ENSINO DE TÉCNICAS ARQUITETURAIS NUM CONTEXTO DE MANUTENÇÃO

Júlio H. N. O. Guimarães [1]; Reginaldo Arakaki [2];Renato M. Andrade [3]
USP - Escola Politécnica - Departamento de Engenharia de Computação e Sistemas Digitais
Av. Prof. Luciano Gualberto - travessa 3 - 158
CEP: 05508-900, São Paulo, SP
[1]julio.guimaraes@poli.usp.br
[2]reginaldo.arakaki@poli.usp.br
[3]renato.manzan@poli.usp.br

Resumo: *Diversos negócios hoje são suportados por sistemas de software. Com o dinamismo dos negócios muitas vezes ocorre que o conjunto de requisitos que levou à construção de uma determinada arquitetura para tais sistemas não é mais o mesmo. Em diversas situações é preciso conviver com sistemas legados, portanto é preciso alterá-los para as novas necessidades. Arquitetura de Software é tida como um dos principais meios para atingir alcançar as metas de negócio e qualidade. Manutenção de sistemas usando técnicas de evolução arquiteturais tem se mostrado um eficaz caminho para alterar um sistema à nova situação. Portanto, é importante que estudantes de Engenharia de Computação detenham os conhecimentos de Manutenção e Arquitetura de Software bem como seu uso em conjunto. Para isso montou-se um laboratório onde alunos de graduação fazem uso prático de técnicas arquiteturais em um contexto de manutenção. Propôs-se um modelo geral para manutenção centrada em arquitetura cujas fases podem ser adaptadas para expor os alunos ao uso das técnicas selecionadas pelos dirigentes do curso. Além deste roteiro, apresenta-se também uma aplicação feita no laboratório de Engenharia de Software de alunos de graduação do penúltimo ano onde foram usadas técnicas de avaliação de arquitetura, métricas de software e provas de conceito.*

Palavras-chave: *Manutenção de Arquitetura de Software, Técnicas arquiteturais, Laboratório de Engenharia de Software, Ensino de Arquitetura de Software, Evolução de Arquitetura de Software*

1 INTRODUÇÃO

Pesquisas têm indicado a importância do o ciclo de manutenção de sistemas e tem sido um desafio fazer manutenções que consigam atingir as expectativas de qualidade.

Arquitetura de software tem sido tema de pesquisa muito forte atualmente pois acredita-se que é um bom caminho para atingir metas de negócio e qualidade nos sistemas de software. Por isso, uma série de métodos têm sido propostos para auxiliar o arquiteto no desempenho de suas atividades.

Dada a importância destes dois assuntos acima, foi proposto um curso na forma de laboratório para que os alunos pudessem experimentar na prática os conceitos de arquitetura de software e vivenciar, mesmo que de forma controlada, um contexto de manutenção. É importante ressaltar que muitos alunos de graduação só experimentam o desenvolvimento “do zero” em disciplinas de graduação e isso os deixa despreparados

quando em diversos domínios de aplicação na indústria a principal atividade gira em torno do ciclo de manutenção.

No curso proposto utilizou-se um roteiro de manutenção onde as fases pudessem ser especializadas de acordo com as restrições de tempo, experiência dos alunos e com os assuntos e conceitos que os professores desejassem aprofundar.

A seção 2 apresenta o relacionamento existente entre a arquitetura de software e a qualidade de software, bem como a necessidade de qualidade no ciclo de manutenção. Quando aplicações deste curso proposto foram feitas, decidiu-se aprofundar em algumas técnicas arquiteturais que também são apresentadas nesta seção.

O roteiro de manutenção de software através de técnicas arquiteturais é exposto na seção 3. As fases são descritas e são dadas algumas sugestões quanto a como desempenhá-las.

Na seção 4 é mostrado um exemplo de aplicação do roteiro no laboratório e são mostradas as restrições que foram feitas nesta aplicação, exemplificando uma possível especialização do curso para destacar determinados conceitos.

