

Mestrado em Engenharia Informática  
Universidade do Minho

## **Engenharia de Segurança**

### **Projeto 2 - Ferramentas e técnicas de Fan out**

João Miranda - PG41845  
Sandro Cruz - PG41906

12 de Maio de 2020

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Indicadores de qualidade do software</b>	<b>3</b>
2.1	Contextualização . . . . .	3
2.2	Importância da escolha dos indicadores . . . . .	3
<b>3</b>	<b>Fan out</b>	<b>4</b>
3.1	Definição . . . . .	4
3.2	Características . . . . .	4
3.2.1	Visibilidade . . . . .	4
3.2.2	Conectividade . . . . .	4
3.3	Linguagens de programação e a sua contagem . . . . .	4
3.4	Fan out interno e externo . . . . .	5
3.5	Cálculo do Fan out . . . . .	5
<b>4</b>	<b>Ferramentas que ajudam na utilização de fan-out</b>	<b>6</b>
4.1	AWS Lambda . . . . .	6
4.1.1	Utilizações . . . . .	6
4.1.2	Aplicações do AWS Lambda . . . . .	9
4.1.3	Yubl . . . . .	10
4.2	PubNub Realtime Communication Platform . . . . .	10
4.3	Zato . . . . .	11
4.3.1	Exemplo . . . . .	11
<b>5</b>	<b>Ferramentas para calculo do fan out</b>	<b>12</b>
5.1	Utilização do JHawk . . . . .	13
<b>6</b>	<b>Conclusão</b>	<b>14</b>

# Capítulo 1

## Introdução

No segundo projeto da unidade curricular de Engenharia de Segurança iremos desenvolver o tópico dos Ferramentas e técnicas de *Fan out*.

Com este trabalho pretendemos que seja efetuado uma pesquisa de indicadores de qualidade do código/software com uma explicação do indicador de qualidade de software analisado, para que serve e quais as ferramentas utilizadas consoante a linguagem de programação.

## Capítulo 2

# Indicadores de qualidade do software

### 2.1 Contextualização

Os indicadores de qualidade do *software* são um requerimento importante. O desenvolvimento de um *software* é semelhante à construção de um produto. Para a melhoria da qualidade de um *software* é necessária a recorrência da medida em vários aspetos. A indústria define o *benchmark* padrão para cada métrica e é possível uma verificação do *software* bem como o que é preciso ser feito para uma correção dos *benchmarks* (se cumprem ou não).

### 2.2 Importância da escolha dos indicadores

A escolha das métricas para a medição do *software* pode influenciar no sucesso de um projeto. Depende de muitos aspetos como o tamanho, complexidade, missão crítica e a manutenção do *software*. Muitas empresas de Tecnologia de Informação (TI) possuem um departamento de qualidade que identifica o que precisa de ser medido ao longo do projeto. Podem-se prescrever métricas adicionais com base nos requisitos pretendidos. O processo vai ser apresentado de seguida.

- **Estabelecer meta:** envolve identificar em qual aspeto do *software* está a ser feita uma tentativa de melhoria. Por exemplo, a identificação de módulos de falha no início do projeto.
- **Construir as perguntas indicadas:** envolve fazer uma pergunta certa para nós mesmos sobre o que nos impede de alcançar a meta.
- **Identificar as medidas:** para a maioria da construção das perguntas, existem medidas padrão definidas que se podem utilizar.

## Capítulo 3

# Fan out

### 3.1 Definição

O *Fan out* é uma medida estrutural da complexidade entre módulos. Corresponde ao número de módulos chamados por um determinado módulo. Este tipo de medida pode ser aplicada no nível do módulo e no nível da função. Têm como função disponibilizar um número que identifica a complexidade da interligação de módulos ou funções diferentes. O número de *Fan out* de uma função pode ser restringido para evitar um sobrecarregamento, apesar de não ser uma prática amplamente aceite.

