UNIVERSIDADE FEDERAL DE CAMPINA GRANDE - UFCG
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA - CEEI
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO - UASC

Relatório de Estágio

ANÁLISE AUTOMÁTICA DE IMPACTO CAUSADO POR MUDANÇAS EM SISTEMAS DE SOFTWARE

João Arthur Brunet Monteiro
jarthur@dsc.ufcg.edu.br
Curso de Bacharelado em Ciência da Computação - CEEI/UFCG

Campina Grande, Outubro de 2007.

Sumário

Lista de Figuras

1 Introdução

Ao longo da evolução na indústria de construção *softwares*, os processos de desenvolvimento tornaram-se de extrema importância para que as empresas de *software* sejam cada vez mais competitivas. O motivo para este cenário é que os processos visam sistematizar as atividades de maneira a organizar a construção de *softwares*. Neste cenário, destacam-se várias metodologias, tais como processos orientados a planos [9] e processos ágeis, como o o eXtreme Programming (XP) [6]. Estes processos diferem entre si na forma como conduzem o desenvolvimento dos projetos de *software*. Processos ágeis adaptam-se melhor a equipes pequenas e priorizam a posse coletivas do código, além de advogar por ciclos de iteração curtos. Por outro lado, os processos orientados a planos dividem o desenvolvimento em requisitos, projeto e implementação, seguindo um processo padrão definido [8].

Mudanças em sistemas de *software* são inevitáveis por diversos motivos. Dentre os principais está o atendimento aos requisitos demandados pelos clientes. Para uma empresa manter-se competitiva no mercado é preciso que ela esteja em constante contato com o cliente e atendendo os seus requisitos com rapidez e eficiência. Os processos de *software* lidam de maneira distintas em relação à mudanças nos sistemas de *software*. No entanto, a maioria deles leva em consideração este aspecto. Boas práticas de programação [10], dentre outros artifícios, são utilizadas para que uma mudança no código do sistema seja simples de ser implementada.

Neste contexto, a necessidade de estimar o custo de efetuar-se mudanças nos sistemas de *software* se faz muito importante para as empresas que produzem tais sistemas. Este custo está comumente relacionado ao número de horas que vão ser gastas para efetuar tais mudanças. A técnica utilizada para estimar custos de mudanças em sistemas de *software* é chamada de análise de impacto.

Analisar o impacto de mudanças em sistemas de *software* é um processo onde pretende-se identificar os componentes que serão afetadas por uma mudança, e a partir destes, calcular o custo dessa mudança. A partir do uso da análise de impacto, as empresas visam errar menos ao estimar o custo de um requisito ou de um determinado componente a ser desenvolvido. Quanto mais precisa for essa estimativa, menos prejuízo as empresas terão, uma vez que o pagamento de multas por quebra de contrato em relação ao tempo de desenvolvimento ainda é um grande problema na indústria de *software*.

Como vamos explanar com mais detalhes na seção 4, a análise de impacto causado por mudanças no código dos sistemas de *software* é comumente feita manualmente. Além de estar sujeita a erros, esta atividade é lenta e laboriosa, visto que o encarregado de fazer a análise deve verificar em todo o código do

sistema as possíveis entidades impactadas pela mudança.

O foco do estágio realizado foi desenvolver uma ferramenta que provê suporte a análise de impacto causado por mudanças no código de sistemas de *software*. Para isso, fez-se necessário um estudo sobre a evolução de *software* e as técnicas existentes que utilizam métricas da evolução de *software* para analisar características evolutivas do sistema.

O trabalho efetuado durante o estágio é parte de um projeto executado pelo Grupo de Métodos Formais, em parceria com a empresa *CPM Braxis* [1], intitulado *Design Checker*. Este projeto tem por objetivo desenvolver técnicas e ferramentas que suportem a verificação, o rastreamento da evolução e a avaliação dos impactos gerados por mudanças em *designs* de *software* no desenvolvimento. Tais ferramentas oferecerão suporte automatizado às fábricas de *software* da *CPM Braxis* para o controle da qualidade dos produtos desenvolvidos.