2 TÉCNICAS ARQUITETURAIS E MANUTENÇÃO

A manutenção de um sistema nem sempre está ligada a requisitos funcionais (novas funcionalidades ou alteração de alguma funcionalidade). Quando está ligada a requisitos não funcionais (ou de qualidade), as modificações geralmente estão ligadas a alterações arquiteturais. Daí, os uso de algumas técnicas arquiteturais são importantes para obter-se sucesso na manutenção, tornando a arquitetura adequada.

2.1 Arquitetura de software adequada e requisitos de qualidade

Segundo BASS *et al*. (2003), Arquitetura de Software pode ser definida como a estrutura ou estruturas do sistema, que compreende os elementos de software, as propriedades visíveis externamente desses elementos, e o relacionamento entre eles. Adicionalmente IEEE-1471 (2000) constata que ela é a organização fundamental de um sistema, incorporada em seus componentes, seus relacionamentos entre si e o ambiente, e os princípios que governam seu projeto e evolução.

Ainda de acordo com IEEE-1471 (2000) uma arquitetura é projetada de forma que enderece os desejos e preocupações dos *stakeholders*. BASS *et al*. (2003) acrescenta que esses desejos e preocupações podem ser entendidos como um conjunto de requisitos que podem ser requisitos funcionais — funcionalidades que o sistema deve fornecer — ou requisitos de qualidade — o nível de qualidade que o sistema deve possuir em seus diversos aspectos, podendo basear-se num modelo de referência de qualidade.

Para que uma arquitetura seja considerada adequada, CLEMENTS *et al.* (2002) afirma que o sistema que resulta dela deve atingir seus objetivos de qualidade e deve poder ser construído com os recursos que estão disponíveis (deve ser "construível").

A Figura 1 foi adaptada de IEEE-1471 (2000) e ilustra como se relacionam estes conceitos.

