Understand and Use Software Metrics

Confoo.ca

Kore Nordmann (@koredn)

27. Feb 2013

About me

- Degree in computer sience

About me

- Degree in computer sience
- Professional PHP since 2000

About me

- Degree in computer sience
- Professional PHP since 2000
- Open source enthusiast

About me

- Degree in computer sience
- Professional PHP since 2000
- Open source enthusiast
- Passion for
  - Software Design
  - Automated Testing

Co-founder of

Co-founder of

Helping people to create high quality web applications.

`http://qafoo.com`

Co-founder of

Helping people to create high quality web applications.

`http://qafoo.com`

- Expert consulting
- Individual training

Co-founder of

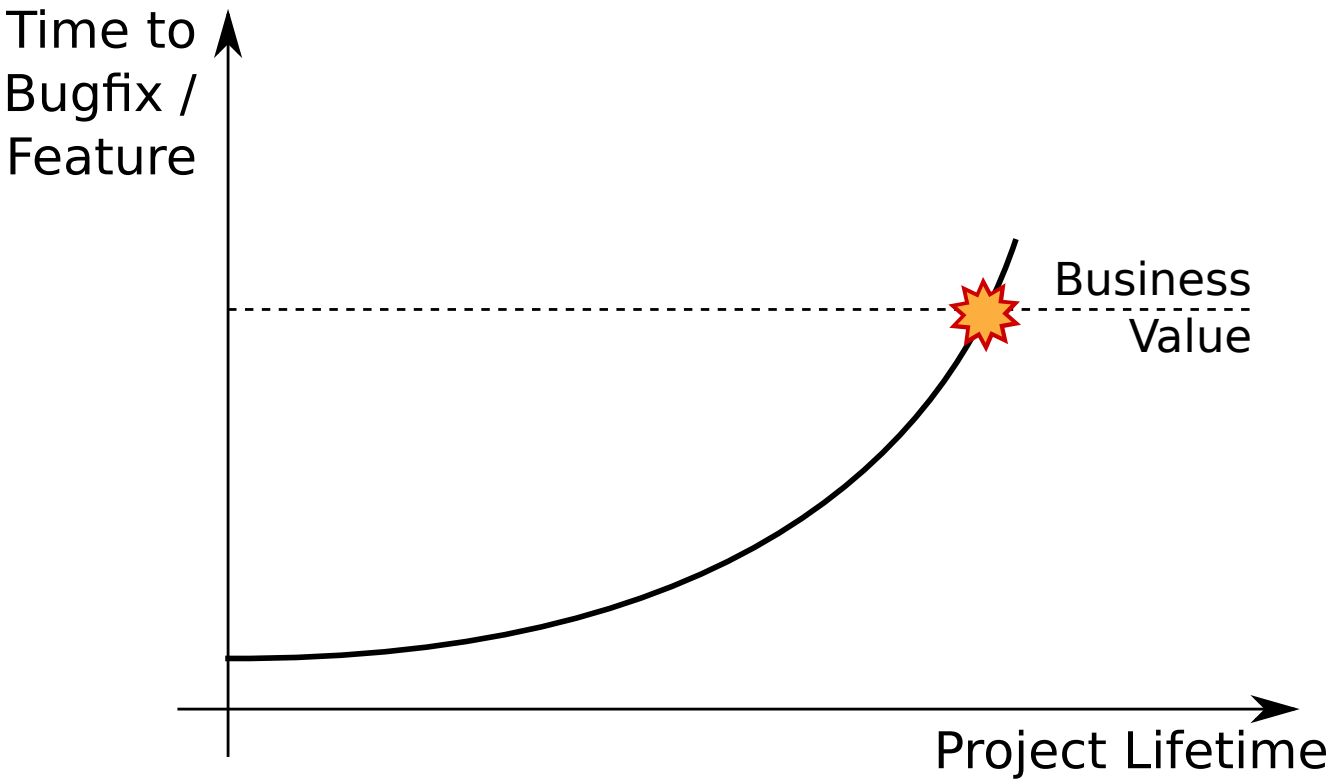
Helping people to create high quality web applications.