A ferramenta foi desenvolvida durante o período de estágio e será aperfeiçoada durante o curso do projeto *Design Checker*. Ela tem por objetivo identificar as entidades direta e indiretamente impactadas por uma mudança no *design* do código de um sistema. À esta ferramenta, demos o nome de *Impala*. Na seção 5 vamos caracterizar mais detalhadamente esta ferramenta.

2 Ambiente do Estágio

O estágio foi desenvolvido no Laboratório de Métodos Formais (pertencente ao Grupo de Métodos Formais) do Departamento de Ciências da Computação (DSC) da Universidade Federal de Campina Grande (UFCG).

O grupo de pesquisadores formado de professores e estudantes possui a meta de investigar, desenvolver e disseminar técnicas e métodos com base formal, bem como desenvolver ferramentas para dar suporte ao desenvolvimento de sistemas de software da alta qualidade.

O LMF ocupa uma área de aproximadamente 100 m2, disponibilizando cerca de 25 postos de trabalho e 2 postos de impressão, além de dispositivos especializados para digitalização de imagens e gravação de CDs. Além disso, o laboratório faz parte da rede corporativa da UFCG que tem tecnologia ATM a 155 Mbps e está ligada à Internet via o POP-PB da Rede Nacional de Pesquisa (RNP).

Endereço: Universidade Federal de Campina Grande
Departamento de Sistemas e Computação
Laboratório de Métodos Formais
Av. Aprígio Veloso, 882 Bodocongó, Bloco BB
Telefone: (83) 33101430.

2.1 Supervisão

A supervisão técnica do estágio foi realizada por Geovani Santangelo de Araujo Lima Santos, gerente de Pesquisa e Desenvolvimento da *CPM Braxis* e Lile Hattori, mestranda em Ciência da Computação pelo Grupo de Métodos Formais. Geovani é responsavel pela gestão de projetos de pesquisa realizados pela empresa, atuando fortemente com software livre, utilizando plataformas J2EE e .NET, os mais diversos ambientes de desenvolvimento e IDEs e bancos de dados de mercado e free. A supervisão acadêmica foi efetuada pelo professor doutor Dalton Serey, pesquisador do Grupo de Métodos Formais.

Dados do supervisor acadêmico
Nome: Dalton Dario Serey Guerrero
Endereço: Rua José de Alencar, 752
Bairro: Bodocongó
Cidade: Campina Grande - PB

CEP: 58109997 Telefone: 8333101429 Fax: 3101046
E-Mail: dalton@dsc.ufcg.edu.br
Cargo: Coordenador Geral

Dados do supervisor técnico
Nome: Geovani Santangelo de Araujo Lima Santos
Endereço: Rua Helio de Oliveira, 67
Bairro: Brotas
Cidade: Salvador UF: BA
CEP: 40280630 Telefone: 7133509721 Fax: 713509778
E-Mail: geovani@unitech.com.br
Cargo: Gerente de Pesquisa

Dados do supervisor técnico
Nome: Lile Palma Hattori
Endereço: Rua João Machado, 412
Bairro: Prata
Cidade: Campina Grande UF: PB
CEP: 58101-300
E-Mail: lhattori@gmail.com
Cargo: Mestranda em Ciência da Computação

3 Metodologia de Trabalho

Nesta seção, iremos detalhar como o processo de desenvolvimento do estágio foi conduzido e quais as técnicas utilizadas para gerenciá-lo, uma vez que a equipe estava distribuída geograficamente.

Quatro pessoas estiveram envolvidas diretamente neste trabalho: O orientador acadêmico, Dalton Serey. O orientador técnico, Geovani Santangelo. Lile Hattori, aluna de mestrado do Grupo de Métodos Formais e cliente de nossa solução. Finalmente, João Arthur, estudante de graduação, foi responsável por desenvolver a ferramenta. A composição de tarefas e delegação das mesmas eram efetuadas em reuniões quinzenais com o cliente direto do módulo a ser desenvolvido. Devido ao fato de um integrante da equipe estar separado geograficamente, foi necessário também efetuar reuniões por telefone.

