesquisa e construção do método aqui proposto.

O capítulo 2 ressalta a importância da Arquitetura de Software como ponte para o alcance da adequação da arquitetura de um sistema e como isto se relaciona com a qualidade do produto de software. Neste capítulo também são relacionadas normas e técnicas importantes tanto influenciando o método como sendo adotadas como ferramentas de apoio na execução das atividades das fases do método.

O capítulo 3 contém o método proposto, tratando do fluxo das atividades, relacionando os participantes, produtos intermediários e finais.

No capítulo 4, aplicações do método realizadas em contextos de ensino e de indústria são apresentadas mostrando como as alternativas apontadas no método foram utilizadas.

O capítulo 5 encerra com as conclusões finais e propostas de possíveis trabalhos futuros para continuidade da pesquisa.

O Anexo A que contém alguns resultados finais da aplicação do método para exemplificação.

2 ARQUITETURA DE SOFTWARE

Este capítulo apresenta a Arquitetura de Software, especialmente em como ela oferece uma ponte para o alcance da adequação de um sistema. Mostra-se também que ela está intimamente relacionada com a qualidade do produto de software. Este capítulo ainda apresenta uma série de normas e técnicas nestes assuntos que foram importantes tanto influenciando o método como sendo adotadas como ferramentas de apoio na execução das atividades das fases do método.

2.1 Qualidade de software, modelos de qualidade e a ISO 9126-1

Um modelo de qualidade é a especificação das características necessárias que um sistema de software deve exibir. Arquitetos podem usar modelos de qualidade documentados como *templates* e derivar um modelo de qualidade específico para o sistema em questão. Às vezes, um tipo de aplicação pode não exibir todas as características de qualidade que um modelo de qualidade define. Pode inclusive nem precisar usar todas as métricas especificadas. Por outro lado, pode ser que o arquiteto tenha que definir outras características, sub-características e métricas de acordo com o tipo de aplicação (ALBIN, 2003).

A Figura 2.1 demonstra que modelos de qualidade agregam características que relacionam vários atributos de qualidade e, em alguns casos, identificam práticas de engenharia para endereçá-los e métricas apropriadas para medi-los ou observá-los. Cada modelo usa uma terminologia diferente. Um modelo de qualidade é uma instância específica do meta-modelo de qualidade e define características específicas, atributos de qualidade e métricas (ALBIN, 2003).

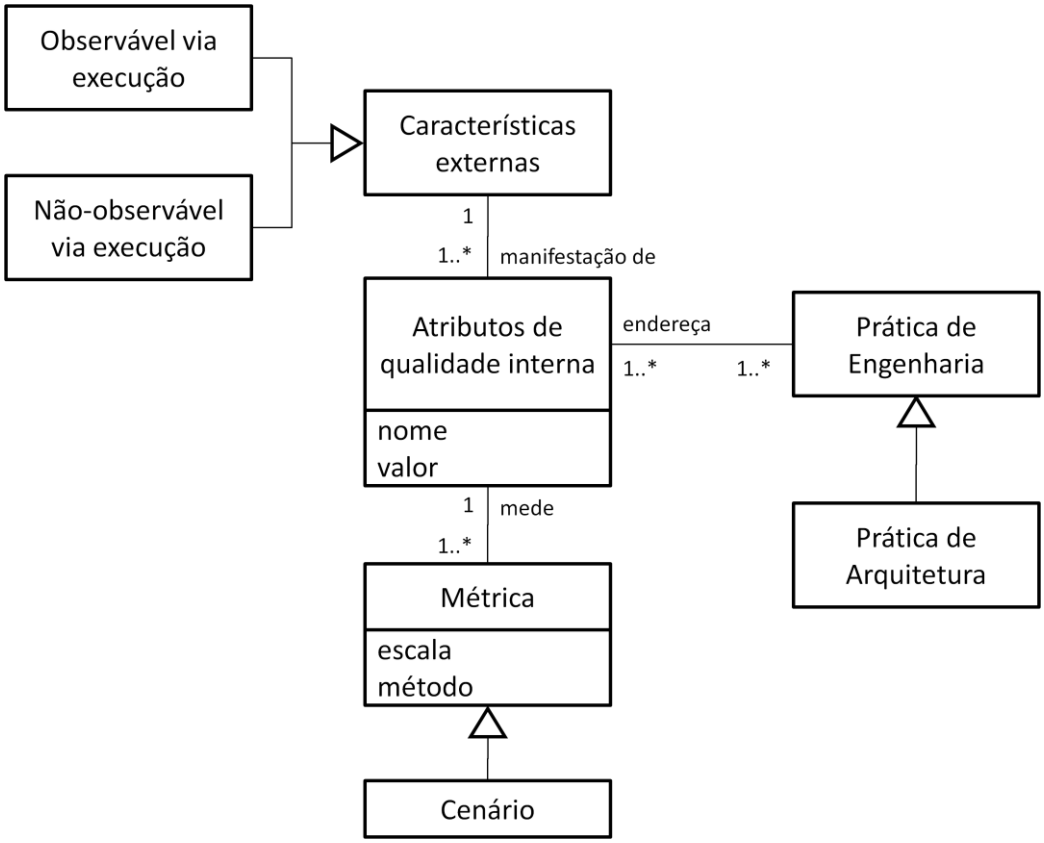


Figura 2.1 – Meta-modelo de qualidade (notação UML). (ALBIN, 2003)

Como benefícios que podem ser obtidos pelo uso de um modelo de qualidade, a especificação (ISO, 2001) lista:

- Validação da completeza da definição de requisitos;
- Identificação de requisitos de software;
- Identificação de objetivos de projeto de software;
- Identificação de objetivos de teste de software;
- Identificação do critério de aceitação do usuário para um produto de software produzido.

Uma referência de qualidade de âmbito internacional é a ISO 9126, onde a primeira seção (ISO, 2001) define o modelo de qualidade para um produto de software. Tomando-se como princípio que esta referência propõe uma ampla cobertura no que tange a qualidade, ela foi muitas vezes usada nos experimentos como base para analisar uma arquitetura de software.

O padrão organiza a qualidade de software em 6 características ou objetivos, onde cada um é composto de vários atributos de qualidade ou sub-características.

Qualidades externas podem ser experimentadas pelos usuários do sistema e as sub-características são os atributos de qualidade internos que, quando balanceados apropriadamente, alcançam as qualidades externas. As 6 características externas são apresentadas a seguir com suas sub-características (ISO, 2001):

- **Funcionalidade:** é a capacidade do produto de software possuir um conjunto de funções e propriedades específicas que satisfazem necessidades explícitas e implícitas.
    - **Adequação**: presença das funções especificadas.
    - **Acurácia**: gera resultados precisos ou dentro do esperado.
    - **Interoperabilidade**: capacidade de interagir e inter-operar com outros sistemas, de acordo com o especificado.
    - **Segurança**: capacidade para prevenir o acesso não autorizado.
    - **Conformidade:** observância a padrões, convenções ou regras estabelecidas.
- **Confiabilidade:** é a capacidade do produto de software de manter seu nível de desempenho em determinadas condições por um tempo determinado.
    - **Maturidade**: indicação de baixa freqüência de falhas.
    - **Tolerância a falhas**: capacidade do produto para manter determinados níveis de desempenho mesmo na presença de problemas.
    - **Recuperabilidade**: capacidade do produto para restabelecer o nível de desempenho desejado e recuperar dados em caso de ocorrência de falha.
    - **Conformidade relacionada à confiabilidade**.
- **Usabilidade:** é a capacidade do produto de software de ser entendido, aprendido e ser atrativo ao usuário quando usado sob determinadas condições (o esforço necessário para usar).
    - **Inteligibilidade**: medida da facilidade do usuário para reconhecer a lógica de funcionamento do produto e sua aplicação.
    - **Apreensibilidade**: medida da facilidade encontrada pelo usuário para aprender a utilizar o produto.

  - **Operacionabilidade**: medida da facilidade para operar o produto.
  - **Conformidade relacionada à usabilidade**.

- **Eficiência:** é a capacidade do produto de software de prover um desempenho apropriado, relativo à quantidade de recursos usados, sob determinadas condições (o que o software faz para cumprir necessidades).
  - **Comportamento com relação ao tempo**: medida do tempo de resposta e do processamento, assim como as taxas de processamento (*throughput*) ao executar as funções prescritas.
  - **Comportamento com relação ao uso de recursos**: medida da quantidade de recursos necessários (p. e. CPU, disco, memória) e a duração do seu uso ao executar as funções prescritas.
  - **Conformidade relacionada à eficiência**.
- **Manutenibilidade:** é a capacidade do produto de software de ser modificado. Modificações incluem correções, melhorias ou adaptações do software a mudanças no ambiente e nos requisitos e especificações funcionais (o esforço para ser modificado).
  - **Analisabilidade**: medida do esforço necessário para diagnosticar deficiências ou causas de falhas, ou localizar as partes a serem modificadas para corrigir os problemas.
  - **Modificabilidade**: medida do esforço necessário para realizar alterações, remover falhas ou para adequar o produto a eventuais mudanças de ambiente operacional.
  - **Estabilidade**: medida do risco de efeitos inesperados provenientes de modificações.
  - **Testabilidade**: medida do esforço necessário para testar o software alterado.
  - **Conformidade relacionada à manutenibilidade**.
- **Portabilidade:** é a capacidade do produto de software ser transferido de um ambiente para outro. O ambiente pode ser organizacional, hardware ou software.

Em quais ambientes (sistemas operacionais), o software se adapta e a facilidade de configuração em ambientes diferentes.

- **Adaptabilidade**: medida da facilidade de se adaptar o produto para funcionar em outros ambientes operacionais diferentes do originalmente especificado.
- **Instalabilidade**: medida do esforço necessário para se instalar o produto.
- **Coexistabilidade**: medida do nível de conformidade do produto com padrões referentes à portabilidade.
- **Substituibilidade**: medida do esforço necessário para usar o produto em substituição a outro produto previamente especificado.
- **Conformidade relacionada à portabilidade**.

2.2 Significado de arquitetura de software

O termo “arquitetura de software” é muito utilizado por pesquisadores e desenvolvedores, porém com diferenças quanto à definição. Como é um termo utilizado em diversos contextos da Engenharia de Software, percebe-se um crescimento de sua importância, mas também que é um conceito ainda não muito bem definido (KRUCHTEN, 2000).

Como breve histórico, apresentam-se algumas definições que são bastante abrangentes e procuram expressar a extensão, a complexidade e as diferentes visões sobre o conceito de arquitetura de software:

Como uma das primeiras definições, (PERRY; WOLF, 1992) dispõem que arquitetura de software representa a organização de coleções de componentes interconectados de acordo com determinadas restrições quanto à forma de interação.

(GARLAN; SHAW, 1996) definem arquitetura de software como a representação do sistema por meio de seus componentes computacionais e os relacionamentos entre eles.

Com uma definição mais abrangente, e esta é a que é utilizada no restante deste trabalho, (BASS; CLEMENTS; KAZMAN, 2003) apresentam a arquitetura de software de um programa ou sistema computacional como a estrutura ou estruturas do sistema, que compreendem elementos de software, as propriedades externas visíveis desses elementos e o relacionamento entre eles. São suprimidos os detalhes dos elementos se estes não mudam como eles são utilizados ou como eles utilizam outros elementos. Isto evidencia o desacoplamento entre os elementos de uma arquitetura, conceito muito importante para gerenciar a complexidade do sistema.

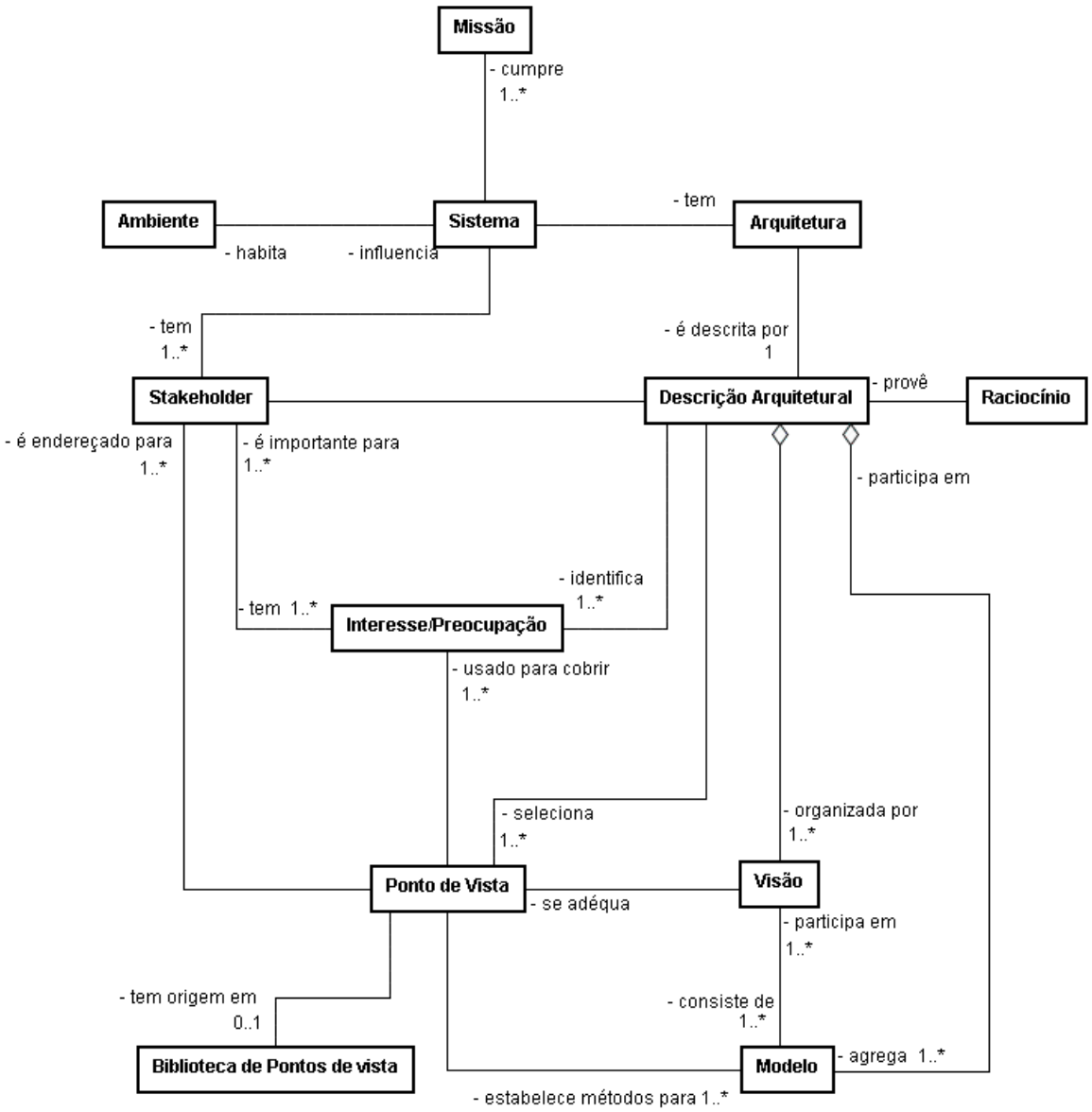


Figura 2.2 – Modelo conceitual da descrição arquitetural (notação UML). (IEEE, 2000)

O documento IEEE Std. 1471-2000 (IEEE, 2000) define arquitetura de software como sendo a organização fundamental de um sistema, englobando os componentes e seus relacionamentos, o ambiente e os princípios que governam o design e a evolução da arquitetura. Este documento inclusive descreve um modelo conceitual que denota bem a distinção entre uma arquitetura e sua descrição. A descrição arquitetural é organizada por um conjunto de visões que são regidas pelos pontos de vista (*viewpoints*). A Figura 2.2 foi extraída do documento e demonstra o modelo conceitual apresentado.

2.3 Importância da arquitetura de software

A partir da década de 90, a arquitetura de software tem recebido muita atenção como uma importante área da Engenharia de Software, devido à sua criticidade como fator de sucesso nos projetos de sistemas computacionais (GARLAN, 2000).

Este amadurecimento não ocorre somente na academia. Muitas empresas que reconhecem o valor da arquitetura de software a consideram como um dos artefatos intelectuais mais valiosos, considerando-se recursos humanos e financeiros e procuram maximizar o retorno deste investimento buscando novas formas de utilização das arquiteturas e submetendo-as a atualizações e melhorias contínuas (BASS; CLEMENTS *et al.*, 2003).

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

- **Comunicação entre *stakeholders***: a arquitetura de software representa uma abstração comum do sistema que a maioria (se não a totalidade) dos *stakeholders* pode utilizar como base para entendimento mútuo, negociação, consenso e comunicação;
- **Decisões de projeto iniciais**: a arquitetura de software manifesta as decisões de projeto iniciais de um sistema e estas ligações iniciais são levadas para todo o ciclo de desenvolvimento, *deployment* e manutenção. Este é o ponto mais cedo

em que se podem analisar as decisões de projeto que governam o sistema a ser construído. Citam-se alguns benefícios da arquitetura relacionados:

- A arquitetura define restrições na implementação;
- A arquitetura rege a estrutura organizacional;
- A arquitetura inibe ou permite os atributos de qualidade de um sistema;
- O estudo da arquitetura ajuda a prever qualidades do sistema;
- A arquitetura facilita o entendimento e a gerência de mudanças;
- A arquitetura ajuda em prototipagem evolutiva;
- A arquitetura permite estimativas de custo e prazo mais precisas.

- **Abstração transferível de um sistema**: a arquitetura de software constitui-se de um modelo relativamente pequeno e intelectualmente compreensível de como um sistema é estruturado e como seus elementos trabalham junto. Este modelo é transferível para outros sistemas. Em particular, pode ser aplicável a outros sistemas que exibam atributos de qualidade e requisitos funcionais similares e podem promover o reuso em larga escala. Alguns benefícios citados:
  - Linhas de produto de software dividem uma arquitetura comum;
  - Sistemas podem ser produzidos usando elementos desenvolvidos externamente;
  - Restringir-se quanto às alternativas de projeto podem ajudar a aumentar a expertise nestas alternativas;
  - Uma arquitetura permite o desenvolvimento baseado em *templates*;
  - Uma arquitetura pode ser a base para treinamento.

2.4 Utilização da arquitetura de software

Para (JAZAYERI; RAN; LINDEN, 2000), a arquitetura de software pode ser vista como uma ferramenta de gerência da complexidade do software. Reforça também

que uma arquitetura de software tem como papel fundamental satisfazer os requisitos funcionais e não funcionais.

Corroborando nessa linha de pensamento, (HOFMEISTER; NORD; SONI, 2000) colocam que a arquitetura é a ponte entre os requisitos do sistema e sua implementação e serve de guia para todas as atividades do processo de desenvolvimento de software.

De acordo com (SOMMERVILLE, 2004), uma arquitetura representa o framework fundamental para a estruturação e a organização de um sistema de software que engloba os componentes e as comunicações entre eles.

Em (PAULISH; BASS; PAULISH, 2001) propõe-se que os propósitos principais para a arquitetura de software são planejar o projeto e abstrair o produto a ser implementado. Com a visão arquitetural, os engenheiros de software têm mais chances de implementar o produto com sucesso, visto que esta visão auxilia a tomada de decisões (pois podem ter uma visão melhor dos impactos de suas decisões) e minimiza os riscos e incertezas de projetos de software.

A arquitetura é também um ponto de referência comum para as demais atividades que são executadas após sua definição. Isto porque a arquitetura é a manifestação antecipada das decisões de projeto, pois deve ocupar-se de fatores como tempo de desenvolvimento, custo e manutenção, definição das restrições de implementação e definição da estrutura organizacional, enfatizando os atributos de qualidade que o sistema deve ter e medindo através de avaliações a satisfação das qualidades necessárias (BASS; CLEMENTS *et al.*, 2003).

Segundo (GARLAN, 2000), a arquitetura de software desempenha um papel fundamental no processo de desenvolvimento de software, nos seguintes aspectos do desenvolvimento de software:

- **Análise**: a definição da arquitetura de software facilita a tarefa de análise em um projeto, pois inclui verificação de consistência, conformidade com atributos de qualidade e análise de dependências;
- **Entendimento**: a arquitetura de software simplifica a compreensão de sistemas complexos, por meio de níveis de abstração;

- **Reuso**: a arquitetura promove o reuso em diversos níveis – especificações, classes, componentes e frameworks;
- **Construção**: a arquitetura de software provê subsídios para o desenvolvimento, elucidando os componentes, suas interfaces e dependências entre eles;
- **Evolução e manutenção**: a definição da arquitetura permite identificar as ramificações e os impactos das mudanças, o que auxilia na manutenção (corretiva e evolutiva);
- **Gerenciamento**: a experiência tem mostrado que projetos de sucesso tratam a arquitetura de software como elemento chave no processo de desenvolvimento. A avaliação crítica da arquitetura permite o entendimento mais claro dos requisitos, das estratégias de implementação e de potenciais riscos.