`http://qafoo.com`

- Expert consulting
- Individual training

Get a training on object oriented design for your team!

Why quality?



Software metrics

"A software metric is a measure of some property of a piece of software or its specifications"
(Wikipedia)

Applications

- Code Review
  - Find weak spots
  - Find high impact code

Applications

- Code Review
  - Find weak spots
  - Find high impact code
- Measure Progress
  - Watch change rate over time
  - Watch quality over time

Outline

Classic software metrics

Object oriented software metrics

Conclusion

Scale metrics

- How big is my project?

Scale metrics

- How big is my project?
  - Lines Of *

LOC	Lines Of Code
ELOC	Executable Lines Of Code
CLOC	Comment Lines Of Code
NCLOC	Non-Comment Lines Of Code

Scale metrics

- How big is my project?
  - Lines Of *

LOC	Lines Of Code
ELOC	Executable Lines Of Code
CLOC	Comment Lines Of Code
NCLOC	Non-Comment Lines Of Code

  - Number Of *

NOC	Number Of Classes
NOM	Number Of Methods
NOP	Number Of Packages

Lines Of *, Number Of *

```php
<?php
namespace foo\bar;

abstract class FooBar {
    abstract function bar();
}

class Foo extends FooBar {
    /* Does this ... */
    public function bar() {
        return;
    }

    /* Does that ... */
    public function baz() {
        // Comment
        return;
    }
}

class Bar extends Foo {
    public function foo(Foo $f) {
        return;
    }
}
```

- Lines Of *
- Number Of *

Lines Of *, Number Of *

```php
<?php
namespace foo\bar;

abstract class FooBar {
    abstract function bar();
}

class Foo extends FooBar {
    /* Does this ... */
    public function bar() {
        return;
    }

    /* Does that ... */
    public function baz() {
        // Comment
        return;
    }
}

class Bar extends Foo {
    public function foo(Foo $f) {
        return;
    }
}
```

- Lines Of *

  LOC 24

- Number Of *

Lines Of *, Number Of *

```php
<?php
namespace foo\bar;

abstract class FooBar {
    abstract function bar();
}

class Foo extends FooBar {
    /* Does this ... */
    public function bar() {
        return;
    }

    /* Does that ... */
    public function baz() {
        // Comment
        return;
    }
}

class Bar extends Foo {
    public function foo(Foo $f) {
        return;
    }
}
```

- Lines Of *

  LOC 24

  ELOC 3

- Number Of *

Lines Of *, Number Of *

```php
<?php
namespace foo\bar;

abstract class FooBar {
    abstract function bar();
}

class Foo extends FooBar {
    /* Does this ... */
    public function bar() {
        return;
    }

    /* Does that ... */
    public function baz() {
        // Comment
        return;
    }
}

class Bar extends Foo {
    public function foo(Foo $f) {
        return;
    }
}
```

- Lines Of *

  LOC 24
  ELOC 3
  CLOC 3

- Number Of *

Lines Of *, Number Of *

```php
<?php
namespace foo\bar;

abstract class FooBar {
    abstract function bar();
}

class Foo extends FooBar {
    /* Does this ... */
    public function bar() {
        return;
    }

    /* Does that ... */
    public function baz() {
        // Comment
        return;
    }
}

class Bar extends Foo {
    public function foo(Foo $f) {
        return;
    }
}
```

- Lines Of *

LOC	24
ELOC	3
CLOC	3
NCLOC	21

- Number Of *

Lines Of *, Number Of *

```php
<?php
namespace foo\bar;

abstract class FooBar {
    abstract function bar();
}

class Foo extends FooBar {
    /* Does this ... */
    public function bar() {
        return;
    }

    /* Does that ... */
    public function baz() {
        // Comment
        return;
    }
}

class Bar extends Foo {
    public function foo(Foo $f) {
        return;
    }
}
```