### 3.2 Características

#### 3.2.1 Visibilidade

A visibilidade indica o conjunto de componentes de programas que pode ser invocado ou usado como dados por uma determinada componente. Por exemplo, um módulo de um sistema orientado a objeto pode ter acesso a uma ampla sucessão de objetos de dados que ele herdou, mas só faz uso de um pequeno número desses objetos dados.

#### 3.2.2 Conectividade

A conectividade indica o conjunto de componentes invocado ou usado como dados por determinada componente. Por exemplo, um módulo que efetue diretamente outro módulo iniciar a execução é conectado a ele.

### 3.3 Linguagens de programação e a sua contagem

Para C e C++, o número de instruções é usada, para Java, o número de instruções de importação. Os caracteres nas instruções de importação em Java pare-

cem ser complicados porque são importados vários módulos ao mesmo tempo. É por isso que a contagem deve ser feita como 5 instruções. A situação é ainda mais complexa para C porque é utilizada uma importação com um mecanismo diferente. A instrução em C importa um espaço para o nome completo, que pode consistir em centenas de classes e só algumas delas estão a ser devidamente utilizadas. Por isso, é exigido que no C seja contado o número real de dependências exclusivas por ficheiro.

### 3.4 Fan out interno e externo

O *Fan out* externo preocupa-se com as importações exteriores ao sistema de *software* enquanto o *Fan out* interno relaciona-se com as referências do sistema em si. As importações externas são aplicadas principalmente para uma reutilização do *software* existente e, portanto, são muito melhores do que importações internas. Portanto, as importações internas têm um impacto negativo quatro vezes maior no TQI (TIOBE Quality Indicator) do *Fan out* do que as importações externas.

### 3.5 Cálculo do Fan out

O *Fan out* médio de um sistema é mapeado numa escala normativa utilizando-se a fórmula:

$$score = \min(\max(120 - (8 * internalfan\_out + 2 * externalfan\_out), 0), 100)$$

De seguida será apresentada uma tabela que mapeia o *Fan out*, a respetiva pontuação TQI e a sua categoria. É assumido o rácio entre a importanção interna e externa de 1:1.







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

Figura 3.1: Pontuações do *Fan out*.

## Capítulo 4

# Ferramentas que ajudam na utilização de fan-out

### 4.1 AWS Lambda

Este *AWS (Amazon Web service)* permite a utilização de um serviço sem servidor, que permite uma enorme paralelização com uma gestão simplificada das infraestruturas. O que faz com que o AWS Lambda seja um candidato perfeito para a implementação de fan-out e fan-in.

O AWS Lambda está disponíveis em varias linguagens sendo que é necessário utilizar o respectivo *sdk* para cada linguagem, por exemplo para Python o *sdk* a ser utilizado é o boto3.

#### 4.1.1 Utilizações

##### Processamento de dados

O AWS Lambda pode ser utilizado para executar código como resposta a diferentes tipos de *triggers*, como alterações nos dados, mudanças no estado do sistema ou intervenções dos utilizadores. O Lambda pode ser acionado diretamente pelos serviços da AWS como S3, DynamoDB, Kinesis, SNS e CloudWatch, ou pode ser orquestrado em fluxos de trabalho pelo AWS Step Functions. Isso permite que você crie uma variedade de sistemas de processamento de dados sem servidor em tempo real.

Diferentes formas como pode ser utilizado:

- **Processamento de arquivos em tempo real** - o processamento dos arquivos imediatamente depois de serem enviados, por exemplo o Lambda pode ser utilizado para imagens em miniatura, transcodificação de vídeos, indexação de arquivos, processamento de logs, validação de conteúdo, além de agregação e filtragem de dados em tempo real. Este tipo de utilização específica é utilizada pelo "The Seattle Times" para

redimensionar imagens para visualização em dispositivos diferentes, como computadores desktop, tablets e smartphones.

