

Relatório TP2 - Ferramentas e técnicas de Fan out

Roberto Cachada (a81012) Pedro Gonçalves (a82313)

Braga, maio de 2020

Universidade do Minho
Mestrado Integrado em Engenharia Informática
Engenharia de Segurança
(2^o Semestre/2019-2020)
Grupo 7

Conteúdo

1	Introdução	3
2	Fan Out	4
2.1	Cálculo do Fan Out em diferentes linguagens	5
3	Ferramentas	7
3.1	JHawk	7
4	Conclusão	8

Lista de Figuras

1	Exemplo de módulo com Fan Out baixo	4
2	Exemplo de módulo com Fan Out alto	4
3	Escala Fan Out da TIOBE	5
4	Resultados pertinentes da análise em JHawk	7

1 Introdução

Nos dias que correm, são criados inúmeros *softwares* com as mais amplas utilizações. Esses *softwares*, além de necessitarem de cumprir com os seus objetivos, necessitam de possuir qualidade. Tendo isso em conta, foi criado um ISO, ISO 25010, que define quais os fatores que servem para avaliar a qualidade de um produto de *software*. Existem 8 fatores sendo eles:

- *Functional Suitability*;
- *Reliability*;
- *Performance Efficiency*;
- *Operability*;
- *Security*;
- *Compatibility*;
- *Maintainability*;
- *Transferability*.

No entanto, avaliar a qualidade de um *software* usando estes fatores torna-se difícil, pois estes são formas muito teóricas e gerais de medição. Assim sendo, o que se utiliza para medir a qualidade de *software* são as chamadas métricas de *software*. As métricas de *software* mais utilizadas são as seguintes:

- *Code Coverage*;
- *Abstract Interpretation*;
- *Cyclomatic Complexity*;
- *Compiler Warnings*;
- *Coding Standards*;
- *Code Duplication*;
- *Fan Out*;
- *Security*.

Apesar de existirem todas estas métricas, este relatório vai apenas focar-se na métrica denominada *Fan Out*.

2 Fan Out

Tendo em conta que um programa nada mais é do que uma conjunção de diferentes módulos ou componentes, e que esses componentes utilizam outros módulos/componentes, podemos definir *Fan Out* como o grau de interdependência entre os diferentes componentes do *software*, ou seja, quantos módulos são necessários para que um dado componente funcione corretamente. Se um componente necessita de muitos módulos para funcionar corretamente, diz-se que apresenta um alto *Fan Out*, o que também significa que se torna mais difícil modificar esse componente. Assim sendo, a métrica *Fan Out* serve para avaliar o fator de manutenção do ISO 25010.

```
import java.util.List;
import java.util.ArrayList;
```

Figura 1 - Exemplo de módulo com Fan Out baixo.

```
import java.io.IOException;
import java.util.Map;

import org.apache.commons.exec.CommandLine;
import org.apache.commons.exec.DefaultExecutor;
import org.apache.commons.exec.ExecuteException;
import org.apache.commons.exec.ExecuteResultHandler;
import org.apache.commons.exec.ExecuteWatchdog;
import org.apache.commons.exec.Executor;
import org.apache.commons.exec.PumpStreamHandler;
import org.apache.commons.exec.environment.EnvironmentUtils;
import org.apache.commons.io.output.NullOutputStream;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.resources.IResource;

import com.tiobe.plugins.eclipse.console.ITICSConsole;
import com.tiobe.plugins.eclipse.console.TICSConsole;
import com.tiobe.plugins.eclipse.util.EclipseUtils;
```

Figura 2 - Exemplo de módulo com Fan Out alto.

Na prática, a medição de *Fan Out* é efetuada contando o número de *imports* efetuados em cada módulo/componente. Contudo, a forma como

esta contagem é efetuada varia de linguagem para linguagem, como iremos analisar nos capítulos seguintes.

Além disto, é importante mencionar que existem dois tipos de *Fan Out*, *Fan Out* interno e *Fan Out* externo. *Fan Out* interno, consiste nas chamadas de módulos pertencentes ao próprio sistema, enquanto *Fan Out* externo refere-se às chamadas de módulos externos ao sistema, isto é, consiste nas reutilizações de *software* já existente. Para efeitos de medida, o TIOBE considera que *Fan Out* interno possui um efeito 4 vezes mais negativo num sistema comparativamente a *Fan Out* externo.

Existem duas formas de calcular o *Fan Out*. A primeira forma denominada *Fan Out* médio consiste em, simplesmente, somar todos os *Fan Out* de cada ficheiro e dividir esse resultado pelo número total de ficheiros de um programa. A segunda maneira, recomendada pelo TIOBE, calcula um *score* denominado *TQI Score*, utilizando a fórmula apresentada em baixo, e consoante esse resultado atribui uma classificação numa escala de A a F, sendo A a melhor categoria e F a pior.

$$score = \min(\max(120 - (8 * internalFanOut + 2 * externalFanOut), 0), 100) \quad (1)$$







Fan Out	TQI Score	Category
<= 6	>= 90%	 A
<= 8	>= 80%	 B
<= 10	>= 70%	 C
<= 14	>= 50%	 D
<= 16	>= 40%	 E
> 16	< 40%	 F

Figura 3 - Escala Fan Out da TIOBE.