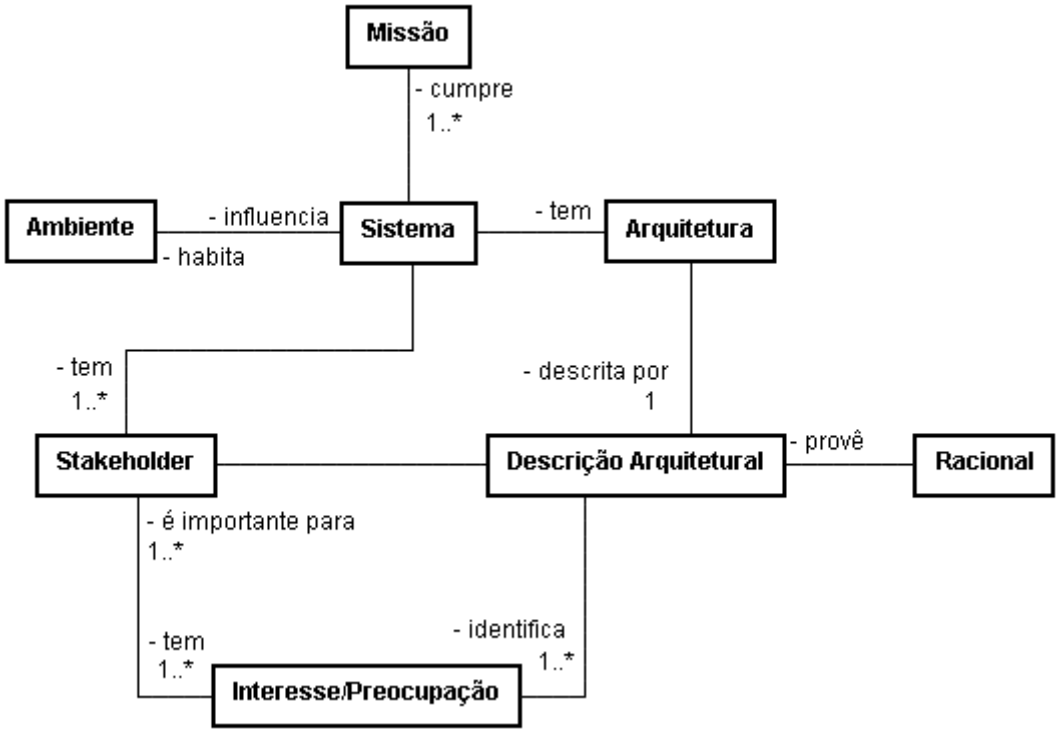

Figura 1 - Diagrama conceitual relacionando a arquitetura com os interesses e preocupações (requisitos funcionais e de qualidade) dos *stakeholders* (notação UML).

2.2 Manutenção

Diversos negócios hoje são suportados por sistemas de software. Com o dinamismo dos negócios muitas vezes ocorre que o conjunto de requisitos que levou à construção de uma determinada arquitetura para tais sistemas não é mais o mesmo. Em diversas situações é preciso conviver com sistemas legados, portanto é preciso alterá-los para as novas necessidades.

Como o processo de manutenção de software pode consumir mais de 60% do orçamento destinado ao desenvolvimento de um sistema de acordo com Pressman (2001), é muito importante que ele seja bem sucedido, inclusive para evitar perdas na forma de retrabalhos e custos.

Este trabalho sugere um roteiro de forma a alinhar a manutenção aos requisitos de qualidade desejada pelos *stakeholders*, tornando a manutenção bem-sucedida.

2.3 Técnicas Arquiteturais

Existem diversas técnicas que utilizam ao conceito de Arquitetura de Software em diversas fases do ciclo de vida do software. Um conjunto dessas técnicas pode ser visto em SEI (2008).

O processo descrito na seção 3 utiliza algumas técnicas arquiteturais e são descritas a seguir:

Avaliação de Arquitetura de Software

Existem diversos métodos para fazer avaliações de arquitetura. Segundo CLEMENTS *et al.* (2002), uma avaliação arquitetural tem como saída um relatório cujo conteúdo deve responder dois tipos de perguntas:

- A arquitetura é adequada para o sistema ao qual ela foi projetada?
- Qual dentre duas ou mais arquiteturas é mais adequada para um dado sistema?

Como visto na subseção 2.1, para que uma arquitetura seja considerada adequada, o sistema que resulta dela deve atingir seus objetivos de qualidade e deve poder ser construído com os recursos que estão disponíveis.

Uma arquitetura pode ser avaliada baseando-se em diversos atributos de qualidade que podem inclusive ser pertinentes a apenas um determinado contexto.

Um dos métodos de avaliação de arquitetura que tem tido muito destaque é o *Architecture Tradeoff Analysis Method* (ATAM). Em CLEMENTS *et al.* (2002) e SEI (2008), listam-se como os principais resultados que são obtidos:

- Uma lista priorizada de enunciados de requisitos de qualidade;
- Mapeamento das decisões arquiteturais em atributos de qualidade endereçados;
- Riscos e não-riscos;
- Um catálogo das decisões arquiteturais usadas;
- Pontos de sensibilidade e pontos de tradeoff.

Resultados como estes podem ser muito úteis no processo de manutenção e isto é mostrado em uma das fases do roteiro de manutenção.

Métricas de Software

Em BASS *et al.* (2003), demonstram que atingir os requisitos de qualidade de um sistema está intimamente ligado à arquitetura deste sistema.

A combinação desejada dos atributos de qualidade deve ser claramente definida, senão a avaliação da qualidade é deixada para a intuição.

Em CLEMENTS *et al.* (2002), percebe-se que embora os atributos de qualidade formem a base para uma avaliação arquitetural, simplesmente listá-los como requisitos (p. e. “o sistema deve ser robusto” ou “o sistema deve ser seguro”) não é uma base suficiente para julgar se uma arquitetura é adequada. Sem uma elaboração melhor esse tipo de requisito fica sujeito 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), AMMAR *et al.* (2005) constatam que estimativas da qualidade de software baseando-se em métricas dinâmicas são mais precisas e realistas.

