Sistema de *Log* de Ações em aplicações *Web* Utilizando Programação Orientada a Aspectos

Maxwell Queiroz Francisco[1], Felype Nery de O. Vasconcelos[1], Richardson Bruno[1], José Inácio Nery da Fonseca[1], Aida Araújo Ferreira[2]

[1]Graduando em Análise e Desenvolvimento de Sistemas – IFPE. e-mail: {maxwell2k,felypenery,richardsonbruno,inacionery}@gmail.com
[2]Docente do curso de Análise e Desenvolvimento de Sistemas – IFPE. email: aidaaf@gmail.com

Resumo: A segurança dos dados e principalmente a integridade das transações são objetos de muita preocupação para as empresas, que possuem sistemas *on-line*, trabalhando direta ou indiretamente com seus ativos, visto que o interesse pela modalidade de comércio eletrônico se difundiu em diversos tipos de mercado, devido ao baixo custo operacional e seu alcance global, causando certa dependência em constantes aperfeiçoamentos no quesito segurança. Este artigo demonstra a construção de um sistema *Web* utilizando Programação Orientada a Aspectos (POA), que realiza o *log* de acessos a chamadas de métodos utilizando a linguagem AspectJ. Foi utilizado a IDE NetBeans como ambiente de desenvolvimento e adotada uma sintaxe alternativa conhecida como *Annotations* por meio de *interfaces*, que serviram para relacionar o método chamado ao perfil de usuário, permitindo assim uma maior rastreabilidade de suas ações no sistema. Este modelo foi implementado em um dos sistemas do Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco-IFPE, que está em operação e nomeado como Programa Monitoria.

Palavras–chave: Programação Orientada a Aspectos, AspectJ, *Log* de Ações, Segurança da Informação

1. INTRODUÇÃO

Devido ao significativo crescimento tecnológico mundial e os benefícios que ela proporciona, atualmente é difícil encontrar uma empresa, e muito menos uma rede de negócios, que não possua um Sistema de Informação Computadorizado manipulando de alguma forma os seus ativos, sendo assim as informações trafegadas assumem uma importância cada vez maior para estas instituições. Neste contexto, a exposição destas informações, sejam por falta de políticas de segurança ou até mesmo diretrizes básicas de Segurança de Informação, podem maximizar ameaças e possíveis prejuízos para a empresa, visto que a maioria destes sistemas está conectada a redes de computadores locais e muitas vezes também na internet.

Para Marciano (2006), os conteúdos e continentes digitais estão expostos a vários tipos de ameaças, sejam elas físicas ou virtuais, que muitas vezes comprometem gravemente a segurança das informações e até a segurança das pessoas e informações relacionadas a elas. Contudo, a Tecnologia da Informação (TI) dispõe de parte da solução para este problema, não sendo capaz, de resolver e garantir totalmente as falhas do universo tecnológico que a empresa está envolvida. O autor ainda afirma que em alguns casos a TI pode contribuir e agravar a possível falha.

Corroborando com esta afirmação, o processo de identificação e reparo de falhas, pode ser lento e oneroso, e se tratando de software, muitas vezes são oriundas de metodologias de desenvolvimento infundadas da empresa fornecedora do mesmo, que afetam a segurança e desempenho do produto utilizado pelo cliente, resultando na ineficácia de toda a estrutura de segurança da informação adotados pela empresa, pois o problema localiza-se em um nível intangível a esta, pois seria a segurança do software. De acordo Rangel (2003), o custo relativo para correção de um erro gira em torno de duzentas vezes maior na fase de manutenção que na fase de levantamento de requisitos, isto devido a muitos erros passarem despercebidos para estágios mais avançados de desenvolvimento, aumentando este custo relativo cada vez mais que problema perdura.

Com isto, aumenta-se a necessidade de adoção de padrões, metodologias e políticas devidamente testadas e consolidadas. São requisitos como esses para obter-se notoriedade e confiança do mercado de desenvolvimento de software, que é um nicho carente de mão de obra especializada e

ISBN 978-85-62830-10-5

em constante crescimento. Segundo a ABES (2011), o mercado de desenvolvimento, produção e distribuição de software é explorado por cerca de 8.520 empresas, sendo 94% classificadas como micro e pequenas empresas.