Existem basicamente dois clientes de nossa solução: Lile Hattori e a empresa *CPM Braxis*, representada por Geovani. Lile é interessada em fazer uso do trabalho desenvolvido neste estágio como parte de validação do seu trabalho de mestrado. Geovani, foi responsável por aplicar a nossa solução na empresa *CPM Braxis* com intuito de aumentar o controle de qualidade do processo de desenvolvimento da empresa, fazendo com que esta passe de CMMI 3 para CMMI 5. Geovani e Lile foram responsáveis por levantar requisitos específicos que a solução deveria possuir e repassar para o desenvolvedor como forma de tarefas a serem implementadas.

O desenvolvimento do código da solução não seguiu um processo de desenvolvimento de software específico, no entanto, algumas técnicas de XP [7] foram utilizadas. Esse processo foi tido como referência por adequar-se ao nosso contexto, uma vez que que a nossa equipe era pequena e os requisitos eram mudados com frequência.

Com o intuito de manter a qualidade do código desenvolvido, bem como manter sua conformidade com os requisitos solicitados, fizemos o uso de testes de unidade e testes de aceitação respectivamente. Os testes de aceitação eram compostos pelos clientes de forma textual e deveriam ser endereçados pelo código. Arquivos de entrada e suas respectivas saídas eram gerados manualmente pelos clientes. Durante o desenvolvimento do código, os testes foram executados sistematicamente e um requisito só era considerado implementado com sucesso se a saída fosse a mesma que a especificada pelos testes de aceitação. Testes de unidade também foram amplamente utilizados pela equipe de desenvolvimento com intuito de diminuir a quantidade de bugs correntes no código. Fizemos uso do framework JUnit [3] para compor e executar testes de unidade. Quando os testes de aceitação não eram suficientes para descrever detalhadamente o requisito a ser implementado, reuniões eram

executadas para haver um consenso entre o problema e a solução adequada. Diagramas e figuras ilustrativas também foram utilizados para este fim.

É importante salientar que as atividades desenvolvidas no estágio fizeram parte também de um trabalho de mestrado, acrescentando assim um caráter de pesquisa ao trabalho. O aluno foi responsável também por ler artigos científicos que abordam do tema análise de impacto [5, 12]. A leitura destes artigos foi responsável por aumentar o conhecimento do aluno em relação ao problema abordado.

O gráfico 1 mostra em termos percentuais a alocação de tempo para as atividades desenvolvidas. Note que a atividade desenvolvimento da ferramenta se fez predominante durante o estágio. Embora esta seja a atividade principal a ser executada pelo estagiário, a atividade de pesquisa também demandou tempo. A equipe estava segura de que alocar tempo para esta atividade aumentaria o entendimento do mesmo em relação ao problema abordado, diminuindo assim problemas na fase de implementação gerados pela falta de embasamento teórico. As atividades de acompanhamento e planejamento eram executadas durante as reuniões semanais. As atividades foram registradas no *XPlanner* [4] - uma ferramenta *web* de gerenciamento de tarefas. Por ser uma ferramenta de uso intuitivo, o *XPlanner* acelerou o processo de acompanhamento de tarefas, além de servir como registro de todo o trabalho de estágio.