2.5 O processo da arquitetura de software

O processo da arquitetura de software é o processo que envolve as atividades para criar uma arquitetura de software, usar uma arquitetura de software para realizar um projeto e implementar ou gerenciar a evolução de um sistema. As atividades são as seguintes (BASS; CLEMENTS *et al.*, 2003):

- **Criar o modelo de negócio para o sistema**: o arquiteto deve avaliar o custo, o público-alvo, o *time-to-market*, a interface com outros sistemas e as limitações que o sistema deve ter;
- **Entender os requisitos**: o arquiteto usa de técnicas de levantamento de requisitos para obter o modelo de domínio;
- **Criar ou selecionar a arquitetura**: o arquiteto faz isso por identificar os componentes e suas interações, identificar as dependências de construção e escolher as tecnologias que suportam a implementação;
- **Documentar e divulgar a arquitetura**: os participantes envolvidos precisam entender a arquitetura, pois esta será insumo para o desenvolvimento das suas atividades;

- **Analisar ou avaliar a arquitetura**: devem-se avaliar as decisões tomadas na arquitetura para se garantir que elas satisfazem as necessidades dos *stakeholders*;
- **Implementar o sistema baseando-se na arquitetura**: os desenvolvedores devem manter-se fiéis às estruturas e protocolos de interação definidos pela arquitetura;
- **Garantir adequação a uma arquitetura**: durante a fase de manutenção e preciso que seja mantida vigilância constante para garantir que a arquitetura real e a sua representação se mantêm fiéis entre si.

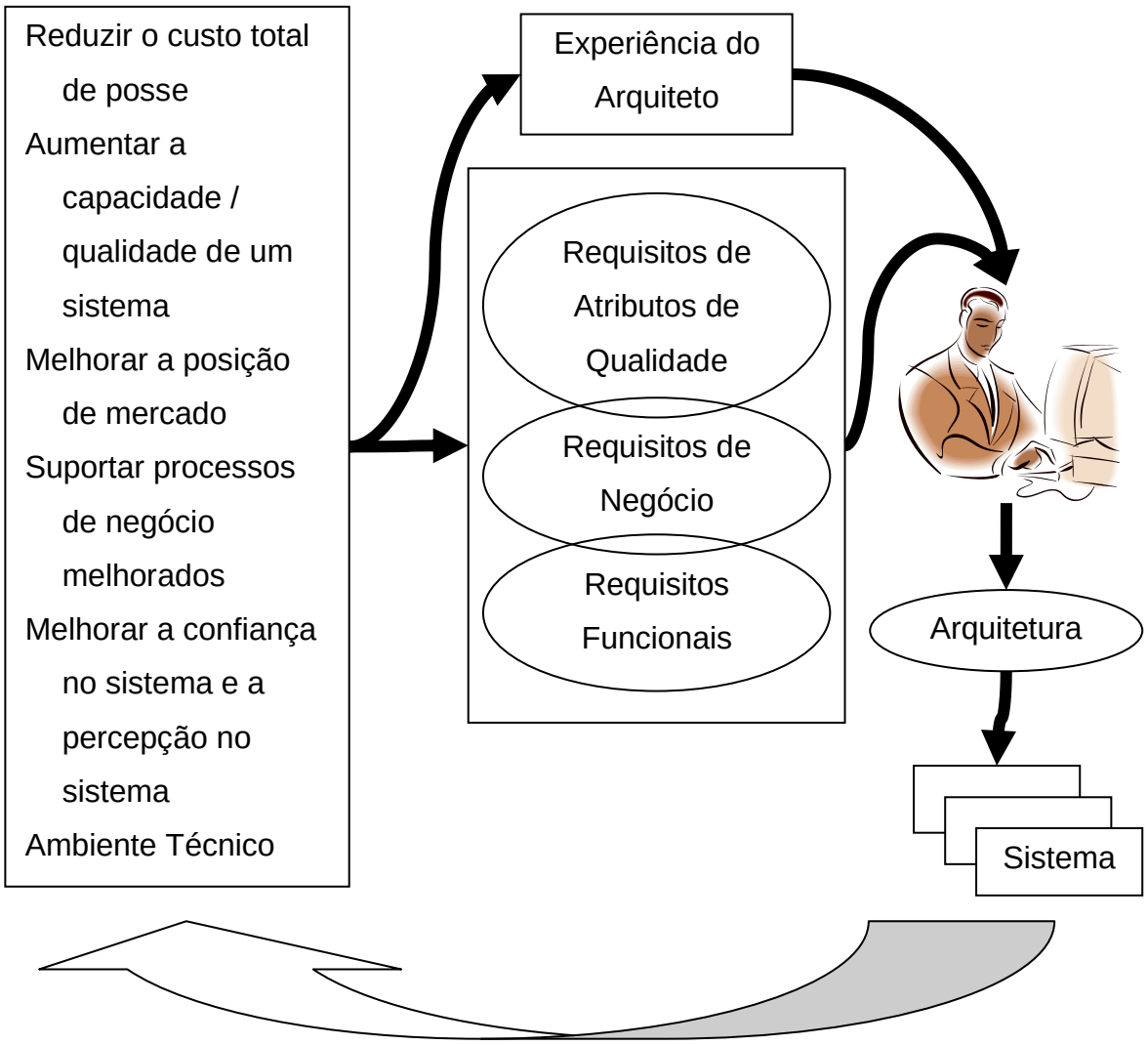


Figura 2.3 – O ciclo da arquitetura (KAZMAN; BASS, 2005).

Em (BASS; CLEMENTS *et al.*, 2003) também é apresentado um ciclo iterativo que apresenta a origem e evolução da arquitetura que é refinado em (KAZMAN; BASS,

2005), um trabalho de categorização de objetivos de negócio. este ciclo refinado é apresentado na Figura 2.3,

No ciclo de vida da arquitetura, o arquiteto (ou grupo de arquitetos) sofre diversas influências, tais como os requisitos (qualidades) exigidos pelos *stakeholders* e pela organização, o ambiente técnico e a experiência prévia do arquiteto. Sob tais influências, o arquiteto produz a arquitetura e o sistema. O ciclo se fecha quando estes resultados afetam aqueles fatores influenciadores (por exemplo, alterando os objetivos de negócio da organização, alterando os requisitos para um próximo sistema ou alterando a experiência do arquiteto).

2.6 Algumas técnicas arquiteturais

2.6.1 Métricas de software

A combinação desejada dos atributos de qualidade deve ser claramente definida, senão a avaliação da qualidade é deixada para a intuição. Em (CLEMENTS; KAZMAN; KLEIN, 2002) percebe-se que sem uma elaboração melhor os requisitos de qualidade ficam sujeitos à má interpretação porque o conceito de qualidade é subjetivo. Assim, o uso de métricas de software pode ajudar a reduzir a abstração. Definir a qualidade de software de um sistema é equivalente a definir uma lista de atributos de qualidade de software para este sistema e identificar um conjunto de métricas de software associadas.

O propósito de métricas de software é fazer avaliações por todo o ciclo de vida para verificar se os requisitos de qualidade de software estão sendo atendidos. O uso de métricas de software reduz a subjetividade na avaliação e controla a qualidade do software porque fornece uma base quantitativa para tomar decisões quanto à qualidade do software. Porém, o uso de métricas de software não elimina a necessidade de julgamento humano nas avaliações de software.

Embora pesquisas na área de métricas de software tendam a focar predominantemente em métricas estáticas (obtidas de análise estática de artefatos de software), (GUNNALAN; SHERESHEVSKY; AMMAR, 2005) constatam que estimativas da qualidade de software baseando-se em métricas dinâmicas são mais precisas e realistas. Portanto, este trabalho enfatiza o uso de métricas– e, por sua vez, o uso de medições – estáticas e dinâmicas de software.

Os tipos de medição aplicáveis variam de acordo com o domínio de aplicação. Em aplicações do método foram realizados diversos tipos de medição. Alguns tipos de medições estáticas incluíram tamanho de componentes, linhas de código, quantidade de classes que referenciam determinado método, dentre outros. Pode-se citar exemplos de medições dinâmicas que também foram utilizadas, tais como tempo de resposta de partes do sistema ou do todo, quantidade de usuários simultâneos, quantidade de registros na base de dados, uso de memória e CPU, dentre outros. Tais medições foram bastante usadas para aumentar o conhecimentos sobre a arquitetura em maiores detalhes e permitir a investigação de limites do sistema.

2.6.2 Provas de conceito (POC's)

Define-se prova de conceito (usa-se no texto a sigla POC, do inglês *Proof of Concept*) como uma técnica que permite demonstrar que uma determinada idéia é tecnicamente possível, ou seja, pode ajudar a verificar se uma arquitetura é "construível". Provas de conceito permitem demonstrar a viabilidade de construção de um sistema de software utilizando um determinado estilo arquitetural, o que aumenta as chances de que a solução adotada satisfaça os requisitos funcionais e não funcionais.

Para este trabalho, definiu-se usar os termos "POC construtiva" e "POC destrutiva" conforme descrito a seguir:

- Uma **POC construtiva** potencialmente reduz o risco de fracasso e cobre muitos aspectos importantes no desenvolvimento de uma aplicação, como por exemplo: integração entre componentes, escalabilidade e desempenho.
- Uma **POC destrutiva** explicita os pontos fracos do software, expondo seus limites e falhas antes do mesmo ir para a produção. Permite que se mostre também “o que não fazer”, ou seja, situações de insucesso de implementação de solução.

Provas de conceito permitem uma separação clara do trabalho dos desenvolvedores, possibilitando o uso de técnicas de engenharia simultânea (paralelismo de esforços), o que diminui o tempo de desenvolvimento de um sistema. Ambos os tipos são importantes para reduzir os riscos de insucesso do projeto e viabilizar a satisfação dos requisitos funcionais e principalmente os não funcionais.

2.7 Avaliação de arquitetura de software

2.7.1 Importância da avaliação de arquitetura de software

A arquitetura de software é uma ferramenta importante para alcançar atributos de qualidade de um sistema como um todo. Por conseguinte, torna-se importante avaliar a arquitetura de um sistema com respeito aos requisitos de qualidade desejados.

2.7.2 Avaliação da arquitetura de software em estágios iniciais e finais

Em (CLEMENTS; KAZMAN *et al.*, 2002) percebem-se duas possíveis fases de avaliação: em fases iniciais e em fases finais do ciclo de vida de um sistema.

A avaliação em estágios **iniciais** é realizada quando existem apenas fragmentos da descrição arquitetural de forma que questionários, *checklists* e métodos baseados em cenários são mais usados para avaliação porque nessa fase não há informação tangível suficiente disponível para a coleta de medições ou simular o comportamento. As bases principais desse tipo de avaliação são a experiência dos desenvolvedores e cenários baseados em requisitos que estão nos documentos de requisitos.

A avaliação em estágios **finais** é realizada em estágios mais tardios do processo de desenvolvimento quando há ao menos um projeto detalhado disponível no qual métricas mais concretas podem ser coletadas, o que significa que técnicas de métricas arquiteturais são usadas para avaliar a arquitetura de software com respeito a um ou mais atributos de qualidade.

Devem-se usar técnicas de avaliação em fases iniciais e finais para assegurar o controle de qualidade e a garantia de qualidade. Por meio disso, é possível garantir que os requisitos dos *stakeholders* são considerados e implementados na arquitetura.

No método apresentado neste trabalho, foca-se principalmente na avaliação em estágios finais juntamente com o apoio de medições estáticas e dinâmicas. Isto é feito para verificar o atendimento dos requisitos conforme também é feito em outros métodos tradicionais.

2.7.3 Objetivos da avaliação de arquitetura de software

Há 3 propósitos principais para avaliação de arquitetura de software (CLEMENTS; KAZMAN *et al.*, 2002):

- O primeiro propósito é a identificação antecipada de insuficiência que foram feitas durante a fase de levantamento dos requisitos ou fases de projeto iniciais. Aqui, insuficiência significa que a arquitetura não atinge as expectativas dos *stakeholders* com respeito a um ou mais atributos de qualidade. Quanto mais cedo for a fase de desenvolvimento em que tal insuficiência for encontrada, menores serão os custos e a quantidade de recursos necessários para sua eliminação. Já que a arquitetura descreve todo o sistema de software, fraquezas existentes não reconhecidas causam erros em saídas posteriores do desenvolvimento, como a implementação ou, no pior dos casos, no produto final. Por conseguinte, mudanças causadas por fraquezas que foram descobertas tardiamente também causarão mudanças necessárias nas saídas destes desenvolvimentos posteriores. Estas mudanças precisam de muito mais recursos que apenas mudanças na própria arquitetura;
- O segundo propósito principal é a comparação entre sugestões arquiteturais alternativas relativas a um ou mais atributos de qualidade. Pode haver muitas arquiteturas candidatas à realização de um sistema de software que implementam a mesma funcionalidade, mas endereçam diferentemente os atributos de qualidade;
- O terceiro propósito principal é avaliar se uma arquitetura possui riscos para certos atributos de qualidade. A avaliação da arquitetura de software pode ser vista como uma parte de outro processo de avaliação. Um sistema de software poderia ter sido planejado com o propósito de ser parte de uma arquitetura empresarial porque nos dias de hoje, muitos processos de negócio de empresas utilizam software para atingir seus propósitos (por exemplo, através de aplicações, telecomunicações, workflow, e sistemas de bancos de dados). Uma arquitetura empresarial também descreve um sistema complexo e dinâmico que tem de cumprir certa funcionalidade ou prover certos serviços. Estes sistemas envolvem muito mais *stakeholders* que um sistema de software e estes *stakeholders* também esperam um número de requisitos funcionais e não funcionais. Portanto a descrição arquitetural de um sistema de software, assim como um sistema empresarial, deveria observar que o sistema tem de atingir funcionalidade, abertura, desempenho, confiabilidade e manutenibilidade. Para criar, desenvolver, implementar e manter uma arquitetura que atinge os requisitos

dos *stakeholders* é preciso também um processo de avaliação para forçar o controle de qualidade e garantia.

2.7.4 Métodos de avaliação de arquitetura de software

Existem diversos métodos voltados à avaliação de arquitetura de software. (BABAR; ZHU; JEFFERY, 2004) apresentam alguns dos principais métodos comparando-os numa série de critérios.

O método proposto na seção 3, possui uma fase de avaliação da adequação da arquitetura (3.3). Nesta fase, alguns métodos podem ser utilizados como auxílio na execução, desde que forneçam os resultados necessários. Alguns métodos foram avaliados e dentre os principais métodos para avaliação de arquitetura usados no método proposto, dois se destacam e são descritos em maior detalhe a seguir.

2.7.4.1 O processo de avaliação de software da ISO 14598-5

A norma ISO 14598, na sua quinta parte (ISO, 1998) define um processo para avaliar produtos de software. A Figura 2.4 apresenta as macro-atividades deste processo.

Na análise de requisitos de avaliação, definem-se os objetivos da avaliação, quais partes serão avaliadas e qual o modelo de qualidade que será utilizado. Recomenda-se que seja usada em conjunto com o modelo de qualidade definido na ISO 9126-1.

Durante a atividade de especificação da avaliação, os padrões e critérios utilizados para a avaliação são definidos por meio da seleção das métricas, do estabelecimento de níveis de pontuação e do estabelecimento de critérios de julgamento.

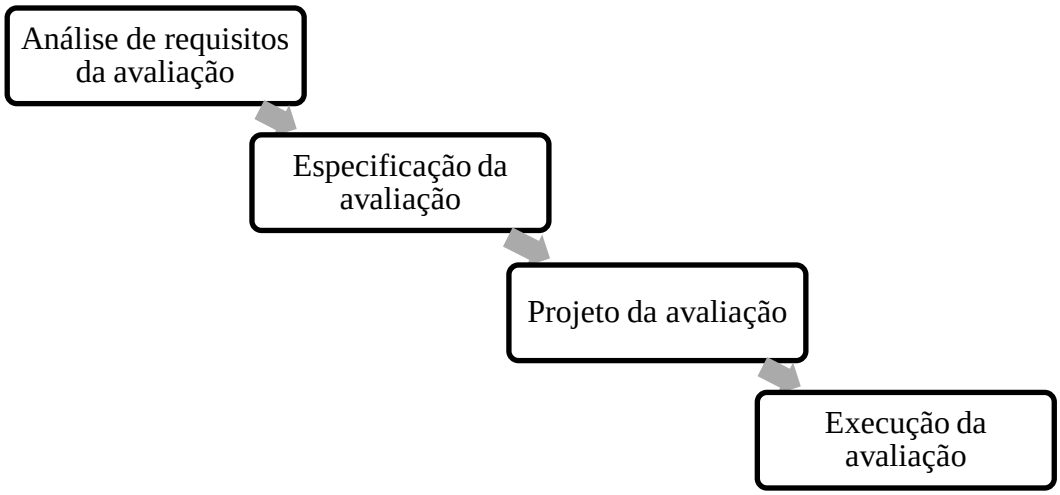


Figura 2.4 – Seqüência das macro-atividades da ISO 14598-5 (ISO, 1998).

Na atividade seguinte, realiza-se o projeto da avaliação. O objetivo deste projeto é produzir um plano de avaliação para a etapa seguinte.

Por fim, é feita a execução da avaliação por intermédio de medições, comparações com os critérios definidos e avaliação crítica dos resultados.

A fase de avaliação de arquitetura estabelecida para o método proposto foi também influenciada pela ISO 14598-5. A primeira atividade da referência (análise de requisitos de avaliação) já é uma pré-concepção do método, ou seja, já se sabe que o objetivo é avaliar a discrepância entre a arquitetura adequada e a existente.

2.7.4.2 ATAM: Método de Análise de Tradeoff de Arquitetura

O método *Architecture Tradeoff Analysis Method* (ATAM) descrito em (KAZMAN; KLEIN; CLEMENTS, 2000) foi bastante usado em aplicações do método proposto neste trabalho na fase de avaliação de arquitetura de adequação de arquitetura do sistema.

O ATAM é um método de análise organizado sobre a idéia de que estilos arquiteturais são os principais determinantes dos atributos de qualidade arquiteturais. O método foca na identificação dos objetivos de negócio que levam aos atributos de qualidade alvos. Daí, baseando-se nos atributos de qualidade alvos,

usa-se o ATAM para analisar como os estilos arquiteturais ajudam a atingir estes alvos.

Os passos do método são os seguintes:

1) **Apresentar o ATAM.** O método é descrito à equipe de *stakeholders* (tipicamente representantes dos clientes, o arquiteto ou equipe de arquitetura, representantes dos usuários, mantenedores, administradores, gerentes, testadores, integradores, etc.).
2) **Apresentar os condutores de negócio.** O gerente de projeto descreve quais objetivos de negócio motivam o esforço de desenvolvimento e, portanto, quais serão os condutores arquiteturais principais (por exemplo, alta disponibilidade ou *time do market* ou alta segurança).
3) **Apresentar a arquitetura.** O arquiteto descreve a arquitetura proposta, focando em como ela endereça os condutores de negócio.
4) **Identificar decisões arquiteturais.** As decisões arquiteturais são identificadas pelo arquiteto, mas não são analisadas.
5) **Gerar a *utility tree* dos atributos de qualidade**. Os fatores de qualidade que compreendem a "utilidade" do sistema (desempenho, disponibilidade, segurança, modificabilidade, etc.) são extraídos, especificados até o nível de cenários, anotados com estímulo e respostas, e priorizados.

   A tabela a seguir exemplifica como é uma *utility tree*. A sua construção é feita partindo-se dos atributos de qualidade principais (primeira coluna), depois seus sub-atributos (segunda coluna) e por fim a especificação em cenários (terceira coluna).

Tabela 2.1 – Exemplo de forma tabular da utility tree . (KAZMAN; KLEIN *et al.*, 2000)

Atributo de qualidade	Sub-atributo de qualidade	Cenário
Desempenho	Latência de dados	Minimizar a latência de banco de dados do cliente para 200 ms
		Entregar vídeo em tempo real
	Throughput de transações	-

Atributo de qualidade	Sub-atributo de qualidade	Cenário
Modificabilidade	Novas categorias de produto	-
	Mudar COTS	Adicionar middleware CORBA em menos de 20 pessoas-mês
		Mudar a interface de usuário web em menos de 4 pessoas-semana
Disponibilidade	Falha de hardware	Interrupção de energia no Site 1 requer redirecionamento para o Site 2 em menos de 3 s
		Reinicialização após falha de disco em menos de 5 min
		Falha de rede é detectada e recuperada em menos de 1,5 min
	Falhas de software COTS	-
Segurança	Confidencialidade dos dados	Transações de cartão de crédito são seguras 99,999% do tempo
		Autorização do banco de dados do cliente funciona 99.999% do tempo
	Integridade dos dados	-

6) **Analisar as decisões arquiteturais.** Baseando-se nos fatores de maior prioridade que foram identificados no passo 5, as decisões arquiteturais que endereçam tais fatores são extraídas e analisadas (por exemplo, uma decisão arquitetura para atingir objetivos de desempenho será sujeita a análises de desempenho). Durante este passo identificam-se riscos arquiteturais, pontos de sensibilidade e pontos de *tradeoff*.