Este trabalho enfatiza o uso de métricas– e, por sua vez, o uso de medições – estáticas e dinâmicas de software.

Provas de conceito (POCs)

Define-se prova de conceito (do inglês *Proof of Concept*) como uma técnica que permite demonstrar que uma determinada idéia é tecnicamente possível, ou seja, ajuda 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.

Uma **POC construtiva** potencialmente reduz o risco de fracasso e cobrem 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. É tão importante quanto a POC construtiva para aumentar a possibilidade de sucesso do projeto e satisfação dos requisitos funcionais e principalmente não funcionais.

Provas de conceito permitem uma separação clara do trabalho dos desenvolvedores, possibilitando o uso de técnicas de engenharia simultânea (paralelismo), o que diminui o tempo de desenvolvimento de um sistema.

3 MANUTENÇÃO DE SOFTWARE ATRAVÉS DE TÉCNICAS ARQUITETURAIS

O roteiro aqui apresentado é dividido em 5 fases. São elas:

- Fase 1: Apresentação do sistema
- Fase 2: Apresentação dos objetivos de negócio
- Fase 3: Avaliação da adequação da arquitetura de software
- Fase 4: Projeto da manutenção visando adequação da arquitetura
- Fase 5: Implementação da manutenção

As fases do processo são encadeadas segundo o diagrama na Figura 2 a seguir.

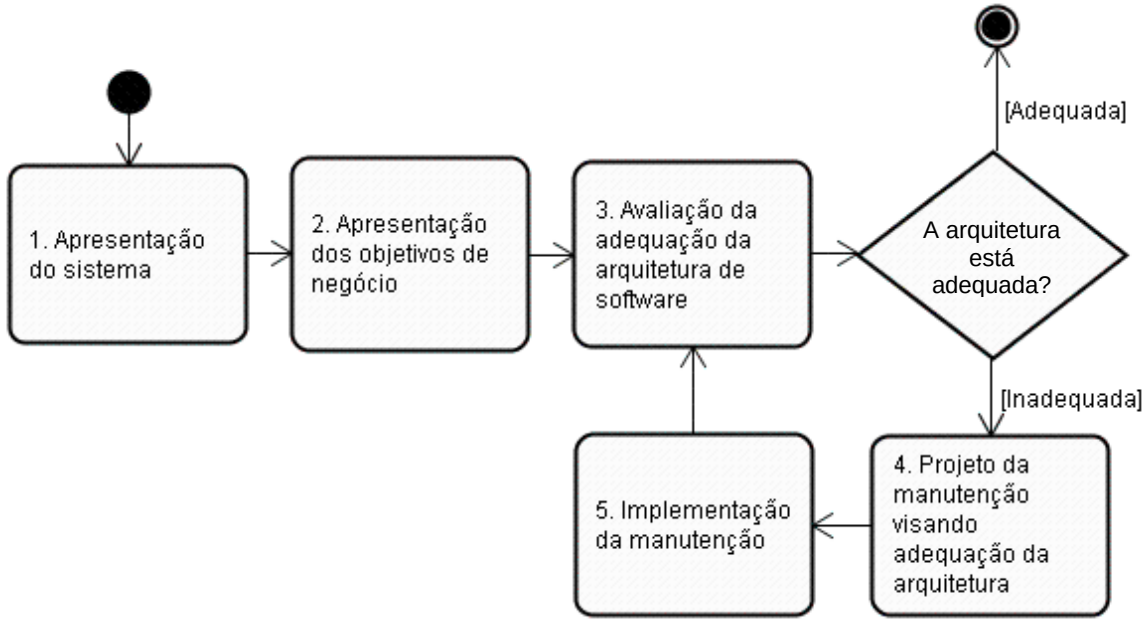

Figura 2-Diagrama do processo do roteiro de manutenção (notação UML).