- Lines Of *

LOC	24
ELOC	3
CLOC	3
NCLOC	21

- Number Of *

NOC	3

Lines Of *, Number Of *

```php
<?php
namespace foo\bar;

abstract class FooBar {
    abstract function bar();
}

class Foo extends FooBar {
    /* Does this ... */
    public function bar() {
        return;
    }

    /* Does that ... */
    public function baz() {
        // Comment
        return;
    }
}

class Bar extends Foo {
    public function foo(Foo $f) {
        return;
    }
}
```

- Lines Of *

LOC	24
ELOC	3
CLOC	3
NCLOC	21

- Number Of *

NOC	3
NOM	4

Lines Of *, Number Of *

```php
<?php
namespace foo\bar;

abstract class FooBar {
    abstract function bar();
}

class Foo extends FooBar {
    /* Does this ... */
    public function bar() {
        return;
    }

    /* Does that ... */
    public function baz() {
        // Comment
        return;
    }
}

class Bar extends Foo {
    public function foo(Foo $f) {
        return;
    }
}
```

- Lines Of *

LOC	24
ELOC	3
CLOC	3
NCLOC	21

- Number Of *

NOC	3
NOM	4
NOP	1

Run yourself

```
$ pear install pear.phpunit.de/phploc
$ phploc src/main/

Lines of Code (LOC):                            4699
Comment Lines of Code (CLOC):                   1792
Non-Comment Lines of Code (NCLOC):              2907

Namespaces:                                       12
Interfaces:                                        1
Classes:                                          32
  Abstract:                                        4 (12.50%)
  Concrete:                                       28 (87.50%)
  Average Class Length (NCLOC):                   88
Methods:                                         135
  Scope:
    Non-Static:                                  134 (99.26%)
    Static:                                        1 (0.74%)
  Average Method Length (NCLOC):                  20
```

Run yourself

```
$ pear install pear.phpunit.de/phploc
$ phploc src/main/

Lines of Code (LOC):                            4699
Comment Lines of Code (CLOC):                   1792
Non-Comment Lines of Code (NCLOC):              2907

Namespaces:                                       12
Interfaces:                                        1
Classes:                                          32
  Abstract:                                        4 (12.50%)
  Concrete:                                       28 (87.50%)
  Average Class Length (NCLOC):                   88
Methods:                                         135
  Scope:
    Non-Static:                                  134 (99.26%)
    Static:                                        1 (0.74%)
  Average Method Length (NCLOC):                  20
```

Run yourself

```
$ pear install pear.phpunit.de/phploc
$ phploc src/main/

Lines of Code (LOC):                          4699
Comment Lines of Code (CLOC):                 1792
Non-Comment Lines of Code (NCLOC):            2907

Namespaces:                                     12
Interfaces:                                      1
Classes:                                        32
  Abstract:                                      4 (12.50%)
  Concrete:                                     28 (87.50%)
  Average Class Length (NCLOC):                 88
Methods:                                       135
  Scope:
    Non-Static:                                134 (99.26%)
    Static:                                      1 (0.74%)
  Average Method Length (NCLOC):                20
```

Complexity metrics

- How complex is my code?

Complexity metrics

- How complex is my code?
  - Control structures are the key point to complexity
    - if, elseif, for, while, foreach, catch, case, xor, and, or, &&, ||, ?:

Complexity metrics

- How complex is my code?
  - Control structures are the key point to complexity
    - if, elseif, for, while, foreach, catch, case, xor, and, or, &&, ||, ?:
- Cyclomatic Complexity (CCN)
  - Number of *branches*

Complexity metrics

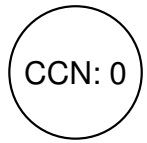
- How complex is my code?
  - Control structures are the key point to complexity
    - if, elseif, for, while, foreach, catch, case, xor, and, or, &&, ||, ?:
- Cyclomatic Complexity (CCN)
  - Number of *branches*
  - Extended Cycomatic Complexity (CCN2) actually minds all those control structures