   Para a documentação das análises feitas, o método propõe a seguinte estrutura:

   a) **Cenário:** um cenário da *utility tree* (do passo 5) ou do *brainstorm* de cenários (do passo 7);

b) **Atributo:** atributo de qualidade envolvido no cenário;

c) **Ambiente:** suposições relevantes sobre o ambiente no qual o sistema reside;

d) **Estímulo:** um enunciado preciso do estímulo de atributo de qualidade (por exemplo: falha, ameaça, modificação) incorporado pelo cenário;

e) **Resposta:** um enunciado preciso da resposta de atributo de qualidade (por exemplo: tempo de resposta, medida da dificuldade de modificação);

f) Para cada decisão arquitetural envolvida:

i) **Nome da decisão arquitetural;**

ii) **Riscos;**

iii) **Não-riscos;**

iv) **Pontos de sensibilidade;**

v) ***Tradeoffs;***

g) **Racional:** serve como um memorial qualitativo ou quantitativo explicando por que as decisões arquiteturais contribuem para atingir os requisitos de resposta de atributo de qualidade. Pode vir também acompanhado de diagramas que auxiliem a explicação.

7) ***Brainstorm* e priorização de cenários.** Baseando-se nos cenários de exemplo gerados no passo da *utility tree*, todo o grupo de *stakeholders* produz um número maior de cenários. Este conjunto de cenários é priorizado por intermédio de um processo de votação envolvendo todo o grupo de *stakeholders*.

8) **Analisar as decisões arquiteturais.** Este passo reitera o passo 6, mas aqui os cenários mais votados pelo passo 7 são considerados para ser os casos de teste para a análise das decisões arquiteturais determinadas até o momento. Estes cenários de casos de teste podem não cobrir decisões arquiteturais adicionais, riscos, pontos de sensibilidade e pontos de *tradeoff* que, então, são documentados.

9) **Apresentar resultados.** Baseando-se nas informações coletadas no ATAM (estilos, cenários, questões específicas para atributos, a *utility tree*, riscos, pontos de sensibilidade, *tradeoffs*) a equipe do ATAM apresenta as descobertas ao

grupo de *stakeholders* e potencialmente escreve um relatório detalhando esta informação junto com propostas de estratégias de mitigação.

2.8 Conclusão do capítulo

Neste capítulo apresentou-se o conceito de modelo de qualidade, em especial o modelo de qualidade da ISO 9126-1. Este modelo é utilizado no método proposto como referência de atributos de qualidade, atuando como referência especialmente para o mapeamento de requisitos.

Viu-se também que a arquitetura de software está intimamente ligada à capacidade de um sistema atingir sua adequação e a importância de um processo para construí-la e gerenciá-la. Outro ponto importante que o capítulo traz é mostrar o processo envolvido na geração e evolução da arquitetura e como funciona o ciclo de influências numa organização, conforme ilustrado na Figura 2.3.

Continuando, o capítulo comenta sobre a avaliação de arquitetura e como ela pode ser especialmente útil para uso em manutenção de softwares. Um dos métodos inclusive bastante utilizado nos estudos de caso foi o uso do ATAM como referência técnica.

Por fim, comenta-se sobre algumas técnicas arquiteturais que são importantes e utilizadas na execução do método proposto.

3 MÉTODO PARA MANUTENÇÃO DE SISTEMAS DE SOFTWARE UTILIZANDO TÉCNICAS ARQUITETURAIS

Neste capítulo é apresentado o método proposto para auxiliar a manutenção de sistemas que se tornaram inadequados e cuja arquitetura não fora construída levando-se em conta variabilidades (linhas de produto) ou as modificações necessárias fogem ao escopo das variabilidades. O método promove a aliança da Arquitetura de Software com Manutenção na forma de fases iterativas.

A fim de facilitar o entendimento do método proposto, ele foi dividido em 5 grandes fases. Segue abaixo uma rápida descrição destas fases:

- **Fase 1 - Apresentação do sistema e do método de trabalho:** fornece um primeiro contato da Equipe de Manutenção com o sistema e o método de trabalho a toda a equipe envolvida no processo;
- **Fase 2 - Apresentação dos objetivos de negócio:** obtenção dos objetivos de negócio (passados e novos) que vão guiar a manutenção;
- **Fase 3 - Avaliação da adequação da arquitetura de software:** análise realizada antes e depois da manutenção para saber se a arquitetura é ou não adequada;
- **Fase 4 - Projeto da manutenção visando adequação da arquitetura:** preparação do plano de ação para a manutenção;
- **Fase 5 - Implementação da manutenção:** a implementação da manutenção seguindo o plano de ação.

O encadeamento das fases do método obedece o fluxo que é representado na Figura 3.1. Pode-se notar que há um ciclo incremental formado pelas fases 3, 4 e 5. reduzir-se a quantidade de iterações neste ciclo, é proposto o uso de técnicas arquiteturais práticas que conduzam à convergência mais rápida ao resultado desejado.

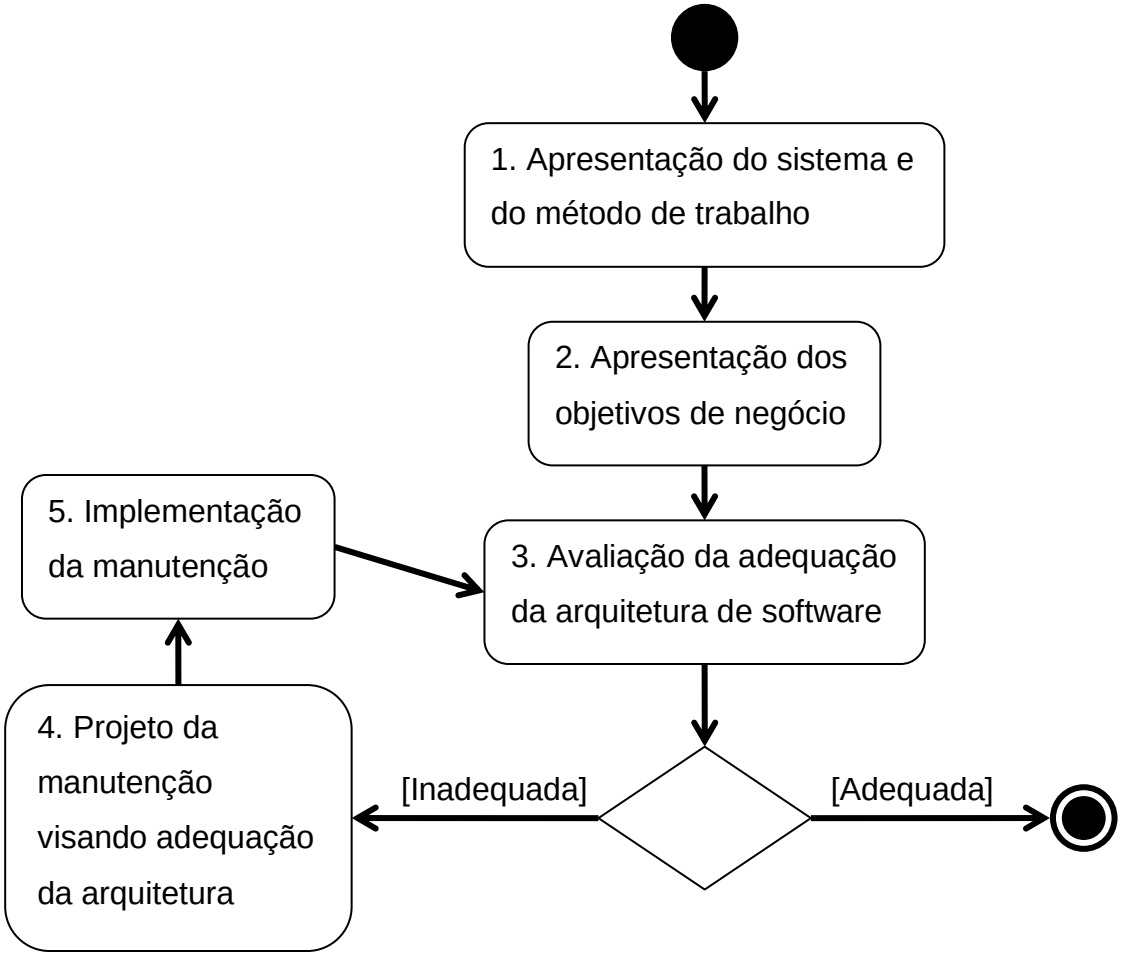


Figura 3.1 – Diagrama do processo do método de manutenção (notação UML).

Nas subseções a seguir, cada uma das fases é aprofundada mostrado-se seus objetivos, quais as atividades a serem desempenhadas, os papéis dos responsáveis por desempenhar cada atividade, e ferramentas apropriadas. Dependendo da atividade, sugerem-se algumas possibilidades de uso de técnicas ou métodos cujos resultados podem ser utilizados como auxílio ou substituição.

Embora não seja explicitado nas fases, um dos papéis importantes no uso do método é o do Facilitador. Este papel é responsável por direcionar o andamento do método, controlando cada fase e verificando que cada atividade esteja sendo realizada a contento. Ele auxilia muitas vezes relembrando o que deve ser realizado em cada fase e como deve ser desempenhada cada atividade.

3.1 Fase 1: Apresentação do sistema e do método de trabalho

O objetivo desta fase é permitir que a Equipe de Manutenção obtenha uma visão geral do sistema, um primeiro contato com o sistema caso a equipe não tenha tido oportunidade anteriormente. Note que esta apresentação é mais ligada à visão do usuário.

O responsável pela execução desta fase pode ser um usuário experiente do sistema. Alternativamente pode-se apresentar alguma documentação ou até submeter a equipe ao uso do sistema por um período. Em muitos casos, membros da própria Equipe de Manutenção já possuem um bom conhecimento do sistema, até porque muitas vezes convivem com o sistema há bastante tempo. Neste caso, pode-se abreviar a apresentação do sistema.

É importante que seja feita a apresentação do método de trabalho para que os participantes e envolvidos possam se programar para as atividades. Em aplicações na indústria percebe-se também a importância de se obter o comprometimento dos responsáveis a desempenhar os papéis em cada fase.

Esta fase é tida como uma forma de preparação. Não desempenhá-la ou fazê-lo de forma muito superficial pode dificultar especialmente as fases 2 e 3 seguintes.

Ao terminar esta fase, segue-se à Fase 2.

3.2 Fase 2: Apresentação dos objetivos de negócio

O objetivo desta fase é apresentar à Equipe de Manutenção quais eram os objetivos de negócio para o qual o sistema fora construído (se esta informação estiver disponível) e principalmente quais os objetivos de negócio atuais.

Conforme visto em 1.2, uma arquitetura adequada é aquela que atende os seus objetivos de negócio (as preocupações e desejos dos *stakeholders* traduzidos na forma de requisitos para a arquitetura). Entretanto também viu-se que uma

arquitetura existente pode tornar-se inadequada e inclusive vimos que um dos principais motivos é a mudança dos objetivos de negócio. Tal como vários métodos de desenvolvimento centrados em arquitetura, o método também parte dos objetivos de negócio.

Para facilitar a realização das fases seguinte é importante levar em conta a forma de apresentação dos resultados desta fase. Tais resultados podem ser descritos como uma lista de regras e diretrizes de negócio. Esta lista pode, melhor ainda, ser estruturada na forma de cenários conforme os que são exemplificados em 2.7.4.2.

Um dos principais responsáveis pela execução desta fase é o Responsável de Negócio com o apoio do Facilitador. A colaboração do Responsável de Negócio é crítica para obter informação relevantes sobre o negócio ao qual o sistema dá suporte.

O Facilitador tem uma tarefa crítica nesta fase. Visto que as influências que darão base à verificação da adequação do sistema são aquelas obtidas nesta fase, habilidades de consultoria do Facilitador e métodos de levantamento de requisitos são altamente recomendadas. Um exemplo de método que pode ser de auxílio é o *Quality Attribute Workshop* (QAW) (BARBACCI et al., 2003). Como ferramenta de apoio neste sentido, o uso de algum modelo de qualidade pode ser muito útil pois, no mínimo, serve como uma espécie de guia ou *checklist* promovendo uma cobertura maior dos requisitos de qualidade. Entretanto, tal *checklist* também pode conter outras características de qualidade dependentes do domínio de aplicação.

No caso de não se captar todos os requisitos necessários, o método prevê que a lista de requisitos fica em aberto. Conforme restrições de tempo, por exemplo, pode-se seguir adiante na execução das fases do método, incrementado a lista quando necessário. Como novos requisitos podem representar impacto na base de comparação da adequação da arquitetura, deve-se também avaliar o impacto nos resultados das fases 3, 4 e 5.

Terminando esta fase, passa-se para a Fase 3.

3.3 Fase 3: Avaliação da adequação da arquitetura de software

Nesta fase, cada grupo efetua uma série de avaliações do sistema de forma a verificar se sua arquitetura é adequada aos seus objetivos. Em especial, é importante poder-se mapear onde (quais os mecanismos) a arquitetura está atendendo os objetivos.

Relembrando o fluxo do método (Figura 3.1), é interessante notar que esta fase pode ser executada após a implementação de modificações (Fase 5). Neste caso, o objetivo é verificar e evidenciar se os objetivos foram atendidos pelo sistema resultante da manutenção.

Alguns métodos de avaliação de arquitetura de software podem ser utilizados para desempenhar esta fase desde que atendam algumas características. Em primeiro lugar, eles devem estar alinhados com a abordagem que este método segue de serem dirigidos pelos guias de negócio. Em segundo lugar, precisam prover resultados que dêem subsídios para verificar se a arquitetura é adequada e caso ela não seja, sejam utilizados para planejar uma solução na fase seguinte. Tais resultados incluem riscos, não-riscos, pontos de sensibilidade, *tradeoffs*, decisões arquiteturais e seus racionais.

Alguns métodos baseados no uso de cenários que foram analisados e que seguem tais critérios são o *Architecture Tradeoff Analysis Method* (ATAM) (KAZMAN; KLEIN *et al.*, 2000) ou até mesmo o *Active Reviews for Intermediate Designs* (ARID) (CLEMENTS, 2000).

Para obter evidências em suas avaliações, os participantes podem usar outras técnicas de apoio para delinear os limites da arquitetura do sistema. Por exemplo, em (BHATTACHARYA; PERRY, 2007) os autores desenvolvem um modelo para rastrear a evolução do software e propõe medidas que indicarão objetivamente a extensão do desvio ou divergência num sistema de software. Outras técnicas de reengenharia podem ser úteis quando não se têm disponíveis as documentações ou pessoal com conhecimento em detalhes do sistema.

Recomenda-se fortemente que sejam utilizadas avaliações estáticas (estruturais) e dinâmicas, com amplo uso de medições. Desta forma pode-se verificar, por exemplo,

padrões de carga e comportamento do sistema. O tipo de medições aplicáveis pode variar bastante, dependendo muito do domínio da aplicação tratada.

Outra técnica interessante é o uso dos chamados POC's destrutivos. Seus objetivos não são de destruir o sistema. São na verdade usados para estressar limites e verificar uma ou mais possíveis deficiências.

Os participantes desta fase são diversos, porém no mínimo precisa-se do Arquiteto – que apresenta onde a arquitetura cumpre os objetivos – e da Equipe de Avaliação (que pode ser a própria Equipe de Manutenção).

No caso do resultado da avaliação indicar que a arquitetura já está adequada, encerra-se o processo, visto que não há necessidade de se realizar a manutenção. Note que, por exemplo, no caso da arquitetura precisar estar preparada para mudanças futuras, isto deve ser encarado como mais um requisito de qualidade e a avaliação deve levar isso em conta. Logo, a avaliação deve levar em conta todos os requisitos.

No caso do resultado da avaliação indicar que há inadequação, segue-se para a Fase 4.

3.4 Fase 4: Projeto da manutenção visando adequação da arquitetura

O objetivo desta fase é produzir uma especificação de manutenção com modificações na arquitetura de forma a atender os objetivos de negócio, tornando a arquitetura de software adequada.

A especificação é similar na forma a uma especificação de projeto de desenvolvimento. Porém o tipo aqui descrito é focado nas mudanças, ou seja, nos mecanismos novos e nos alterados.

É apropriado que esta fase esteja seguindo uma fase de avaliação arquitetural. Assim, os tomadores de decisão usam os riscos descobertos para focar em projetos e alternativas arquiteturais que os ajudem a mitigar tais riscos (NORD et al., 2003).

Neste momento de propostas de solução para os riscos, pode-se usar uma série de técnicas. O uso conjunto de táticas arquiteturais é uma possível alternativa pois sua intenção é ser um meio para obter uma medida satisfatório da resposta de um atributo de qualidade (BACHMANN; BASS; KLEIN, 2002). Em (BASS; KLEIN; BACHMANN, 2001), também é proposto o uso de *attribute primitives* que são padrões arquiteturais que são primitivos para o alcance de atributos de qualidade.

Na seleção das alternativas de solução, deve-se levar em conta os impactos positivos e negativos que elas proporcionam nos atributos de qualidade. Daí a importância de se ter os subsídios tais como pontos de sensibilidade, tradeoffs e decisões arquiteturais e seus racionais estabelecidos na fase de avaliação arquitetural anterior. As medições realizadas na fase anterior também fornecem informações de comportamento e carga do sistema que auxiliam como forma de restrição para selecionar as alternativas.

Tratando-se do uso de padrões arquiteturais, usar um conjunto deles pode ser útil para a montagem da solução. Inclusive como tais padrões representam soluções bastante empregadas e seus resultados e impactos com respeito às qualidades são bem conhecidos tal estratégia permite a composição mais fácil da solução candidata. Outro ponto beneficiado é que os testes para verificação da solução candidata podem ser melhor dirigidos para observar eventuais dúvidas referentes à integração dos padrões.

Tudo isso vai permitir um primeiro conjunto de decisões candidatas à mitigação dos riscos. É então que este método proposto ressalta fortemente a importância de testar a viabilidade da solução candidata na forma de POC's construtivos. Assim, pode-se avaliar dinamicamente a solução candidata, reduzindo-se o esforço de retrabalho no caso de uma solução inadequada ou ainda fornecendo um exemplo de implementação que sirva de modelo e guia para a fase de implementação a seguir.

A construção dos POC's deve ser cuidadosa para que os testes sejam aderentes às situações reais ou o mais próximo possível disto. Não sendo assim, pode-se ter resultados de testes ineficazes que provavelmente levarão a mais uma iteração nas fases 3, 4 e 5 do método.

Como o método proposto não define uma regra para seleção de decisões arquiteturais usadas na manutenção, ele também não define uma forma para

selecionar baseando-se no custo. Para isso, pode-se usar métodos como, por exemplo, o *Cost Benefit Analysis Method* (CBAM) (KAZMAN; ASUNDI; KLEIN, 2002) ou ainda usar metrificações de custo e esforço como, por exemplo, Análise de Pontos de Função (APF) (IFPUG, 2008) ou até mesmo a base histórica da organização.

Ao terminar esta fase, tem início a Fase 5.

Dependendo do contexto de criticidade, pode-se opcionalmente aplicar uma avaliação (como na Fase 3) entre as fases 4 e 5 para verificar antecipadamente possíveis problemas.

3.5 Fase 5: Implementação da manutenção

Esta fase é realizada pela Equipe de Manutenção. Ela corresponde à implementação propriamente dita das modificações especificadas na fase anterior. Esta implementação deve ser feita de forma cuidadosa, sendo sempre guiada pelo projeto fornecido na fase anterior.

Os POC's construtivos gerados na fase anterior podem ser muito úteis como base inicial dos trabalhos. Visto que proporcionam um caminho ou exemplo, podem ser seguidos para fazer as modificações. Muitas vezes pode-se inclusive usá-los como *templates*.

Implementações descuidadas podem causar muitas iterações no método nas fases 3, 4 e 5. Portanto, visando-se a convergência para bons resultados mais rapidamente, recomenda-se que nesta fase de implementação sejam feitos testes sucessivos voltados para a identificação dos problemas.

As ferramentas para auxiliar a implementação são basicamente aquelas que são utilizadas em ambientes de desenvolvimento, como é o caso de IDE's.

Terminada esta fase, segue-se para a Fase 3.