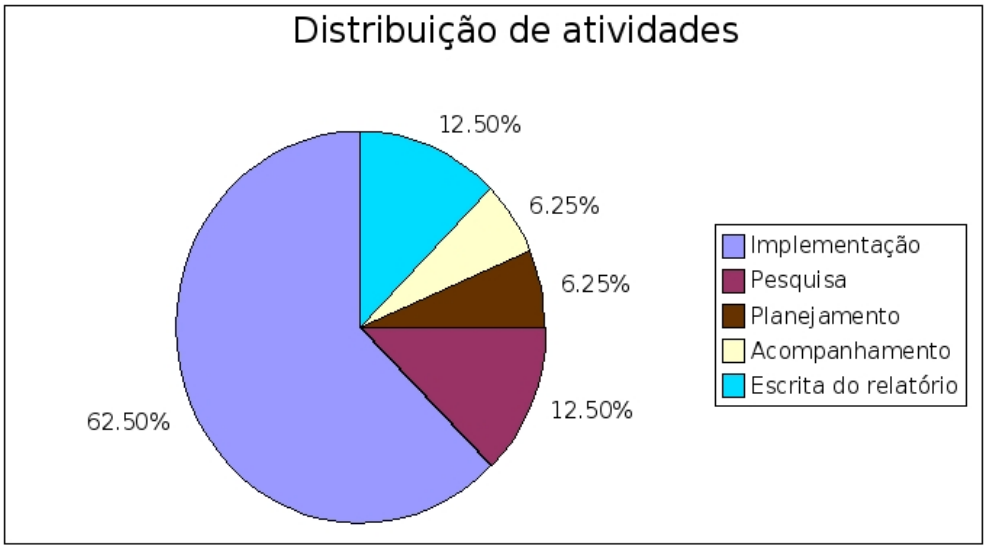

Figura 1: Distribuição das atividades

4 Problema Abordado

Analisar impacto de mudanças é uma tarefa que vem sendo utilizada com frequência pelas empresas que constroem sistemas de software. Um dos objetivos desta atividade é estimar, da forma mais precisa possível, o custo de efetuar mudanças no código. Para essas empresas, é extremamente importante que essa estimativa se aproxime cada vez mais do custo real, tendo em vista que o gastos com multas decorrentes de atrasos na entrega de produtos, bem como pagamento de horas extras para os desenvolvedores, podem ser minimizados.

Na empresa *CPM Braxis*, o processo de análise de impacto de mudanças no código é feito manualmente. Um profissional da empresa é encarregado de analisar as mudanças propostas e estimar, baseado em métricas específicas, quantas horas de trabalho serão gastas para aplicar tais mudanças. O problema de se fazer análise manual de impacto é que esta análise é relativamente laboriosa e lenta, uma vez que o profissional responsável pela análise deve verificar em todo o código quais entidades serão impactadas pelo conjunto de mundanças proposto. Ainda, esta técnica exige a existência de um profissional que tenha pleno conhecimento do código do sistema. Em empresas que fabricam software, o desenvolvimento de sistemas muitas vezes é conduzido por equipes distintas que trabalham em módulos distintos e, por vezes, separada geograficamente. Este cenário contribui para que o conhecimento coletivo do código como um todo seja praticamente impossível.

A atividade de se analisar o código manualmente está sujeita a erros pela sua inerente complexidade. A *CPM Braxis*, e as empresas em geral, necessitam que o processo seja automatizado de maneira a gerar estimativas mais próximas do custo real, além de agilizar o processo desta análise.

O trabalho realizado durante o período de estágio foi o de desenvolver uma ferramenta que lista automaticamente, dado um conjunto de mudanças, as entidades impactadas por tais mudanças no código.

5 A Solução

Como resultado deste estágio, foi implementado o *Impala*, uma ferramenta cujo objetivo principal é dar suporte à análise de impacto causado por mudanças no código fonte de um sistema. Ou seja, esta ferramenta surge como solução para o problema descrito na seção 4.

Nas próximas seções descreveremos como a ferramenta é estruturada, bem como seu funcionamento.

5.1 Arquitetura e Funcionamento

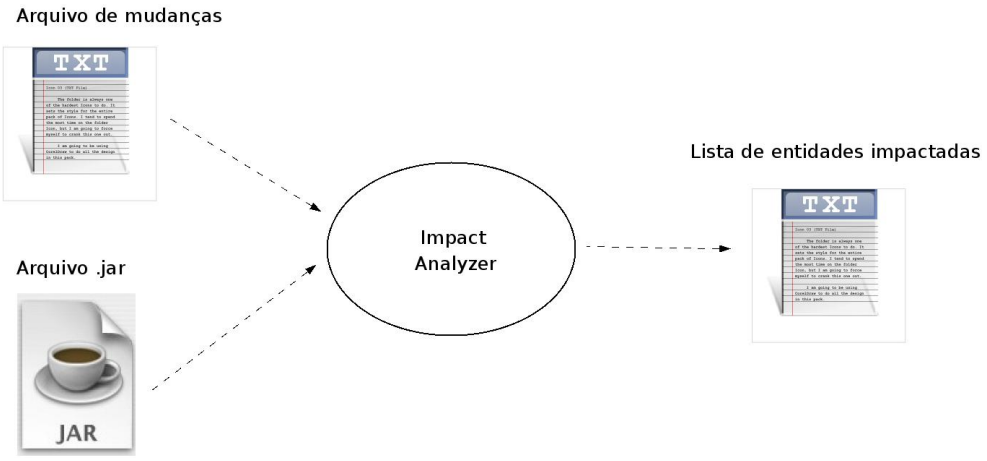

Figura 2: Analisando impacto automaticamente

A Figura 2 ilustra os componentes envolvidos na análise automática de impacto. A ferramenta recebe como entrada um arquivo executável *JAR* [1] correspondente ao código do sistema a ser analisado e um arquivo composto por um conjunto de mudanças especificados textualmente e seguindo um padrão pré-definido por nossa equipe. Ao final da análise, o *Impala* gera um arquivo com as entidades impactadas pelas mudanças.