Em meados de 2011, os autores deste artigo desenvolveram um sistema para o Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco, no qual foi continuado e finalizado em 2012, intitulado de Programa Monitoria[1] e encontra-se em plena operação desde o edital de seleção de monitoria 2012.2, para cursos superiores e técnicos do campus Recife. O intuito era que este sistema fosse desenvolvido seguindo todas as etapas utilizando conceitos de Engenharia de Software, ou seja, levantamento de requisitos, casos de uso, plano de testes, diagramas de classes, até a implantação do mesmo, na intenção de que este fosse idealizado como modelo a ser replicado em sistemas futuros da instituição.

Contudo, este artigo tem como objetivo elucidar um dos requisitos não funcionais relacionado a segurança deste sistema, o *log* de acesso a chamada de métodos, por meio de POA utilizando uma sintaxe alternativa conhecida como *Annotations* utilizando marcações predefinidas como *interfaces*, promovendo uma maior segurança e descentralização desta responsabilidade, que normalmente é atribuída a todos os desenvolvedores da equipe.

2. MATERIAIS E MÉTODOS

2.1 Segurança da Informação

A segurança da informação em uma organização surge da necessidade da construção de produtos seguros. Segundo Howard e Leblanc (2005) "um produto seguro é um produto que protege a confidencialidade, integridade e disponibilidade das informações dos clientes e a integridade e disponibilidade dos recursos de processamento sob controle do dono ou administrador do sistema".

A ISO/IEC 27002 (2005), afirma que a segurança da informação é importante para os negócios, tanto do setor público quanto o privado, e para proteger as infraestruturas críticas. A interconexão de redes públicas e privadas e o compartilhamento de recursos de informação aumentam a dificuldade de se controlar o acesso.

Campos (2007) defende que um sistema de segurança baseia-se em três princípios básicos: Confidencialidade, Integridade e Disponibilidade. Andress (2011) corrobora afirmando que esses princípios são comumente referidos como tríade CIA (do inglês *Confidentiality*, *Integrity* e *Availability*). A tríade CIA nos dá um modelo, que possibilita pensar e discutir sobre conceitos de segurança no que diz respeito aos dados.

2.2 Programação Orientada a Aspectos

Comumente, as linguagens tradicionais de programação articulam em sintonia com um projeto de *software*, até onde estas requeiram apenas abstrações e mecanismos de composição conhecidos e suportados pelos tipos descritos no projeto, ou seja, sub-rotinas, procedimentos, funções, classes, APIs e *Frameworks*. Entretanto, esta abordagem pode ser insuficiente dado a necessidade de um determinado projeto em propriedades não tradicionais, como tratamento de exceções, controle de concorrência, etc. Onde estas na maioria das vezes estão distribuídas em diversos componentes do sistema afetando o desempenho da aplicação. (PIVETA, 2001)

Rotinas de rastreamento de ações de usuários e controle de sessões também são tidas como trabalhosas de serem implementadas e fortemente suscetíveis "quebras", visto que a menor alteração em sua lógica repercute na necessidade de revisão ou modificação de parte do código das entidades que a referenciam, além de tornar o código muitas vezes de difícil compreensão, isto por que as funcionalidades pertinentes as regras do negócio se misturam com outras referentes a segurança e tratamento de exceções, por exemplo.

[1] Disponível em: <http://monitoria.recife.ifpe.edu.br>, Acesso em: 06 de Agosto de 2012.

ISBN 978-85-62830-10-5

Segundo Piveta (2001), Programação Orientada a Aspectos (POA) é uma abordagem que permite a separação dessas propriedades, tidas como ortogonais, dos componentes funcionais do sistema de forma natural e concisa, através de abstrações para representação desses aspectos, para em seguida "combiná-los" de forma transparente para o desenvolvedor, resultando em um executável.

