



# DR-Tools

A tool quality suite to help the  
developers to maintain health and code evolution

**[drtools.dev](https://drtools.dev)**



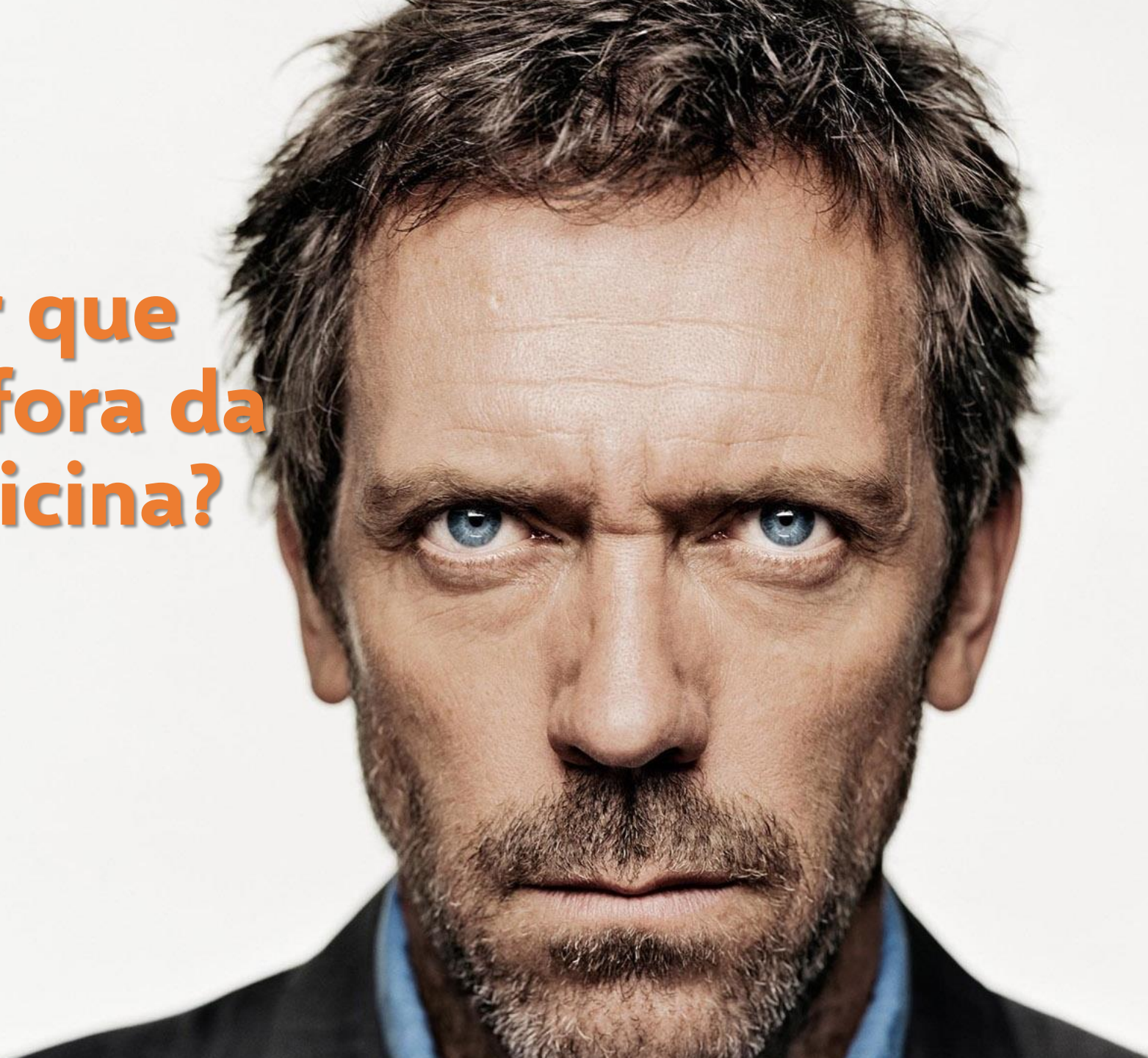
# Quem sou eu?

glacerda@wildtech.com.br  
@guilhermeslac

- ✓ Mestre e Doutorando em Ciência da Computação (UFRGS)
- ✓ Professor de Graduação (UniRitter) e Pós-Graduação (UniRitter, Unisinos, UFRGS)
- ✓ Consultor associado da Wildtech
- ✓ Pioneiro em Metodologias Ágeis no Brasil
- ✓ Fundador do XP-RS/GUMA
- ✓ Membro da ScrumAlliance , IASA, SBC e ACM