Cada linha presente no arquivo de mudanças corresponde a uma mudança que pretende-se efetuar no código. Essas mudanças serão analisadas automaticamente e o usuário do *Impala* terá acesso às entidades impactadas em decorrência das mudanças. As mudanças são especificadas como se segue:

- Tipo: Tipo da mudança a ser efetuada.

[1] Java Archive (JAR) é um arquivo compactado usado para distribuir um conjunto de recursos de uma aplicação Java. É usado para armazenar classes compiladas e metadados associados que podem constituir um programa.

- Entidades: Entidades (Classes, Métodos ou Atributos) envolvidas na mudança.

O número de entidades envolvidas nas mudanças varia de acordo com o tipo da mesma. Atualmente, os seguintes tipos estão disponíveis para análise:

- RemoveCommand - Remoção de uma entidade
- AddCommand - Adição de herança entre classes ou adição de entidades
- ChangeAttributeCommand - Mudança em um atributo
- ChangeReturnTypeCommand - Mudança no tipo de retorno de um método
- ChangeSignatureCommand - Mudança na assinatura de um método
- ChangeVisibilityCommand - Mudança na visibilidade de uma entidade

5.2 Implementação

5.2.1 Aquisição de Informações

Para listar as entidades que são impactadas direta e indiretamente por uma mudança no código, é preciso possuir informação sobre a estrutura do mesmo. O *Impala* faz uso da biblioteca *Design Wizard* [11] para ter acesso a essas informações. *Design Wizard* é uma biblioteca desenvolvida pelo Grupo de Métodos Formais que é utilizada para extrair informações de um arquivo *JAR* e modela essas informações em um conjunto de Classes, Métodos, Atributos e suas respectivas relações. Esta biblioteca provê uma API composta por uma série de métodos que podem ser usados para consultar sobre as relações entre as entidades do código extraído. Exemplos destes métodos são listados abaixo:

- getCallers() - Retorna os métodos que acessam diretamente a entidade.
- getSubClasses() - Retorna as subclasses de uma determinada classe.
- getStaticMethods() - Retorna os métodos estáticos de uma determinada classe.

As informações providas pelo *Design Wizard* em sua versão original não eram suficientes para que a análise revelasse todas as entidades efetivamente impactadas por determinada mudança. Isto porque a biblioteca não levava

em consideração questões como herança de atributos e métodos. Por este motivo, algumas entidades impactadas por determinada mudança não constavam como impactadas. Para fins de clareza, vamos analisar o cenário de herança através de exemplos.

Código 1 Super Classe

```
1 class SuperClasse {
2  void m1() {}
3 }
```

Código 2 Sub Classe

```
1 class SubClasse extends SuperClasse{
2 }
```

Código 3 Cliente

```
1 class Cliente {
2  public static void main(String[] args) {
3   SubClasse sub = new SubClasse();
4   sub.m1();
5  }
6 }
```

Para cenário especificado pelos Códigos 1, 2, 3, a biblioteca *Design Wizard* interpretava que o método *main* da classe *Cliente* (Código 3) invoca o método *m1* da classe *SubClasse* (Código 2). No entanto, como a classe SubClasse não sobrescreve o método *m1*, o método executado será o m1 pertencente à *SuperClasse* (Código 1).

Houve então a necessidade de modificar a biblioteca *Design Wizard*, incorporando algoritmos que lidam com questões de herança entre entidades. Esta atividade fez com que essa biblioteca evoluisse de maneira positiva em relação às informações providas. Ao longo do estágio, uma nova versão da biblioteca foi desenvolvida e disponibilizada através do endereço `http://www.designwizard.org`.