Entretanto, a POA não trabalha de forma isolada, é um paradigma que complementa outros paradigmas como a Programação Orientada a Objetos (POO), abstraindo elementos não inerentes ao negócio, a fim de facilitar uma rotina onerosa para o desenvolvedor que seria suscetível a erros, e promover diminuição do acoplamento entre os interesses e os componentes. Ou seja, boa parte do código é desenvolvida em linguagens tradicionais como Java e C++ e o restante são escritas em linguagem "aspectual". De maneira geral, o emprego deste modelo de programação propicia uma implementação mais ágil dos interesses sistêmicos (requisitos não funcionais) existentes no sistema, que na maioria das vezes estão espalhadas ao longo dos componentes que o integram, onde esses interesses (*concerns*) são centralizadas em entidades, promovendo uma separação mais clara. (RESENDE & SILVA, 2005)

A Figura 1 ilustra a separação dos conceitos apresentados em três camadas: interesses, componentes e código emaranhado (*tangled code*). A camada superior se refere os interesses, onde cada nuvem representa um requisito não funcional a ser mesclado no sistema, por exemplo, uma rotina de *log*, tratamento de exceções, controle de concorrência ou uma conexão ao banco de dados. Na camada inferior são apresentados os componentes do sistema e o relacionamento entre eles, responsáveis pelos negócios do sistema e escrito em linguagem tradicional. A camada central representa o código emaranhado, resultado da combinação (*aspect weaver*) entre as outras camadas.

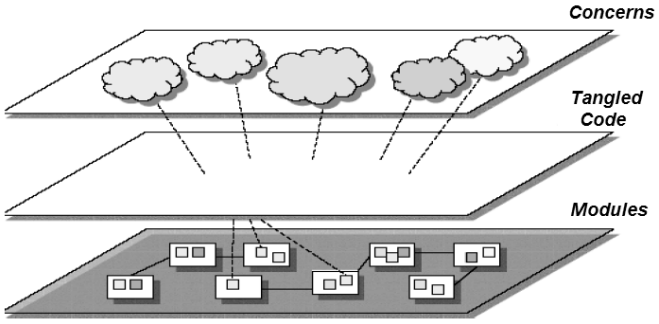

Figura 1 - Representação dos interesses ortogonais e *tangled code.*
Fonte: Winck e Junior, 2006.

De acordo com Piveta (2001), uma implementação baseada no paradigma de Programação Orientada a Aspectos é composta normalmente por: uma linguagem de componente, uma ou mais linguagens de aspectos, um combinador de aspectos (*Aspect Weaver*), um programa escrito na linguagem de componentes, um ou mais programas escritos na linguagem de aspectos.

2.2.1 AspectJ - Principais elementos

Dado que o AspectJ é uma extensão da linguagem Java, Laddad (2003) apresenta alguns elementos importantes da sintaxe para serem destacados, que tem como objetivo manipular os elementos transversais do paradigma:

a. *Joinpoint*: É a forma abstrata para representar um ponto identificável de um programa, como a chamada ou execução de métodos e construtores, acesso a atributos, inicialização de objetos, execução de inicialização estática e tratamento de exceções.

b. *Pointcut*: Tem como finalidade selecionar os *joinpoints* e coletar o contexto, como por exemplo os argumentos passados pelos mesmos. Contudo, os *pointcuts* não se restringem a captura de execução de métodos, pode conter os "designadores" de chamada de métodos, execução e chamada de construtores, leitura e escrita de atributos, dentre outros. No AspectJ

ISBN 978-85-62830-10-5

ainda são utilizadas algumas *wildcards(..,*,+)* e operadores(!,||,&&), que auxiliam a construção dos *pointcuts*, afim de capturar *pointcuts* que partilham características similares. (PIVETA,2001)

c: *Advice*: São trechos de códigos a serem executados quando um *joinpoint* for selecionado por um *pointcut.* Esses trechos podem ser chamados antes (*before*), depois (*after*) ou durante (*around*) a execução do *pointcut.* Onde destaca-se o momento *around*, que pode modificar e até mesmo substituir a execução de um determinado *joinpoint.* Por meio de *advices* é possível, por exemplo, enviar ao usuário uma mensagem depois que um método for executado.

d.*Aspect*: O *aspect* é a unidade central de AspectJ, da mesma forma que as classes são para a linguagem Java. Ele contém as regras expressas e elementos, ou seja, encapsula os *pointcuts* e *advices*, que serão mesclados ao longo do sistema. Pelo fato do AspectJ ser uma extensão da linguagem Java, nele é possível utilizar normalmente métodos e atributos, bem como é numa classe Java.

