2
Fundamentação

Apresentamos inicialmente uma análise de problemas de código orientado a objetos. Essa análise, apesar de não poder ser aproveitada diretamente em códigos declarativos como a NCL, deve servir de inspiração para a criação de uma taxonomia de problemas. Posteriormente, discutiremos sobre possíveis abordagens para se escrever regras de inspeção em linguagens baseadas em XML e também seus prós e contras.

2.1
Problemas de qualidade de código

Martin Fowler e Kent Beck deram uma importante contribuição para o assunto, catalogando diversos problemas de código orientado a objetos (Fowler et al. 1999), denominados por eles como "maus cheiros" (*Bad Smells*). Além disso, esse catálogo contém possíveis soluções para cada mau cheiro, chamadas de refatoração (*Refactoring*).

Definição 2.1 *Refatoração é uma modificação feita na estrutura interna de um software para torná-lo mais fácil de se entender e mais barato de se modificar sem alterar seu comportamento externo. – Martin Fowler (Fowler et al. 1999) p. 53*

Podemos entender os maus cheiros como uma metáfora normalmente usada para descrever projetos de software ruins e más práticas de programação. Os maus cheiros normalmente podem ser vistos como uma indicação de problemas de código que podem ser resolvidos por meio da refatoração.

Para cada mau cheiro descrito na literatura (Fowler et al. 1999), existe um detalhamento de quais refatorações podem ser aplicadas a fim de resolvê-los. Porém, essas refatorações não serão apresentadas por não fazerem parte do escopo deste trabalho. Muitos ambientes de desenvolvimento modernos oferecem refatorações automáticas como uma de suas funcionalidades.

Os maus cheiros enumerados a seguir foram os identificados em (Fowler et al. 1999). Quando necessário, há uma explicação do problema.

- Código Duplicado (*Duplicated Code*)
- Método Longo (*Long Method*)
- Classe Grande (*Large Class*)
- Lista de Parâmetros Longa (*Long Parameter List*)
- Modificação Divergente (*Divergent Changes*): ocorre quando uma classe precisa ser modificada para entrar em conformidade com alterações de naturezas diferentes no sistema, indicando um problema de coesão.
- Cirurgia com Espingarda (*Shotgun Surgery*): o oposto de Modificação Divergente. Ocorre quando diversas classes precisam ser modificadas para uma alteração de um aspecto do sistema.
- Inveja de Características (*Feature Envy*): ocorre quando, por exemplo, um método está mais interessado em campos que estão contidos em outra classe do que a qual ele próprio está contido.
- Aglomerados de Dados (*Data Clumps*): dados que aparecem sempre juntos repetidamente em diversos locais do código, devem ser colocados em seu próprio objeto.
- Obsessão por Primitivas (*Primitive Obsession*): iniciantes em orientação a objetos são relutantes em usar pequenos objetos para pequenas tarefas e preferem usar tipos primitivos, o que na maioria das vezes é uma decisão errada.
- Sentenças Switch (*Switch Statements*): o uso de *switch* diminui a manutenibilidade do sistema.
- Hierarquias de heranças paralelas (*Parallel Inheritance Hierarchies*): é um caso especial da Cirurgia com Espingarda. Ocorre quando ao criar uma subclasse é necessário também criar uma subclasse de outro tipo.
- Classe Preguiçosa (*Lazy Class*): uma classe que não vale a pena existir.
- Generalidade Especulativa (*Speculative Generality*): é o caso em que é criado um sistema genérico para acomodar supostas funcionalidades que não são requeridas.
- Campo temporário (*Temporary Field*): é o caso em que um objeto utiliza suas variáveis de instância em apenas algumas circunstâncias. Isso torna o código difícil de entender, pois é esperado que o objeto necessite de todas as suas variáveis.