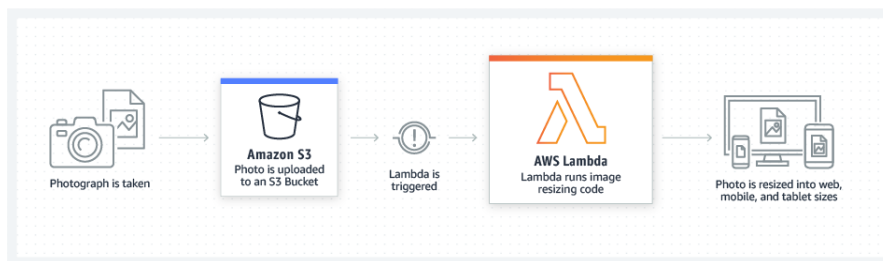


Figura 4.1: Exemplo de uma utilização deste serviço.

Exemplo de uma aplicação deste serviço: <https://github.com/aws-samples/lambda-refarch-fileprocessing>

- **Processamento de stream em tempo real** - a fim de processar dados de streaming em tempo real para monitorização de atividades de aplicações, processamento de pedidos de transações, análise de sequências de cliques, refinamento de dados, geração de métricas, filtragem de logs, indexação, análise de redes sociais, e telemetria e medição de dados de dispositivos da IoT.

A Localytics processa bilhões de pontos de dados em tempo real e usa o Lambda para processar dados históricos e ativos armazenados no S3 ou oriundos do Kinesis (Dois serviços da Amazon).

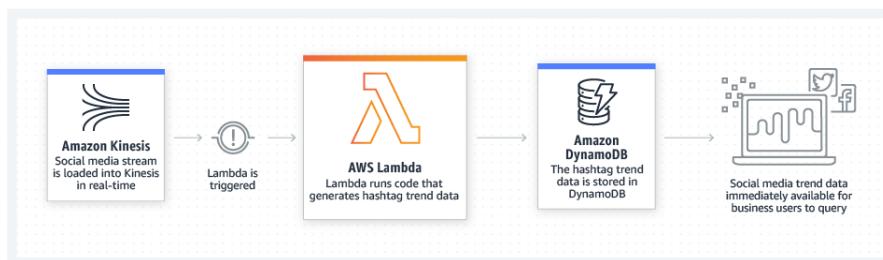


Figura 4.2: Exemplo de uma utilização deste serviço.

Exemplo de uma aplicação deste serviço: <https://github.com/aws-samples/lambda-refarch-streamprocessing>

- **Extrair, Transformar, Carregar** - pode ser usado para executar a validação, a filtragem e a classificação de dados ou outras modificações para



cada alteração de dados ocorrida em uma tabela do DynamoDB, um serviço de base de dados da Amazon, e carregar os dados modificados em outro armazenamento de dados.

A Zillow usa o Lambda e o Kinesis para monitorar um subconjunto de métricas móveis em tempo real. Com o Kinesis e o Lambda, nós conseguimos desenvolver e implantar uma solução econômica em duas semanas.

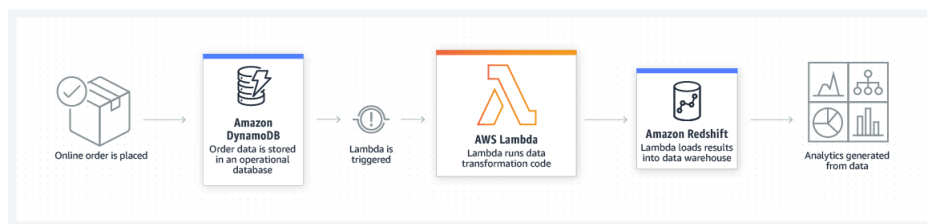


Figura 4.3: Exemplo de uma utilização deste serviço.

## Back-ends

O AWS Lambda pode ser utilizado para administrar solicitações de API da web, móveis, da Internet das Coisas (IoT) e de terceiros.

Diferentes formas como pode ser utilizado:

- **Aplicações Web** - Ao combinar o AWS Lambda com outros serviços da AWS, os desenvolvedores conseguem criar aplicativos da Web potentes que expandem e diminuem automaticamente e que são executados numa uma configuração altamente disponível em vários datacenters, exigindo zero esforço administrativo para escalabilidade, backups ou redundância de vários datacenters.