2.2.2 AspectJ - Sintaxe alternativa (Annotations)

Informalmente conhecida como "@AspectJ", este modelo suporta o desenvolvimento baseado em anotações para POA em Java. Esta modalidade permite a declaração de *joinpoints* ao longo do código de componentes através de *Annotations*, funcionando como classes de marcadores. Contudo, qualquer que seja o estilo de desenvolvimento adotado, o combinador (*weaver*) de AspectJ garante o mesmo funcionamento e semântica, que o estilo tradicional de desenvolvimento, sendo permitido inclusive a mistura das duas modalidades no mesmo arquivo. (ECLIPSE, 2006)

Serão apresentadas abaixo a relação de equivalências entre as modalidades citadas para os conceitos de *joinpoints, pointcuts, advices* e *aspects.*(ECLIPSE, 2006)

***a. Joinpoint*:** Diferentemente do modelo apresentado anteriormente, onde os *joinpoints* eram tratados de forma abstrata, neste agora apresentado se faz necessária uma declaração literal. Para qualquer tipo de elemento anotado (classe, método, construtor, pacote, etc), um padrão de anotação pode ser usado para corresponder a um conjunto de anotações, como por exemplo uma classe inteira da linguagem de componente. Esse tipo de anotação possui duas formas básicas:

- @<qualified-name>, por exemplo, **@Foo** ou **@org.xyz.Foo.**
- @(<type-pattern>), por exemplo, **@(org.xyz .. *)** ou **@ (Foo | | Boo)**

b. *Pointcut:* De maneira geral, um método anotado como *pointcut* não deve possuir corpo e não deve levantar cláusulas *throws.* Contudo, é possível utilizar parâmetros ou argumentos de ligação. As *Wildcards* mencionadas anteriormente também podem ser utilizadas para a construção dos *pointcuts* baseados em anotações;

c. *Advices*: Os *advices* baseados neste estilo, também possuem trechos de código escritos em linguagem Java tradicional e são representados pelas cláusulas *Before, After, "AfterReturning", "AfterThrowing" e Around*, onde os métodos marcados por estes devem ser declarados como públicos. Excepcionalmente no caso do *advice* do tipo *Around*, o retorno do método deverá ser void. A diferença entre os tipos *"AfterReturning", "AfterThrowing" e After*", é que o primeiro é chamado caso o *pointcut* que o mesmo indicou resultou em sucesso, o segundo é levantado caso ocorra alguma exceção no *pointcut* indicado e o terceiro chamado em todos os casos.

ISBN 978-85-62830-10-5

d. *Aspect:* A Figura 2 ilustra a estrutura básica de um aspecto construído utilizando o modelo baseado em anotações. Podemos observar que sua forma é similar a uma classe comum em Java, contudo possui os marcadores @{Aspect, Pointcut, Before}, que o torna característico para a sintaxe @AspectJ.

3. RESULTADOS E DISCUSSÃO

O objeto de estudo desta pesquisa foi realizado em um dos requisitos não funcionais relativo a segurança do sistema Programa Monitoria do IFPE, campus Recife. Onde consistiu em modelar um procedimento, que auxiliasse no processo de identificação de ações realizadas pelos usuários do sistema, de forma que isto auxiliasse o requisito de segurança do mesmo.

3.1 Ambiente de desenvolvimento

De acordo com Laddad (2003), AspectJ oferece suporte integrado para edição, compilação, execução e depuração em diversos IDEs populares, como Eclipse, Netbeans e JBuilder. Contudo, esta informação está desatualizada, pois o módulo de extensão, também conhecido como *plugin*, *AspectJ Development Environment* (AJDE) para NetBeans não é atualizada desde versão 3.5 da IDE[2], limitando o desenvolvimento apenas à nível de experiência, visto que esta versão está obsoleta.

