CURSO DE ENGENHARIA DE SOFTWARE

RELATÓRIO – TRABALHO FINAL QUALIDADE DE SOFTWARE

Canarinho

Equipe:

Eduardo Henrique Brito da Silva

537777

Francisco Jerônimo da Silva Júnior

433399

Professora:

Carla Ilane Moreira Bezerra

QUIXADÁ

Junho, 2022

SUMÁRIO

1 DESCRIÇÃO DO PROJETO.. 2

2 AVALIAÇÃO DO PROJETO.. 2

2.1 Medição 1 – Antes de refatorar o projeto. 2

2.2 Detecção dos Code Smells. 3

2.3 Medição 2 – Após Refatorar Code Smell Cyclically-dependent Modularization

2.4 Medição 3 – Após Refatorar Code Smell Unnecessary Abstraction

2.5 Medição 4 – Após Refatorar Code Smell Dead Code

2.6 Medição 5 – Após Refatorar Code Smell Magic Number

2.7 Medição 6 – Após Refatorar Code Smell Too Many Arguments

2.8 Medição 7 – Após a refatoração de todos os code smells do projeto

3 COMPARAÇÃO DOS RESULTADOS. 4

REFERÊNCIAS. 4

APÊNDICE A.. 5

1 DESCRIÇÃO DO PROJETO

Nessa Seção deve ser descrito o projeto. É importante incluir uma Tabela com as seguintes informações: Número de Linhas de Código (LOC), Número de Classes e Número de Releases. Incluir o link do github do projeto. Falar sobre a natureza do projeto, se é JAVA OO, se é de código aberto ou fechado, quais tecnologias utilizadas no projeto.

Link do projeto: https://github.com/concretesolutions/canarinho.git

Esta biblioteca é um conjunto de utilitários para trabalhar com padrões brasileiros no Android, por exemplo: validação e formatação de CPF, CNPJ e outros cadastros comuns no Brasil. Inspirado em: https://github.com/caelum/caelum-stella. O foco aqui é o Android. Portanto, não é compatível com aplicações Java puras.

Tabela 1 – Características do Projeto

Projeto	LOC	# de classes	# de releases
Canarinho	2555	62	2.0.2

2 AVALIAÇÃO DO PROJETO

2.1 Medição 1 – Antes de refatorar o projeto

Tabela 2 – Medição dos atributos antes de refatorar o projeto.

Sistema	Coesão	Complexidade				Herança			Acoplamento	Tamanho			
	LCOM2	ACC	SCC	EVG	MaxNet	DIT	NOC	IFANIN	CBO	LOC	CLOC	NIM	CDL
S1 antes da refatoração	1071	412	140	1573	184	288	34	96	90	10122	2231	134	739
S1 após refat. CS X													
S1 após refat. CS X													

2.2 Detecção dos Code Smells

Tabela 3 – Code smells do projeto.

Nome do Code Smell	Quantidade
Cyclically-dependent Modularization	1
Unnecessary Abstraction	12
Dead code	2
Magic Number	18
Too many arguments	7

2.3 Medição 2 – Após Refatorar Code Smell Cyclically-dependent Modularization

Esse Smell surge quando duas ou mais abstrações em nível de classe dependem umas das outras direta ou indiretamente (criando um forte acoplamento entre as abstrações). A possível solução para esse tipo de smell é:

1. Introduzir uma interface para uma das abstrações envolvidas no ciclo.
2. Caso uma das dependências seja desnecessária e possa ser removida com segurança, remova essa dependência. Por exemplo, aplique a refatoração “move method” (e “move field”) para mover o código que introduz dependência cíclica para uma das abstrações participantes
3. Mova o código que introduz a dependência cíclica para uma abstração completamente diferente.
4. Caso as abstrações envolvidas no ciclo representam um objeto semanticamente único, mescle as abstrações em uma única abstração.

A Solução aplicada foi a (3), criando uma nova classe fora do arquivo original e mudando a forma que é instanciado. Assim quebrando a dependência.

NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT
195	2554	395	3	3	875	57	57

2.4 Medição 3 – Após Refatorar Code Smell Unnecessary Abstraction

Removido Unnecessary Abstraction nos pacotes de Validator e Formatador. Nas classes do Validator apenas teve que ser removido as declarações e no Formatador em uma classe Formatador teve que ser removido o SingletonHeader e movido os atributos para parte inicial da classe.

NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT
195	2494	395	3	3	875	45	45

2.5 Medição 4 – Após Refatorar Code Smell Dead Code

Removido dead code no pacote Watcher na classe ValorMonetarioWatcher. O trecho do código não é utilizado em nenhuma parte do código, logo ele é devidamente removido.

NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT
193	2471	391	3	3	875	43	43

2.6 Medição 5 – Após Refatorar Code Smell Magic Number

Os Magic numbers foram analisados e refatorados principalmente na classe ValidadorBoleto.java do pacote Validador.
Um magic number é um valor numérico encontrado no código, mas sem significado óbvio, esse “antipadrão” torna mais difícil entender o programa e refatorar o código.

Ainda mais dificuldades surgem quando você precisa alterar esse número mágico. Localizar e substituir não funcionará para isso: o mesmo número pode ser usado para propósitos diferentes em lugares diferentes, o que significa que você terá que verificar cada linha de código que usa esse número.

NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT
193	2496	391	3	3	875	43	43

2.7 Medição 6 – Após Refatorar Code Smell Too Many Arguments

Os métodos com o code smell “Too Many Arguments” (também conhecido por outra denominação chamada Long Parameter List) foram analisados e refatorados principalmente na classe ValidadorBoleto.java do pacote Validador. O smell tem a característica de possuir mais de quatro parâmetros para um método. Uma longa lista de parâmetros pode acontecer depois que vários tipos de algoritmos são mesclados em um único método. Uma longa lista pode ter sido criada para controlar qual algoritmo será executado e como. A melhor maneira

de tentar corrigir tal defeito é diminuir o número de argumentos encapsulando alguns argumentos em um único objeto.

NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT
194	2513	392	3	3	0	43	43

2.8 Medição 7 – Após a refatoração de todos os code smells do projeto

Após a descoberta dos code smells e as operações de refatoração do código, é perceptível que alguns valores diminuem e outros não. Como a maioria dos smells eram classificados como Design Smells, já era esperado que a mudança nas métricas não fosse relevante.

Antes:

NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT
195	2555	395	3	3	875	57	57

Depois:

NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT
194	2513	392	3	3	0	43	43

REFERÊNCIAS

AZEEM, Muhammad. Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. Information and Software Technology, v. 108, p. 115-138, 2019.

SABIR, Fatima. A systematic literature review on the detection of smells and their evolution in object‐oriented and service‐oriented systems. Software: Practice and Experience, v. 49, n. 1, p. 3-39, 2019.