Como se pode notar, o processo é incremental e iterativo, procurando atingir os objetivos de qualidade até a adequação da arquitetura do sistema.

As atividades, os atores e ferramentas apropriadas de cada fase são descritos a seguir.

3.1 Fase 1: Apresentação do sistema

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.

Nesta fase, a documentação do sistema é apresentada à Equipe de Manutenção. A utilização do sistema por essa equipe também contribui para a absorção dos principais conceitos do sistema. A duração desta fase é variável de acordo com o tamanho do sistema e das necessidades do usuário. O responsável pela execução desta fase pode ser um usuário experiente do sistema. Ao terminar esta fase, segue-se à Fase 2.

3.2 Fase 2: Apresentação dos objetivos de negócio

Conforme visto na subseção 2.1, uma arquitetura adequada é aquela que atende os seus objetivos (as preocupações e desejos dos *stakeholders*). No contexto de manutenção, um dos fatores que pode fazer com que a arquitetura existente torne-se inadequada é a mudança dos objetivos de negócio. Portanto, 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 quais os novos objetivos de negócio.

A forma de apresentação pode ser uma lista de regras e diretrizes de negócio. Muitas vezes as diretrizes de negócio podem ser normatizadas para determinado nicho de aplicações. Ou ainda, alguns ramos de negócio possuem objetivos em comum, e o conhecimento de tais objetivos auxilia a Equipe de Manutenção em sua documentação.

O responsável pela execução desta fase é o Responsável de Negócio, que conhece os objetivos de negócio.

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 mapear onde a arquitetura está atendendo os objetivos.

Técnicas de avaliação de arquitetura como o ATAM podem ser utilizados para o cumprimento desta fase.

Para obter evidências em suas avaliações, os participantes podem usar outras técnicas de apoio. Recomenda-se o uso de avaliações estáticas e dinâmicas, com uso de medições. Outra técnica interessante é o uso de POCs destrutivos.

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).

Por exemplo: de acordo com o tipo de negócio, há a necessidade do sistema fornecer uma resposta dentro de cinco segundos, mesmo com uma carga de trezentos usuários acessando simultaneamente. O sistema no momento consegue atender tal necessidade? Se sim, como o faz? Que evidências se tem disto? Uma avaliação arquitetural vai permitir responder perguntas como estas.

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.

É interessante notar que esta fase pode ser executada após a implementação de modificações (Fase 5), pois o processo é incremental. Neste caso, o objetivo é verificar e evidenciar se os objetivos foram atendidos pelo sistema resultante da manutenção.

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 a uma especificação de projeto de desenvolvimento. Porém o tipo aqui descrito é focado nas mudanças, ou seja, nos mecanismos novos e nos alterados. Esta especificação deve conter um detalhamento da estrutura existente, evidenciando os pontos de modificação e quais as modificações em si.

Uma vez que se conhecem as necessidades, pode-se fazer POCs construtivos para testar a viabilidade das modificações propostas, reduzindo-se o risco de impor muito esforço numa possível modificação mal-sucedida. Inclusive recomenda-se que eles sejam parte integrante da especificação. Eles servirão de base inicial para a fase de implementação seguinte.

Por exemplo, no caso em que se tem um tempo de resposta estabelecido para uma determinada carga, pode-se fazer a alteração em uma pequena parte do sistema em um ambiente controlado e simular a carga, verificando se o tempo de resposta ficou dentro do desejado. Se o teste foi bem-sucedido, tal modificação já é um exemplo a ser seguido na implementação das modificações nas outras partes do sistema.

Ao terminar esta fase, tem início a Fase 5.

3.5 Fase 5: Implementação da manutenção

Esta fase corresponde à implementação propriamente dita das modificações especificadas na fase anterior. Inclusive POCs construtivos da fase anterior podem ser a base inicial dos trabalhos.

As ferramentas para auxiliar a implementação variam, mas são praticamente as mesmas utilizadas em ambientes de desenvolvimento.

Esta fase é realizada pela Equipe de Manutenção.