Complexity metrics

- How complex is my code?
  - Control structures are the key point to complexity
    - if, elseif, for, while, foreach, catch, case, xor, and, or, &&, ||, ?:
- Cyclomatic Complexity (CCN)
  - Number of *branches*
  - Extended Cycomatic Complexity (CCN2) actually minds all those control structures
- NPath Complexity
  - Number of *execution paths*
  - Minds the structure of blocks

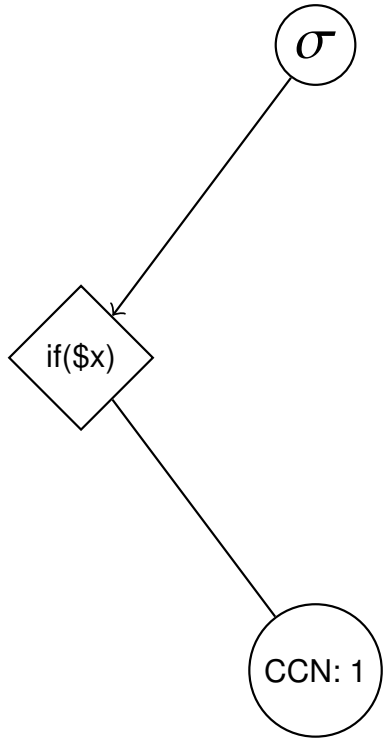
Cyclomatic Complexity


```
1   <?php
2   class Foo {
3       public function foo() {
4           if ($x) { }
5           if ($y) { }
6           if ($z) { }
7           return $x;
8       }
9   }
```



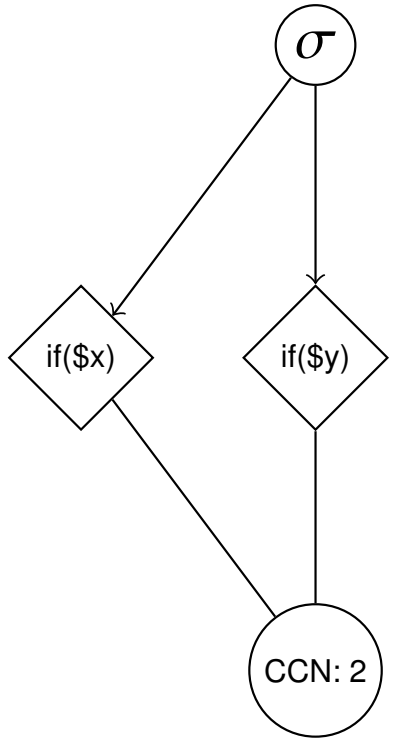
Cyclomatic Complexity

```
1  <?php
2  class Foo {
3      public function foo() {
4          if ($x) { }
5          if ($y) { }
6          if ($z) { }
7          return $x;
8      }
9  }
```



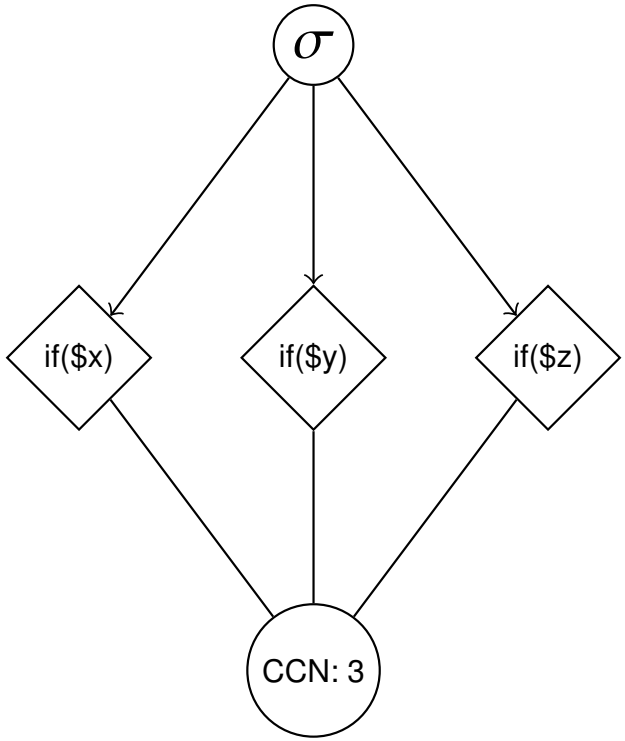
Cyclomatic Complexity

```
1  <?php
2  class Foo {
3      public function foo() {
4          if ($x) { }
5          if ($y) { }
6          if ($z) { }
7          return $x;
8      }
9  }
```