- Cadeias de Mensagens (*Message Chains*): ocorre, por exemplo, em uma longa sequencia de `a.getXX().getZZ().getDadosNecessarios()`. Isso aumenta o acoplamento da classe cliente e causa um problema de manutenção caso a estrutura de relacionamentos seja modificada.
- Intermediário (*Middle Man*): esse mau cheiro ocorre quando há delegações de métodos desnecessárias.
- Intimidade Inapropriada (*Inappropriate Intimacy*): é chamado desta forma quando uma classe conhece muitos detalhes de outra. Pode acontecer, por exemplo, em heranças quando uma subclasse tem acesso a muitos detalhes da classe pai.
- Classe Incompleta de uma Biblioteca (*Incomplete Library Class*): Uma classe de uma biblioteca que não faz exatamente tudo que você deseja.
- Classes alternativas com diferentes interfaces (*Alternative Classes with Different Interfaces*)
- Classe de Dados (*Data Class*): uma classe que apenas contém campos, métodos `get` e `set` e mais nada.
- Herança Recusada (*Refused Bequest*): uma subclasse que utiliza poucos ou nenhum método ou dados herdados de seu pai.
- Comentários: quando todos os maus cheiros forem removidos através da refatoração, os comentários na maioria dos casos se tornarão supérfluos.

Apesar de não poderem ser aplicados à NCL, os Maus Cheiros de Fowler serviram de inspiração para catalogar problemas de NCL e também para inspirar a criação de um esquema taxonômico de problemas NCL, como discutido na Seção 6.1.1 – Esquema taxonômico baseado nas instâncias de problemas NCL .

2.2 Estendendo XML Schemas

Para aproveitar melhor o uso de tecnologias XML, que já possuem diversas ferramentas disponíveis e reduzir o esforço de implementação, abordaremos o problema de criar um sistema de críticas para NCL de outra forma. Podemos ver esse problema como o problema de estender um XML Schema, de modo que, este Schema se torne mais expressivo e possa representar, além de restrições sintáticas ou léxicas, também restrições semânticas.

Como exemplo, considere uma restrição semântica dos atributos `top`, `bottom` e `height` de `<region>` (o mesmo se aplica aos atributos `left`, `right` e `width`). Quando estes três atributos aparecem juntos no mesmo elemento `<region>`, podem causar inconsistências em relação ao real tamanho da `<region>`. Essa restrição não é possível de ser avaliada usando um XML Schema.

Para expressar essa restrição, as opções disponíveis, de nosso conhecimento, são: outra linguagem baseada no Schema, como por exemplo o Schematron; uma linguagem de programação de uso geral; Transformações em Folhas de Estilo XML (XSLT). Todas essas três opções possuem vantagens e desvantagens, descritas nas seções a seguir.

2.2.1 Linguagens baseadas no XML Schema

Existem diversas linguagens baseadas no XML Schema. Dentre as mais conhecidas estão: Schematron, TREX e RELAX NG. Essas linguagens foram desenvolvidas exclusivamente com o propósito de realizar validações e assertivas em documentos XML.

Sendo assim, o mais natural seria pensar que o NCL-Inspector utiliza uma, ou talvez um conjunto dessas linguagens, para expressar suas regras de inspeção de forma declarativa. Porém durante o projeto do NCL-Inspector optou-se por não utilizar nenhuma linguagem baseada no XML Schema. A justificativa desta não adoção se encontra nas Seções 2.2.2 e 2.2.3, e também nas características das linguagens baseadas no XML Schema, que serão apresentadas a seguir:

Documento único: um dos resultados obtidos ao usar o Schematron é ter apenas um documento contendo todas as informações relativas ao XML Schema. Isto a princípio pode parecer uma vantagem, em alguns casos pode até mesmo ser. Dessa forma, é possível ter um XML Schema com todas as informações e regras de validação em apenas um lugar, ao invés dessas informações estarem dispersadas em múltiplos documentos. Os padrões das assertivas são incluídas através do elemento `<appinfo>` do XML Schema.

Simplicidade: As linguagens baseadas foram criadas como reação às limitações e complexidades existentes no XML Schema. Portanto, a maioria dessas linguagens são relativamente simples e fáceis de usar.