**Por que  
metáfora da  
medicina?**

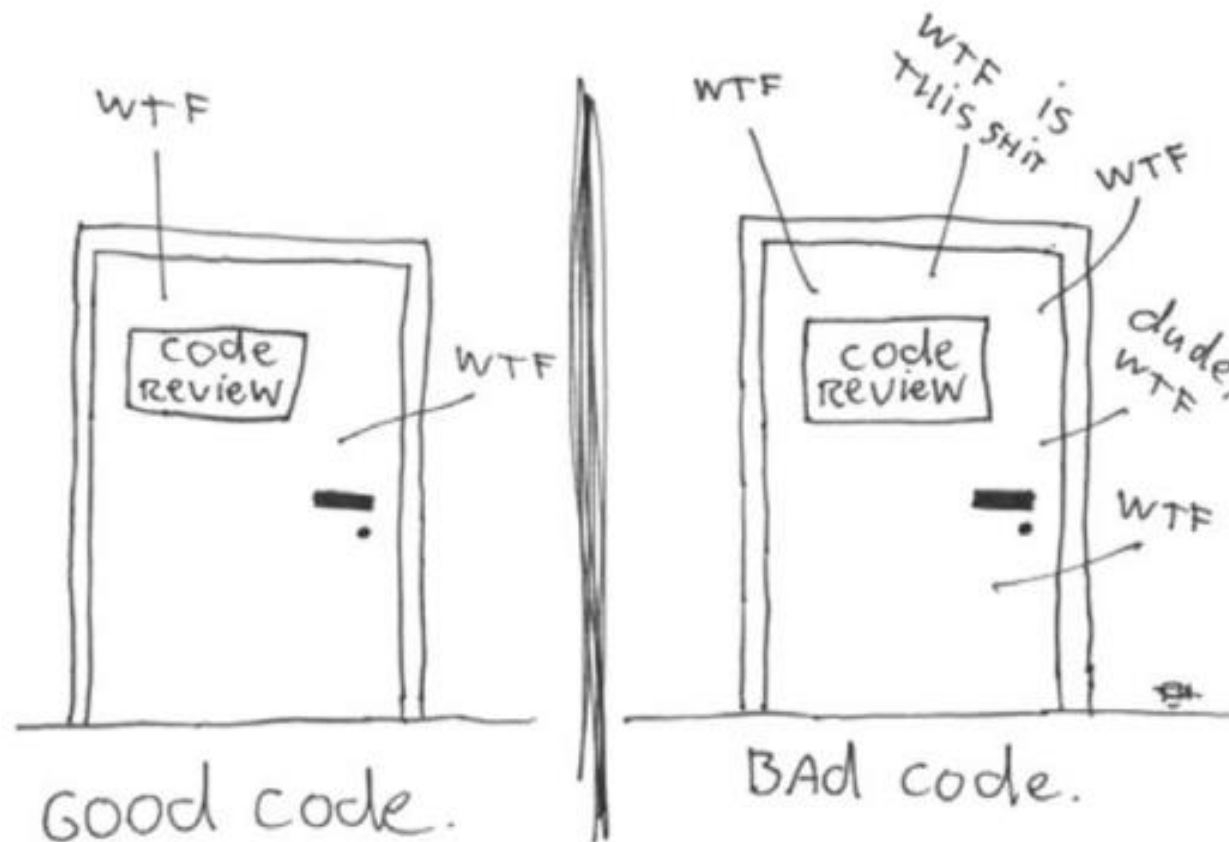




**metric**  
**DR-Tools**

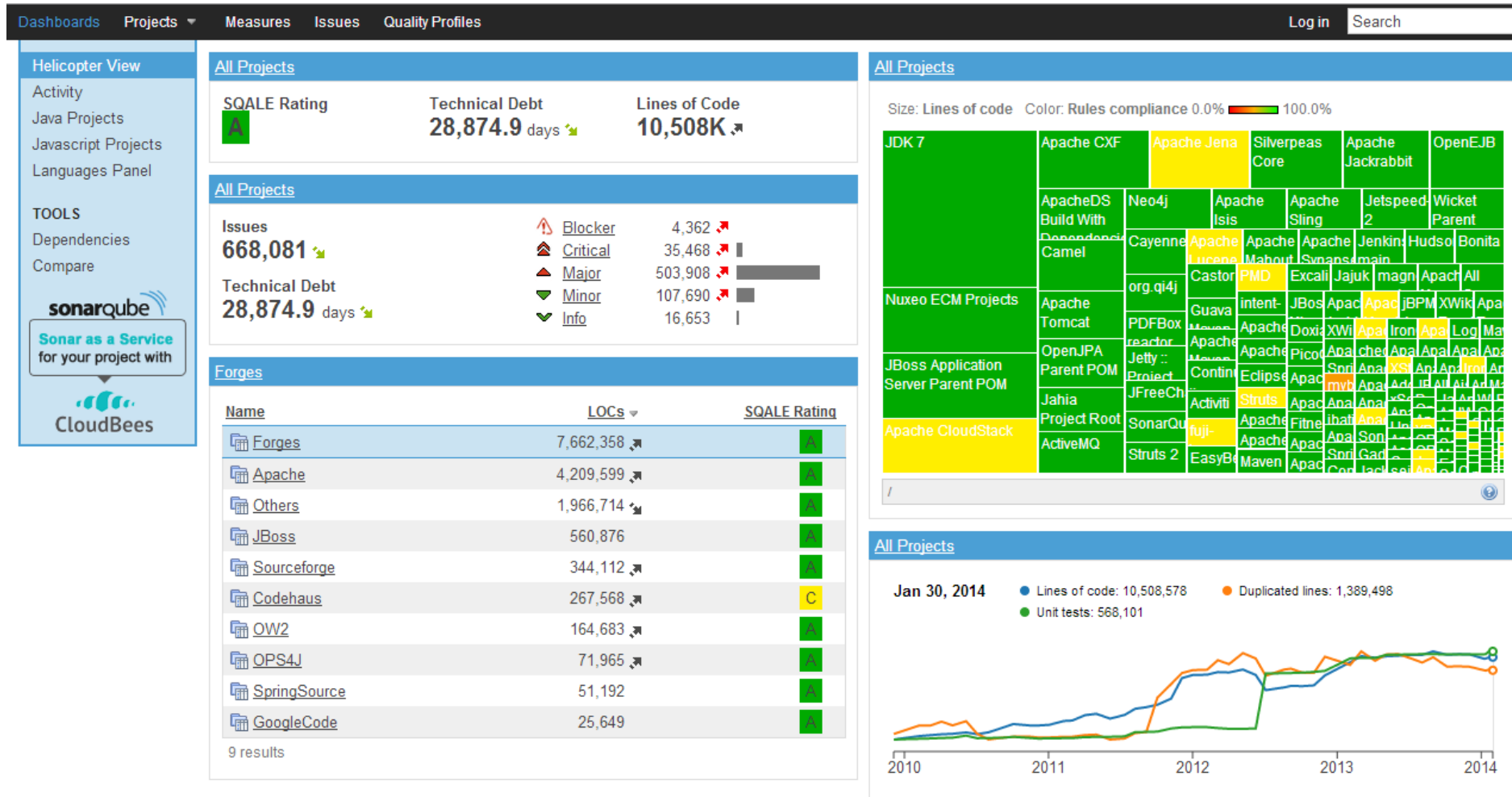
# Como você mede a qualidade do código?

The ONLY VALID MEASUREMENT  
OF CODE QUALITY: WTFs/MINUTE





# “ah... Nós temos o SonarQube”



**Se eu baixar um projeto do  
GitHub,  
onde estão os maiores  
problemas?**



# Características

- ✓ **Simplicidade, valor do XP**
- ✓ **CLI (*command line interface*)**
- ✓ **Versatilidade**
- ✓ **Download and run! (pré-requisito: JRE 8)**
- ✓ **Seleção de métricas, correlacionadas e ordenadas**
- ✓ **Visualizações usando Google Chart e D3.js**
- ✓ **Atualmente analisa projetos Java**





# Funções

- ✓ **Resultados em diferentes formatos**  
*Console, JSON, CSV*
- ✓ **Filtragem dos 'Top X'**
- ✓ **Resultados contextualizados**  
*Projeto, Namespaces, Classes, Métodos, Acoplamento, Dependências*



# Métricas Contextualizadas



## Sumário do Projeto

*Total de Namespaces, Classes, SLOC, Métodos e CYCLO*



## Namespaces

*NOC, NAC*



## Classes

*SLOC, NOM, WMC/CYCLO, DEP, I-DEP/Fan-Out, NPM, NOA*



## Métodos

*MLOC, CYCLO, CALLS, NBD, PARAM*



## Acoplamento

*CA, CE, Instability, Abstractness, Normalized Distance*



## Dependências

*Externas, Internas, Ciclic Dependency*



**DR-Tools**

# Uso

```
λ drtools-metric
```

```
drtools-metric - helping you to improve the health of your source code and reduce technical debt!
```

```
Developed by Guilherme Lacerda (guilhermeslacerda@gmail.com)
```

```
Usage: drtools-metric <project-directory> <OPTIONS> <OUTPUT> [--top <number>]
```

```
OPTIONS = <-a|-ac|-s|-n|-t|-m|-d|-cd|-id|-c|-mt> OUTPUT = <--console|--csv|--json>
```

## Where

-a	list ALL metrics (namespaces/types/methods)	--console	show the results to console
-ac	list ALL metrics about COUPLING/DEPENDENCIES	--csv	generate results in CSV format
-s	list a SUMMARY of project	--json	generate results in JSON format
-n	list information about NAMESPACES	--top	list top 'number' records, based on used format
-t	list information about TYPES (classes)		
-m	list information about METHODS (functions)		
-d	list information about DEPENDENCIES of types/classes		
-cd	list information about CYCLIC DEPENDENCIES of types/classes		
-id	list information about INTERNAL DEPENDENCIES of types/classes		
-c	list information about COUPLING of namespaces		
-mt	list information about METRIC THRESHOLDS		

## Metrics

CA	- Number of types/classes outside this component that depends on types/classes inside this component (Afferent Coupling)		
CE	- Number of types/classes inside this component that depends on types/classes outside this component (Efferent Coupling)		
I	- Instability of namespace (range between 0=Maximally stability and 1=Maximally instability)		
A	- Abstractness degree of namespace (range between 0=Minimally abstractness and 1=Maximally abstractness)		
D	- Normalized distance of namespace from the main sequence		
NAC	- Number of abstract types/classes of package/namespace	WMC	- Weighted methods per types/classes (sum the CYCLO of each method)
NOC	- Number of types/classes of package/namespace	SLOC	- Number of lines of source code
DEP	- Number of type/classes external dependencies	I-DEP	- Number of type/classes internal dependencies (Fan-Out)
NOA	- Number of attributes/variables	NOM	- Number of methods/functions of a type
NPM	- Number of public methods/functions of a type	NBD	- Number of nested block depth of a method/function
MLOC	- Number of lines of a method/function	PARAM	- Number of parameters of a method/function
CYCLO	- Cyclomatic complexity (McCabe) of a method/function	CALLS	- Number of invocations made from within a method/function