3.6 Subprodutos do método

Durante a execução do método obtém-se uma série de informações que são de grande importância na formação de uma documentação arquitetural. Inclusive tais subprodutos podem ser dispostos em visões como descrevem (CLEMENTS et al., 2002) e (IEEE, 2000). Inclusive, um interessante trabalho foi feito para verificar a aderência entre estes dois trabalhos em (CLEMENTS, 2005).

Os subprodutos produzidos no decorrer do método estão relacionados na tabela a seguir de acordo com as fases onde são produzidos.

Tabela 3.1 - Lista de subprodutos do método por fase.

Fase	Subprodutos
Fase 2 - Apresentação dos objetivos de negócio	• Lista de requisitos na forma de cenários contendo requisitos funcionais e não-funcionais
Fase 3 - Avaliação da adequação da arquitetura de software	• Lista de riscos e não-riscos • Lista de pontos de sensibilidade e *tradeoffs* • Decisões arquiteturais utilizadas e seus racionais • Conjunto de medições realizadas • POC's destrutivos
Fase 4 - Projeto da manutenção visando adequação da arquitetura	• Especificação de projeto (com foco nas alterações) • POC's construtivos (e destrutivos)[1]
Fase 5 - Implementação da manutenção	• O sistema alterado

[1] POC's construtivos e destrutivos são considerados neste trabalho como uma importante parte da documentação como apoio e justificativa de tomadas de decisão.

3.7 Conclusão do capítulo

Neste capítulo apresentou-se o método proposto para realizar manutenção em um sistema utilizando técnicas arquiteturais visando a adequação deste sistema.

O processo iterativo envolvido no método foi descrito também. Para cada uma das fases mostrou-se o objetivo e apontou-se técnicas que podem ser utilizadas, bem como outros métodos que podem ser usados em conjunto. Destacou-se também o papel dos participantes, mostrando suas atividades em cada fase.

Por fim, também foram explicitados o conjunto de resultados (entregáveis) que são gerados na execução do método.

4 ESTUDOS DE CASO DE APLICAÇÃO DO MÉTODO

O método apresentado na seção anterior é resultante de um trabalho desenvolvido durante a pesquisa. Sucessivas aplicações de uso dele serviram como meio de avaliação do método e auxiliaram no seu aperfeiçoamento. Assim, para pontos de dificuldade na execução foram propostas modificações no método introduzindo outras técnicas que ajudassem a mitigar as deficiências.

As aplicações do método foram feitas em ambientes acadêmicos e de indústria. No ambiente acadêmico, houve o enfoque para o ensino de conceitos e técnicas de Arquitetura de Software e Manutenção. Já no ambiente de indústria, o foco principal era a identificação de deficiências de uma arquitetura existente e a geração e implementação de planos de ação para torná-la adequada.

A seguir apresentam-se os dois tipos de aplicações acima, restrições necessárias em cada tipo e os resultados obtidos.

4.1 Aplicações num contexto de ensino

O método apresentado na seção anterior endereça temas muito importantes como manutenção e técnicas arquiteturais. Portanto, uma possível aplicação do método, além de servir para manutenção na indústria, é utilizá-lo como meio de apresentar tais temas a alunos em aulas ou treinamentos relacionados a estes assuntos.

Diversas aplicações do método foram realizadas em um laboratório de graduação de Engenharia de Software do curso de Engenharia de Computação. Durante o curso, os alunos cursam o segundo semestre do quarto ano (de cinco no total) de Engenharia de Computação.

Neste laboratório, o objetivo é apresentar aos alunos o tema de arquitetura de software e seu relacionamento com requisitos de qualidade de software, ou seja, além da preocupação apenas com a funcionalidade. Outro ponto endereçado é o

processo de manutenção, assunto que é menos discutido na graduação. Algumas das técnicas arquiteturais apresentadas também foram eleitas como importantes para aprofundamento. Por fim, julgou-se necessário que os alunos aprimorem o contato com tecnologias atuais.

O contexto da disciplina envolve a simulação de uma situação em que uma equipe recebe a tarefa de efetuar a manutenção em um sistema desconhecido por eles. Mais ainda, tal sistema sofreu alteração dos objetivos de negócio, fazendo-se necessária uma manutenção de forma a atender os novos objetivos. Outro ponto importante é que as tecnologias utilizadas não necessariamente são conhecidas por eles e o seu aprendizado é mais um desafio, mas aproxima os alunos do conhecimento técnico.

Por causa da complexidade do método, em aplicações dele no contexto de ensino deve-se fazer uma especialização de acordo com restrições que incluem quais conceitos se deseja ressaltar, qual o tempo disponível e qual a experiência dos alunos.

Nas aplicações descritas neste estudo de caso, os conceitos mais relevantes que se desejou destacar foram qualidade de produto de software, uso de POC's construtivos e negativos, medições estáticas e dinâmicas, a importância da arquitetura de software como ponte para adequação de um sistema a suas necessidades, manutenção de software e implementação. No período de um semestre letivo com aulas de aproximadamente quatro horas semanais (além de trabalho individual fora de aula), somando um total de onze aulas.

Para auxiliar o entendimento do tipo de resultados que foram obtidos por meio do uso do método no ensino, os resultados finais de alguns grupos foram incluídos neste trabalho e estão apresentados no ANEXO A.

4.1.1 Estrutura do curso

A turma é dividida em grupos de 2 alunos cada. Todos os grupos atuam no mesmo sistema, porém em vertentes distintas. Por conseguinte, os trabalhos não têm

obrigação de serem complementares ou integráveis entre si. Cada par de grupos atua no balanceamento de atributos de qualidade, porém em situações distintas.

Na Figura 4.1 é apresentado um mapa que demonstra como foram divididos os grupos em uma aplicação da disciplina. Nesta aplicação, a turma havia sido dividida em dez grupos que tratariam de cinco tipos de balanceamento de atributos de qualidade, ficando com dois grupos para cada tipo.

Os “nomes” dos grupos foram dados posteriormente, mas estão relacionados na Figura 4.1 para facilitar o entendimento da distribuição. Estes nomes foram dados de acordo com o tipo de solução que foi adotada para a implementação das modificações no sistema.

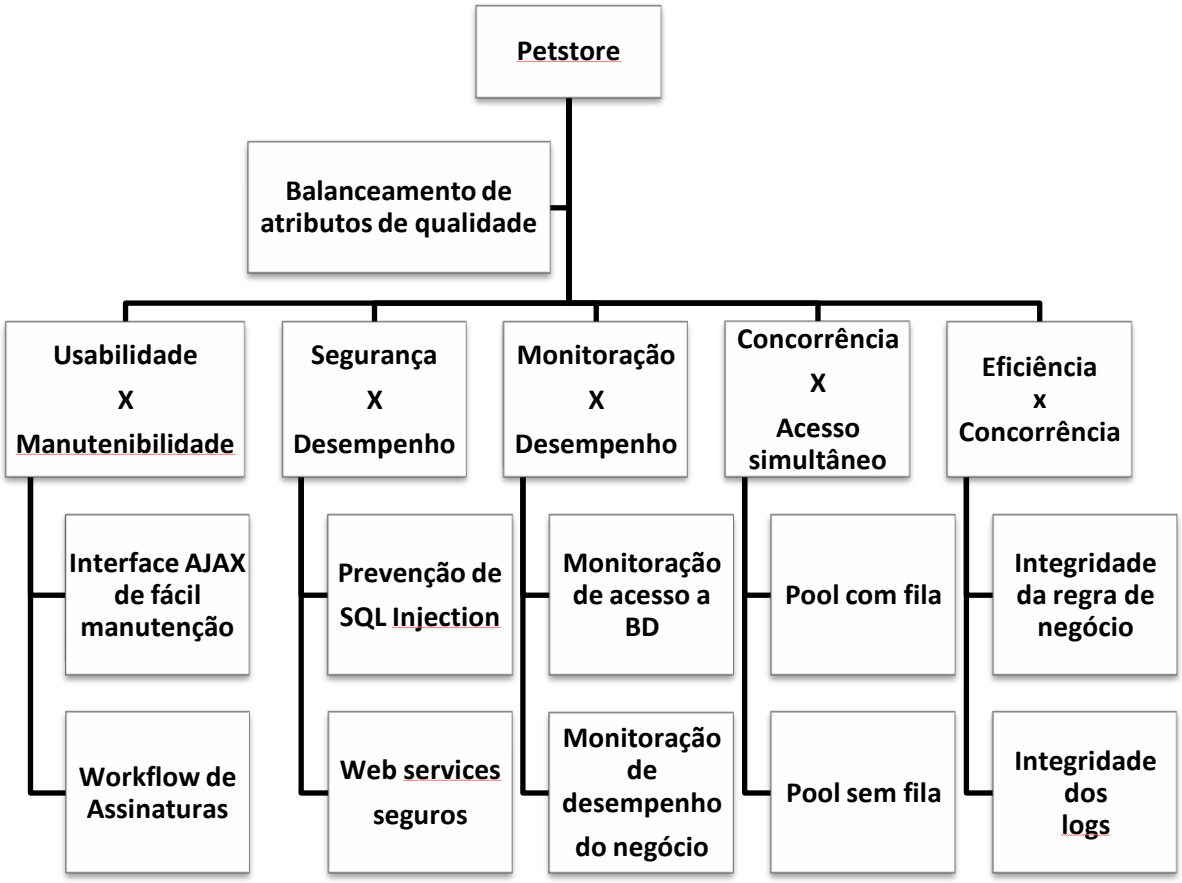


Figura 4.1 – Mapa de grupos em uma aplicação do método numa disciplina de laboratório da graduação de alunos do quarto ano de Engenharia de Computação.

Tomando como exemplo o tema “Concorrência X Acesso simultâneo”, tivemos 2 grupos, sendo que um deles tratou do “Pool com fila” (acesso a informações de dados com pool de conexões e uso de filas) e outro tratou do “Pool sem fila” (acesso

a informações de dados com uso de pool de conexões sem uso de fila) como será visto mais à frente.

4.1.2 Fase 1: Apresentação do sistema e do método de trabalho

O sistema-alvo é apresentado a todos os alunos da mesma forma, visto que todos atuam no mesmo sistema. Isso é feito mostrando-se os objetivos iniciais do sistema, seus procedimentos de instalação e seu funcionamento na visão do usuário.

O sistema-alvo utilizado foi o Java Pet Store, um portal de anúncios de animais pela internet. Neste portal, os usuários podem cadastrar animais para serem comprados, informando o tipo (cachorro, gato, peixe, réptil), subtipo (por exemplo, gato de pelo longo e gato de pelo curto), nome, preço, localização (integrado com o Google Maps), etc. Outros usuários visitantes do site podem buscar animais para comprar. Mais detalhes em (SUN MICROSYSTEMS, 2006).

Os professores apresentam também os passos que serão realizados na execução do método, mostrando inclusive o calendário.

Nesta fase, os alunos também receberam um kit contendo ferramentas e instruções de instalação do sistema e foram instruídos a seguir as instruções para a instalação do sistema com o apoio dos monitores.

4.1.3 Fase 2: Apresentação dos objetivos de negócio

No contexto da disciplina, o que causa a inadequação da arquitetura existente é a mudança dos objetivos de negócio.

Uma simplificação feita nesta fase é que os professores realizam a atividade de fornecer um determinado conjunto de objetivos de negócio. Porém, cada grupo recebe um conjunto diferente. Visto que uma arquitetura adequada é aquela que

atende os objetivos (as preocupações e desejos dos *stakeholders*) e cada grupo visa atender objetivos diferentes, então se espera que os resultados finais dos grupos acabem sendo inerentemente diferentes.

Outra simplificação na aplicação da disciplina foi limitar a poucos objetivos em cada conjunto. Por exemplo, em um dos conjuntos, os alunos deveriam balancear “Concorrência X Acesso simultâneo”, ou seja, o objetivo é atender muitos usuários simultaneamente, concorrendo pelos recursos e gerando assim problemas de concorrência a serem resolvidos.

4.1.4 Fase 3: Avaliação da adequação da arquitetura de software

Nesta fase, cada grupo efetua uma série de avaliações do sistema de forma a verificar se sua arquitetura é adequada aos seus objetivos. Como referência, utilizou-se o ATAM apoiado por medições estáticas e dinâmicas.

A Fase 3 tem o fluxo alternativo de terminar o processo caso a arquitetura já seja considerada adequada. Porém, para que os alunos passem por todo o processo, os professores já escolheram propositadamente situações que sabem que o sistema não atenderia.

Como exemplo, observe o do grupo “Pool sem fila”. Após uma análise estática do código-fonte mais aprofundada, verificou-se que na arquitetura inicial várias classes utilizam a classe CatalogFacade que, por sua vez, fornece acesso aos dados. Em testes comportamentais em tempo de execução, quando há vários usuários fazendo uso de dados simultaneamente (usaram o JMeter para simular esta situação), o sistema inicial tenta atender a todos até que, em um determinado ponto (os alunos chegaram ao número de usuários simultâneos propriamente dito, dentro das condições do laboratório), não consegue responder a tantas requisições e o usuário não recebe uma resposta apropriada (recebe uma tela de erro).

Portanto, os alunos conseguiram observar os limites do sistema dentro dos requisitos impostos. Verificaram então que a arquitetura não era adequada e a manutenção precisaria ser feita.

4.1.5 Fase 4: Projeto da manutenção visando adequação da arquitetura

Nesta fase, cada grupo desenvolve a especificação das modificações na arquitetura de forma a atender os objetivos de negócio, tornando a arquitetura de software adequada. Para atender o objetivo de negócio do grupo “Pool sem fila” (responder a muitos usuários simultaneamente sem que o sistema retorne uma mensagem de erro de HTTP ou que estoure o limite de timeout) seria aceitável que, ao ultrapassar a quantidade de usuários em atendimento simultâneo, o sistema respondesse com uma mensagem ao usuário informando o congestionamento temporário.

A sugestão de implementação foi o uso de um pool de acesso aos dados sem o uso de filas.

Os alunos primeiramente tentaram uma solução e experimentaram na forma de um POC que envolvia uma pequena alteração. Viram então logo que aquela solução não era viável. Esta informação é muito rica, visto que já descobriram um caminho por onde não seguir na fase de implementação.

Os alunos tentaram uma segunda solução. A Figura 4.2 destaca o ponto avaliado na arquitetura inicial e a modificação proposta. Na primeira situação, não há controle da quantidade de acessos simultâneos. Na segunda, o controle é feito pedindo-se a uma classe auxiliar (AccessController) que implementa o controle do pool de acessos. Quando uma requisição é feita, chama-se o método testAccess de AccessController. Este método verifica se o contador interno chegou ao limite (tamanho do pool). Se não, o contador é incrementado e retorna-se VERDADEIRO, permitindo o tratamento da requisição. Quando a requisição termina de ser atendida, o contador é decrementado. Se o contador chega até um limite especificado (tamanho do pool), o método testAccess retorna FALSO, impedindo o tratamento da requisição e fazendo com que a aplicação retorne uma mensagem informando o usuário adequadamente.

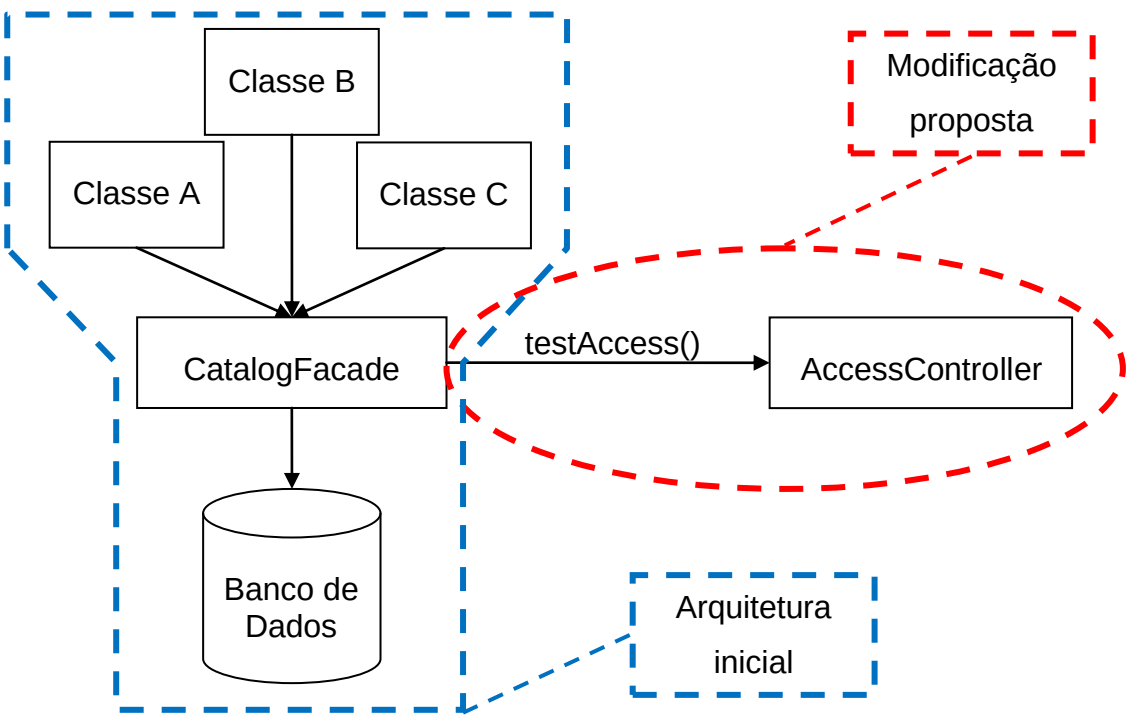


Figura 4.2 – Exemplo de diagrama identificando a Arquitetura inicial e a modificação proposta.

4.1.6 Fase 5: Implementação da manutenção

Esta fase corresponde à implementação propriamente dita das modificações especificadas na fase anterior. Aqui, os grupos utilizaram os POC's construtivos que fizeram parte da especificação da manutenção como base inicial para o desenvolvimento.

O grupo que tomamos como exemplo partiu da solução que havia especificado na fase anterior e validado com o POC, fazendo com que esta fase tivesse seu risco bastante minimizado. E realmente foram bem-sucedidos, como vemos na fase a seguir.

4.1.7 Fase 6: Evidenciação de adequação da arquitetura

Na verdade, o método não possui uma “Fase 6”. Trata-se realmente da Fase 3. Mas esta fase pode agora ser enxergada como uma fase de validação, onde o grupo verifica se a implementação foi bem-sucedida, tornando a arquitetura adequada às necessidades, ou seja, se ela atingiu os requisitos.

O grupo fez as medições e testes e evidenciaram que a solução foi bem implementada e atingiu os requisitos.

4.1.8 Ferramentas utilizadas

Segue uma lista com as principais ferramentas e programas utilizados na aplicação deste curso:

- **Java SE (Standard Edition) e EE (Enterprise Edition):** linguagem Java de programação e extensões para desenvolvimento e execução de sistemas corporativos;
- **Netbeans:** IDE para edição de código, geração de modelos UML, administração do servidor de aplicação e do banco de dados;
- **Sun Application Server:** servidor de aplicações JEE.
- **JavaBD:** SGBD feito em Java e integrado ao IDE Netbeans.
- **JMeter:** teste de carga que simula diversos usuários acessando o sistema simultaneamente.

4.1.9 Resultados

As aplicações do método serviram como boa forma de experimentação e levaram ao aperfeiçoamento do método proposto. Por este motivo, na última aplicação do método (em que ele estava mais “maduro”), o sucesso nas manutenções foi bem maior.

Outro resultado interessante é que como algumas modificações foram bastante localizadas, algumas soluções de grupos distintos poderiam ser facilmente integradas. Entretanto, mais interessante ainda eram as soluções que eram de difícil ou inviável integração. Estas refletiam justamente requisitos de qualidade conflitantes ou ainda que a solução adotada por um dos grupos introduzia um impacto em outra característica de qualidade (que para eles não importava) que era essencial para outro grupo.

Embora houvesse uma restrição de tempo, o andamento das fases foi razoavelmente suave, com entregas parciais marcadas pelo encerramento de cada fase e sem “correrias” ou “super-esforço” nos períodos finais.

Por fim, cabe ressaltar que o uso do método permitiu a geração de resultados efetivos, ou seja, os sistemas foram considerados adequados, em todos os dez grupos, sendo que em nove deles terminaram com conforto de prazo.

4.2 Aplicações na indústria

O método proposto foi aplicado também em ambiente de indústria, especialmente no domínio de sistemas voltados à área financeira.

A seguir, apresenta-se como foi a aplicação do método em um dos casos.

4.2.1 Fase 1: Apresentação do sistema e do método de trabalho

Como a equipe participante do projeto já possuía um conhecimento prévio do sistema devido a interações com o cliente anteriormente, as apresentações iniciais puderam ser abreviadas. Mesmo assim, foi importante para o desenrolar dos trabalhos a apresentação do contexto geral do sistema e onde ele se encaixa no negócio da organização.