Terminada esta fase, é feita uma nova avaliação da adequação da arquitetura, representada pela Fase 3.

4 EXEMPLO DE APLICAÇÃO DO ROTEIRO

O roteiro apresentado na seção 3 endereça temas muito importantes como manutenção e técnicas arquiteturais. Portanto, uma possível aplicação do roteiro, 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.

Este roteiro tem sido aplicado em um laboratório de graduação de Engenharia de Software do curso de Engenharia de Computação. Nas aplicações aqui mencionadas, os

alunos cursam o segundo semestre do quarto ano (de cinco no total) de Engenharia de Computação. Desde 2003 tem-se procurado ressaltar pontos de Arquitetura de Software e Qualidade. Este roteiro é resultado de refinamentos desde seu início em 2003 até a publicação deste artigo.

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. As técnicas arquiteturais apresentadas na seção 2 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.

Restrições no uso do roteiro nesta aplicação serão apresentadas nas respectivas fases e estas são apresentadas a seguir.

4.1 Estrutura do curso

A turma é dividida em grupos de 2 alunos cada. Todos os grupos atuam no mesmo sistema, porém em vertentes distintas, ou seja, 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.

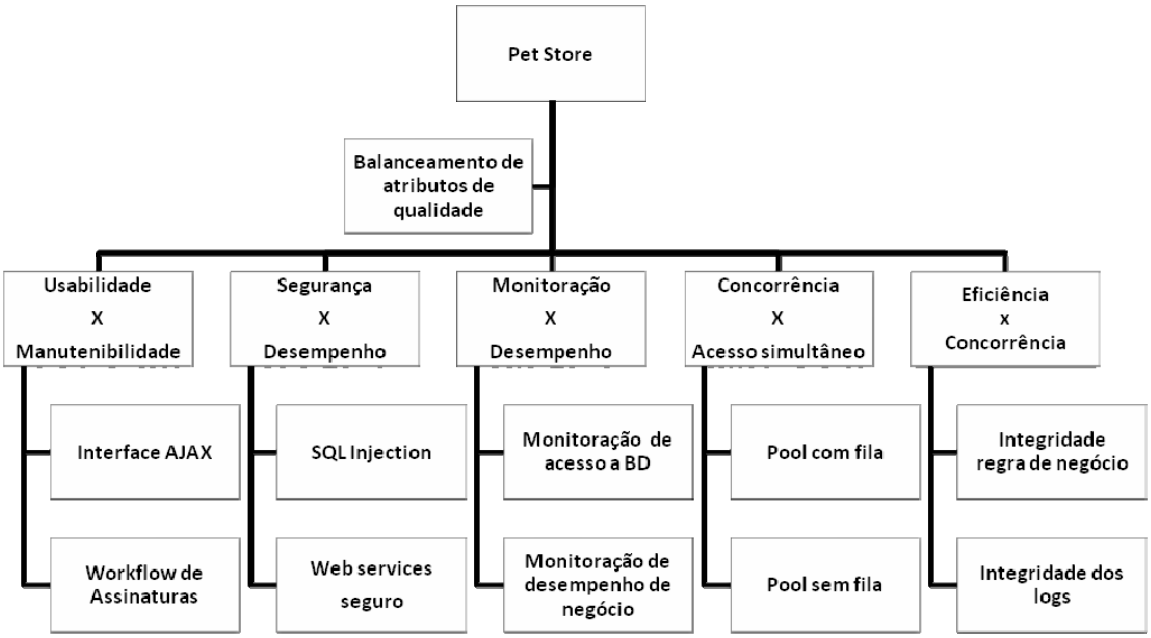

Figura 3 - Exemplo de mapa de grupos.

Na Figura 3 acima, vemos um mapa que demonstra como foram divididos os grupos em uma aplicação da disciplina. Vemos que esta turma teve 10 grupos, onde cada um

atuou no balanceamento entre 2 atributos de qualidade. Por exemplo: no caso do tema "Concorrência X Acesso simultâneo", tivemos 2 grupos: um que tratou do "Pool com fila" (acesso a informações de dados com pool de conexões e uso de filas) e outro que tratou do "Pool sem fila" (acesso a informações de dados com uso de pool de conexões sem uso de fila).