## Usage examples:

```
Example 1 : # drtools-metric \Project\Java\src -a --console
```

```
Example 2 : # drtools-metric \Project\Java\src -t --csv
```

```
Example 3 : # drtools-metric \Project\Java\src -m --console --top 10
```

```
C:\Program Files\cmderr
```

```
λ |
```

# Uso

```
λ drtools-metric D:\JavaApps\Plugins\DrToolsMetric\src -s --console
```

## SUMMARY OF METRICS

```
-----
Total of Namespaces: 11
Total of Types: 39 - 3,55 (number of types/namespaces)
Total of SLOC: 2039 - 52,28 (number of SLOC/types)
Total of Methods: 269 - 6,90 (number of methods/types)
Total of CYCLO: 432 - 11,08 (number of CYCLO/types)
Processing time: 531 milliseconds
```

```
C:\Program Files\cmdr
```

```
λ drtools-metric D:\JavaApps\Plugins\DrToolsMetric\src -mt --console
```

## INFORMATION ABOUT METRIC THRESHOLDS

```
-----
Small Project (SMALL)                small project with < 50 KLOC or 200 < classes
Medium Project (MEDIUM)              medium project with (50 KLOC <= project <= 250 KLOC) or (200 <= classes <= 1000)
Large Project (LARGE)                 large project with > 250 KLOC or > 1000 classes
Number of Types/Classes (NOC)         Good: <= 11; Regular: between 11 and 28; Bad: > 28
Number of Abstract Types/Classes (NAC) without references
Type/Class Line of Code (SLOC)        Bad: > 500
Number of Functions/Methods (NOM)     Good: <= 6; Regular: between 6 and 14; Bad: > 14
Weighted Methods per Class (WMC)      Good: <= 11; Regular: between 11 and 34; Bad: > 34
Number of external types/classes dependencies (DEP) Bad: > 20
Number of internal types/classes dependencies (I-DEP) Bad: > 15
Number of Public Methods (NPM)        Good: <= 10; Regular: between 11 and 40; Bad: > 40
Number of Attributes/Fields (NOA)     Good: <= 3; Regular: between 3 and 8; Bad: > 8
Method Lines of Code (MLOC)           Good: <= 10; Regular: between 10 and 30; Bad: > 30
Cyclomatic Complexity (CYCLO)         Good: <= 2; Regular: between 2 and 4; Bad: > 4
Number of Invocations (CALLS)         without references
Nested Block Depth (NBD)              Good: <= 1; Regular: between 1 and 3; Bad: > 3
Number of Parameters (PARAM)          Good: <= 2; Regular: between 2 and 4; Bad: > 4
Afferent Coupling (CA)                Good: <= 7; Regular: between 7 and 39; Bad: > 39
Efferent Coupling (CE)                Good: <= 6; Regular: between 6 and 16; Bad: > 16
Package Instability (I)               range between 0=Maximally stability and 1=Maximally instability
Abstractness Degree (A)               range between 0=Minimally abstractness and 1=Maximally abstractness
Normalized Distance (D)               range between 0=exactly located in the main sequence and 1=far from the main sequence
```

```
C:\Program Files\cmdr
```

```
λ |
```



# Uso

```
C:\Program Files\cmdr
```

```
λ drtools-metric D:\JavaApps\Doutorado\repos\hibernate-orm-master\ -s --console
```

## SUMMARY OF METRICS

```
-----
Total of Namespaces: 1371
Total of Types: 9954 - 7,26 (number of types/namespaces)
Total of SLOC: 681725 - 68,49 (number of SLOC/types)
Total of Methods: 55503 - 5,58 (number of methods/types)
Total of CYCLO: 76955 - 7,73 (number of CYCLO/types)
Processing time: 24 seconds
```

```
C:\Program Files\cmdr
```

```
λ drtools-metric D:\JavaApps\Doutorado\repos\hibernate-orm-master\ -cd --console | more
```

## Types with Cyclic Dependencies

```
-----
org.hibernate.Criteria - org.hibernate.criterion.Criterion
org.hibernate.Criteria - org.hibernate.criterion.Order
org.hibernate.Criteria - org.hibernate.criterion.Projection
org.hibernate.boot.MetadataSources - org.hibernate.boot.spi.MetadataBuilderFactory
org.hibernate.boot.model.naming.ObjectNameNormalizer - org.hibernate.boot.spi.MetadataBuildingContext
org.hibernate.boot.registry.StandardServiceRegistryBuilder - org.hibernate.service.spi.ServiceContributor
org.hibernate.boot.spi.MetadataBuilderFactory - org.hibernate.boot.MetadataSources
org.hibernate.boot.spi.MetadataBuildingContext - org.hibernate.boot.model.naming.ObjectNameNormalizer
org.hibernate.bytecode.enhance.spi.interceptor.EnhancementAsProxyLazinessInterceptor - org.hibernate.persister.entity.EntityPersister
org.hibernate.bytecode.enhance.spi.interceptor.EnhancementHelper - org.hibernate.mapping.Property
org.hibernate.cache.cfg.spi.DomainDataRegionBuildingContext - org.hibernate.cache.spi.RegionFactory
org.hibernate.cache.spi.RegionFactory - org.hibernate.cache.cfg.spi.DomainDataRegionBuildingContext
org.hibernate.cache.spi.access.CollectionDataAccess - org.hibernate.persister.collection.CollectionPersister
org.hibernate.cache.spi.access.EntityDataAccess - org.hibernate.persister.entity.EntityPersister
org.hibernate.cache.spi.access.NaturalIdDataAccess - org.hibernate.persister.entity.EntityPersister
org.hibernate.cfg.AnnotationBinder - org.hibernate.cfg.annotations.CollectionBinder
org.hibernate.cfg.Environment - org.hibernate.internal.util.ConfigHelper
org.hibernate.cfg.SetSimpleValueTypeSecondPass - org.hibernate.cfg.annotations.SimpleValueBinder
org.hibernate.cfg.annotations.CollectionBinder - org.hibernate.cfg.AnnotationBinder
org.hibernate.cfg.annotations.SimpleValueBinder - org.hibernate.cfg.SetSimpleValueTypeSecondPass
org.hibernate.collection.spi.PersistentCollection - org.hibernate.engine.spi.SharedSessionContractImplementor
org.hibernate.collection.spi.PersistentCollection - org.hibernate.persister.collection.CollectionPersister
```

# Uso

C:\Program Files\cmdr

λ drtools-metric D:\JavaApps\Doutorado\PathFinder\softwarepathfinder\src\ -t --console --top 30