A fim de promover o comprometimento dos participantes em cada fase, o grupo facilitador também forneceu um calendário de trabalho e estabeleceu-se qual seria o contato com o cliente para a realização dos trabalhos em conjunto.

4.2.2 Fase 2: Apresentação dos objetivos de negócio

Nesta fase, o levantamento de requisitos foi feito utilizando-se um conjunto inicial fornecido pelo próprio cliente. Interessante notar que o conjunto de requisitos foi refinado cada vez mais com o passar do tempo com o andamento da fase seguinte, de avaliação.

O sistema-alvo era constituído de um conjunto de tipos de módulo encadeados. Cada módulo era consumidor do resultado do módulo anterior na cadeia e produtor de resultados para o próximo módulo da cadeia. Observando-se externamente, o sistema provia o acesso a transações bancárias a serviços externos.

Algumas restrições levantadas nesta fase regravam quanto à obediência a determinadas normas corporativas e uso de alguns módulos adquiridos.

4.2.3 Fase 3: Avaliação da adequação da arquitetura de software

Uma vez obtido um conjunto inicial de requisitos, passou-se a investigar se a arquitetura existente era adequada. Para isso, utilizou-se uma série de estratégias de análise.

O grupo avaliador estava principalmente interessado na identificação dos riscos (fortes candidatos a fonte de inadequação da arquitetura). Então, a abordagem inicial utilizada foi de mapear que tipo de táticas arquiteturais (relacionadas a atributos de qualidade) já estavam sendo aplicadas no sistema. Com isto, pôde-se vislumbrar a cobertura das iniciativas existentes e, mais ainda, perceber que tipos de táticas não estavam sendo utilizados, o que representaria um potencial foco de melhorias.

Outra forma utilizada foi o estabelecimento de um conjunto de cenários representativos que foram validados junto ao cliente. Daí, para cada cenário, identificou-se possíveis “sub-cenários” que representavam possíveis problemas que poderiam ocorrer e como o sistema o trataria. Mais ainda, esta abordagem ajudou a identificar situações não tratadas e que representavam riscos para o sistema.

O uso de medições do sistema também forneceu informações importantes como uso da infra-estrutura utilizada (incluindo memória, CPU, uso de rede), a carga a ser suportada em quantidade de acessos simultâneos, uso de banco de dados (quantidade de registros e tempo de acesso), padrão de comportamento de uso do sistema, tamanho de filas, gargalos na estrutura de módulos, etc.

Tudo isso ajudou a incrementar a lista de requisitos.

4.2.4 Fase 4: Projeto da manutenção visando adequação da arquitetura

Uma vez com a lista de requisitos, foram sugeridas diversas decisões arquiteturais para atacar os riscos e requisitos identificados.

Foi feito o empacotamento das modificações propostas e, para cada pacote, utilizou-se metrificações para fornecer estimativas de esforço e custo. Tal empacotamento acabou gerando um pacote mínimo de modificações que permitiam então que outros pacotes pudessem ser implantados, ou seja, os outros pacotes tinham dependências para este pacote mínimo.

4.2.5 Resultados

A aplicação neste exemplo foi interrompida ao final da Fase 4, proporcionando uma especificação para modificações. Este material será então utilizado pelo cliente para a tomada de decisões quanto ao quê e quando passar à fase de implementação.

Foi interessante notar que na fase de avaliação da arquitetura existente, em geral havia uma espécie de percepção da inadequação por parte dos *stakeholders*. Inclusive em praticamente todas as vezes esta percepção era a motivação da contratação deste tipo de serviço.

O uso de medições estáticas e dinâmicas foi fundamental para a identificação de pontos críticos na arquitetura, identificando suas deficiências. Isto auxiliou a formação de objetivos mais claros para a formação dos planos de ação. Também nas fases de avaliação da arquitetura na forma de validação, as medições formavam evidências claras do cumprimento ou não dos objetivos.

Nas aplicações voltadas à indústria, os planos de ação foram empacotados de forma que as modificações pudessem ser feitas em partes, visando à adequação iterativa. Nestas situações, métricas e estimativas de custo foram aplicadas a cada pacote de modificações, permitindo que gestores do negócio pudessem tomar a decisão de implementação ou não das modificações.

5 CONCLUSÕES

5.1 Análise dos resultados dos estudos de caso

Pode-se pensar inversamente em como seria não seguir um método assim. Neste caso, os sucessos acabam sendo casuais e muito retrabalho ocorre por conta de tentativas e erros. Mais ainda, se as modificações não seguirem uma estratégia guiada pelos objetivos de negócio e qualidade, muito provavelmente gerarão sistemas inadequados às necessidades.

Resultados obtidos em aplicações no contexto de ensino e na indústria mostraram os benefícios do uso do método em tornar os sistemas adequados.

5.2 Resumo das contribuições do trabalho

O método proposto fornece uma abordagem para realizar manutenção de um sistema de software utilizando técnicas arquiteturais. Avaliando os resultados das aplicações do método, viu-se que o método foi bem-sucedido em auxiliar o encadeamento das técnicas e os sistemas resultantes das manutenções foram considerados adequados, ou seja, alinhados com os objetivos de negócio.

Durante a construção do método foram estudadas diversas técnicas arquiteturais e foram selecionadas algumas para serem aplicadas de forma prática como facilitadores no processo de manutenção. Tal aplicação permitiu avaliar os resultados de tais técnicas. Os pontos fortes foram então incorporados no método. Já para os pontos críticos, verificou-se que outras técnicas poderiam ser utilizadas em conjunto para o benefício do método.

Viu-se também que há aplicações do método também no contexto de ensino, visto que no uso do método, os participantes são expostos a conceitos importantes de manutenção, arquitetura de software e técnicas de apoio arquitetural.

5.3 Conclusão

Viu-se a fundamental importância do uso de arquitetura e técnicas arquiteturais para realização de manutenção de sistemas de software. Este trabalho compõe uma iniciativa para abordar o problema de manutenção sobre sistemas que se tornam inadequados.

Embora as aplicações do método proposto tenham sido diferentes, tanto no objetivo principal quanto no desenrolar das atividades, seus resultados foram considerados bastante satisfatórios, tanto no contexto de ensino quanto no contexto de indústria.

5.4 Trabalhos futuros

O método proposto é resultado da pesquisa que vem sendo desenvolvida que procura aplicar de forma prática os conhecimentos nas áreas de melhoria do processo de manutenção, melhoria de qualidade de software, arquitetura de software e aplicações no ensino, dentre outras.

A aplicação no ensino apresentada neste trabalho tem sido muito útil no ensino de conceitos de Arquitetura de Software e Manutenção a alunos de graduação. Ele permite que os alunos experimentem na prática os conceitos, consolidando o aprendizado. Outros trabalhos futuros nesta linha podem ser a especialização do método de forma a ressaltar conceitos diferentes dos que foram apresentados no curso descrito em 4.1.

O uso de técnicas arquiteturais tem ajudado muito nos ciclos de desenvolvimento de software e aplicando-se tais técnicas também no ciclo de manutenção se mostrou

bastante útil. Aplicando o método em diversos contextos na indústria (tendo-se como objetivo a manutenção) e na educação (tendo-se como objetivo o ensino), tem-se conseguidos resultados muito positivos. Trabalhos futuros neste sentido compreendem a aplicação do método em domínios de aplicação variados visando o seu refinamento.

As técnicas aqui apresentadas são apenas um subconjunto das técnicas arquiteturais existentes. Trabalhos futuros podem envolver o uso de outras técnicas aplicadas à manutenção ou até mesmo estas técnicas aqui apresentadas aplicadas em outras fases do ciclo de vida do software. Inclusive o método pode ser refinado também para comportar outras técnicas.

REFERÊNCIAS

ALBIN, S. **The Art of Software Architecture: Design Methods and Techniques**: John Wiley & Sons, Inc. 2003. 352 p.

BABAR, M. A., L. ZHU e R. JEFFERY. **A Framework for Classifying and Comparing Software Architecture Evaluation Methods**. Proceedings of the 2004 Australian Software Engineering Conference (ASWEC'04): IEEE Computer Society 2004.

BACHMANN, F.*, et al.* **The Architecture Based Design Method**. Software Engineering Institute. Pittsburgh: January 2000. 2000. (CMU/SEI-2000-TR-001).

BACHMANN, F., L. BASS e M. KLEIN. **Illuminating the Fundamental Contributors to Software Architecture Quality**. Software Engineering Institute. Pittsburgh: August 2002. 2002. (CMU/SEI-2002-TR-025).

BARBACCI, M. R.*, et al.* **Quality Attribute Workshops (QAWs), Third Edition**. Software Engineering Institute. Pittsburgh: August 2003. 2003. (CMU/SEI-2003-TR-016).

BASS, L., P. CLEMENTS e R. KAZMAN. **Software Architecture in Practice, Second Edition**: Addison-Wesley Professional. 2003

BASS, L. J., M. KLEIN e F. BACHMANN. **Quality Attribute Design Primitives and the Attribute Driven Design Method**. Revised Papers from the 4th International Workshop on Software Product-Family Engineering: Springer-Verlag 2001.

BHATTACHARYA, S. e D. E. PERRY. **Architecture Assessment Model for System Evolution**. Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture: IEEE Computer Society 2007.

CLEMENTS, P. **Active Reviews for Intermediate Designs**. Software Engineering Institute. Pittsburgh: August 2000. 2000. (CMU/SEI-2000-TN-009).

CLEMENTS, P. **Comparing the SEI's Views and Beyond Approach for Documenting Software Architectures with ANSI-IEEE 1471-2000**. Software Engineering Institute. Pittsburgh: July 2005. 2005. (CMU/SEI-2005-TN-017).

CLEMENTS, P.*, et al.* **Documenting Software Architectures: Views and Beyond**: Pearson Education. 2002. 512 p.

CLEMENTS, P., R. KAZMAN e M. KLEIN. **Evaluating Software Architectures: Methods and Case Studies**: Addison-Wesley Professional. 2002

GARLAN, D. **Software architecture: a roadmap**. Proceedings of the Conference on The Future of Software Engineering. Limerick, Ireland: ACM Press 2000.

GARLAN, D. e M. SHAW. **Software Architecture: Perspectives on an Emerging Discipline**: Prentice Hall. 1996

GRAAF, B. **Model-Driven Evolution of Software Architectures**. Proceedings of the 11th European Conference on Software Maintenance and Reengineering: IEEE Computer Society 2007.

GUNNALAN, R., M. SHERESHEVSKY e H. H. AMMAR. **Pseudo dynamic metrics [software metrics]**. Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications: IEEE Computer Society 2005.

HOFMEISTER, C.*, et al.* **Generalizing a Model of Software Architecture Design from Five Industrial Approaches**. Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture: IEEE Computer Society 2005.

HOFMEISTER, C., R. NORD e D. SONI. **Applied software architecture**: Addison-Wesley Longman Publishing Co., Inc. 2000. 397 p.

IEEE. **IEEE Std 1471: Recommended practice for architectural description of software-intensive systems**. 2000

IFPUG. **IFPUG: International Function Point Users Group**. Disponível em: <http://www.ifpug.org/>. 2008. Acesso em: jun. 2008.

ISO. **ISO 14598-5: Information technology - Software product evaluation - Part 5: Process for evaluators**. 1998

ISO. **ISO 9126-1: Software engineering - Product quality - Part 1: Quality model**. 2001

JAZAYERI, M., A. RAN e F. V. D. LINDEN. **Software architecture for product families: principles and practice**: Addison-Wesley Longman Publishing Co., Inc. 2000. 257 p.

KAZMAN, R., J. ASUNDI e M. KLEIN. **Making Architecture Design Decisions: An Economic Approach**. Software Engineering Institute. Pittsburgh: September 2002. 2002. (CMU/SEI-2002-TR-035).

KAZMAN, R. e L. BASS. **Categorizing Business Goals for Software Architectures**. Software Engineering Institute. Pittsburgh: December 2005. 2005. (CMU/SEI-2005-TR-021).

KAZMAN, R., M. KLEIN e P. CLEMENTS. **ATAM: Method for Architecture Evaluation**. Software Engineering Institute. Pittsburgh: August 2000. 2000. (CMU/SEI-2000-TR-004).

KRUCHTEN, P. **The Rational Unified Process: An Introduction, Second Edition**: Addison-Wesley Longman Publishing Co., Inc. 2000. 320 p.

NORD, R. L.*, et al.* **Integrating the Architecture Tradeoff Analysis Method (ATAM) with the Cost Benefit Analysis Method (CBAM)**. Software Engineering Institute. Pittsburgh: December 2003. 2003. (CMU/SEI-2003-TN-038).

PARNAS, D. L. **Software aging**. Proceedings of the 16th international conference on Software engineering. Sorrento, Italy: IEEE Computer Society Press 1994.

PAULISH, D. J., L. BASS e D. J. PAULISH. **Architecture-Centric Software Project Management: A Practical Guide**: Addison-Wesley Longman Publishing Co., Inc. 2001. 320 p.

PERRY, D. e A. WOLF. Foundations for the study of software architecture. SIGSOFT Softw. Eng. Notes, v.17, n.4, p.40-52. 1992.

SADOU, N., D. TAMZALIT e M. OUSSALAH. **A unified Approach for Software Architecture Evolution at different abstraction levels**. Proceedings of the Eighth International Workshop on Principles of Software Evolution: IEEE Computer Society 2005.

SOMMERVILLE, I. **Software Engineering (7th Edition)**: Pearson Addison Wesley. 2004

SUN MICROSYSTEMS. **Blueprints: Java Pet Store Reference Application**. Disponível em: <https://blueprints.dev.java.net/petstore/>. 2006. Acesso em: mar. 2008.

ANEXO A - TRABALHOS DOS ALUNOS

Neste apêndice será apresentada uma pequena parte dos trabalhos desenvolvidos por alunos que participaram de estudos de casos de aplicações do método em um contexto de ensino.

Exemplo 1 - PetStore - Concorrência e Simultaneidade

Contexto Considerado

O foco de trabalho de nossa equipe foi o tradeoff entre Concorrência e Acesso Simultâneo mais precisamente no tratamento de erros oriundos da grande quantidade de acessos à base de dados.

O sistema recebido não conta com nenhum tratamento para os erros estudados, retornando ao usuário uma tela com código Java descrevendo o erro encontrado.

Objetivo

Nosso objetivo é controlar esse erro oriundo da grande quantidade de acessos, retornando uma pagina mais amigável ao usuário quando não for possível o acesso aos dados, melhorar a resposta do sistema implementando um timeout aos acessos e criar um log de acessos ao sistema de banco de dados.

Requisitos do Sistema

Requisitos Funcionais

RF1. Tela amigável ao cliente, quando não for possível acessar o banco de dados;

Descrição: Quando não for possível acessar o banco de dados, o sistema vai retornar uma tela em jsp com uma mensagem informando ao usuário que nosso sistema está em uma situação critica e como proceder nesse caso.

Requisitos Não Funcionais

.

RNF1. Geração de arquivo de log com os motivos de erro de acesso para uso de estatística de erro, que ficara disponível;

Descrição: Será gerado um arquivo de log com as informações do sistema captadas periodicamente, que mostrarão o comportamento do sistema de acordo com o número de acessos e hora de acesso.

RNF2. Controlar o tempo de resposta de acesso ao banco de dados incluindo um time-out.

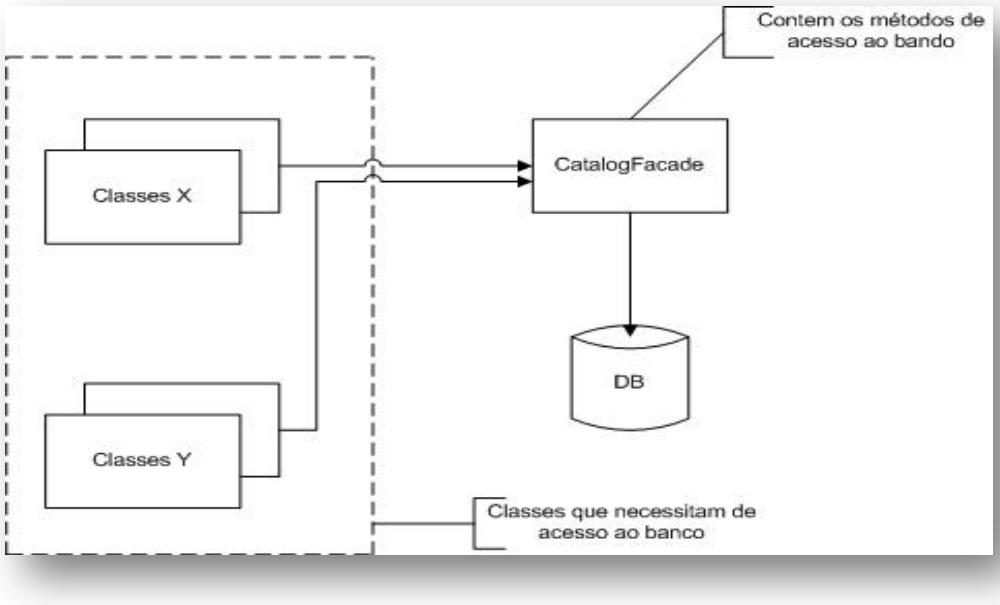
Descrição: O tempo de resposta do sistema ao usuário será controlado por uma função que mede o tempo desde o pedido do acesso até a o recebimento do dado do banco. Se esse tempo for superior à um valor estipulado o sistema retornará uma tela como descrevendo o que aconteceu como ocorre quando não consegue acessar o banco de dados.

Especificação Técnica

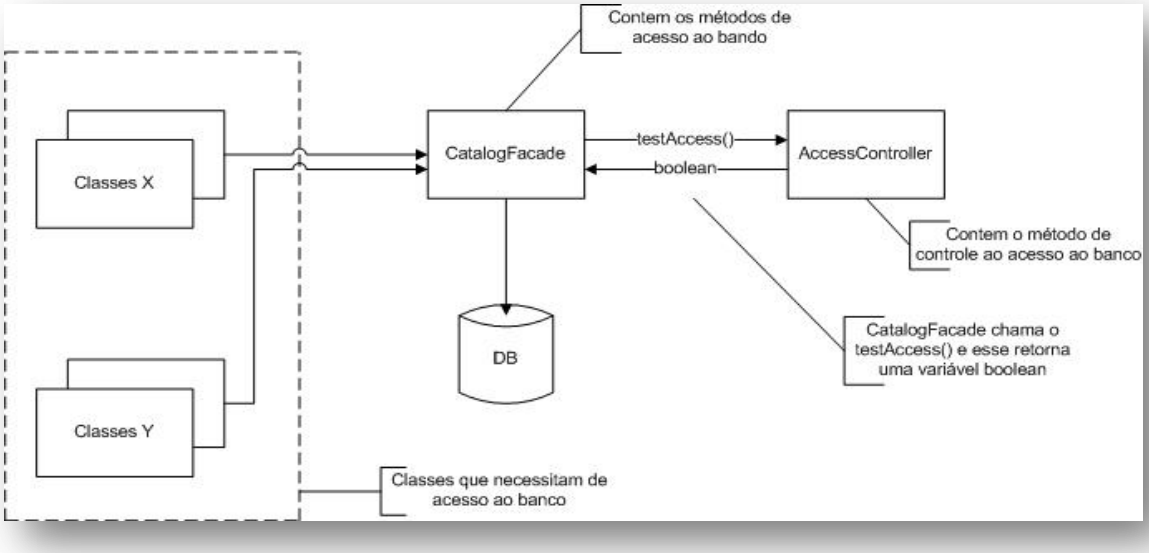
Primeiramente pensamos como fazer um monitoramento e controle efetivo do uso do banco de dados. Com base nas etapas anteriores, sabíamos que a classe que faz o acesso ao banco é a CatalogFacade. Nela temos os métodos com as queries.

Foi feito então um classe (AccessController) que tem o intuito que monitorar e controlar esse acesso. Para tanto, mudamos a CatalogFacade de modo que para cada método de acesso, antes dele executar a query, chamamos um método da AccessController(testAccess) que testa se o numero de acessos não ultrapassou o limite estipulado e caso não, atualizamos o controlador de acesso e autorizamos a execução da query. Caso o limite de acesso exceda ao especificado, a query não é executada.

