UNIVERSIDADE FEDERAL DO RIO GRANDE
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

NICOLAS BENTO

Framework AOP utilizando técnicas de Byte Code Engineering

Trabalho de Conclusão apresentado como requisito parcial para a obtenção do grau de Tecnólogo em Análise e Desenvolvimento de Sistemas

Prof. Márcio Torres
Orientador

Rio Grande, fevereiro de 2014

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Bento, Nicolas

Framework AOP utilizando técnicas de Byte Code Engineering / Nicolas Bento. – Rio Grande: TADS/FURG, 2014.

39 f.: il.

Trabalho de Conclusão de Curso (tecnólogo) – Universidade Federal do Rio Grande. Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Rio Grande, BR–RS, 2014. Orientador: Márcio Torres.

1. AOP, Byte Code Engineering, Meta-programação, Framework, Soc. I. Torres, Márcio. II. Título.

FOLHA DE APROVAÇÃO

Monografia sob o título *"Framework AOP utilizando técnicas de Byte Code Engineering"*, defendida por Nicolas Dias Bento e aprovada em ?? de ?? de ???, em Rio Grande, estado do Rio Grande do Sul, pela banca examinadora constituída pelos professores:

Prof. Márcio Torres
Orientador

Prof. NOME
IFRS - Campus Rio Grande

Prof. NOME
IFRS - Campus Rio Grande

"A mente que se abre a uma nova ideia jamais volta ao seu tamanho original."
— ALBERT EINSTEIN

AGRADECIMENTOS

Agradecimentos ...

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AOP	Programação Orientada a Aspectos (*Aspect Oriented Programming*)
SoC	Separação de interesses (*Separation of concerns*)
OOP	Programação Orientada a Objetos (*Object Oriented Programming*)
PARC	Centro de Pesquisa Palo Alto (*Palo Alto Research Center*)
SQL	Linguagem de Consulta Estruturada (*Structured Query Language*)
YAGNI	Você não vai precisar dele (*You aren't gonna need it*)
XP	Programação Extrema (*Extreme Programming*)

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Resumo ...

Palavras-chave: AOP, Byte Code Engineering, Meta-programação, Framework, Soc.

1 INTRODUÇÃO

2 ASPECT ORIENTED PROGRAMMING

Neste capítulo será abordado de forma geral o paradigma de programação orientada a aspectos, trazendo de forma objetiva as informações necessárias para a superficial compreensão desta abordagem.

2.1 O que é?

AOP é um paradigma de programação que foi contruído tomando como base outros paradigmas(OOP e *procedural programming*), cujo o principal objetivo seria a modularização de interesses transversais, utilizando um dos paradigmas base na implementação dos interesses centrais. A forma como AOP e o paradigma base se integram se dá com a utilização de aspectos que determinam a forma como os diferentes módulos se relacionam entre si na formação do sistema final. (LADDAD, 2003)

2.2 História

Após um grande período de estudos, pesquisadores chegaram a conclusão que para desenvolver um software de qualidade era fundamental separar os interesses do sistema, ou seja, deveria então ser aplicado o princípio de *Separation of Concerns* (SoC)[1]. Em 1972, David Parnas escreveu um artigo, que tinha como proposta aplicar SoC através de um processo de modularização, onde cada módulo deveria esconder as suas decisões de outros módulos.Passado alguns anos, pesquisadores continuaram a estudar diversas formas de separação de interesses. OOP foi a melhor, se tratando de separação de interesses centrais, mas quando se tratava de interesses transversais, acabava deixando a desejar. Diversas metodologias— *generative programming, meta-programming, reflective programming, compositional filtering, adaptive programming, subject-oriented programming, aspect oriented programming,* e *intentional programming*— surgiram como possíveis abordagens para modularização de interesses transversais. AOP acabou se tornando a mais popular entre elas. (LADDAD, 2003)

Em 1997 Gregor Kiczales e sua equipe descreveram de forma sólida o conceito de Programação Orientada a Aspectos, durante um trabalho de pesquisa realizado pelo PARC, uma subsidiária da *Xerox Corporation*. O documento descreve uma solução complementar a OOP, ou seja, seriam utilizados "aspectos", que iriam encapsular as preocupações transversais, de forma a garantir a reutilização por outros módulos de um sistema.Sugeriu também diversas implementações de AOP, servindo como base para a criação do AspectJ[2],

[1]Para saber mais sobre SoC consulte o Glossário.

[2]No final dos anos 90, a Xerox Corporation, transferiu o projeto AspectJ para a comunidade Open

uma linguagem AOP muito difundida nos dias de hoje.(GROVES, 2013)

2.3 Principais conceitos

Para entender melhor o funcionamento de AOP, é preciso compreender seus principais conceitos.