TYPES	SLOC	NOM	NPM	WMC	DEP	I-DEP	NOA
om.softwarepathfinder.visualization.prefuse.PathMethodsGraph	141	2	3	6	29	3	0
com.softwarepathfinder.visualization.jgraph.PathView	116	5	2	13	17	5	4
com.softwarepathfinder.visualization.yEd.GraphmlBuilder	71	1	1	7	11	6	2
com.softwarepathfinder.model.Project	175	14	14	28	11	2	2
com.softwarepathfinder.visualization.blender.Geometry	51	12	12	12	1	1	6
com.softwarepathfinder.model.Field	53	11	11	11	6	0	6
com.softwarepathfinder.visualization.yEd.TypeGraphML	199	19	9	38	10	6	7
com.softwarepathfinder.model.Locus	109	16	16	21	13	2	7
com.softwarepathfinder.visualization.prefuse.TypeGraph	142	2	3	8	31	5	0
com.softwarepathfinder.model.Path	140	16	16	18	11	1	3
com.softwarepathfinder.parsing.php.InvocationVisitor	181	14	3	46	15	3	5
com.softwarepathfinder.model.Namespace	57	8	8	8	12	1	3
com.softwarepathfinder.visualization.prefuse.NamespaceGraph	114	0	1	11	31	5	0
com.softwarepathfinder.parsing.java.NamespaceParser	52	2	1	5	11	4	3
om.softwarepathfinder.visualization.prefuse.PathMethodLayout	69	1	1	13	8	2	1
com.softwarepathfinder.model.Type	246	37	37	51	14	2	11
com.softwarepathfinder.visualization.prefuse.PathGraph	92	0	1	2	33	5	0
com.softwarepathfinder.parsing.java.InvocationVisitor	220	12	10	48	17	5	3
com.softwarepathfinder.visualization.prefuse.Example	81	2	3	1	22	0	0
com.softwarepathfinder.model.Invocation	75	14	14	14	9	1	6
com.softwarepathfinder.parsing.php.GenerateAstPhp	170	17	1	32	17	12	2
oftwarepathfinder.visualization.prefuse.AggregateDragControl	300	11	8	19	40	5	10
com.softwarepathfinder.model.Method	146	26	26	28	14	2	13
oftwarepathfinder.visualization.prefuse.NamespaceGraphByPath	115	0	1	10	32	6	0
com.softwarepathfinder.visualization.prefuse.MethodGraph	115	0	1	10	31	4	0
com.softwarepathfinder.parsing.java.TypeVisitor	91	5	5	16	14	5	1
com.softwarepathfinder.visualization.prefuse.SimpleGraph	80	0	1	2	24	0	2
com.softwarepathfinder.parsing.java.GenerateJava2JPA	154	8	1	16	16	8	3
com.softwarepathfinder.parsing.PathMaker	59	2	1	8	10	7	1
com.softwarepathfinder.parsing.php.TypeVisitor	97	7	6	15	18	4	4

Processing time: 899 milliseconds

C:\Program Files\cmdr

λ |





```
λ drtools-metric D:\JavaApps\Doutorado\PathFinder\softwarepathfinder\src -m --console --top 30
```

METHODS	MLOC	CYCLO	CALLS	NBD	PARAM
com.softwarepathfinder.visualization.prefuse.NamespaceGraphByPath.main(String[] argv)	125	10	66	6	1
com.softwarepathfinder.visualization.prefuse.NamespaceGraph.main(String[] argv)	130	11	69	6	1
.InvocationVisitor.identifyStaticMethodInvocation(Identifier parentReference, Identifier methodName)	13	5	9	3	2
com.softwarepathfinder.visualization.prefuse.TypeGraph.main(String[] argv)	151	8	69	5	1
PathMaker.calcPath(Type type, Path path, int level, HashMap<String,Locus> pathLocs, EntityManager em)	26	6	19	4	5
com.softwarepathfinder.model.Project.getTypeByName(Type typeParent, String typeName)	52	12	25	6	2
com.softwarepathfinder.parsing.java.TypeVisitor.visit(MethodDeclaration node)	42	8	31	3	1
com.softwarepathfinder.visualization.prefuse.PathMethodsGraph.main(String[] argv)	164	6	93	3	1
com.softwarepathfinder.visualization.yEd.TypeGraphML.build()	69	10	28	2	0
com.softwarepathfinder.model.Type.getConcretChildren(Type parent, List<Type> types)	16	5	10	2	2
com.softwarepathfinder.visualization.prefuse.PathMethodLayout.run(double frac)	67	12	21	4	1
com.softwarepathfinder.parsing.php.InvocationVisitor.visit(FunctionInvocation node)	19	4	12	3	1
com.softwarepathfinder.parsing.java.InvocationVisitor.visit(VariableDeclarationStatement node)	14	4	10	7	1
com.softwarepathfinder.parsing.java.InvocationVisitor.visit(ArrayInitializer node)	16	5	18	7	1
com.softwarepathfinder.parsing.java.InvocationVisitor.visit(Assignment node)	20	4	15	7	1
com.softwarepathfinder.visualization.yEd.GraphmlBuilder.main(String[] args)	55	4	40	2	1
com.softwarepathfinder.model.Type.hasMethod(String methodName)	11	4	12	2	1
com.softwarepathfinder.visualization.prefuse.MethodGraph.main(String[] argv)	122	10	61	5	1
com.softwarepathfinder.parsing.java.InvocationVisitor.visit(MethodInvocation node)	60	13	33	7	1
com.softwarepathfinder.visualization.jgraph.PathView.createEdges(Object pathNode, Path path)	13	4	13	3	2
com.softwarepathfinder.parsing.java.InvocationVisitor.visit(SuperConstructorInvocation node)	15	4	5	7	1
com.softwarepathfinder.parsing.java.InheritanceVisitor.makeInterface(TypeDeclaration node)	15	5	8	3	1
com.softwarepathfinder.parsing.java.InheritanceVisitor.makeParent(TypeDeclaration node)	9	5	8	3	1
com.softwarepathfinder.parsing.php.InvocationVisitor.identifyFunctionInvocation(Method method)	20	7	10	3	1
com.softwarepathfinder.parsing.php.InvocationVisitor.identifyClassInstanceCreation(Identifier newCall)	14	6	7	3	1
com.softwarepathfinder.parsing.java.InvocationVisitor.visit(SuperMethodInvocation node)	14	4	7	7	1
com.softwarepathfinder.model.Locus.getTargets(Invocation invocation, Locus locus)	29	4	18	1	2
com.softwarepathfinder.parsing.php.InvocationVisitor.doCastNodesOf(Expression identifiedExpression)	32	5	6	3	1
com.softwarepathfinder.parsing.php.InvocationVisitor.identifyPolymorphicCall(String methodName, Type typeCaller)	12	5	4	3	2
com.softwarepathfinder.parsing.java.InvocationVisitor.visit(CastExpression node)	22	4	14	7	1

Processing time: 1 seconds

C:\Program Files\cmdr

λ |



**metric  
visualization**  
**DR-Tools**

# Uso



## Metric Visualization

A tool quality suite to help the developers to maintain health and code evolution

PROJECT SUMMARY

### Software Pathfinder

[View Thresholds](#)

**13**

Namespaces

**4665**

SLOC

**65**

Number of Types

**402**

Number of Methods

**DR-Tools**

# Uso



## Thermometer

Summary Visualization



## Namespaces

Using NOC and NAC



## Types

Using NOM, SLOC, and WMC



## Methods

Using CYCLO, MLOC, and CALLS



## Internal Dependencies

Internal dependencies of types/classes



## Type Coupling

Coupling between types/classes (input/output)



## Namespace Coupling

Using CA and CE



## Instability/Abstractness/Distance

Using LA and D



## Instability and Abstractness

Using LA and A



**DR-Tools**



## Metric Thresholds Information

Project: Software Pathfinder

[Back](#)

### PROJECT

Acronym	Name	Description
<b>SMALL</b>	Small Project	small project with < 50 KLOC or 200 < classes
<b>MEDIUM</b>	Medium Project	medium project with (50 KLOC <= project <= 250 KLOC) or (200 <= classes <= 1000)
<b>LARGE</b>	Large Project	large project with > 250 KLOC or > 1000 classes

### NAMESPACE

Acronym	Name	Description
<b>NOC</b>	Number of Types/Classes	Good: <= 11; Regular: between 11 and 28; Bad: > 28
<b>NAC</b>	Number of Abstract Types/Classes	without references