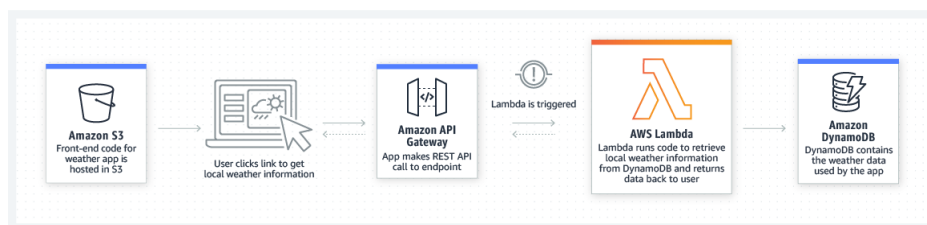


Figura 4.4: Exemplo de uma utilização deste serviço.

- **Back-ends da IOT** - permite criar back-ends sem servidor, que utilizam o AWS Lambda para administrar solicitações de API da Web, móvel, da Internet das Coisas (IoT) e de terceiros.

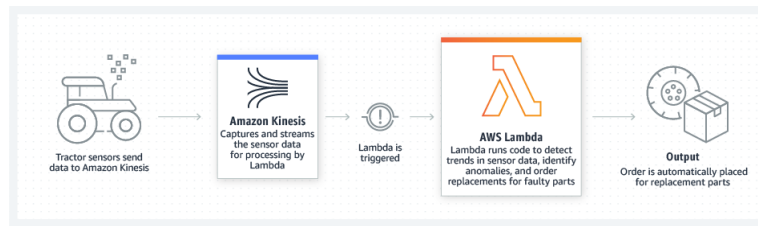


Figura 4.5: Exemplo de uma utilização deste serviço.

Exemplo de uma aplicação deste serviço: <https://github.com/aws-samples/lambda-refarch-iotbackend>

- **Back-ends móveis** - O AWS Lambda facilita a criação de experiências de aplicativo avançadas e personalizadas. Pode ser utilizado para criar back-ends usando o AWS Lambda e o Amazon API Gateway para autenticar e processar solicitações de API.

O Bustle executa um back-end sem servidor para sua aplicação Bustle iOS e websites usando o AWS Lambda e o Amazon API Gateway. As arquiteturas sem servidor permitem que a Bustle nunca tenha que lidar com o geração de infraestrutura.



Figura 4.6: Exemplo de uma utilização deste serviço.

Exemplo de uma aplicação deste serviço: <https://github.com/aws-samples/lambda-refarch-mobilebackend>

### 4.1.2 Aplicações do AWS Lambda

#### Coca-Cola Company

A Coca-Cola Company, uma empresa de bebidas multinacional americana, usou o AWS Lambda e o AWS Step Functions para criar uma solução sem servidor econômica.

#### iRobot

A iRobot, uma empresa de robôs para consumidores, está a criar uma próxima geração de dispositivos conectados para a casas inteligente usando uma

arquitetura sem servidor.

### Benchling

A Benchling, uma empresa de software de ciências biológicas, criou uma técnica usada por investigadores para modificar partes de um genoma com precisão extrema, usando uma arquitetura sem servidor.

### Thomson Reuters

A Thomson Reuters usa o AWS Lambda para processar até 4.000 eventos por segundo para o seu serviço de estatística de uso, e levou cinco meses para colocar o serviço em produção.

#### 4.1.3 Yubl

O Yubl era uma rede social que utilizava o AWS Lambda para o envio de notificações aos seus utilizadores. Infelizmente a rede social já não existe, mas encontramos um *dev blog* que nos explicou como era feita a utilização do Lambda.