Reduzido poder de expressão: Cada linguagem possui suas próprias capacidades e limitações. Com o Schematron não é possível expressar todas as restrições. Podemos tomar a validação do ISBN (International Standard Book Number) como um exemplo trivial onde o Schematron não possui poder suficiente para expressar esta restrição. O último dígito deve ser a soma de alguns outros dígitos e depois aplicado o módulo 11 nesta soma. A falta de variáveis e estruturas de repetição também reduzem de forma considerável o poder de expressão do Schematron, tornando difícil a criação de determinadas restrições como a mencionada anteriormente, por exemplo.

Curva de aprendizado: As linguagens baseadas no Schema são muito particulares, cada uma possui sua própria sintaxe e semântica, como é possível ser visto em (Lee et al. 2000). Eventualmente pode ser necessário encontrar uma linguagem (ou mais de uma) que permita expressar todas as restrições necessárias para o fim que se deseja dar. Mesmo as linguagens sendo simples em sua grande maioria, ainda sim é preciso gastar algum tempo de aprendizado.

Reduzido grupo de desenvolvedores: Com exceção do Schematron e algumas outras, a grande maioria das linguagens baseadas em Schema são desenvolvidas apenas por um único desenvolvedor. O risco de não haver mais suporte a essas linguagens em um curto prazo deve ser considerado.

Porém no caso do NCL-Inspector existem regras gerais, que poderiam ser especificadas através do Schematron, mas também existem regras específicas de determinadas aplicações ou alguma equipe de desenvolvimento como exemplificada na Seção 4.1.4. A adoção do Schematron inviabilizaria a criação dessas regras específicas, portanto, apesar das outras desvantagens mencionadas anteriormente, o documento único foi realmente determinante para a não adoção do Schematron pelo NCL-Inspector.

2.2.2 Linguagens de Programação

Esta opção consiste em utilizar uma linguagem de programação de propósito geral como Java, C/C++, Lua, etc. para expressar as restrições necessárias.

A vantagem desta abordagem é disponibilizar todo o poder de expressão disponível nessas linguagens. Se não for possível escrever uma restrição utilizando uma linguagem como essa, é impossível que outra linguagem o faça.

Porém, a solução utilizando essa abordagem pode ser mais custosa. É necessário realizar alguns passos a mais, como o passo de compilação e ligação por exemplo. Além disso, uma solução que utiliza uma linguagem de programação tende a ser mais prolixa. Essas duas características geram um esforço maior de implementação e posteriormente de manutenção.

A opção do NCL-Inspector de fornecer Java para criação de suas regras é devido ao grande poder de expressão das linguagens de programação de uso geral. Mesmo que todas as regras pudessem ser expressas através de XSL e Schematron, algumas delas podem ser mais fáceis de serem expressas através de uma linguagem de programação como Java, como é discutido na Seção 5.2.1.

2.2.3 Transformações em Folhas de Estilos XML (XSLT)

O XSLT foi a opção utilizada pelo NCL-Inspector para permitir que as regras sejam escritas de forma declarativa. As características do XSLT atendem de maneira satisfatória os objetivos estabelecidos para o projeto do NCL-Inspector:

Múltiplos documentos: Cada equipe de desenvolvimento pode criar seu conjunto de folhas de estilos e, com isso, estender o NCL-Inspector possibilitando a criação de novas regras sem intervenção no XML Schema. Cada regra pode ser implementada como uma folha de estilos, como pode ser visto na Subseção 4.3.2.

Tecnologia-chave: É uma tecnologia-chave em se tratando de desenvolvimento utilizando XML. Possui um bom suporte da comunidade de desenvolvedores, vasta documentação e um enorme número de livros publicados a respeito.

Alto poder de expressão: Devido ao seu ótimo mecanismo de casamento de padrões, provido pelo XPath, a maioria das regras de inspeção, ou talvez todas, podem ser expressas usando essa tecnologia. As regras podem ser declarativamente escritas usando a expressão `<template match="">`. Além disso, embora não seja encorajado, com XSLT é possível a utiliza-

ção de variáveis e estruturas de repetição, o que fornece praticamente o poder de uma linguagem de programação de uso geral.

Grande suporte de desenvolvedores: Se comparado às linguagens baseadas no XML Schema, o XSLT é suportado por uma grande comunidade de desenvolvedores. Além disso é um padrão W3C.