### TYPE

Acronym	Name	Description
<b>SLOC</b>	Type/Class Line of Code	Bad: > 500



**DR-Tools**

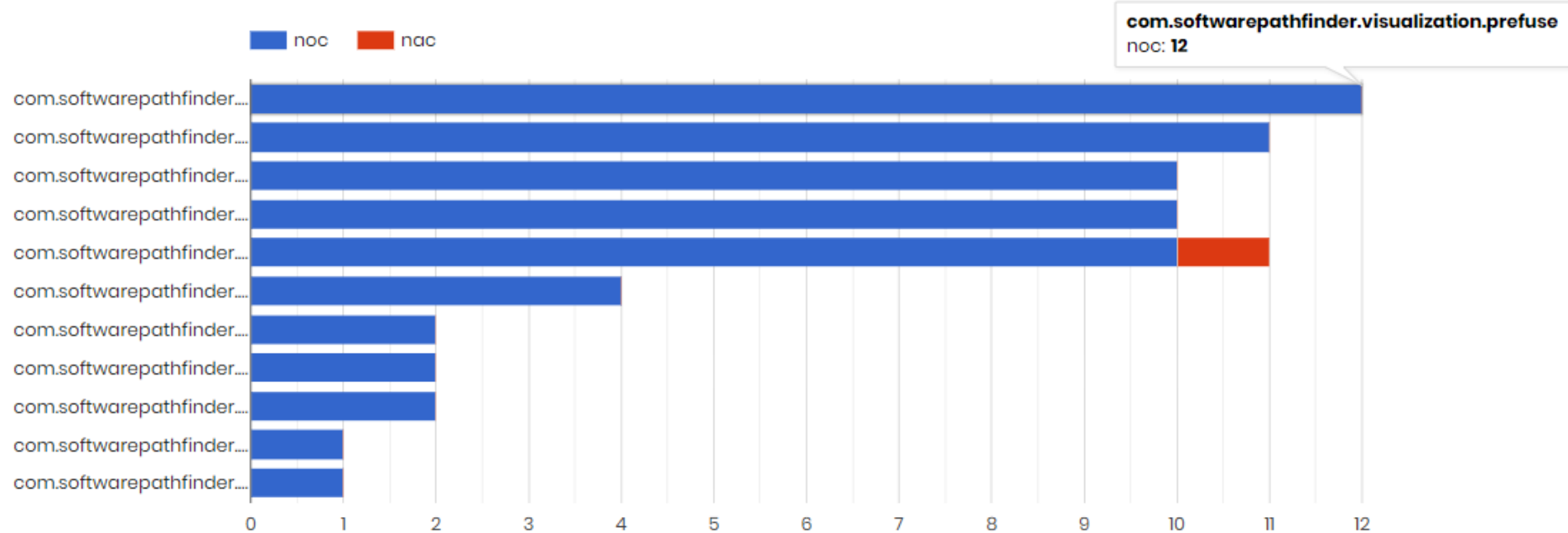


## Namespace Visualization

NOC (Number of Classes/Types) and NAC (Number of Abstract Classes/Types)

Project: Software Pathfinder

[Back](#)



DR-Tools

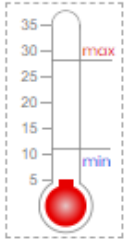


## Thermometer Visualization

Project: Software Pathfinder

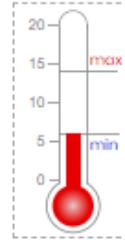
[Back](#)

### Types (types/namespaces)



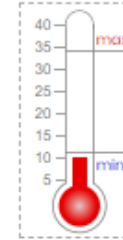
Total of Namespaces: 13  
Total of Types: 65  
Types/namespaces: 5  
Total of SLOC: 4665

### Methods (methods/types)



Total of Methods: 402  
Methods/Types: 6

### Complexity (WMC/types)



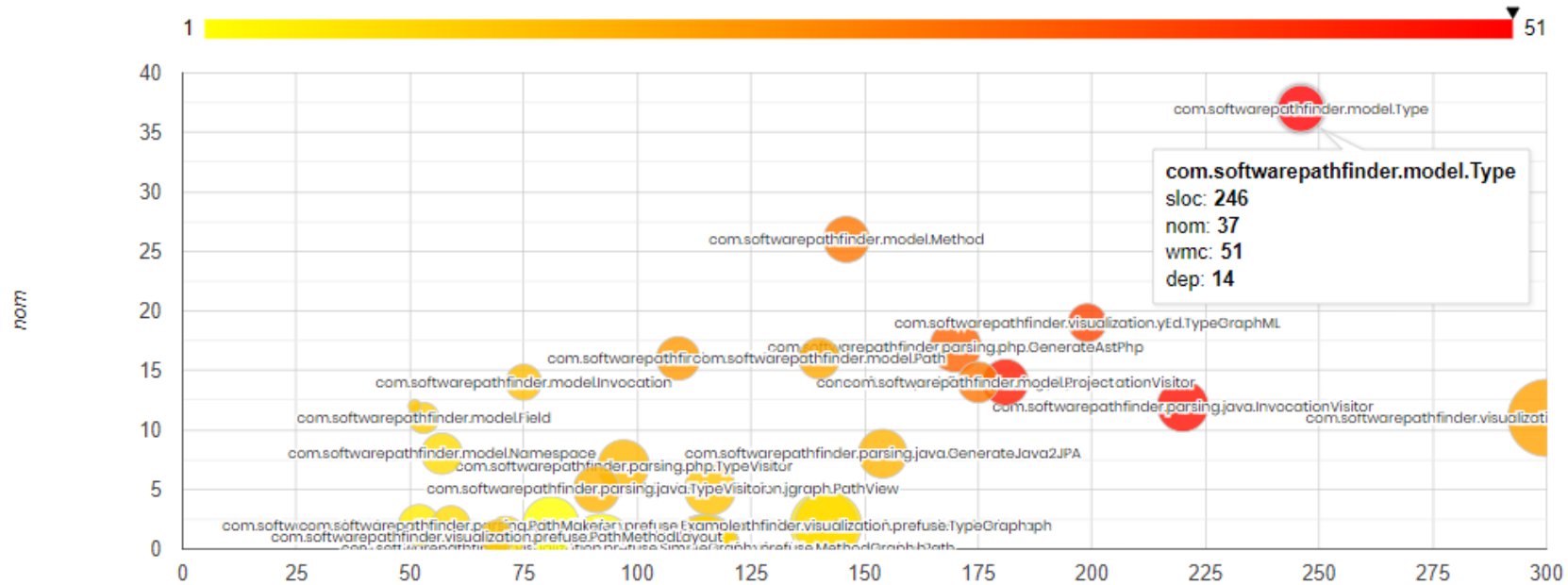
Total of Complexity: 673  
Complexity/Types: 10



## Type Visualization

Types with Number of Methods/Functions (NOM - y axis), Lines of Code (SLOC - x axis), Complexity (WMC - bubble color), and Dependencies (DEP - bubble size)

Project: Software Pathfinder

[Back](#)