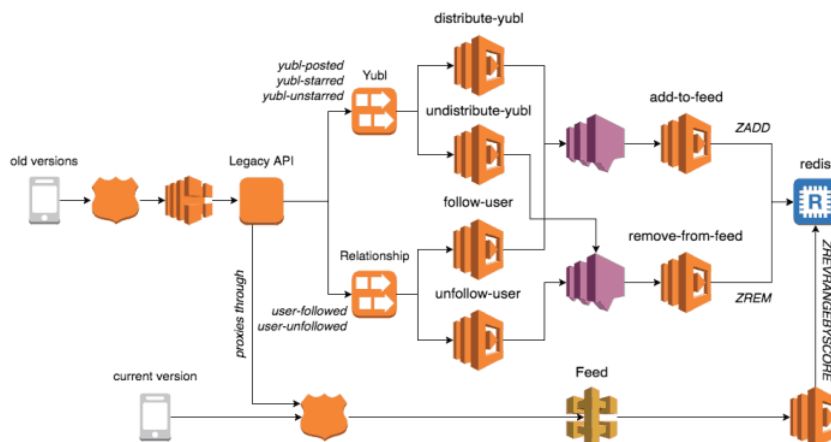


Figura 4.7: Diagrama de como era aplicado o sistema de partilha de publicações, quando um utilizador publicava uma foto para os seus seguidores, dado que um utilizador pode ter milhares de seguidores a tarefa é dividida em pequenas tarefas de forma a enviar a nova publicação aos seguidores.

## 4.2 PubNub Realtime Communication Platform

É uma SKD em conjunto com uma API para JavaScript que permite implementar um conjunto de serviços como de mensagem, notificações, armazenamento, funções e blocos para integração de outros módulos. A parte das funções é a mais interessante dado que é a que implementa as funções de fan out.

Infelizmente este produto é pago e a maior parte da informação está por de trás desta barreira, o que nos limita em saber as potenciais aplicações do fan out, mas do que é de domínio público a PubNub indica que pode ser utilizado num sistema de subscrições e notificações, provavelmente de estrutura parecida com o exemplo da AWS Lambda da Yubt.

## 4.3 Zato

É um ESB, SOA, REST API, APIs e que permite integrações em cloud *open source* em Python. Não nos é dado nenhum exemplo específico, mas na documentação do Zato encontramos alguns enxertos de código de como normalmente é aplicado o fan out.

### 4.3.1 Exemplo

Neste exemplo é utilizado um dicionário para mapear os alvos e os seus pedidos bem como uma lista de *callbacks*, que é fornecida pelo "self.patterns.fanout.invoke". Com isto todos os alvos são lançados em segundo plano e assim que estão acabados, as *callbacks* são invocadas.

```
1 from zato.server.service import Service
2
3 class MyService(Service):
4
5     def handle(self):
6
7         # A dictionary of services to invoke along with requests they receive
8         targets = {
9             'service1': {'hello': 'from-fan-out1'},
10            'service2': {'hello': 'from-fan-out2'},
11        }
12
13        # Callbacks to invoke when both services above finish
14        callbacks = ['my.callback1', 'my.callback2']
15
16        # Call's Correlation ID is returned on output
17        cid = self.patterns.fanout.invoke(targets, callbacks)
```

Figura 4.8: Exemplo de uma aplicação de fan out utilizando o Zato 3.1

## Capítulo 5

# Ferramentas para calculo do fan out

Existem algumas ferramentas disponíveis, mas infelizmente tivemos dificuldades em encontrar essas mesmas ferramentas o que nos atrasou bastante o trabalho. Também tivemos outro problema que é facto de que as poucas que encontramos a grande parte serem pagas e/ou desactualizadas ou para versão extremamente antigas de algumas linguagens de programação.