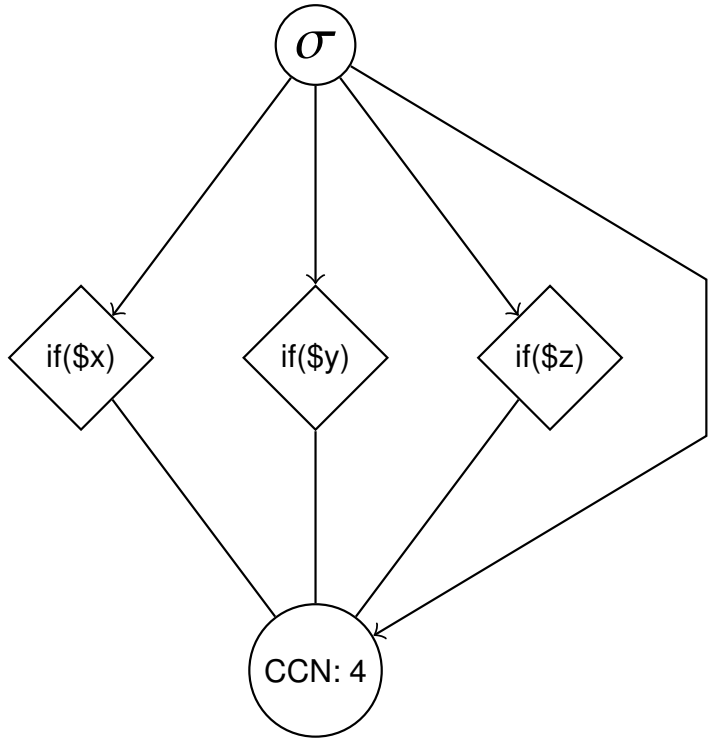
Cyclomatic Complexity

```
1  <?php
2  class Foo {
3      public function foo() {
4          if ($x) { }
5          if ($y) { }
6          if ($z) { }
7          return $x;
8      }
9  }
```



Cyclomatic Complexity

```php
<?php
class Foo {
    public function foo() {
        if ($x) { }
        if ($y) { }
        if ($z) { }
        return $x;
    }
}
```

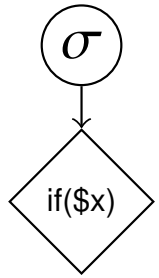


NPath Complexity


```
1   <?php
2   class Foo {
3       public function foo() {
4           if ($x) { }
5           if ($y) { }
6           if ($z) { }
7           return $x;
8       }
9   }
```