4.2 Fase 1: Apresentação do sistema

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 (p.e. 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 (2006).

Embora a apresentação do sistema tenha sido feito pelos professores, os alunos foram incentivados a seguir os procedimentos de instalação e *deployment* do sistema, o que permitiu também o contato com as ferramentas que utilizariam durante o decorrer do curso.

4.3 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 é esperado que os resultados finais de cada um 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.

4.4 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.

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 um feedback apropriado (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.

4.5 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 (*Hypertext Transfer Protocol*) ou que estoure o limite de timeout. Dado esse objetivo 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.

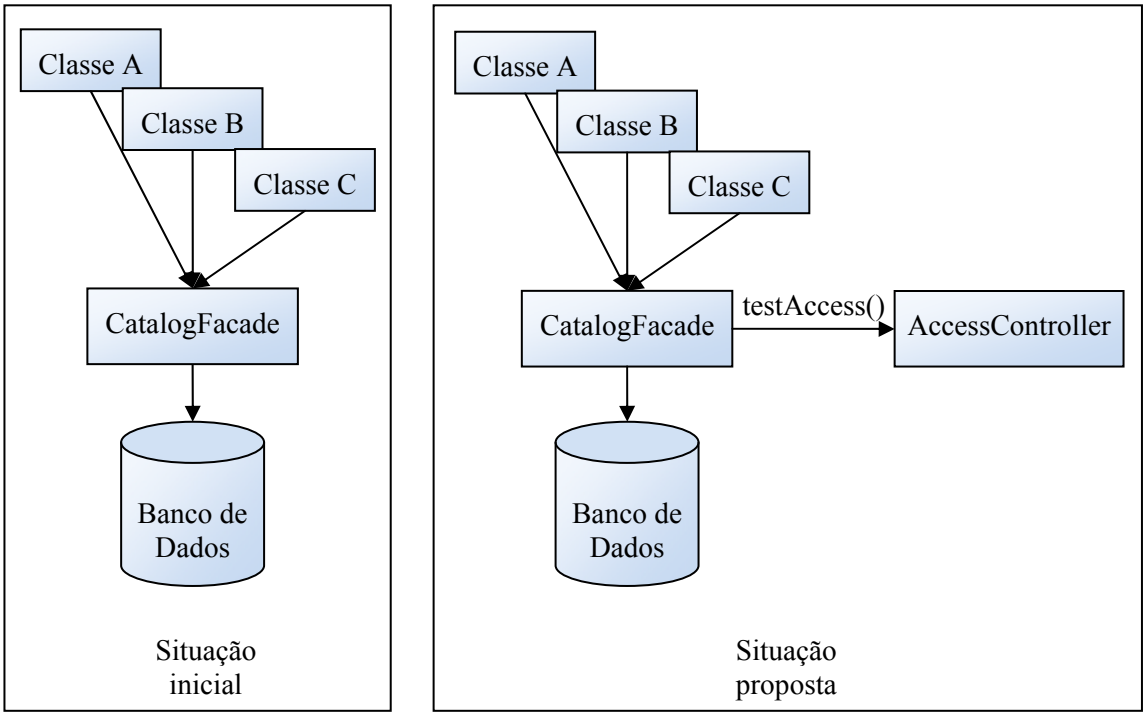

Figura 4 - Exemplo de comparação entre duas arquiteturas.

Os alunos tentaram uma segunda solução. A Figura 4 acima 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.

4.6 Fase 5: Implementação da manutenção

Esta fase corresponde à implementação propriamente dita das modificações especificadas na fase anterior. Aqui, os grupos utilizaram os POCs 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.

4.7 Fase 6: Evidenciação de adequação da arquitetura

Na verdade, o roteiro não possui uma "Fase 6". Esta fase na verdade é a 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.