Os 4 principais conceitos de AOP são:

- Aspecto (*aspect*) - pode ser definido como um interesse transversal, ou seja, interesse onde sua implementação é espalhada por diversos módulos ou componentes de um sistema.
- Ponto de junção (*joinpoint*) - podem ser definidos basicamente como pontos bem definidos na execução de um programa.
- Ponto de corte (*pointcut*) - em AOP um *pointcut* pode ser representado como um agrupamento de pontos de junção.
- Conselho (*advice*) - é a implementação de determinado interesse transversal, sendo executado quando determinado *pointcut* é ativado, podendo ser executado antes(*before*), durante (*around*) ou depois (*after*) da execução de um ponto de junção.

Podemos também comparar os conceitos de AOP com os principais conceitos de OOP:

Tabela 2.1: Comparação entre conceitos de AOP e OOP

AOP	**OOP**
Aspect	Classe
Advice	Método
Joinpoint	Atributo

Pointcut não se encaixa em nenhum conceito de orientação a objetos, mas podemos comparar um *pointcut* com um gatilho(*trigger*) da linguagem SQL (*Structured Query Language*).

2.4 Benefícios de AOP

Em programação podemos dizer que cada paradigma tem seus prós e contras, sendo assim dificilmente encontraremos uma metodologia que resolva todos os nossos problemas da melhor maneira.Com AOP não é diferente, mas devemos considerar sua larga escala de benefícios:

- Separação de interesses e Alta modularização - com AOP podemos separar o projeto em módulos responsáveis por implementar apenas os interesses centrais do sistema, possuindo assim um acoplamento mínimo que extingui o codigo duplicado, fazendo com o sistema fique muito mais fácil de entender e manter, colocando os interesses transversais (aspectos) em módulos completamente diferentes.(LADDAD, 2003)

Source em eclipse.org.

- Fácil evolução - utilizando AOP o sistema fica muito mais flexível quando se trata da adição de novas funcionalidades, fazendo com que a resposta às exigências se tornem mais rápidas.

- Foco na prioridade - o arquiteto do projeto pode se concentar nos requisitos básicos atuais do sistema, os novos requisitos que abordam interesses transverssais poderão ser tatados fácilmente com a criação de novos aspectos. AOP trabalha em harmonia com metodos agéis, por exemplo apóia a prática YAGNI(*"You aren't gonna need it"*) [3], do *Extreme Programming*(XP)[4].(LADDAD, 2003)

- Maior reutilização de código - a chave para a reutização de código definitiva é uma implementação flexível, com AOP é possível pois cada aspecto é implementado como um módulo distinto, se tornando adaptavél a implementações equivalentes convencionais.(LADDAD, 2003)

- Aumento na produtividade - o ciclo do projeto se torna mais rápido, a grande reutilização usada em AOP, faz com que tempo de desenvolvimento seja reduzido, diminuindo também o tempo de implantação e o tempo de resposta às novas exigências de mercado. (LADDAD, 2003)

- Redução de custos e aumento da qualidade - com a implementação dos interesses transversais em módulos distintos, o custo de implementação cai bastante, fazendo com que o desenvolvedor concetre-se mais nos interesses centrais, criando um produto de qualidade e com o custo reduzido.

[3] Em português significa "Você não vai precisar dele". Para saber mais sobre YAGNI consulte o Glossário.

[4] É um dos metodos agéis mais utilizados na atualidade.

3 BYTE CODE ENGINEERING

4 FRAMEWORK QUE SERÁ USADO

5 META-PROGRAMAÇÃO

5.1 Anotações

5.2 Reflexão

6 SOLUÇÕES EXISTENTES

6.1 PostSharp

6.2 AspectJ

7 O FRAMEWORK

7.1 Análise e projeto

7.2 Implementação

8 ESTUDO DE CASO - INSTRUMENTAÇÃO

9 CONCLUSÃO

Conclusões ...

REFERÊNCIAS

BECK, K.; ANDRES, C. **Extreme programming explained**: embrace change. [S.l.]: Addison-Wesley Professional, 2004.

GROVES, M. D. **AOP in. NET**. [S.l.]: Manning Publ., 2013.

LADDAD, R. **AspectJ in action: practical aspect-oriented programming**. [S.l.]: Manning, 2003.

PRESSMAN, R. **Software engineering: a practitioner's approac**. [S.l.]: McGraw Hill, 2010.

GLOSSÁRIO

SoC è um princípio de projeto, criado com a finalidade de subdividir o problema em conjuntos de interesses tornando a resolução do problema mais fácil.Cada interesse fornece uma funcionalidade distinta, podendo ser validado independentemente das regras negócio.(PRESSMAN, 2010)

YAGNI é um princípio de projeto, bastante usado em equipes XP , cuja principal finalidade é implementar apenas o necessário, ou seja, a melhor maneira de implementar o código rapidamente é implementar menos.

Extreme Programming XP é um estilo de desenvolvimento de software com em excelentes técnicas de programação, comunicação clara e trabalho em equipe, que permite grande produtividade no desenvolvimento. (BECK; ANDRES, 2004)