## DR-Tools

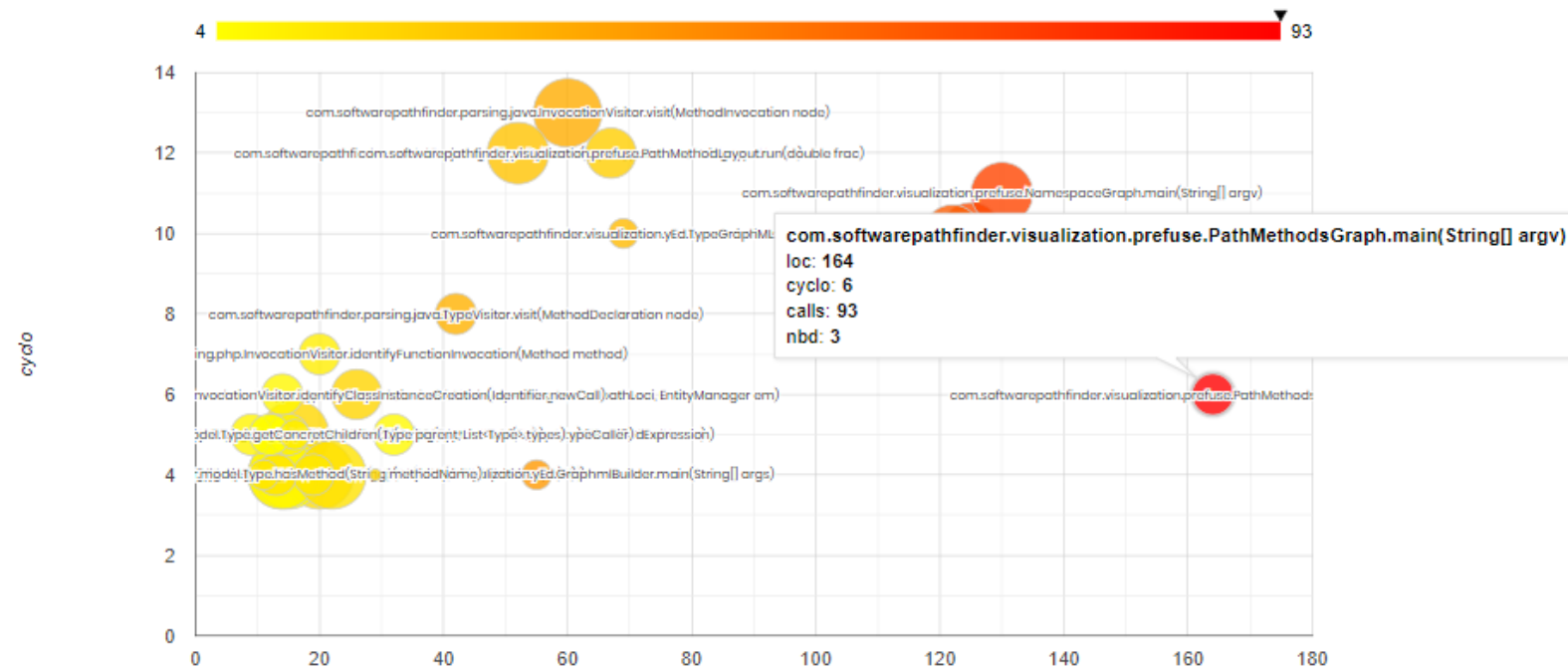


## Method Visualization

Methods with Complexity (CYCLO - y axis), Lines of Code (MLOC - x axis), Number of Invocations (CALLS - bubble color), and Nested Block Depth (NBD - bubble size)

Project: Software Pathfinder

[Back](#)



DR-Tools

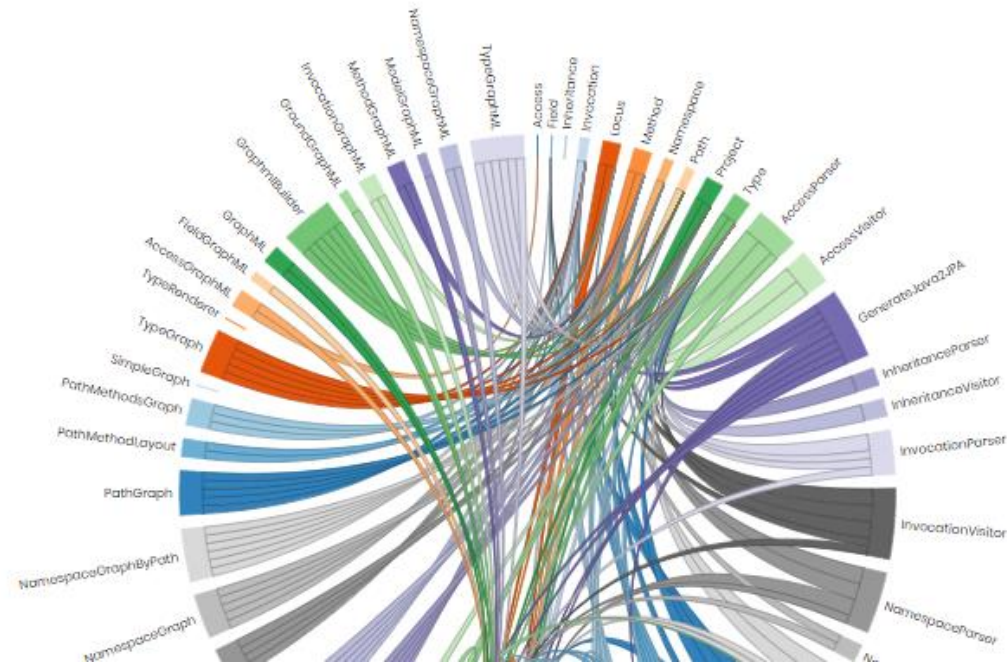


## Internal Dependencies Visualization

Internal dependencies between type/classes

Project: Software Pathfinder

[Back](#)



**DR-Tools**

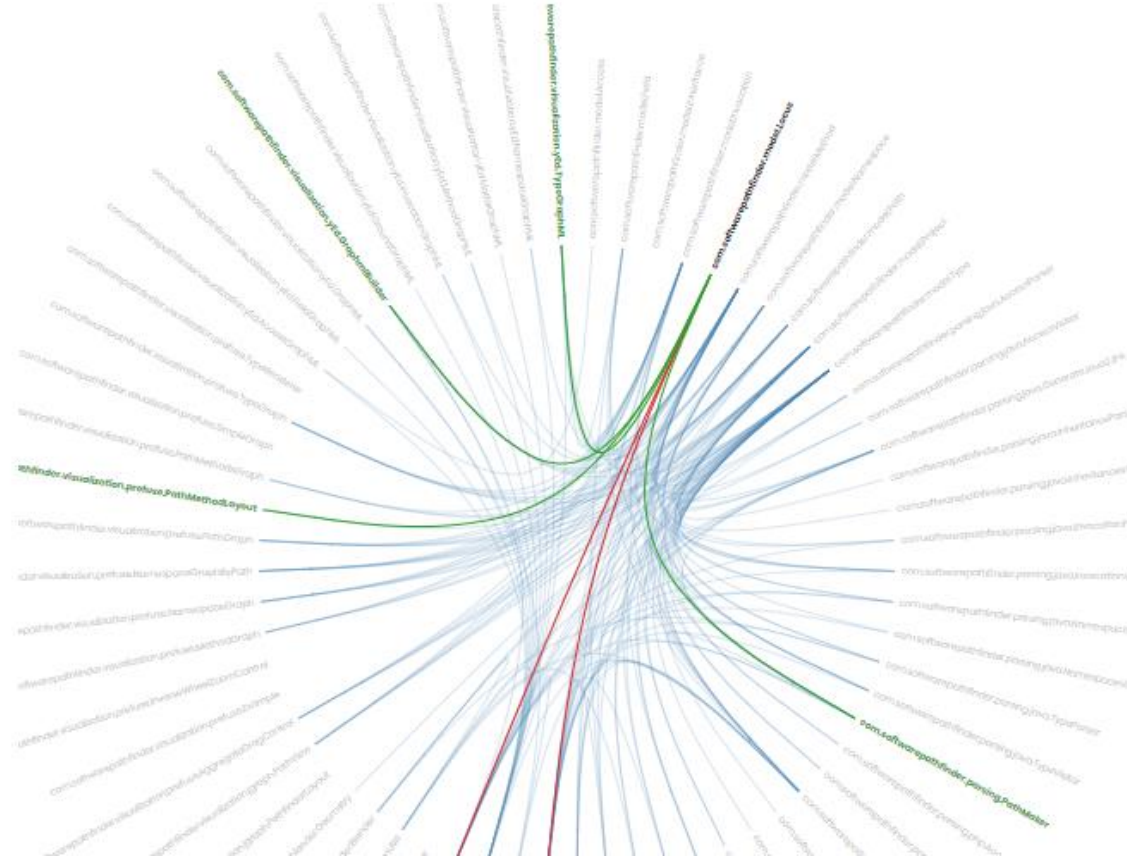
# Uso

## Coupling (Input and Output) Visualization

Red lines (output coupling) and green lines (input coupling)