4.8 Ferramentas utilizadas

Seguem uma lista com as principais ferramentas utilizadas nesta aplicação do curso:

- **Java EE:** 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 aplicação.
- **JavaBD:** sistema gerenciados de banco de dados.
- **JMeter:** teste de carga que simula diversos usuários acessando o sistema simultaneamente.

5 CONSIDERAÇÕES FINAIS

O curso apresentado 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 pretendidos nesta linha é de especializar o roteiro de forma a ressaltar conceitos diferentes destes apresentados na seção 4.

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 roteiro 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 roteiro 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 roteiro pode ser refinado também para comportar outras técnicas.

Os pesquisadores têm utilizado este método ou partes dele com sucesso em projetos da indústria para subsidiar manutenções em sistemas com requisitos não funcionais críticos. Experimentos também têm sido discutidos com especialistas das SEI, como por exemplo nos trabalhos ARAKAKI e ENOBI (2008) ou ainda em DIAS e PEDROSO (2008).

O roteiro proposto é parte de uma pesquisa que vem sendo desenvolvida pelos autores 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.

REFERÊNCIAS BIBLIOGRÁFICAS

AMMAR, H., GUNNALAN, R. E SHERESHEVSKY, M. Pseudo dynamic metrics. In: CONFERENCE ON COMPUTER SYSTEMS AND APPLICATIONS, 2005. **Anais**. p. 117-vii.

ARAKAKI, R, ENOBI, F. Challenges and Observations of Applying the SEI ATAM to a Software Testing Automation Solution. In: SATURN 2008, 2008. **Anais**. Disponível em http://www.sei.cmu.edu/architecture/saturn/2008/recap.html.

BASS, L., CLEMENTS, P. E KAZMAN, R. **Software Architecture in Practice**, Addison Wesley, 2ª edição, 2003.

CLEMENTS, P., KAZMAN, R. E KLEIN, M. **Evaluating Software Architectures**, Addison Wesley, 2002.

DIAS, S., PEDROSO, M. The Value of Sofware Architecture. In: WORKING IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE (WICSA) 2008, 2008. **Anais**. Disponível em http://www.iso-architecture.org/wicsa2008/wval.html.

IEEE STD 1471. Recommended Practice for Architectural Description of Software-Intensive Systems. 2000.

PRESSMAN, R. S. **Software Engineering: A Practitioner's Approach**, McGraw-Hill, 5ª edição, 2001.

SEI. Software Architecture for Software-Intensive Systems, Software Engineering Institute, Carnegie Mellon, http://www.sei.cmu.edu/architecture/, março de 2008.

SUN MICROSYSTEMS. Blueprints: Java Pet Store Reference Application, 2006. https://blueprints.dev.java.net/petstore/, março de 2008.

TEACHING ARCHITECTURAL TECHNIQUES IN A CONTEXT OF MAINTENANCE

Abstract: *Several businesses today are supported by systems software. With the dynamism of business often happens that the set of requirements that led to the construction of an architecture for such systems is no longer the same. In many situations we must live with legacy systems, so we need to change them for new needs. Software Architecture is taken as a major means to achieve business and quality goals. Maintenance of systems using architectural evolution techniques has proven to be an effective way to change a system to the new situation. So it is important that students of Computer Engineering hold the knowledge of Software Architecture and Maintenance and their use together. For this, it has been settled up a laboratory where under graduating students can make practical use of architectural techniques in a context of maintenance. Then, it was proposed a general architecture centered model on whose maintenance phases can be adapted to expose students to use the techniques that the leaders of the course selected. Apart from this roadmap, this paper presents an application in Software Engineering Laboratory with under graduating students on penultimate year where they could use architecture evaluation techniques, software metrics and proofs of concept.*

Key-words: *Software architecture maintenance, Architectural techniques, Software Engineering Laboratory, Teaching of Software Architecture, Software Architecture Evolution*