Para alcançar o objetivo, que era elaboração do modelo proposto, foi necessário realizar a preparação física do ambiente de desenvolvimento e testes. Como foi definido a utilização da IDE NetBeans no plano de projeto do sistema, devido sua licença gratuita para fins não-comerciais e inúmeros recursos disponíveis, utilizou-se a versão 7.0.1. Foi necessário configurá-lo manualmente, para que o mesmo trabalhasse com a versão mais atual do AspectJ, visto que o *plugin* AJDE. Contudo, foi elaborado um tutorial (disponibilizado como projeto no *GoogleCode*[3]), no qual é demonstrado o passo-a-passo desta configuração e onde obter as ferramentas necessárias, como compilador AspectJ e suas bibliotecas, bem como um pequeno projeto *Web* previamente configurado.

3.2 Modelo Proposto

Por se tratar de um requisito não funcional, vislumbrou-se a utilização do paradigma de Programação Orientado a Aspectos devido a sua praticidade na implementação de requisitos sistêmicos, que possivelmente estão espalhados na maioria dos módulos.

O modelo proposto consistia na criação de uma interface que descrevia cada operação básica do sistema, ou seja, métodos de que envolviam consultas, criação, alteração e exclusão de registros. Para garantir uma maior padronização, essas interfaces possuíam a letra "I" seguido da sua funcionalidade, lembrando que o corpo deverá ser vazio.

Com as interfaces bem definidas, as regras referentes ao procedimento de *log* de ações, puderam ser implementadas num aspecto, descrito de acordo com a estrutura do trecho de código apresentado na Figura 2. Onde, **{...}** Significa trechos de código omitidos, relacionados às regras do sistema.

[2] Disponível em: <http://aspectj-netbeans.sourceforge.net>. Acesso em: 06 de Agosto de 2012.

[3] Disponível em: <http://code.google.com/p/projeto-netbeans-poa>. Acesso em: 06 de Agosto de 2012.

ISBN 978-85-62830-10-5

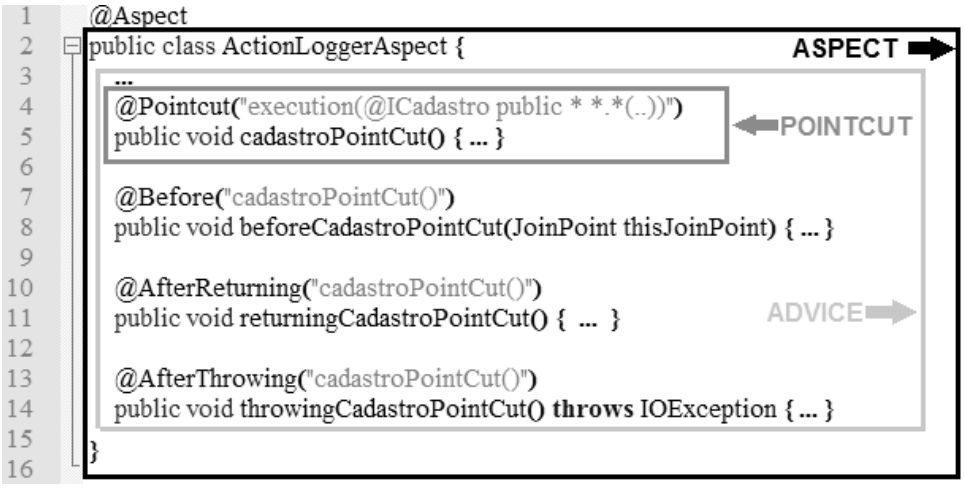

Figura 2 – Exemplo do modelo de anotação para aspectos
Fonte: Os autores

O *Advice* presente no aspecto implementado no trecho de código da Figura 2 resulta na "interceptação" dos métodos dos componentes do sistema, marcados pela interface "ICadastro", representado pelo *Pointcut*. Na cláusula *Before*, é realizada uma coleta das informações do usuário contidas na sessão antes da execução do método, tais como ID do usuário, IP do computador, data e outras, sendo armazenadas em variáveis temporárias. Caso o método executado pelo usuário termine em sucesso, então é chamada a cláusula *AfterReturning*, que é responsável por inserir no banco de dados a ação do usuário e as informações pertinentes a este. Caso o método resulte em falha por qualquer motivo que seja, então é chamada a cláusula *AfterThrowing* é levantada e inserida as informações no banco.