Project: Software Pathfinder

[Back](#)

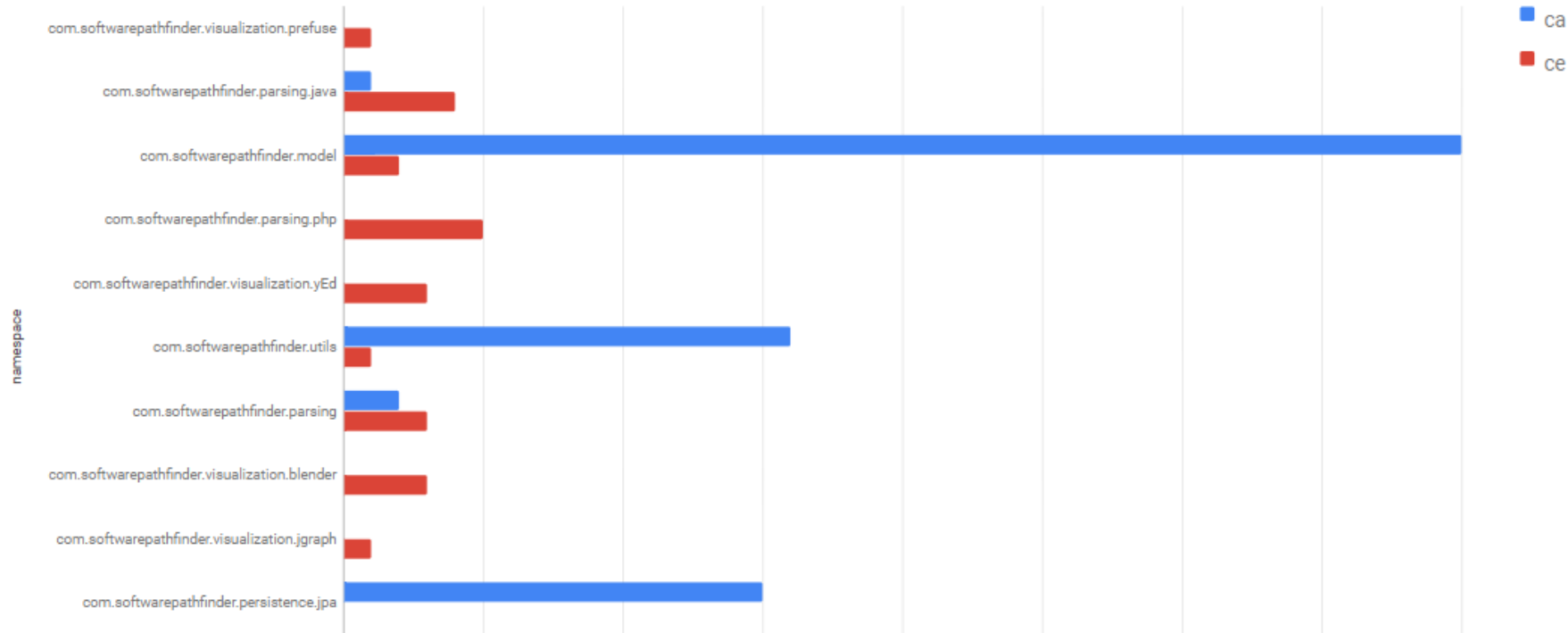


## Namespace Coupling Visualization

CA (Afferent Coupling) and CE (Efferent Coupling)

Project: Software Pathfinder

[Back](#)







# Instability/Abstractness/Distance Visualization

I (Instability), A (Abstractness Degree), and D (Normalized Distance)

Project: Software Pathfinder

[Back](#)



DR-Tools

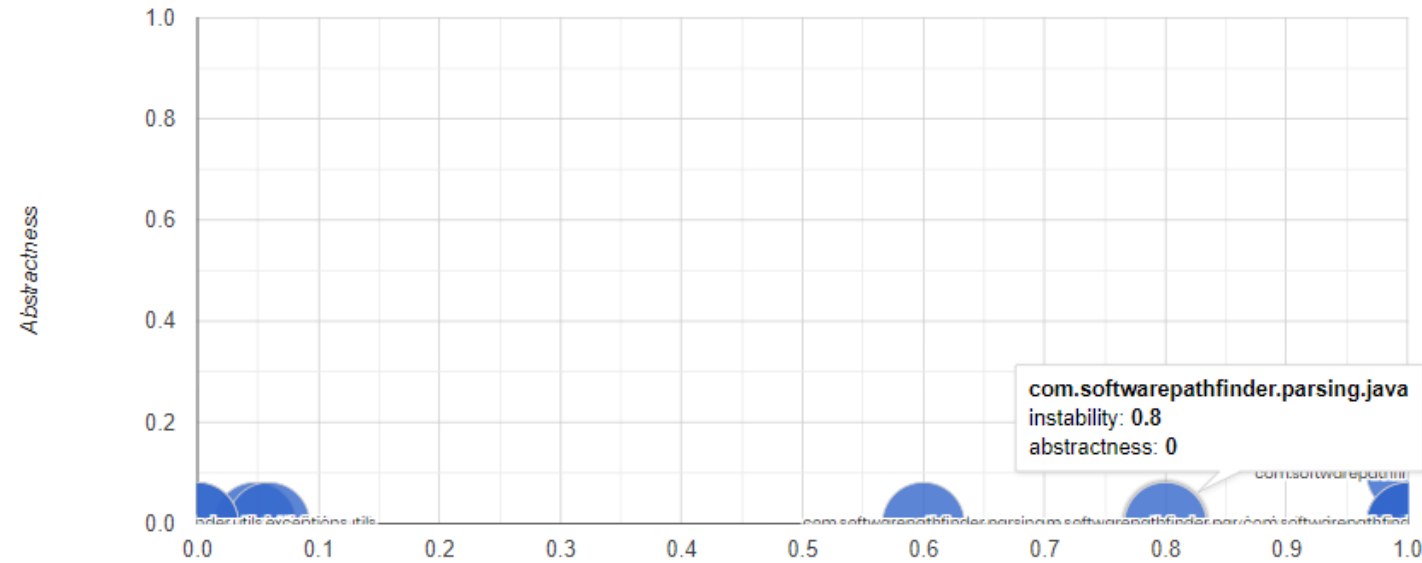


## Instability and Abstractness Visualization

Abstractness degree (y axis) and Instability (x axis)

Project: Software Pathfinder

[Back](#)





# DR-Tools

---

36 Heurísticas de Análise

**[drtools.dev](https://drtools.dev)**

# Heurísticas - Summary

- 1. Contexto que fornece informações gerais sobre dimensões do Projeto**  
*Indicativo de filtrar melhor as informações usando a opção '--top X' para ajudar no entendimento (Classificação: SMALL, MEDIUM e LARGE)*
- 2. Considere o número médio de classes por namespace**  
*Indicativo de que as classes não estão distribuídas uniformemente*
- 3. Avalie o número médio de SLOC por classes**  
*Indicativo de classes muito grandes*
- 4. Observe a distribuição média de métodos por classes**  
*Indicativo de muitos comportamentos por classe*
- 5. Considere a complexidade média por classes**  
*Indicativo de como está a complexidade das classes em geral*



# Heurísticas - Namespaces

## 6. Observe a distribuição de classes por namespace

*Se um namespace tem muitas classes (NOC alto), pode ser um indicativo de 'promiscuous package'*

## 7. Avalie a distribuição de tipos abstratos (classes abstratas, interfaces) por namespaces

*Indicativos para extensão e reuso*

## 8. Avalie a relação das métricas NOC e NAC do namespace

*Uma diferença muito grande entre eles pode indicar uma má distribuição entre tipos abstratos e tipos concretos*