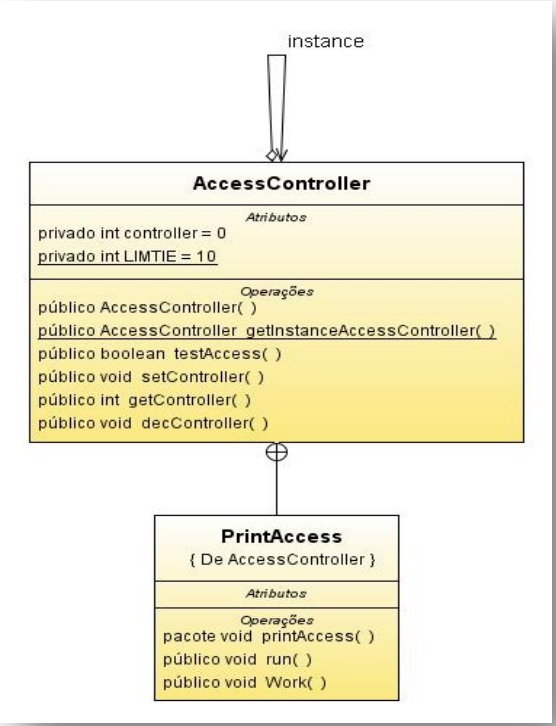
A arquitetura original é mostrada abaixo:



A arquitetura atual depois da modificação:



Esboço de como foi modelado a classe AccessController:



Especificação dos Testes

Para verificar o correto funcionamento do sistema nós realizaremos testes que simularão situações extremas do sistema.

Modificações no programa

Modificações no programa serão feitas dentro do código do Java PetStore para simular atraso, maior quantidade de acessos e demora no acesso.

Para simular demora de acesso ao banco de dados iremos colocar um delay diretamente dentro do código, mais precisamente nas funções da classe CatalogFacade, que fazem query no banco de dados. Algo similar será feito para simular o atraso no acesso, mas os delay serão colocados antes do acesso do banco de dados.

Para simular uma quantidade grande de acessos nós iremos colocar um valor inicial maior que zero no controller da classe AcessController, assim o sistema reagirá

como se tivesse com mais acessos do que os existentes, garantindo a situação de estouro nos acessos.

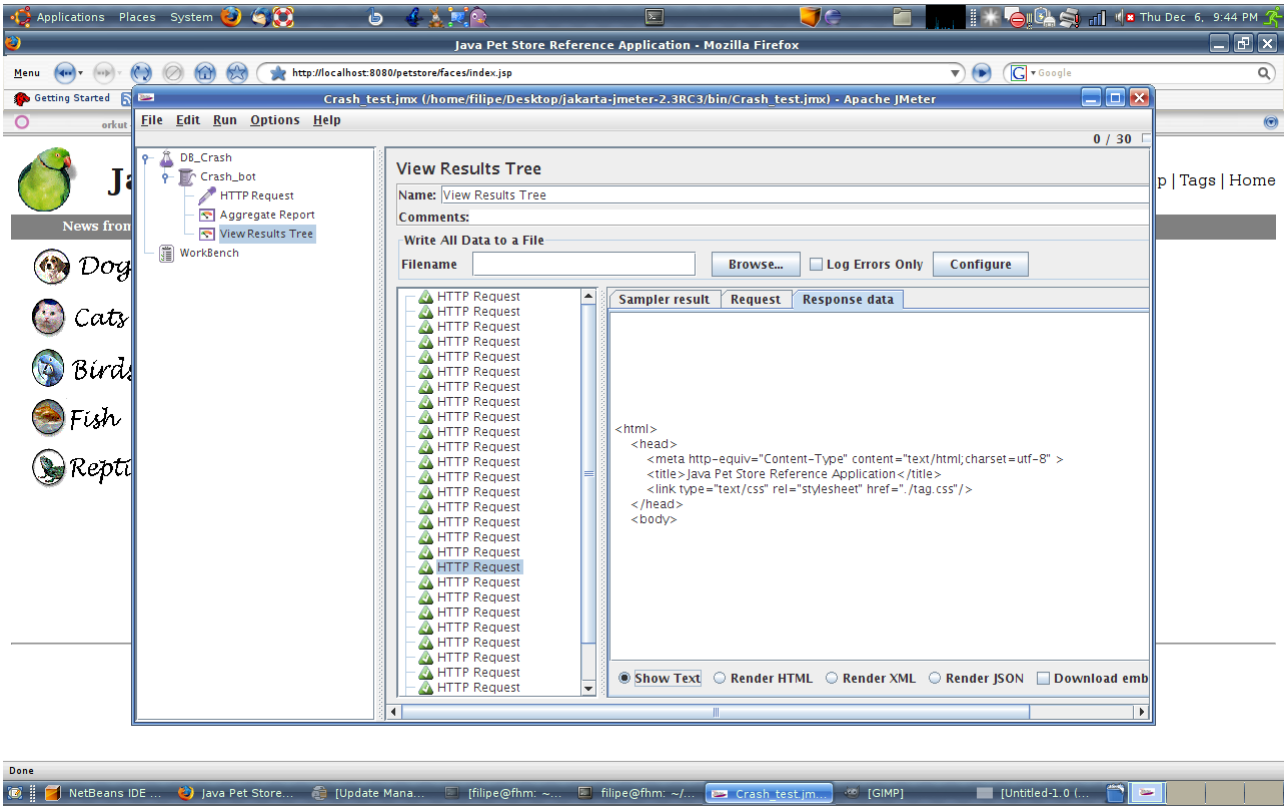
Mesa de Testes

Todos os testes serão feitos utilizando a ferramenta JMeter. Está criará threads concorrentes de acesso a paginas do sistema PetStore que por sua vez farão acesso ao banco de dados e, portanto, cairão no nosso caso de estudo.

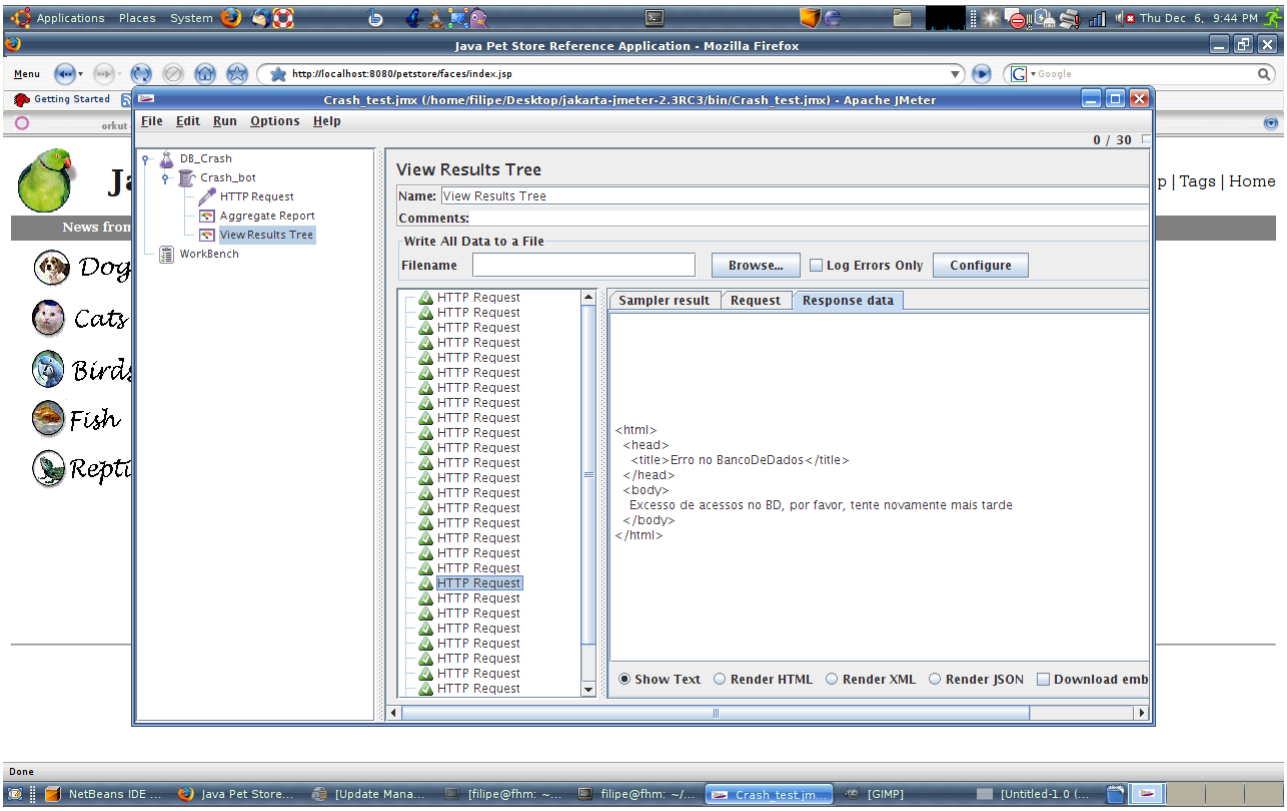
Para os testes realizados no começo do desenvolvimento utilizaremos somente uma maquina acessando o PetStore a partir do JMeter. Mas, com o amadurecimento do projeto, colocaremos mais uma ou duas maquinas instanciando o JMeter e rodando os testes para conseguirmos situações mais extremas e, portando, melhores resultados experimentais.

Resultados dos Testes

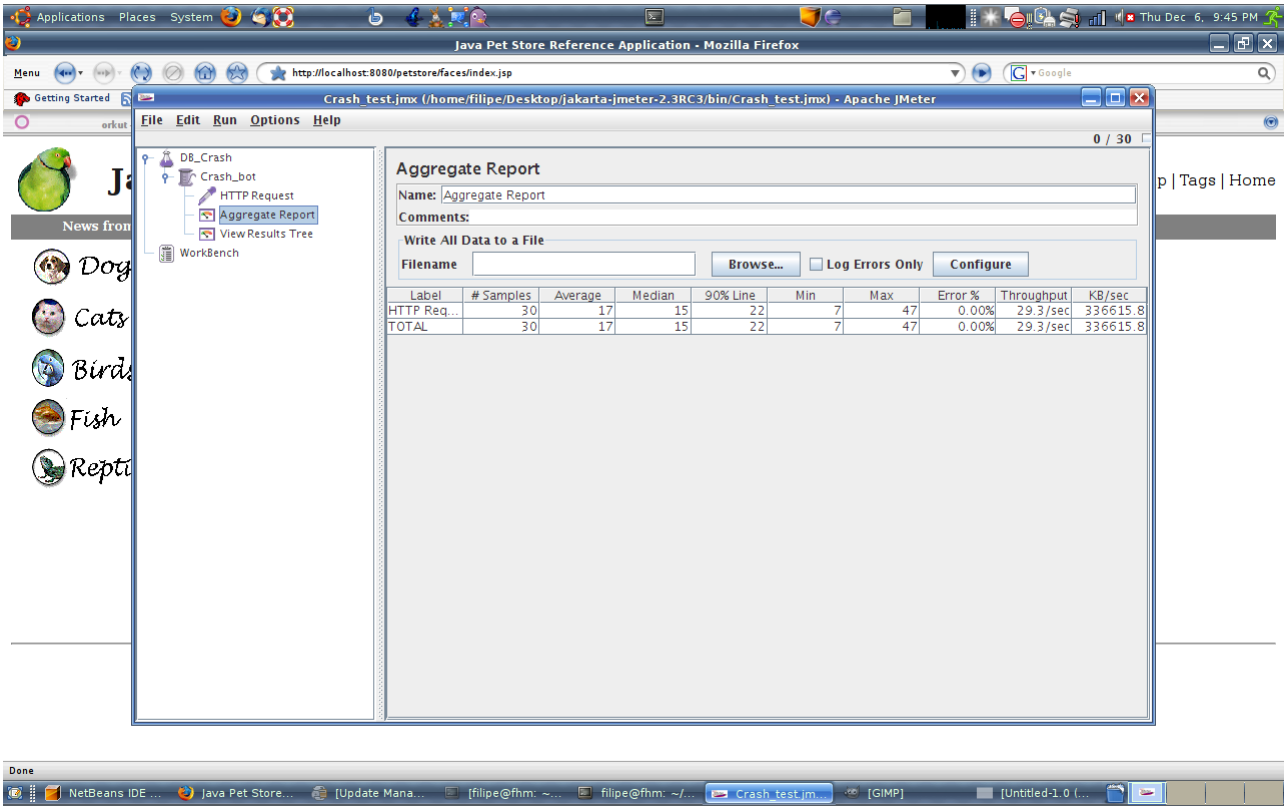
A figura abaixo mostra uma situação em que o acesso ao banco de dados foi bem sucedido. A pagina mostrada pelo JMeter é a pagina principal do JavaPetStore.



Com o limite atualizado para um valor baixo, conseguimos gerar e capturar a pagina de erro de acesso ao banco de dados. Esta por sua vez é mostrada na figura abaixo. Através da interface do JMeter.



Na figura seguinte é mostrado que com o controle de acesso ao banco as requisições HTTP são retornadas com sucesso, mesmo ocorrendo algum estouro de acesso ao banco de dados.



Exemplo do formato do Log

Abaixo é apresentado um exemplo de como ficou o log do sistema.

Ocorreram: 1 acessos 6/11/2007 11:36:30:1196984190148

Ocorreram: 1 acessos 6/11/2007 11:38:31:1196984311501

!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:31:1196984311525

!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:31:1196984311526

!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:31:1196984311529

!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:31:1196984311529

!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:31:1196984311533

!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:31:1196984311536

!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:31:1196984311541

!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:31:1196984311542

!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:31:1196984311544

Ocorreram: 1 acessos 6/11/2007 11:38:57:1196984337182

!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:57:1196984337194

!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:57:1196984337207

!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:57:1196984337207

!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:57:1196984337210

Ocorreram: 1 acessos 6/11/2007 11:38:57:1196984337218
!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:57:1196984337221
Ocorreram: 1 acessos 6/11/2007 11:38:57:1196984337243
!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:57:1196984337252
!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:57:1196984337254
!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:57:1196984337256
Ocorreram: 1 acessos 6/11/2007 11:38:57:1196984337276
Ocorreram: 1 acessos 6/11/2007 11:38:57:1196984337296
!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:57:1196984337300
!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:57:1196984337306
!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:57:1196984337314
!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:57:1196984337318
!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:57:1196984337321
!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:57:1196984337328
!!!Ocorreu um numero maior de acessos que o banco de dados permite!!!6/11/2007 11:38:57:1196984337332

Temos nele então todos os acessos que foram realizados. Ele mostra o dia em que ocorreu o acesso, bem como a hora, o minuto, o segundo e os milisegundos.
Ainda nessa parte faltou a implementação da impressão do tempo que cada acesso durou.

Analise estática

Foi acrescentado ao código cerca de duzentas linhas de código. Isso engloba a classe AccessController e algumas modificações no CatalogFacade e na index.jsp.
Com o AccessController geramos uma impedância em seria no processo de consulta ao banco.

Anexo POC´s e Resultados
Primeira tentativa de implementação(não funcionou)

A primeira tentativa de controle e monitoramento do banco de dados não deu certo. Nela foi criada uma nova classe com o nome de AccessController em que nela seria executada as queries e o CatalogFacade seria simplesmente uma classe de aclopamento onde se poderia fazer o controle de acesso. Isso foi feito porque todas

as classes já chamavam a CatalogFacade, e para não mudar todas as outras classes mudados simplesmente a CatalogFacade.

Portanto a classe AccessController passou a ser instancia pelo Java Application Server.

Abaixo é mostrado um pedaço do codigo da AccessController.

```
    @PersistenceUnit(unitName="PetstorePu")
    private EntityManagerFactory emf;

    @Resource
    private UserTransaction utx;

    private static final boolean bDebug=false;

    public AccessController(){ }

    public void contextDestroyed(ServletContextEvent sce) {
        //close the factory and all entity managers associated with it
        if (emf.isOpen()) emf.close();
    }

    public void contextInitialized(ServletContextEvent sce) {
        ServletContext context = sce.getServletContext();
        context.setAttribute("CatalogFacade", /*new
CatalogFacade()*/this);
    }

   @SuppressWarnings("unchecked")
   public List<Category> getCategories(){
        EntityManager em = emf.createEntityManager();
        List<Category> categories = em.createQuery("SELECT c FROM Category
c").getResultList();
        em.close();
        return categories;
    }
```

Exemplo de como ficou um dos metodos da CatalogFacade:

```
//   @SuppressWarnings("unchecked")
   public List<Category> getCategories(){
//        EntityManager em = emf.createEntityManager();
//        List<Category> categories = em.createQuery("SELECT c FROM
Category c").getResultList();
//        em.close();
//        return categories;
        return ac.getCategories();
    }
```

Em resumo, a classe AccessController passou a ser literalmente a CatalogFacade e esta por fim ficou só como uma classe de acoplamento. Onde futuramente faria o controle de acesso.

Mas essa implementação não funcionou. Mudamos o web.xml onde ele chamava a catalogFacade, mas o problema não foi identificado e ele simplesmente não chegava nem a abrir a pagina.

Segunda Tentativa (esta funcionando)

Na CatalogFacade foi simplesmente implementado em todos os metodos dele de acesso um metodo de condição da AccessController que permite ou não a execução da query.

A classe AccessController é mostrada abaixo:

```
public class AccessController {
    /** Variavel de controle de acesso */
    private int controller = 0;
    /** Limite de acesso */

    private final static int LIMTIE = 10;
    private static AccessController instance = new AccessController();

    /** Creates a new instance of AccessController */
    public AccessController() {
        PrintAccess pa = new PrintAccess();
        pa.start();
    }
```

```
    public static AccessController getInstanceAccessController(){

        return instance;
    }

    public synchronized boolean testAccess(){
        if (getController() < LIMTIE){
            setController();
            //System.out.println("Ocorreram: " + this.controller + "
acessos");
            return true;
        }else{
            System.out.println("!!!Ocorreu um numero maior de acessos que
o banco de dados permite!!!");
            return false;
        }
    }

    public synchronized void setController(){
        this.controller++;
    }

    public int getController(){
        return this.controller;
    }

    public synchronized void decController(){
        this.controller--;
    }
}
```

```
    class PrintAccess extends Thread {
        void printAccess() {
            start();
        }
        public void run() {
            Work();  //some code that executes the functionality of the
thread
```

```
        }
        public synchronized void Work() {
            while(true){
                try {
                    sleep(3000);
                    System.out.println("Numero de Acessos: " +
getController());

                } catch (InterruptedException ex) {
                    ex.printStackTrace();
                }
            }
        }
    }
```

Os métodos da catalogFacade seguem o padrão do mostrado abaixo:

```
    @SuppressWarnings("unchecked")
    public List<Category> getCategories(){
        if(ac.testAccess()){
            EntityManager em = emf.createEntityManager();
            List<Category> categories = em.createQuery("SELECT c FROM
Category c").getResultList();
            em.close();
            ac.decController();
            return categories;
        }else{
            return null;
        }
    }
```

Como podemos ver pelo código acima, retornamos um “null” caso o acesso ultrapasse o limite estipulado. Utilizamos isso para gerar a pagina ao usuário. Retornando um “null”, na pagina index.jsp capturamos ele como uma exceção de NullPointer. Identificamos a exceção , e então geramos a nossa pagina HTML. Uma melhoria seria em vez de usar uma exceção de NullPointer seria nós criarmos o nosso próprio tipo de exceção.

Exemplo 2 - PetStore – Segurança e Desempenho

Contexto Considerado

Nos dias atuais, onde a internet é muito utilizada de diversas maneiras, uma das mais importantes e que merece uma grande atenção é o comércio eletrônico. Os sistemas de comércio eletrônico devem ser muito bem implementados, pois estes sites irão lidar com dinheiro dos clientes.

Neste projeto será tratado um sistema específico, o *Sun Pet Store*, um sistema que foi desenvolvido pela Sun com intuito de demonstrar as possibilidades do uso dos recursos Web da Sun. Este sistema implementa uma loja de animais online, onde é possível comprar animais, cadastrar animais para venda, procurar animais, entre outras funções. O sistema se comunica com o site *Paypal* para simular as transações financeiras.

Serão feitas varias alterações neste projeto, para poder torná-lo melhor, atendendo mais requisitos não funcionais que foram adicionados ao projeto. Este relatório trata a parte relativa à segurança da base de dados do sistema. Iremos tratar da segurança contra acessos indevidos à base de dados, seja via Web, por *SQL Injection*, ou utilizando-se outro programa que se conecta diretamente à base e a altera.

Objetivo

Este projeto visa tornar o sistema mais seguro, no que diz respeito à ataques à base de dados, evitando desta maneira que ataques ocorram, o que poderia causar perda de dados e eventualmente perda de dados também.

Este projeto está sendo desenvolvido para serem aplicados os conceitos, aprendidos na teoria, em um projeto real, com dificuldades e complexidade reais, e, desta maneira, aprender a realizar um projeto de Engenharia de Software completo.

Requisitos

Foram propostas algumas mudanças ao projeto, e com estas mudanças foram gerados novos requisitos, Funcionais e Não Funcionais. Requisitos estes que irão contribuir para melhorar a segurança da base de dados do sistema. Abaixo serão listados e explicados os requisitos que foram adicionados ao sistema.