Em seguida foram marcados todos os principais os métodos dos controladores, que realizavam as regras de negócio, relacionando a funcionalidade com a respectiva interface definida. A marcação foi realizada da maneira descrita no trecho de código da Figura 3, que especificamente resultava no *log* referente ao cadastro de um candidato no sistema.

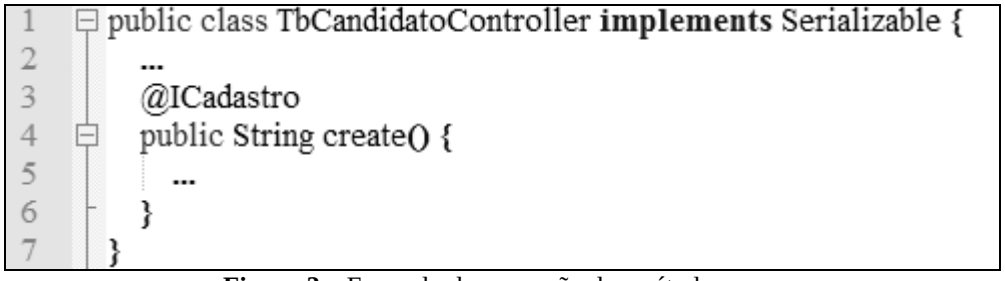

Figura 3 – Exemplo de marcação dos métodos
Fonte: Os autores.

Para realizarmos os testes do modelo implementado, foi criado um perfil no sistema do tipo Professor e realizado diversas operações, tais *login* no sistema, consulta de disciplinas, cadastro de plano de monitoria e simulado uma situação de falha.

ISBN 978-85-62830-10-5

Log Nº	Usuário	Tipo Ação	Método/Invoke	Resultado	Hora/Data	Endereço IP
3	Maria José de Almeida	LOGIN	execution(public java.lang.String controller.LoginController.ValidaUsuario())	SUCESSO	19:54:24 08/06/2012	127.0.0.1
4	Maria José de Almeida	CONSULTA	execution(public void controller.TbComponenteCurricularController.buscarPorDepartamento())	SUCESSO	19:54:52 08/06/2012	127.0.0.1
5	Maria José de Almeida	CADASTRO	execution(public java.lang.String controller.TbPlanoMonitoriaController.create())	SUCESSO	19:55:20 08/06/2012	127.0.0.1
6	Maria José de Almeida	ALTERACAO	execution(public java.lang.String controller.TbOrientadorController.update())	SUCESSO	19:56:08 08/06/2012	127.0.0.1
7	Maria José de Almeida	LOGIN	execution(public java.lang.String controller.LoginController.ValidaUsuario())	SUCESSO	19:56:40 08/06/2012	127.0.0.1
8	Maria José de Almeida	ACESSO SUSPEITO	execution(public java.lang.String controller.TbDepartamentoController.create())	FALHA	19:56:47 08/06/2012	127.0.0.1
9	Administrador	LOGIN	execution(public java.lang.String controller.LoginController.ValidaUsuario())	SUCESSO	19:57:06 08/06/2012	127.0.0.1

Sair

Figura 4 - Tabela de log do sistema
Fonte: Os autores.

O resultado dos testes gerou uma lista de registro de *log* na base de dados do sistema, referente as ações realizadas pelo usuário "Maria José", conforme pode ser visto na Figura 4.

Em seguida, foi realizada uma coleta desses *logs* do sistema, com ele em produção, durante cinquenta e dois dias, onde podemos observar os detalhes dos acessos reais, num período de maior tráfego de operações do edital de seleção de monitoria 2012.1.