# Heurísticas – Types (1)

## 9. Avalie as métricas além do SLOC

*WMC, DEPS (DEP e I-DEP) e NOM/NPM são bons indicativos de como está a classe*

## 10. Classe com NOA alto, mas baixo WMC e NOM alto

*Pode ser um indicativo de POJO (Plain Old Java Object)*

## 11. SLOC alto, mas sem muitos métodos (NOM/NPM baixo)

*Pode ser um indicativo de 'long methods'*

## 12. SLOC e WMC alto, mas sem muitos métodos (NOM/NPM baixo)

*Pode ser um indicativo de 'complex class'*

## 13. NOM/NPM alto pode ser indicativo de classe com muitas responsabilidades

*Indica baixa coesão e possivelmente 'god class'*





# Heurísticas – Types (2)

- 14.** **NOM/NPM alto e NOA baixo pode ser indicativo de classe com muitas responsabilidades**  
*Pode ser um indicativo de uma classe 'controller'*
- 15.** **NOM alto e NPM baixo pode indicar que o comportamento foi dividido**  
*Indicativo de métodos private/protected/default*
- 16.** **NOA alto pode ser indicativo de classe com muitas responsabilidades**  
*Pode ser um indicativo de baixa coesão, dificultando a manutenção*
- 17.** **DEP alto e I-DEP baixo pode indicar uma classe com muitas dependências externas**  
*Dependências de APIs externas (frameworks, libs)*
- 18.** **I-DEP alto (e por consequência DEP alto), pode indicar uma classe com muitas dependências de classe do projeto**  
*Incidência de alto acoplamento*



# Heurísticas – Methods

- 19.** PARAM alto pode ser indicativo de método com baixa coesão  
*Possivelmente é um 'long method'*
- 20.** CYCLO alto e MLOC baixo pode ser um 'complex method'  
*Indicativo de problema de complexidade, legibilidade e entendimento*
- 21.** NBD alto pode ser um 'complex/long method'  
*Indicativo de problema de complexidade, legibilidade e entendimento*
- 22.** CALLS alto pode indicar alto acoplamento  
*Indicativo de problema de várias dependências*
- 23.** MLOC alto, CYCLO alto, CALLS alto e NBD alto é forte indicativo de mais de um problema  
*Pode ser um indicativo de um 'complex/long method'*



# Heurísticas – Coupling (1)

**24. Evite dependência cíclicas**

*Tornam as mudanças complexas e gera a 'síndrome da compilação total'*

**25. CA alto pode indicar que o namespace é estável**

*Se ele mudar, vai fazer com que quem dependa dele seja alterado*

**26. CE alto pode indicar que o namespace é instável**

*A incidência de mudança em outros namespaces que ele depende vai fazer com que ele mude*

**27. I indica como está a instabilidade do namespace**

*I=0, namespace estável ao máximo; I=1, namespace instável ao máximo*



# Heurísticas – Coupling (2)

- 28.** Se  $I=0$ , indica que  $CA > 0$  e  $CE=0$ , indica uma estabilidade total  
*Ele é responsável e independente. Os dependentes tornam difícil alterá-lo e não tem dependência de outros que pode forçar a mudança*
- 29.** A indica como está o grau de abstração do namespace  
*A=0, namespace não tem tipos abstratos; A=1, namespace somente possui tipos abstratos*



# Heurísticas – Coupling (3)

**30. Avalie namespaces estáveis e abstratos**

*ficam no canto superior esquerdo ( $I=0; A=1$ )*

**31. Avalie namespaces instáveis e concretos**

*ficam no canto inferior direito ( $I=1; A=0$ )*

**32. Avalie namespaces estáveis e concretos**

*ficam no canto inferior esquerdo ( $I=0; A=0$ )*

*Namespaces rígidos, não podem ser estendidos e difíceis de mudar. Exemplos: ORM e utils*

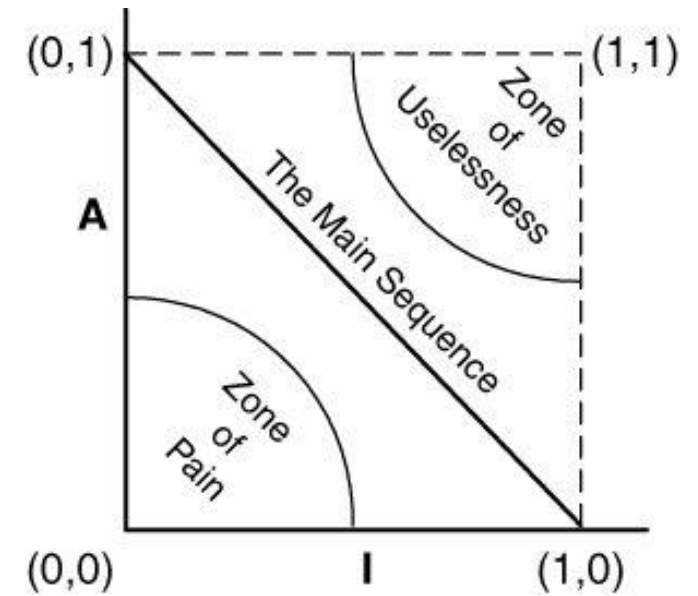
**33. Avalie namespaces instáveis e abstratos**

*ficam no canto superior direito ( $I=1; A=1$ )*

*Namespaces abstratos e sem dependentes, sem utilidade*

**34. Considere namespaces que estão nas zonas de exclusão**

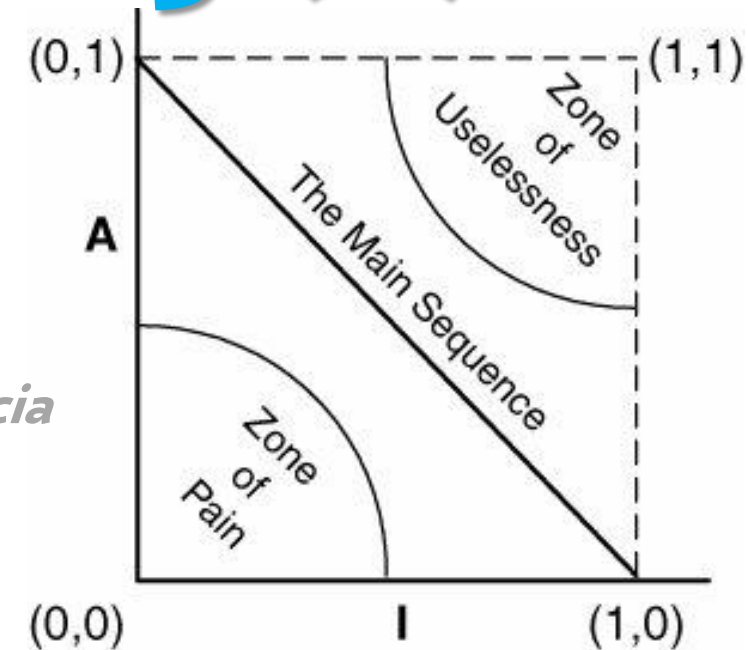
*Zone of Pain e Zone of Uselessness*



# Heurísticas – Coupling (4)

- 35.** Namespace situado próximo a sequência principal indica que não é abstrato nem instável demais

*Valor de  $D$  (entre 0 e 1) que vai indicar a posição na sequência principal*



- 36.**  $D$  indica o quanto longe um namespace está da sequência principal

*$D$  próximo a 0 indica proximidade da sequência principal;  $D$  próximo a 1, indica distância da sequência principal*

*Estes valores (mais próximo a 1) podem indicar quando um namespace está passível de manutenção e menos sensível a mudanças*



# Live demo!



**Questões??**



# DR-Tools

A tool quality suite to help the  
developers to maintain health and code evolution

**[drtools.dev](https://drtools.dev)**