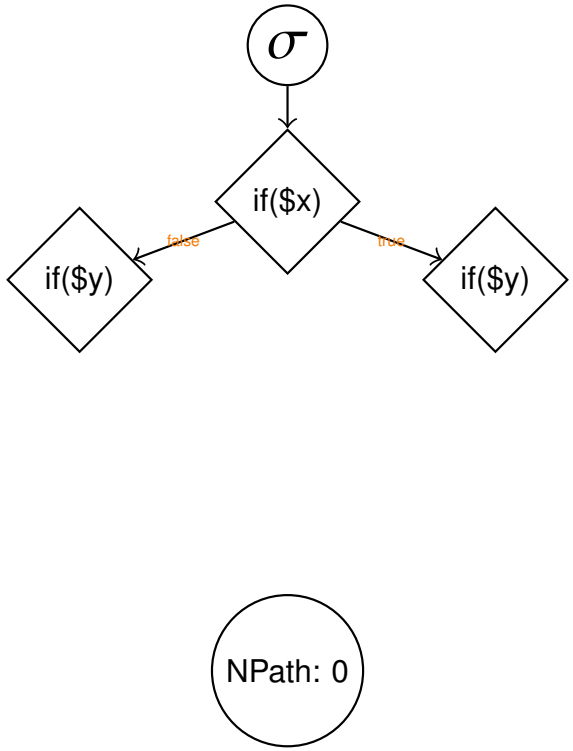
NPath Complexity

```
1  <?php
2  class Foo {
3      public function foo() {
4          if ($x) { }
5          if ($y) { }
6          if ($z) { }
7          return $x;
8      }
9  }
```



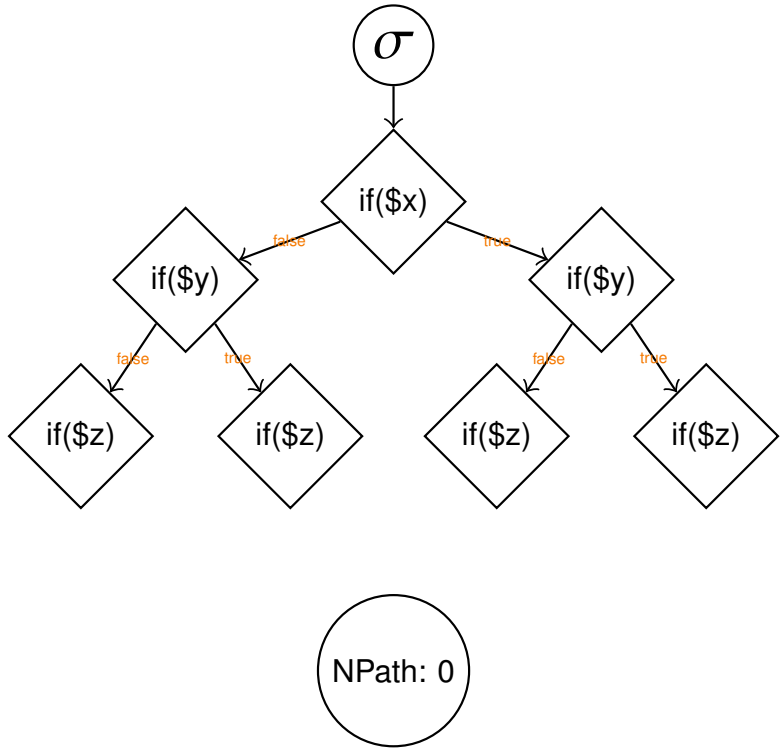
NPath Complexity

```php
<?php
class Foo {
    public function foo() {
        if ($x) { }
        if ($y) { }
        if ($z) { }
        return $x;
    }
}
```