5.2.2 Algoritmos

Os algoritmos que fazem a análise de impacto usam a API do *Design Wizard* para analisar a estrutura do código e determinar quais serão as entidades

impactadas. O Código 4 mostra o código que é responsável por listar as entidades impactadas pela remoção de uma determinada entidade.

Código 4 Análise de impacto referente à remoção de uma entidade.

```
 1 execute(entity, depth){
 2   if (depth==0) return impactedSet;
 3   //Chamadores cuja distância é um.
 4   Set impactedSet = entity.getCallers();
 5   Set impactedChildren;
 6   for each entity in impactedSet{
 7     impactedChildren.add(execute(entity, depth-1));
 8   }
 9   //Includes entity if it is not in the set
10   impactedSet.add(impactedChildren);
11   return impactedSet;
12 }
```

O algoritmo usado para listar as entidades impactadas é recursivo e usa como condição de parada a distância máxima entre os elementos que devem entrar na lista e o elemento a ser modificado. Para fins de clareza, vamos definir o que neste trabalho tratamos por distância entre os métodos analisando a Figura 3.

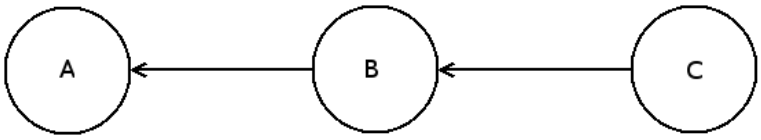

Figura 3: Distância entre métodos

Se as chamadas de métodos forem mapeadas para um grafo não ponderado e dirigido, onde:

- os lugares representam os métodos
- as arestas representam uma chamada dentro de um método a outro
- a direção das arestas representa o método chamado

temos que a distância entre estes métodos é o tamanho do menor caminho simples com origem no método chamador e término no método chamado.

Portanto, o método C está a uma distância de tamanho dois do método A, enquanto que o método B está a uma distância de tamanho um de ambos, C e A. [2]

Usando a distância entre os métodos, os algoritmos de análise de impacto retornam todas as entidades que são impactadas pelas mudanças na distância requerida. Por omissão, a distância máxima analisada pelo algoritmo é 7 (sete). No entanto, este valor é fácil de ser configurado.

Cada tipo de mudança requer um algoritmo que analise o seu impacto. Não iremos descrever todos neste documento por questão de simplicidade. No entanto, vale ressaltar que todos os algoritmos foram extensamente testados através da satisfação aos testes de aceitação e à composição de testes de unidade.

5.2.3 Resultado da Análise

O resultado gerado pela análise de uma mudança no código é um arquivo contendo todas as entidades impactadas dentro da distância requerida. Na Figura 4 podemos ver um exemplo destes arquivos.

Figura 4: Lista de entidades impactadas

Neste caso, uma mudança no método atualizaEmpregados(java.util.Map) da classe EmpregadoDAO tem impacto sobre todas as entidades descritas no resto do arquivo.

[2]Um método está a uma distancia zero dele próprio.

6 Trabalhos Futuros

A ferramenta *Impala* está inserida no contexto do projeto *Design Checker*, um projeto realizado pelo Grupo de Métodos Formais em parceria com a empresa de *software CPM Braxis*. A identificação das entidades impactadas por mudanças foi efetuada com sucesso neste estágio. As proximas atividades relacionadas ao projeto visam combinar a identificação dessas entidades com heurísticas para avaliar o custo das mudanças.

Uma abordagem que já vem sendo implementada no projeto é o uso de dados extraídos do CVS como forma de extrair padrões de comportamento das mudanças. Isto é, poder traçar comportamentos que podem ser utilizados como parâmetros para estimativas de custos, por exemplo, acoplamento entre mudanças.

Além disso, por se tratar de um projeto inserido em um ambiente de pesquisa, serão efetuadas pesquisas bibliográficas e experimentações com o intuito de identificarmos e desenvolvermos um conjunto de heurísticas que, aliados à análise de impacto de mudanças, contribuam para que o custo estimado aproxime-se cada vez mais do custo real das mudanças no ambiente do software.

Outra atividade já em execução é a criação de um *plugin* no ambiente Eclipse [2] como forma de prover uma interface com o usuário mais amigável do que está em vigor atualmente. Este plugin tem o objetivo de facilitar a visualização das entidades impactadas, bem como de informações associadas a essas entidades.