2.1 Cálculo do Fan Out em diferentes linguagens

Como foi referido anteriormente, a métrica de *Fan Out* é medida através da contagem do número de *imports* existentes por módulo/componente, mas a forma como a contagem é efetuada varia de linguagem para linguagem.

C/C++

Nestas duas linguagens, o cálculo da métrica é bastante direta, pois é apenas necessário contabilizar o número de diretivas *include* existentes em cada ficheiro.

C#

O cálculo da métrica em C# é mais complicado comparativamente às linguagens da sua família. A razão para essa maior dificuldade, é devido ao facto das declarações *using* importam *namespaces* inteiros, *namespaces* esses que podem consistir em centenas de classes, mesmo que o programa utilize apenas uma minoria dessas. Assim sendo, o cálculo do *Fan Out* é feito contabilizando o número de dependências únicas existentes em cada ficheiro.

Java

De uma forma geral, em Java, é contabilizado o número de declarações *import* presentes em cada ficheiro. No entanto, se forem utilizados *wildcard imports* a contabilização torna-se mais difícil de calcular pois este tipo de importação importa várias classes em simultâneo. Devido a isto, o TIOBE, aconselha a contabilizar este tipo de imports como 5.

JavaScript

Para esta linguagem, nomeadamente *EcmaScript 6*, é contabilizado para o cálculo da métrica o número de declarações *import* presentes nos ficheiros.

Python

Para o caso do Python, a forma de contabilização é bastante simples, sendo apenas contado o número de *import/import-from* existentes em cada ficheiro do programa.

Swift

Por fim, e tal como no caso do Python, a contabilização é efetuada contando o número de declarações *import* presentes nos ficheiros.

3 Ferramentas

Existem algumas ferramentas informáticas que auxiliam os desenvolvedores a calcular esta métrica. Essas ferramentas variam de linguagem para linguagem, tendo o grupo encontrado as seguintes:

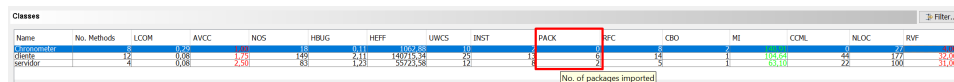
- TICS frameworks (C, C++, Dart, Go, Lua, Java, JavaScript, JSP, Kotlin, Objective-C, PL/SQL, Python, Scala, Swift, TypeScript, VB.NET)
- TICSCil (C#)
- mlint (MATLAB)
- Simulink (Simulink)
- JStyle (Java)
- JHawk (Java)

Apesar do número de ferramentas identificadas ser grande, a maioria destas são ferramentas pagas, o que impediu que o grupo as testasse. No entanto, foi possível testar a versão demo do *JHawk*, teste esse que é descrito no capítulo em seguida. Convém também notar que a ferramenta JStyle já não é suportada pelos criadores da mesma.

3.1 JHawk

Tendo sido testada uma versão demo da ferramenta, existem certas funcionalidades da mesma que não se encontravam disponíveis para utilização, tais como a impossibilidade de exportação dos resultados obtidos e a também a falta de acesso ao visualizador, denominado DataView, que permite analisar com maior detalhe os resultados obtidas nas análises.

Para utilizar esta ferramenta basta importar os ficheiros que pretendemos que sejam analisados e correr o programa. A métrica que mais nos interessa, é a que contabiliza o número de pacotes importados por cada classe.



Name	No. Methods	LCOM	AVCC	NOS	HBUG	HEFF	UWCS	INST	PACK	REC	CBO	MI	COML	NLOC	RVF
cliente	12	0.00	1.75	149	2.11	140715.34	11	1	0	0	14	1	105.65	34	177
servidor	4	0.00	2.50	81	1.25	55723.98	12	1	0	0	1	1	61.00	22	100

Figura 1 - Resultados pertinentes da análise em JHawk

4 Conclusão

Com este relatório o grupo considera que os seus conhecimentos sobre qualidade e métricas de *software* aumentaram, em particular os conhecimentos sobre a métrica *Fan Out*.

No entanto convém referir que a métrica estudada neste projeto, não é na opinião do grupo, uma das mais importantes a ter em conta aquando da construção de *software*, pois apenas aborda um dos 8 fatores abordados no ISO 25010, que é o fator da manutenção. Mesmo não sendo das mais importantes, tem a sua relevância pois quanto mais simples for manter/atualizar um determinado programa melhor é.

Referências

- [1] TIOBE.: https://www.tiobe.com/files/TIOBEQualityIndicator_v4_3.pdf (2020)
- [2] TIOBE.: https://portal.tiobe.com/8.5/docs/metrics/index.html?fbclid=IwAR1wuvdxGi2ddr9H_W4UaDYEq398nIwR8ArGYTI9Gk6ygnltyEkv-fuCb_4 (2020)
- [3] JHawk.: http://www.virtualmachinery.com/jhawkprod.htm?fbclid=IwAR3zGcnjNhEx06g8LwlsTTn_WyHRKZaP2ZhtxbtBeVgZBCzNsXzQkn5F1fM (2020)
- [4] MLint.: <https://www.mathworks.com/help/matlab/ref/mlint.html> (2020)
- [5] TIOBE.: <https://www.tiobe.com/tics/fact-sheet/> (2020)