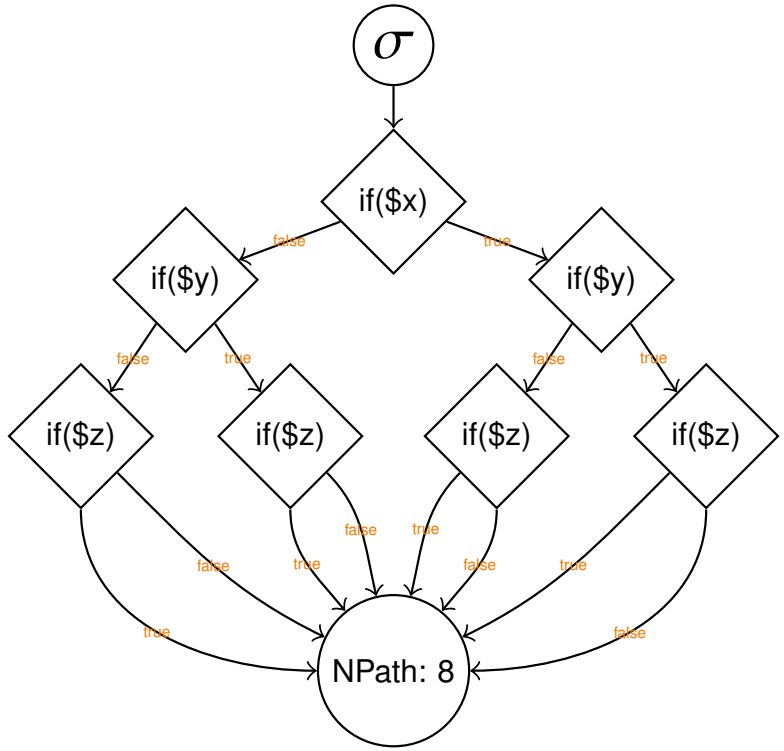
NPath Complexity

```
1  <?php
2  class Foo {
3      public function foo() {
4          if ($x) { }
5          if ($y) { }
6          if ($z) { }
7          return $x;
8      }
9  }
```



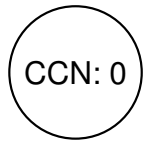
NPath Complexity

```php
<?php
class Foo {
    public function foo() {
        if ($x) { }
        if ($y) { }
        if ($z) { }
        return $x;
    }
}
```



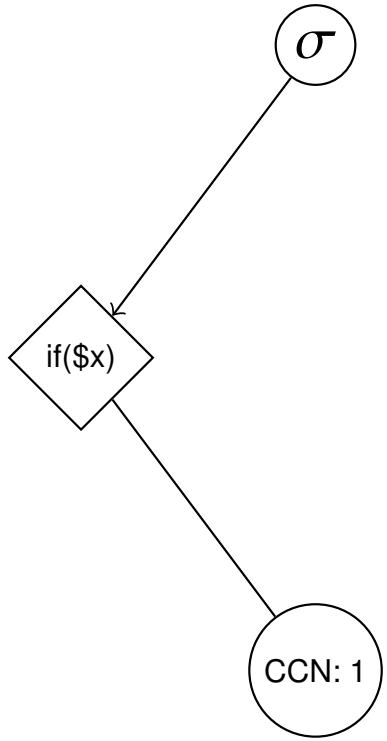
Cyclomatic Complexity


```
1   <?php
2   class Foo {
3       public function foo() {
4           if ($x) {
5               if ($y) {
6                   if ($z) { }
7               }
8           }
9           return $x;
10      }
11  }
```



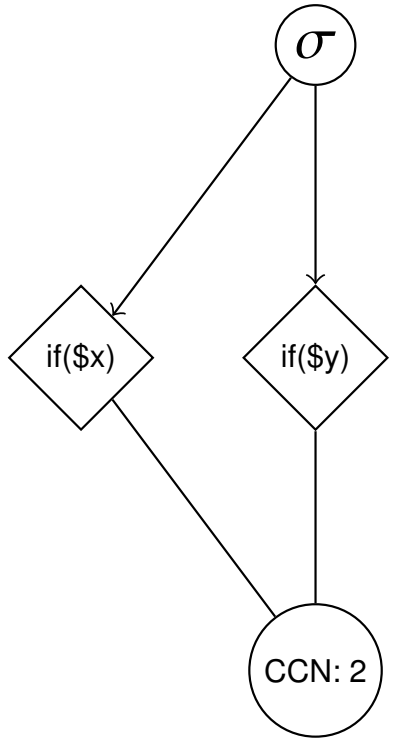
Cyclomatic Complexity

```
1  <?php
2  class Foo {
3      public function foo() {
4          if ($x) {
5              if ($y) {
6                  if ($z) { }
7              }
8          }
9          return $x;
10     }
11 }
```