A Figura 5 representa esses dados sumarizados, em formato de tabelas e gráfico.

SUPERIOR				
AÇÃO	SUCESSO	FALHA	QTD	%
LOGIN	349	0	349	20.02
CONSULTA	1014	0	1014	58.18
CADASTRO	216	0	216	12.39
ALTERAÇÃO	74	0	74	4.25
ACESSO SUSPEITO	0	85	85	4.88
EXCLUSÃO	5	0	5	0.29
TOTAL	1658	85	1743	100.00
MÉDIA DE ACESSOS POR DIA				33.52

TÉCNICO				
AÇÃO	SUCESSO	FALHA	QTD	%
LOGIN	122	0	122	13.17
CONSULTA	370	0	370	39.96
CADASTRO	205	0	205	22.14
ALTERAÇÃO	73	0	73	7.88
ACESSO SUSPEITO	0	146	146	15.77
EXCLUSÃO	10	0	10	1.08
TOTAL	780	146	926	100.00
MÉDIA DE ACESSOS POR DIA				17.81

GERAL				
AÇÃO	SUCESSO	FALHA	QTD	%
LOGIN	471	0	471	17.65
CONSULTA	1384	0	1384	51.85
CADASTRO	421	0	421	15.77
ALTERAÇÃO	147	0	147	5.51
ACESSO SUSPEITO	0	231	231	8.65
EXCLUSÃO	15	0	15	0.56
TOTAL	2438	231	2669	100.00
MÉDIA DE ACESSOS POR DIA				51.33

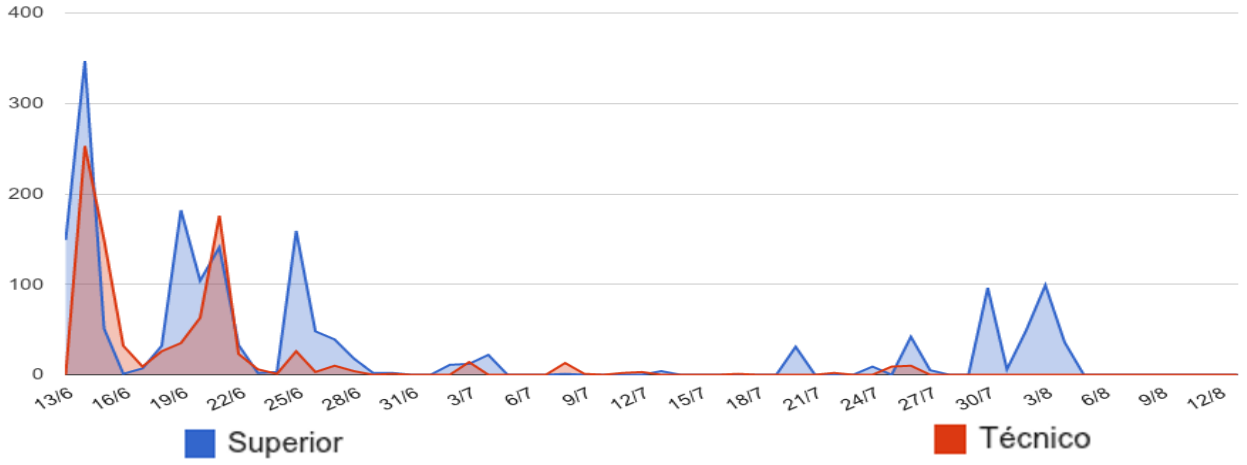

Figura 5 - Estatística entre os dias 13/06/2012 e 03/08/2012
Fonte: Os autores, 2012.

4. CONCLUSÕES

ISBN 978-85-62830-10-5

A Programação Orientada a Aspectos agrega valores inovadores no desenvolvimento do sistema, dentre seus benefícios, pode-se destacar a modularização de requisitos não funcionais do programa, uma vez que a partir do momento em que ela possui a capacidade de agrupar vários interesses em um aspecto, dividindo o problema em partes menores, ela vem a colaborar significativamente no que diz respeito a modularização, trazendo benefícios para o trabalho em equipe, visto que não é mais necessário que os especialistas na parte de segurança, tratamento de exceções, dentre outros, conheçam todo o código referente ao negócio do sistema.