Requisitos Funcionais

Requisito Funcional 1:

Deve haver uma única classe que acessa o banco de dados e todas as requisições de acesso ao banco de dados, seja leitura ou escrita, deve ser feita através desta classe.

Descrição:

Para evitar mudanças indesejadas ao banco de dados, ou mesmo a mudança de dados ao mesmo tempo, o que causaria um erro, deve haver uma e apenas uma classe que acessa o banco de dados. Qualquer outra classe que precise acessar o banco de dados deve fazê-lo através desta classe.

Requisito Funcional 2:

Será permitido que outros programas tenham acesso ao banco de dados, mas estes acessos devem ser feitos através da classe de acesso ao banco de dados que foi citada no item acima.

Descrição:

Este requisito funcional é necessário para permitir o acesso de outros programas ao banco de dados, desde que sejam feitos através da classe de acesso ao banco de dados, que foi especificada no Requisito Funcional 1.

Requisitos Não Funcionais

Requisito Não Funcional 1:

A usabilidade, a disponibilidade, a manutenibilidade, e a portabilidade do sistema não serão alteradas pela implementação das alterações propostas.

Descrição:

Serão feitas algumas mudanças no sistema, mas estas mudanças não podem alterar em nada os critérios de usabilidade, disponibilidade, manutenibilidade e portabilidade do sistema, pois com isso iria se perder o objetivo do trabalho.

Requisito Não Funcional 2:

A confiabilidade do sistema pode ser afetada, devido aos acessos simultâneos à classe intermediária que fará o acesso ao banco de dados. Um controle das requisições através de filas pode ser necessário.

Descrição:

Como uma classe irá centralizar o acesso ao banco de dados, podem ocorrer acessos simultâneos à esta classe, mas esta classe lida com um acesso por vez, portanto pode ser necessária a implementação de uma fila, por exemplo, para poder controlar o acesso a esta classe.

Requisito Não Funcional 3:

A segurança de acesso é fundamental para esta alteração do sistema. O sistema gerenciador de banco de dados deve ser capaz de autenticar a classe que tenta conectar à base de dados.

Descrição:

Como o principal objetivo do projeto é melhorar a segurança deste sistema, se faz necessário um controle de acesso, para que o banco de dados possa autenticar a classe que faz o acesso à ele, para que esta seja de fato a única classe a acessar o banco de dados.

Requisito Não Funcional 4:

Comandos diretos ao sistema gerenciador de banco de dados que podem causar danos à base de dados devem ser bloqueados completamente, obrigando que todos os acessos ao banco de dados sejam realizados através da nova classe intermediária.

Descrição:

Não é possível, seja através de um outro programa, ou através de um formulário, enviar comandos *SQL* diretamente à base de dados, pois isto iria comprometer a segurança da base, então qualquer acesso deve ser feito através da classe de acesso.

Requisito Não Funcional 5:

Devido à existência de uma nova classe intermediária e dos acessos simultâneos à esta classe, o desempenho do sistema será afetado, aumentando o tempo de operações de escrita em banco de dados.

Descrição:

O sistema irá ter o seu desempenho diminuído, em função de haver uma nova classe de controle que centraliza todos os acessos. Esta queda no desempenho é causada tanto pelo fato de haver uma nova classe quanto pela necessidade de se controlar os acessos simultâneos a ela.

Especificação Técnica

Atualmente no sistema temos um acesso ao banco de dados que é feito por algumas classes, e existe a possibilidade de um programa acessar diretamente o banco de dados, como mostrado no diagrama a seguir.

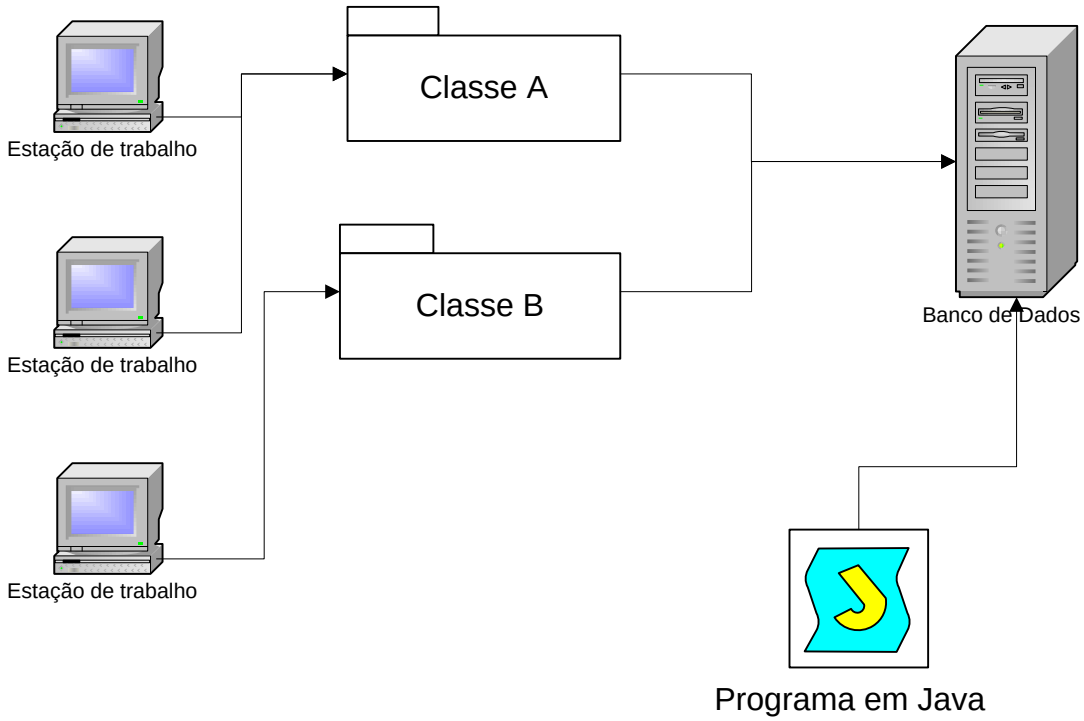


Figura 1: Diagrama atual.

Nesta configuração atual é possível alterar o banco de dados indevidamente, seja através do programa Java, ou através das próprias classes que acessam o banco de dados.

Será desenvolvida uma nova classe, que será a classe de acesso ao banco de dados, e todos os acessos ao banco de dados será feito através desta classe, o que

irá aumentar significantemente a segurança do sistema. O diagrama a seguir mostra como será feita a alteração.

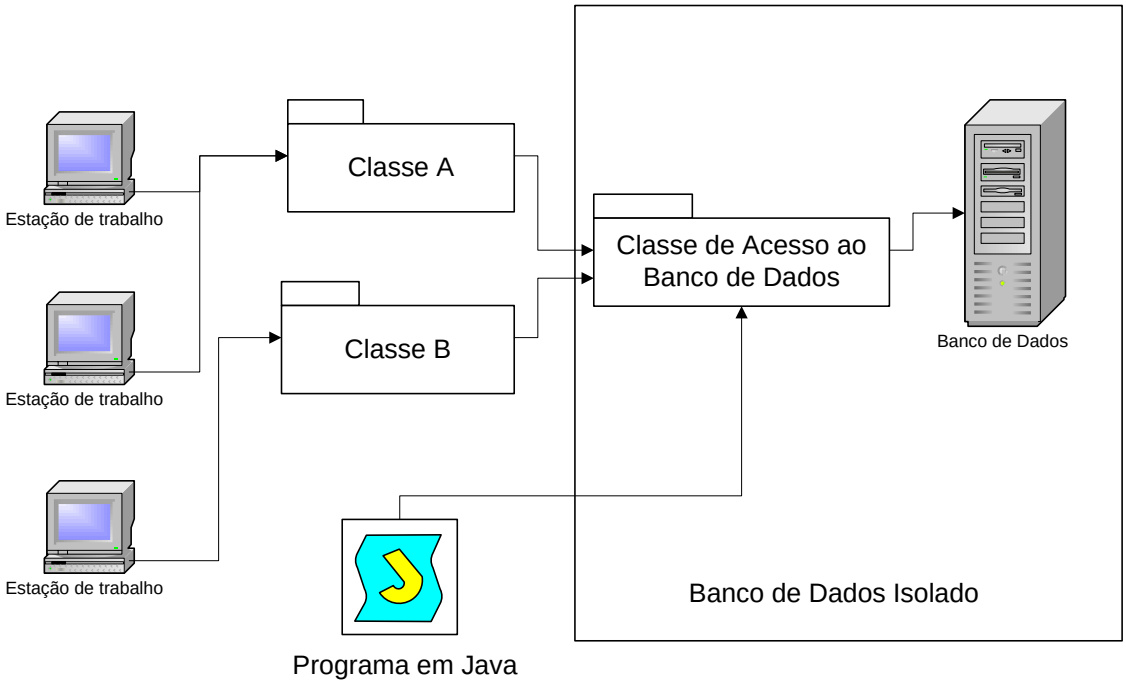


Figura 2: Diagrama Proposto.

Como mostra o diagrama da Figura 2, as alterações serão feitas de maneira que todo e qualquer acesso ao banco de dados seja feito através de uma única classe. Esta classe possui métodos para poder tratar de maneira correta as requisições de acesso ao banco de dados e fazer o acesso sem comprometer a base de dados.

Além da proteção através da classe de acesso ao banco de dados, foi implementada a validação dos campos de entrada do usuário, com o objetivo de impedir que código malicioso seja passado ao banco de dados através de comandos embutidos nas strings de entrada do sistema.

Um exemplo de ataque desta forma é apresentado abaixo, na string que o usuário mal-intencionado digitou no campo “endereço” do formulário de cadastro de animais:

Av. Paulista”; DROP TABLE items;--

As aspas indicam o término do string, ponto-e-vírgura indica o término do comando SQL e os hífens indicam o início de um comentário. Neste caso, existe o risco do usuário poder executar comandos SQL diretamente pelos campos de entrada do sistema.

Para desenvolver a validação, foi realizada uma busca no código fonte do software Pet Store pelo setor que faz as validações já presentes. Estas validações

tratam de campos obrigatórios vazios e campos numéricos preenchidos com caractéres não numéricos (preço, por exemplo). Elas foram incrementadas com testes que verificam a presença de caractéres específicos usados em SQL Injection. Os caractéres testados foram aspas, interrogação, igual e ponto-e-vírgula.

Quando um usuário tenta adicionar um item a venda, se algum dos caractéres citados for encontrado em qualquer campo de entrada, o sistema apresenta uma mensagem de erro indicando uma possível tentativa de SQL Injection.

Especificação de Testes

Nos itens a seguir são descritas as especificações de testes realizadas no projeto.

Inspeção Estática

A inspeção estática foi realizada por observação do código fonte. Foi detectada a forma como o software realiza operações no banco de dados. O arquivo responsável pelo tratamento do upload da figura do animal também é responsável pela escrita em banco de dados. A segurança não é garantida pois a escrita é feita diretamente pelo arquivo que faz o pedido de alteração, sem passar por um módulo que tem permissão para escrita no banco de dados.

Foi inserida uma classe DBAcces, que é a classe responsável pelo acesso ao banco de dados. Por inspeção do código é possível perceber que o upload de imagens e a inserção de animais no banco de dados são feitos através desta classe especifica.

Run-time

Os testes de run-time foram realizados na principal função de alteração do banco de dados do Pet Store, a função de listar um novo animal no catálogo. Essa função é acessada pelo item Seller do menu superior da tela do Pet Store.

Foi observada uma dificuldade para realizar operações de escrita no banco de dados. O software do Pet Store da Sun apresenta um bug no código responsável pelo tratamento da figura do animal. Como as validações da figura falham, a escrita no banco de dados não é realizada, sendo apresentada para o usuário uma mensagem de erro. A mensagem apresentada não reflete o problema encontrado

pelo software, pois considera que o arquivo enviado pelo usuário está danificado ou contém a extensão errada, o que não se aplica a nenhum dos testes de run-time realizados.

Além disso, a figura era colocada no banco de dados do petstore com um nome errado, que não correspondia ao caminho correto da figura. Este bug foi consertado também. Após resolver este problema descobrimos que o petstore não fazia o upload da imagem do seu diretório atual para o diretório de imagens do petstore, este problema foi corrigido com uma rotina de copia de arquivo, esta rotina copia o arquivo de seu local atual para o diretório images do petstore.

Após corrigirmos todos estes problemas, foi possível efetuar o cadastro de novos animais e fazer o upload da imagem correspondente ao animal. Porém há uma limitação nesta operação, a imagem que foi selecionada só fica disponível e aparece na pagina após o petstore ser reiniciado.

Foi testado o cadastro de animais utilizando a classe que foi inserida para realizar o acesso ao banco de dados e verificou-se que o cadastro do animal foi feito corretamente, utilizando a classe de acesso.

Melhorias Futuras

Uma melhoria que pode ser implantada no sistema é a utilização de algum tipo de autenticação entra a classe de acesso ao banco de dados e as outras classe que a acessam, isto deve ser feito para evitar que classes externas ao programa tenham acesso direto ao banco de dados através de uma instância da classe de acesso. A conexão ao banco de dados deve ser feita através da classe de acesso, mas a única segurança neste caso é o usuário e senha, então seria necessária alguma outra medida de segurança para poder barrar programas que acessem diretamente o banco de dados.

Outra melhoria a ser realizada é fazer com que após o upload da imagem ela fique disponível ao usuário imediatamente, não apenas ao reiniciar o petstore. A figura não fica disponível imediatamente pois o petstore carrega seu diretório de imagens antes de ser executado, então seria necessário recarregar este diretório de imagens em tempo de execução.

Referências

- Stop SQL Injection Attacks Before They Stop You – Paul Litwin - http://msdn.microsoft.com/msdnmag/issues/04/09/SQLInjection/

Anexos

POCs

Foi feito um programa em Java que é capaz de acessar o banco de dados do petstore diretamente e alterar qualquer dado. Isto mostrou uma fraqueza do banco de dados, em termos de segurança.w

Para prova de conceito, foi alterado o código da função de listar novo animal ao catálogo. Esta é a principal função do Pet Store para escrita no banco de dados. Para garantir seu funcionamento, foi realizado um rastreamento do local do erro que não permitia a escrita no banco de dados.

Inicialmente as funções de validação da figura foram desabilitadas, para permitir que o software fosse capaz de escrever no banco de dados independentemente do sucesso ou fracasso da operaçao de upload da figura. Após desabilitar todas as validações da figura recebida por upload, foi possível escrever no banco de dados os dados cadastrais do animal a ser inserido no catálogo, obviamente sem a figura enviada.

A seguir foi avaliada a possibilidade de realizar a escrita e incluir uma figura, mas sem utilizar o processo de upload, o software apontaria diretamente para a figura a ser incluída. O teste foi bem sucedido, isentando de problemas a escrita da figura no banco de dados e indicando que o problema inicial encontra-se na função de upload da figura.

Para solucionar o problema foi alterada a função de upload da figura, garantindo que a figura enviada fosse corretamente recebida pelo servidor e posteriormente movida para o diretório onde se encontra a galeria de figuras do catálogo. Com esta alteração no código, e a reabilitação das funções de validação da figura recebida, foi possível realizar a escrita completamente bem sucedida no banco de dados, com os dados cadastrais do animal a ser vendido e a figura enviada pelo vendedor.

Resultados

As alterações no código fonte, especificamente na função de upload de figuras, geraram sucesso na solução do problema de inclusão de animais no catálogo do Pet Store.

A operaçao de cadastro de animais a venda era inicialmente impossível devido a um bug do Pet Store da Sun, pois este bug bloqueava o upload da figura do animal a ser vendido após a execução das funções de validação da figura recebida.

Reparado o bug, o sistema passou a aceitar a figura enviada, armazená-la no diretório onde se encontram as figuras de todos os animais do catálogo e incluir os dados cadastrais do animal e o endereço de sua figura no banco de dados.

Ao acessar o catálogo de animais a venda é possível observar que os novos animais foram incluídos com sucesso no banco de dados e que suas figuras também foram aceitas e estão em exibição no catálogo.

Exemplo 3 - PetStore – Segurança e Desempenho

Contexto Considerado

No projeto que estamos trabalhando na disciplina, a revisão do exemplo Java Pet Store da Sun microsystem, estamos trabalhando com o equilíbrio entre segurança e desempenho, especificamente na área de pagamento on-line via cartão de crédito e similares.

Através deste projeto esperamos aprimorar nosso conhecimento de planejamento e implementação de técnicas de engenharia de software, além de nossas habilidades de análise e observação já que estamos trabalhando em cima de um projeto pronto. Desta maneira, entender e propor alterações sem comprometer a estrutura do projeto com outros módulos é muito importante e nos ajuda no nosso preparo para atuação em situações semelhantes no mercado de trabalho.

Na presente situação devemos propor mudanças no serviço de pagamento para garantir a segurança dos clientes e da própria empresa petStore, sem, contudo, degradar a velocidade desta operação, se tornando incomoda para o cliente.

Objetivo

Na versão original, o acesso ao site de pagamentos PayPal é realizado através de um simples comando POST em HTML, enviando dados importantes (como a conta destino, quantidade e valor da compra) utilizando-se simplesmente do protocolo http, antes mesmo de se inicializar qualquer conexão segura. Assim, tais dados estão sujeitos a ataques de terceiros, como a alteração destes valores ou utilização das informações para fins inescrupulosos. A seguir, está apresentado o trecho de código retirado do projeto petStore, evidenciando nossa afirmação anterior:

```
<form id="buyNow1_form" name="buyNow1_form" method="post"
action="https://www.sandbox.paypal.com/cgi-bin/webscr" target="paypal">
<input id="buyNow1_image" name="buyNow1_image" type="image" target="paypal" alt="BuyNow"
src="http://www.paypal.com/en_US/i/btn/x-click-but01.gif" />
<input type="hidden" id="buyNow1_cmd" name="cmd" value="_xclick" />
<input type="hidden" id="buyNow1_business" name="business" value="donate@animalfoundation.com"
/>
<input type="hidden" id="buyNow1_amount" name="amount" value="100.00" />
<input type="hidden" id="buyNow1_item_name" name="item_name" value="Buy Item One" />
<input type="hidden" id="buyNow1_quantity" name="quantity" value="1" />
<input type="hidden" id="buyNow1_shipping" name="shipping" value="25" />
<input type="hidden" id="buyNow1_return" name="return" value="GET" />
<input type="hidden" id="buyNow1_tax" name="tax" value="10" />
<input type="hidden" id="buyNow1_undefined_quantity" name="undefined_quantity" />
</form>
```

Como é fácil perceber, tais informações podem ser alteradas sem o conhecimento do usuário. Por exemplo, uma pessoa mal intencionada pode interceptar a conexão e mudar a conta de destino para uma outra da qual seja titular. Uma forma de se fazer isso seria o usuário mau intencionado divulgar rotas de entrega falsos e assim receberia os dados que repassaria ao destino final com as informações alteradas, provavelmente a seu favor.

Apesar da falta de segurança acima exposta, o desempenho apresentado pelo projeto atual é alto, uma vez que os dados não estão criptografados, só dependendo mesmo da velocidade de acesso por parte do cliente e servidor do serviço de pagamento, além de possíveis tráfegos na internet. Espera-se que o desempenho caia uma vez que as medidas de segurança sejam tomadas, como utilização de protocolo https, criptografia mais avançada, ou alguma outra técnica mais sofisticada que venhamos a implementar, já que se espera aumentar a carga no cliente e servidor com processamento extra proveniente das fontes que acabamos de citar, além de uma troca maior de dados entre ambos.

Como prova do discutido acima, nas primeiras semanas de projeto pudermos enviar dados a página do petStore de pagamento usando a ferramenta jMeater, comprovando que a estrutura de transferências de dados era completamente obvia para alguem interessado em descobrila e explora-la.

Requisitos Funcionas e Não funcionais

Listamos abaixo uma coleção dos requisitos identificados pelo grupo ao longo de todo o desenvolvimento do projeto, assim sempre que identificamos novos requesitos e necessidades do sistema iamos acrescentando estes a lista, embora devemos deixar claro que esta parou de crescer logo após algumas semanas de início do projeto, tendo em vista que sua baixa complexidade nos permitiu determina-la por completo.

Requisitos Funcionais

- O sistema deverá apresentar os dados da compra efetuada, com seus detalhes: animal e valor pago;
- O sistema deverá solicitar dados da conta bancária do cliente (número da conta e senha);
- O sistema deverá possibilitar mais de uma entidade financeira, oferecendo opções de pagamento on-line para o cliente.
- O sistema ainda deverá exibir mensagens com informações sobre a transação efetuada, seja do sucesso da operação ou da falha, tentando especificala sem, no entanto, disponibilizar falhas na segurança (mensagens sobre fundos insuficientes e de tentativa de login falho, nunca deixando claro a causa desta – se foi a senha ou número de constas errada).