7 Conclusão

A realização do estágio em parceria com a empresa CPM-Braxis trouxe uma série de benefícios que contribuiram para a formação profissional do aluno. Estar em constante contato com a fábrica de software, realizando reuniões periódicas, participando ativamente de todo o processo de desenvolvimento e implementando requisitos reais, foram essenciais na aquisição de uma experiência não oferecida no ambiente acadêmico.

Um dos fatores que dificultaram o trabalho a ser feito foi a separação geográfica dos componentes envolvidos no contexto do estágio. De fato, a falta de comunicação foi um dos obstáculos encontrados no início do projeto. No entanto, a adaptação a esse cenário foi relativamente fácil com o uso de ferramentas *web* de gerenciamento de projetos [4].

Por se tratar de um projeto que está inserido tanto no contexto acadêmico quanto no ambiente empresarial, em certos momentos do projeto, as diferenças entre os requisitos destes dois ambientes era notória, além do prazo fixado para atendimento dos mesmos. O atendimento aos requisitos teve que ser efetuado de forma a priorizar os requisitos mais urgentes e críticos. Em sua maioria, estes eram demandados pela empresa *CPM Braxis*, em função da existência de prazos para a implantação do nível CMMI 5.

O uso de testes de aceitação foi extremamente proveitoso para validar o trabalho que estava sendo feito. Além disso, os testes serviram como boa documentação dos requisitos feitos pelo cliente. Esta característica amenizou os problemas gerados pela distribuição da equipe. Podemos destacar o uso de testes de aceitação como um dos fatores importantes que contribuiram para o sucesso do projeto. A inclusão do cliente no processo de desenvolvimento torna este processo uma tarefa menos laboriosa e mais confiável em relação à entrega do produto esperado.

Por fim, asseguramos que as atividades planejadas na proposta de estágio foram, excetuando-se o desenvolvimento de heurísticas para cálculo do custo de mudanças, desenvolvidas com sucesso. A tarefa não realizada foi postergada para os trabalhos futuros pela sua alta complexidade. De fato, desenvolver boas heurísticas que, aliadas ao trabalho efetuado neste estágio, calculem de maneira aceitável o custo de mudanças no código é um trabalho de complexidade elevada e que será desenvolvido por um aluno de mestrado do Grupo de Métodos Formais em parceria com alunos de graduação que estão alocados para o projeto *Design Checker*.

A ferramenta tem sido utilizada pela empresa *CPM Braxis* com o intuito de aumentar o controle de qualidade do processo de desenvolvimento da empresa. Ao que parece, a *CPM Braxis* está satisfeita com a ferramenta implementada e pretende inserir em todo o seu processo de desenvolvimento a

ferramenta *Impala*. Além disso, o *Impala* está sendo utilizado como validação do trabalho de mestrado de um dos integrantes do Grupo de Métodos Formais.

Referências

[1] Cpm braxis. www.cpmbraxis.com.

[2] Eclipse project. http://www.eclipse.org/.

[3] Junit. www.junit.org.

[4] Xplanner project. http://www.xplanner.org/.

[5] T. Apiwattanapong, A. Orso, and M. J. Harrold. A differencing algorithm for object-oriented programs (best paper award, acm sigsoft distinguished paper award). In *20th IEEE International Conference on Automated Software Engineering (ASE 2004)*, pages 2–13, Linz, Austria, September 2004.

[6] K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[7] K. Beck and M. Fowler. *Planning Extreme Programming*. Addison Wesley, 2001.

[8] B. Boehm and R. Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[9] M. B. Chrissis, M. Konrad, and S. Shrum. *CMMI Guidlines for Process Integration and Product Improvement*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Reading, Massachusetts, 1994.

[11] J. A. B. Monteiro and D. D. S. Guerrero. Extração e verificação automática de modelos de software. In *III Congresso de Iniciação Científica da Universidade Federal de Campina Grande*, Campina Grande, PB, BRA, Outubro 2006. UFCG.

[12] D. L. Parnas. Software aging. In *ICSE '94: Proceedings of the 16th international conference on Software engineering*, pages 279–287, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.