De qualquer das formas, as ferramentas que encontramos foram as seguintes:

- TICS Framework - é uma framework da TIOBE que permite medir a qualidade de software produzido num grande conjunto de parâmetros. Linguagens suportadas: C, C++, Dart, Go, Lua, Java, JavaScript, JSP, Kotlin, Objective-C, PL/SQL, Python, Scala, Swift, TypeScript e VB.NET.
- TICSCil - faz parte da TICS Framework, mas é apenas utilizada para C.
- Mlint ou CheckCode - é um modulo disponível para MatLab da MathWorks que permite medir um conjunto de variáveis referentes a qualidade do código de um programa. Apesar de na página web do Mlint recomendar contra a sua utilização ele ainda é muito utilizado e para todos os efeitos o CheckCode é apenas uma evolução do Mlint.
- Simulink - um software que permite a criação de programas utilizando uma interface puramente gráfica e orientada o design baseada em modelo, que tem a sua própria linguagem a Simulink. As vantagens deste software são que permite criar um programa inteiro sem ter que escrever o código para esse programa, outra vantagem do software é que permite medir o fan out sem a utilização de um modulo ou software externo, o que é sempre agradável.
- Project Analyzer - desenvolvido pela AiVosto e é um programa completamente obsoleto, dado que foi desenvolvido para Visual Basic 6 uma linguagem de programação que recebeu a sua ultima utilização em 1998, pois

foi substituída pelo Visual Basic .NET, o programa em si também parece estar preso a 1998. Mesmo assim faz parte da lista dado que efectivamente mede o fan out, apenas não é muito útil actualmente.

- JHawk - é uma ferramenta de análise de código estático para Java desenvolvido pela Virtual Machinery que fazer todo tipo de análises esperadas numa ferramenta de análise, sendo que a ultima versão o JHawk 6.1.2 saiu em 16 de Outubro de 2016. O site da Virtual Machinery indica que o JHawk existe desde 2006, algo que quase que pode ser adivinhado pelo design da página web e a falta de um certificado https, o que nos deixou um pouco preocupados com o que poderemos estar a fazer download para as nossas máquinas.

Pelo as pesquisas que fizemos a documentação do JHawk conseguimos descobrir que a ferramenta mede o fan out e fan in por cada classe do programa que queremos analisar, mas a falta de segurança no web site em si mais o facto de que os artigos que deveriam certificar o JHawk como uma ferramenta segura remetem para o web site da Virtual Machinery achamos mais prudente não testar e também dado ao facto de que só teríamos acesso a uma demo, mas dado que máquinas virtuais existem e há alguns trabalhos científicos que utilizam o JHawk decidimos testar.

## 5.1 Utilização do JHawk

Dado que é uma demo não temos acesso a todas as funções, mas temos acesso ao número de pacotes que foram importados pelas classes do nosso programa. Neste teste utilizamos uma aplicação android. O JHawk considera o fan out com CBO, sendo assim representado como CBO, a explicação sendo que o JHawk é baseado nas definições de Chidamber e Kemere e a Virtual Machinery considera que ambos se referem a mesma coisa.

PACK	RFC	CBO	M
8	4	0	
5	8	0	
23	9	0	
23	1	0	
8	1	0	

Figura 5.1: Pacotes importados (PAK) e o fan out (CBO) das classes do pacote principal da aplicação.

## Capítulo 6

# Conclusão

Com este trabalho tivemos a oportunidade de explorar uma das métricas e indicadores que constituem a construção de software de qualidade, sendo que o tópico que nos foi apresentado eram as ferramentas de fan out. Ao começar este projeto tínhamos uma vaga ideia do que era isto do fan out, mas saímos deste projeto esclarecidos a importância desta métrica na criação de software de qualidade e a sua crescente necessidade de ser medida, dado as actuais aplicações web e o grande número de módulos interdependentes e as invocações nas mais variadas classes, o que leva muitas vezes ao desenvolvimento de aplicações web extremamente lentas porque o fan out é inexistente e o utilizador tem que carregar um número ridículo de módulos.

Um exemplo perfeito disto é o *launcher* do jogo League Of Legends desenvolvido pela RIOT Games, do qual uma abertura do launcher pode demorar vários minutos e o próprio problema já foi referido em vários *dev blogs* alguns desse com mais de 1 ano, mas ainda nada foi feito. O que também demonstra o quão complexo pode ser o problema do fan out se não for tratado desde do início do desenvolvimento de um software, pois um jogo com milhões de jogadores pelo mundo inteiro sofre de um problema que parece ser tão óbvio.