Requisitos Não Funcionais

- Toda comunicação de dados entre os servidores do Java Pet Store e do simulador da entidade financeira deverá ser em um ambiente seguro, tentando minimizar ao máximo o ataques de terceiros com criptografia e uso de protocolo seguro;
- O tempo de resposta do sistema na área de pagamentos deverá ser não longo o suficiente a ponto de incomodar em demazia o cliente, tentando levar em consideração um possível congestionamento da rede.

Especificação Técnica

Para contornar o problema exposto, foi desenvolvido um servidor que simule a operação de uma instituição financeira. A comunicação entre os servidores deveria se realizada inteiramente por “secure webservices”. Desse modo, os dados transferidos seriam criptografados e enviados atraves de um protocolo e seções seguras, prevenindo ataques comuns. No entanto a obtenção e configuração de certificados se provou um desafio em face da escarsas informações encontrada on-line e o pouco tempo disponível para tal.

Então acabamos optando por uma solução híbrida, onde é feito uma conexão segura via https a página crítica de envio de dados so banco, página esta ainda localizada no petStore, onde contem o código usado na criptografia dos dados enviados ao sevidor do banco, usando o protocolo não seguro http. Na figura abaixo se encontra um esboço de como o projeto esta implementado na versão final, com uma representação simplificada de como suas diferentes funcionalidades estão mapeadas nos diferentes módulos.

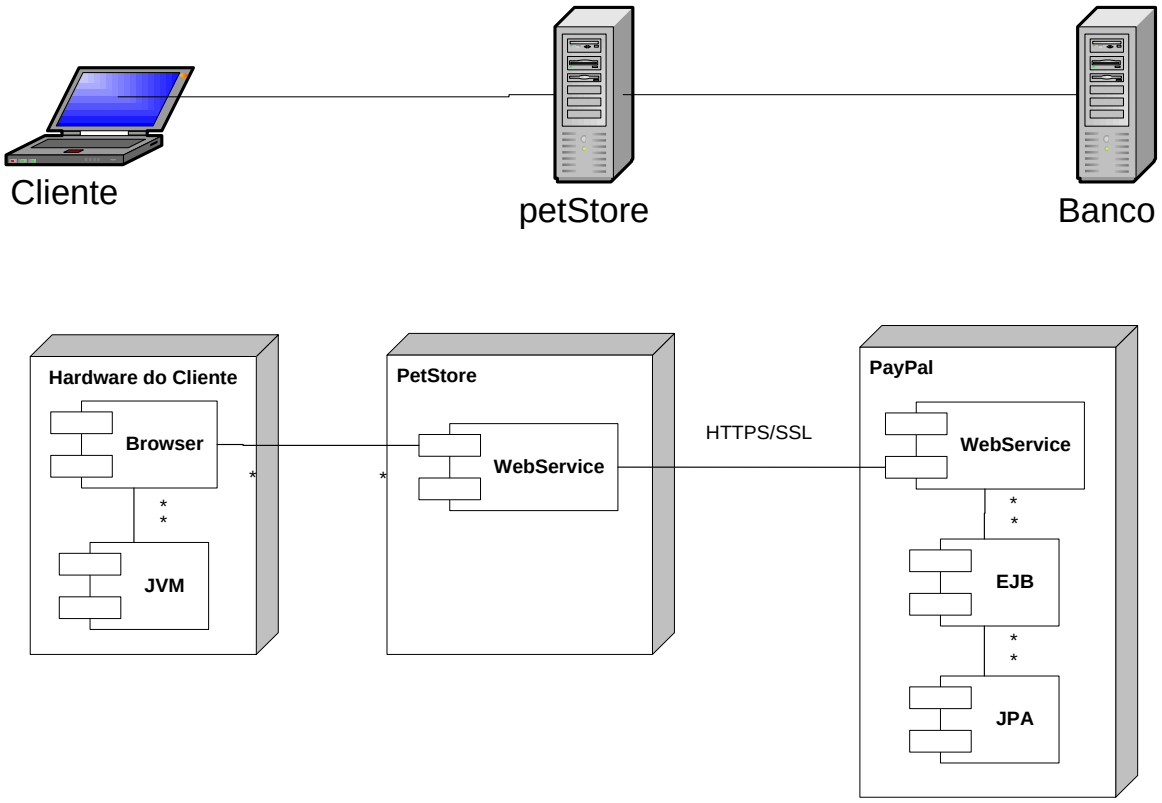


Figura – Diagrama de componentes (somente a parte de acesso a pagamento seguro).

Do lado do servidor teremos:

- O módulo webservice do lado do banco é o responsável por intermediar o acesso entre as requisições de cliente, que ocorrem através do petStore, e os demais módulos prestadores de serviço, nele estão as iterfaces para receber os dados criptografados, emviar as respostas e acessar seus módulos internos, como a base de dados dos clientes, para realizar os pagamentos e conferir senhas e números de contas;

- No módulo EJB estarão todas as regras de negócio definidas para o interfaciamento com o petStore, tais como os dados trocados, tanto dos animais e contas quanto as mesagens de status das transações que são devolvidas para o site da loja e então para o cliente.

- Finalmente o módulo JPA será responsável pelo acesso e persistência dos dados relevantes as transações comerciais, tais como confirmação de pagamento, verificação de senha e login, entre outras que possam vir a ser úteis ao longo do desenvolvimento do projeto.

Do lado do cliente temos:

- O webservice da petStore que acessará o serviço através de uma conexão com protocolo http mas que possui os componentes para uma comunicação com as técnicas de criptografia adequadas.

POCs (Provas de conceito)

Ao longo do projetos foram realizadas provas de conceito, abaixo são listadas algumas delas, aquelas que o grupo achou mais importante ao longo do projeto, mesmo tendo aquelas que foram abansonadas pela equipe e não foram usadas.

Uso de criptografia simétrica

Para testarmos a criptografia que iríamos usar eventualmete no projeto criamos uma pequena aplicação de teste utilizando a classe Cipher que vem inclusa com o java, e a utilização do algorítimo AES. O código abaixo serviu de base para o teste seguinte. Aqui temos o princípio do uso da classe Cipher.

```
package crypto;
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import java.io.*;
/**This program generates a AES key, retrieves its raw bytes, and
 * then reinstantiates a AES key from the key bytes.
 * The reinstantiated key is used to initialize a AES cipher for
 * encryption and decryption.
 */
public class AES {
    /**Turns array of bytes into string
     * @param buf      Array of bytes to convert to hex string
     * @return Generated hex string
     */
    public static String asHex(byte buf[]) {
        StringBuffer strbuf = new StringBuffer(buf.length * 2);
        int i;
        for (i = 0; i < buf.length; i++) {
            if (((int) buf[i] & 0xff) < 0x10)
                strbuf.append("0");
            strbuf.append(Long.toString((int) buf[i] & 0xff, 16));
        }
        return strbuf.toString();
    }
    public static void main(String[] args) throws Exception {
        int origem, destino, senha;
        float valor;
        origem = 111111;
        destino = 222222;
        senha = 333333;
        valor = 500.0f;
        String message = new String(origem + "|" + destino + "|" + senha + "|" + valor);
        // Get the KeyGenerator
        KeyGenerator kgen = KeyGenerator.getInstance("AES");
        kgen.init(128); // 192 and 256 bits may not be available
        // Generate the secret key specs.
        SecretKey skey = kgen.generateKey();
        byte[] raw = skey.getEncoded();
        SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
```

```
        // Instantiate the cipher
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
        System.out.println("original string: " + message);
        byte[] encrypted =
                cipher.doFinal(message.getBytes());
        System.out.println("encrypted string: " + asHex(encrypted));
    }
}
```

A trecho abaixo representa um código de teste usado a cryptografia e decryptografia de mensagens:

```
package crypto;
import javax.crypto.*;
import javax.crypto.spec.*;
import java.util.*;
public class Main {
    /** Creates a new instance of Main */
    public Main() {
        try{
            int origem = 111111;
            int destino = 0;
            int senha = 111111;
            float valor = 0.0f;
            javax.crypto.KeyGenerator keyGen = javax.crypto.KeyGenerator.getInstance("AES");
            keyGen.init(192);
            javax.crypto.spec.SecretKeySpec skeySpec = new
            javax.crypto.spec.SecretKeySpec(keyGen.generateKey().getEncoded(), "AES");
            javax.crypto.Cipher cipher = javax.crypto.Cipher.getInstance("AES");
            cipher.init(javax.crypto.Cipher.ENCRYPT_MODE, skeySpec);
            String codigo = cipher.doFinal(new String(origem + "|" + destino + "|" + senha +
"|" +
            valor).getBytes()).toString();
            mandar(codigo);
            System.out.println("Parametros codificados: " + codigo);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public void mandar(String codigo){
        try{
            KeyGenerator keyGen = KeyGenerator.getInstance("AES");
            keyGen.init(256);
            SecretKeySpec skeySpec = new SecretKeySpec(keyGen.generateKey().getEncoded(),
"AES");
            Cipher cipher = Cipher.getInstance("AES");
            cipher.init(Cipher.DECRYPT_MODE, skeySpec);
            StringTokenizer token = new
```

```
            StringTokenizer(cipher.doFinal(codigo.getBytes()).toString(), "|");
            int origem = Integer.parseInt(token.nextToken());
            int destino = Integer.parseInt(token.nextToken());
            int senha = Integer.parseInt(token.nextToken());
            float valor = Float.parseFloat(token.nextToken());
            System.out.println("Origem, destino, senha, valor: " + "-" + origem + "-" +
destino +
            "-" + senha + "-" + valor);
        }catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        new Main();
    }
}
```

Uso de criptografia assimétrica

Esta prova de conceito foi usada para testar a criptografia asimétrica que seria usada se tivésemos optado por gerar chaves a cada requisissão de pagamento, aumentando assim a segurança com uma chave nova a cada transação, mas a dificuldade de implementação de tal artifício em servidores stateless nos levou a abandonar esta idéia.

```
package javaapplicationcypto;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.Security;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

public class Main {
   private static KeyPairGenerator keysGenerator;
   private static KeyPair pairOfKeys;
   private static Cipher cipher;
   /** @param args the command line arguments
    */
   public static void main(String[] args) {
      Security.addProvider(new BouncyCastleProvider());
      try {
```

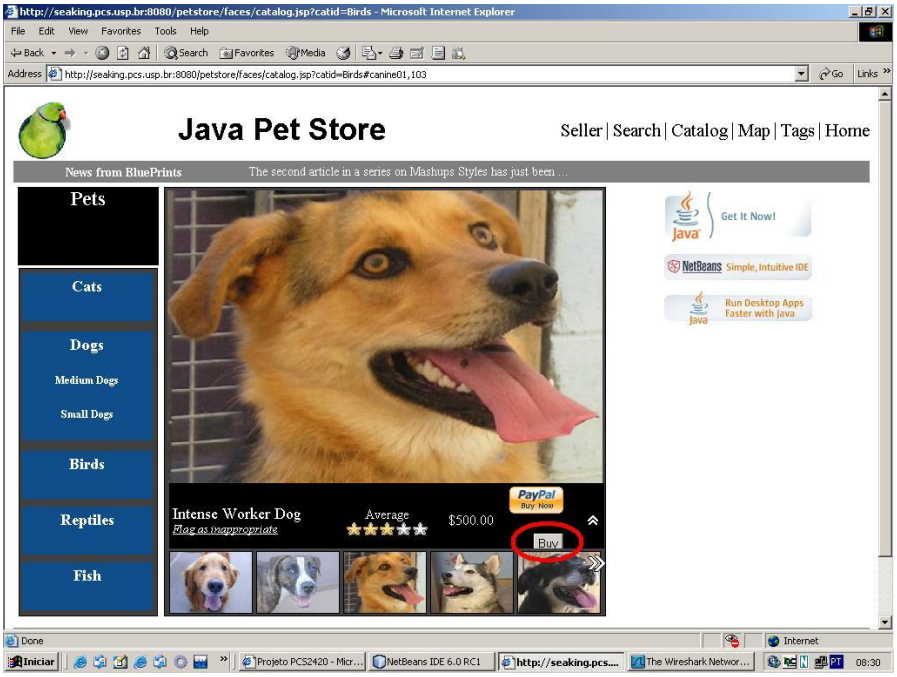
```
            keysGenerator = KeyPairGenerator.getInstance("RSA");
        } catch (NoSuchAlgorithmException ex) {
            ex.printStackTrace();
        }
        keysGenerator.initialize(1024);
        pairOfKeys = keysGenerator.generateKeyPair();
        try {
            cipher = Cipher.getInstance("RSA/NONE/PKCS1PADDING");
            cipher.init(Cipher.ENCRYPT_MODE,pairOfKeys.getPrivate());
            String original = "Teste";
            byte[] teste = original.getBytes();
            System.out.println(original);
            byte[] encrypted = cipher.doFinal(original.getBytes());
            cipher.init(Cipher.DECRYPT_MODE,pairOfKeys.getPublic());
            byte[] decrypted = cipher.doFinal(encrypted);
            System.out.println(new String(decrypted, "UTF-8"));
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

No final optamos por um modelo em que a chave simétrica é previamente conhecida pelas partes involvidas, no caso o perStore e o Banco.

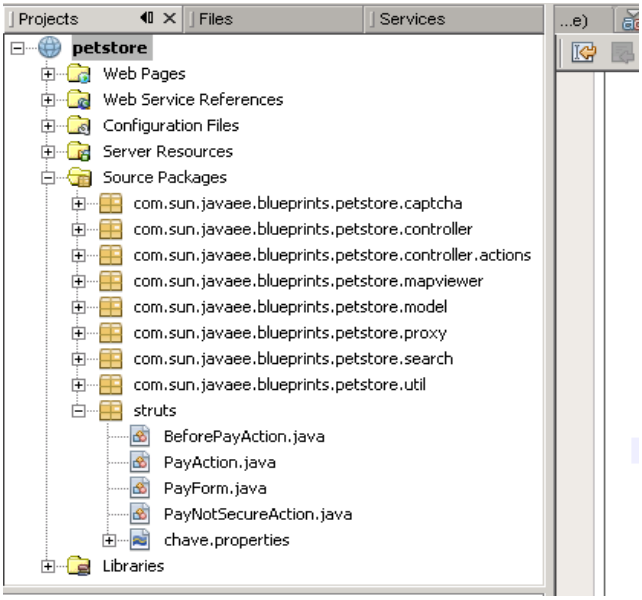
Especificação dos Testes

Inspeção estática

Durante o desenvolvimento do código para a criação do serviço de pagamento e do cliente para acesso do mesmo tentamos identificar todas as possíveis brechas na arquitetura para alterá-las ou eliminá-las de forma que o produto final tivesse um maior grau de segurança e eficiência na troca de dados entre o cliente do site petStore e o webservice, sem, no entanto, alterar a página e serviço básico.



Botão incluido para pagamento seguro.

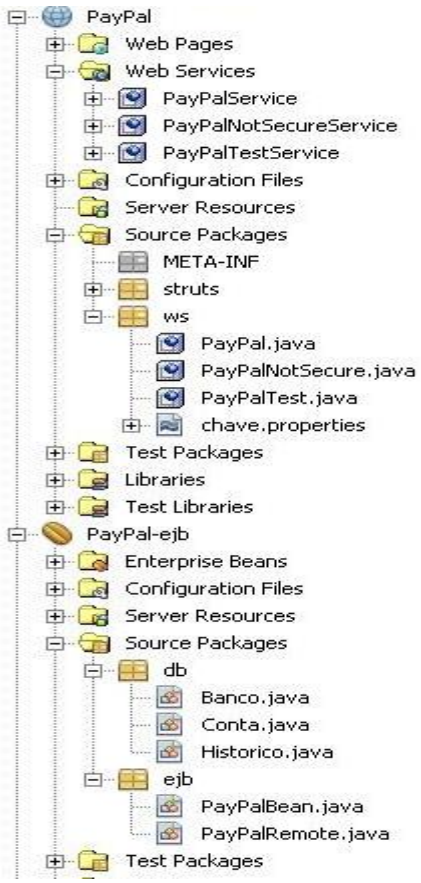


Arquivos adicionados ao projeto original petStore.

Abaixo encontra-se o trecho do código com as alterações na página de escolha de animal no petStore, no arquivo catalog.jsp:

```
<td id="infopaneRating" class="infopaneRating">
                                <f:view>
                                <ui:rating id="rating" maxGrade="5" includeNotInterested="false" includeClear="false"
                                        hoverTexts="#{RatingBean.ratingText}" notInterestedHoverText="Not Interested"
clearHoverText="Clear Rating"
                                        grade="#{RatingBean.grade}"/>
                               <f:verbatim></td><td id="infopanePrice" class="infopanePrice"></td><td id="infopanePayPal"
class="infopanePayPal"></f:verbatim>
                                <ui:buyNow business="donate@animalfoundation.com" id="buyNow1" itemName="Buy Item One"
                                        amount="100.00" quantity="1" type="BuyNow" postData="#{PayPalBean.postData}"
target="paypal"/>
                                </f:view>
                                <input type="button" value="Buy" onclick="buy()">
                               </td>
```

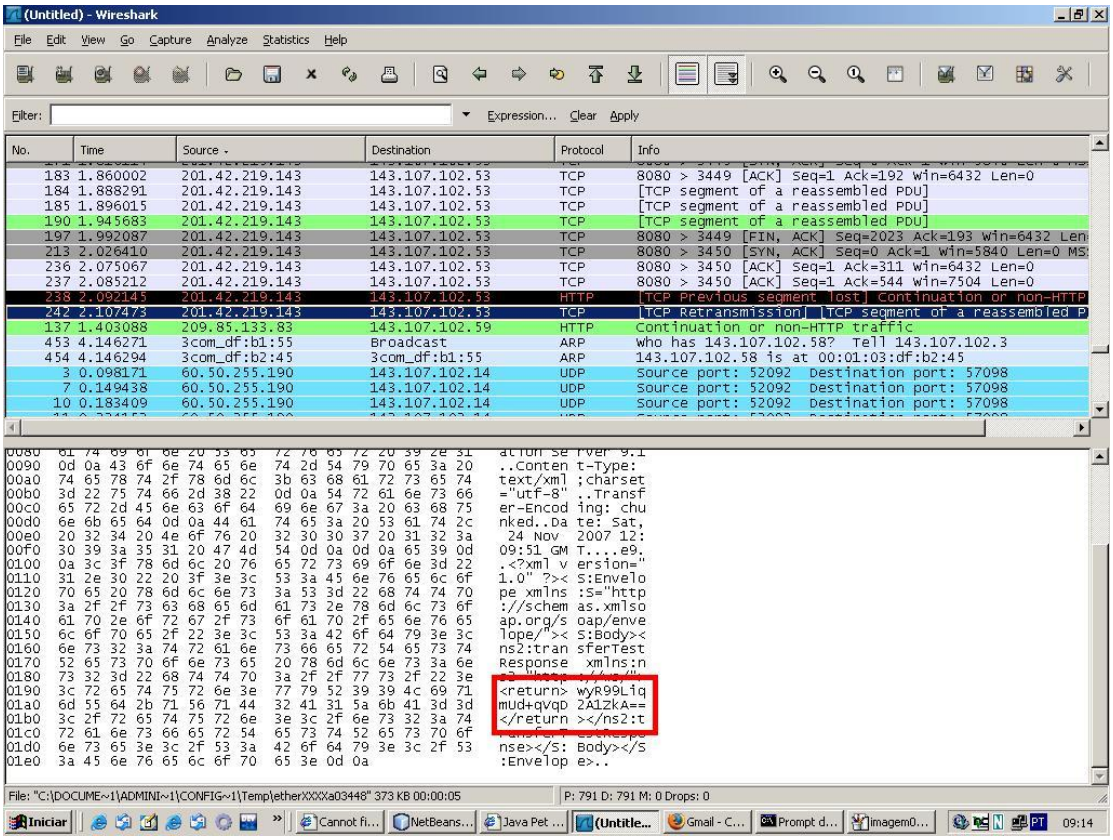
A seguir temos a “árvore” estrutural do projeto do servidor de pagamentos seguro:



Arquivos do sevidor de pagamento seguro.

Inspeção dinâmica

Durante os testes de execução do serviço levamos em consideração o tempo de resposta às requisições ao servidor, sendo considerado somente a paciência do usuário como fator de decisão com relação a satisfação do desempenho, o que foi satisfatório pois o incremeto médio nas transações foi de apenas alguns segundos. Com relação a segurança testamos a operação do pagamento juntamente com um sniffer para verificarmos a criptografia e a utilização do protocolo https.



Comprovação da criptografia dos dados com o sniffer WireShark.