Isto contribui diretamente no que diz respeito à redução do espalhamento e o entrelaçamento de código-fonte por vários componentes, assim, reduzindo a complexidade de tal código-fonte, tornando-o mais simples, de fácil entendimento e consequentemente, facilitando também em sua manutenção e reusabilidade, visto que os interesses ortogonais do sistema estão centralizados em aspectos bem definidos. Contudo, a linguagem AspectJ possibilita a construções de regras bem abrangentes, que devem ser escritas com bastante cautela, pois podem atingir áreas indesejadas do sistema, principalmente ao se utilizar as *Wildcards*.

O sistema desenvolvido, nomeado como Programa Monitoria, foi implementado com este paradigma, proporcionando todos os benefícios relacionados ao mesmo. O modelo de log de ações, comum nos sistemas *Web*, possui muito código entrelaçado e de difícil manutenção com as mudanças de requisitos inerentes ao desenvolvimento de *software*.

Entretanto, após bem definidas a estrutura dos interesses e o respectivo Aspecto, as regras foram implementadas com facilidade utilizando *Annotations* por meio de interfaces, que serviram para relacionar o método chamado a uma ação "reconhecida" pelo sistema. O sistema está integrado ao da instituição de ensino e em produção, sendo aprovado tanto pelo corpo docente como pelo discente, encorajando a prática nas fábricas de *software* e nas salas de aula, formando um profissional mais qualificado e preparado para o mercado de trabalho.

Como trabalhos futuros, espera-se construir uma ferramenta mais robusta de análise desses *logs*, de forma que seja possível sumarizar as ações dos usuários, coordenações e departamentos afim de proporcionar um *feedback* visivelmente mais amigável, que o sistema de *log* atual.

REFERÊNCIAS

ABES – **Associação Brasileira das Empresas de Software, Mercado Brasileiro de Software.** (2001). Recuperado em 28, julho, 2012, de http://www.abes.org.br/UserFiles/Image/PDFs/Mercado_BR2011.pdf

ANDRESS, Jason. (2011). **The Basics Of Information Security – Understanding the Fundamentals of InfoSec In Theory and Pratice**. Ed. Elsevier: USA.

CAMPOS, André L. N. (2007). **Sistema de Segurança da Informação.** Florianópolis: Visual Books.

ECLIPSE. (2006). **The AspectJ 5 Development Kit Developer's Notebook, Eclipse Foundation** – Recuperado em 31, julho, 2012, de http://www.eclipse.org/aspectj/doc/released/adk15notebook/index.html

LADDAD, R. (2003). **AspectJ in Action - Pratical Aspect-Oriented Programming**. Greenwich, Manning.

MARCIANO, João Luiz Pereira. (2006). **Segurança da Informação - Uma Abordagem Social.** Tese de Doutorado – UnB.

ISBN 978-85-62830-10-5
VII CONNEPI©2012

PIVETA, E. K. (2001). **Um modelo de suporte à programação orientada a aspectos**. Dissertação de Mestrado – UFSC.

RANGEL, G. S. (2003). **Uma Ferramenta de Prototipação de Software para o Ambiente PROSOFT**. Dissertação de Mestrado. UFRGS, Brasil.

RESENDE, A. M. P. e SILVA, C. C. da. (2005) . **Programação orientada a aspectos em Java**. São Paulo, Brasport.

SILVA, Pedro Tavares. CARVALHO, Hugo. TORRES, Carina Botelho. (2003). **Segurança dos Sistemas de Informação - Gestão Estratégica da Segurança Empresarial**, Ed. Lisboa: Centro Atlântico.

WINCK, D. V.; JÚNIOR, V. G. (2006). **AspectJ - Programação orientada a aspectos com Java**. São Paulo, Novatec.

ISBN 978-85-62830-10-5