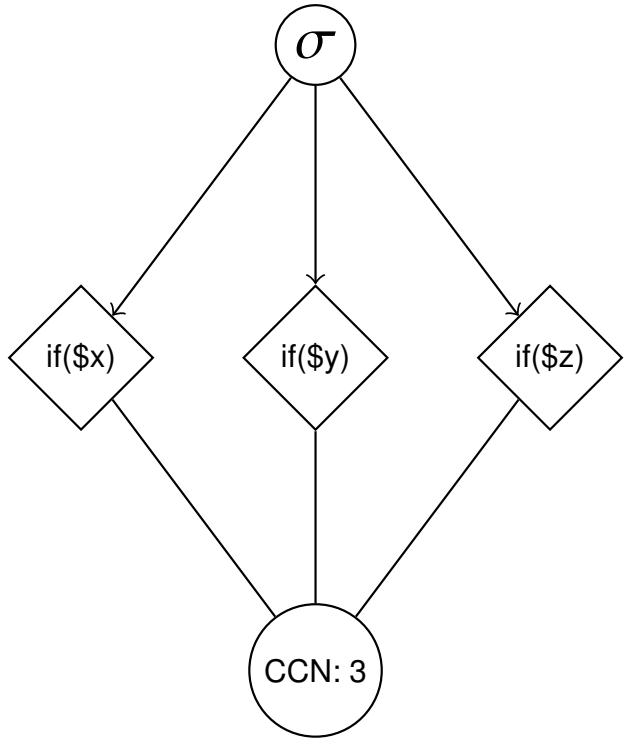
Cyclomatic Complexity

```php
<?php
class Foo {
    public function foo() {
        if ($x) {
            if ($y) {
                if ($z) { }
            }
        }
        return $x;
    }
}
```



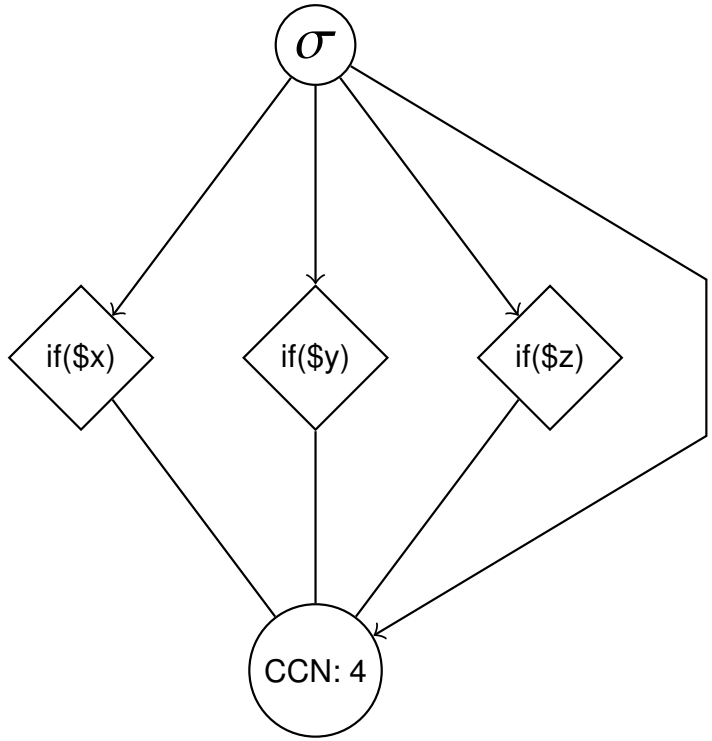
Cyclomatic Complexity

```
1  <?php
2  class Foo {
3      public function foo() {
4          if ($x) {
5              if ($y) {
6                  if ($z) { }
7              }
8          }
9          return $x;
10     }
11 }
```



Cyclomatic Complexity

```
1  <?php
2  class Foo {
3      public function foo() {
4          if ($x) {
5              if ($y) {
6                  if ($z) { }
7              }
8          }
9          return $x;
10     }
11 }
```

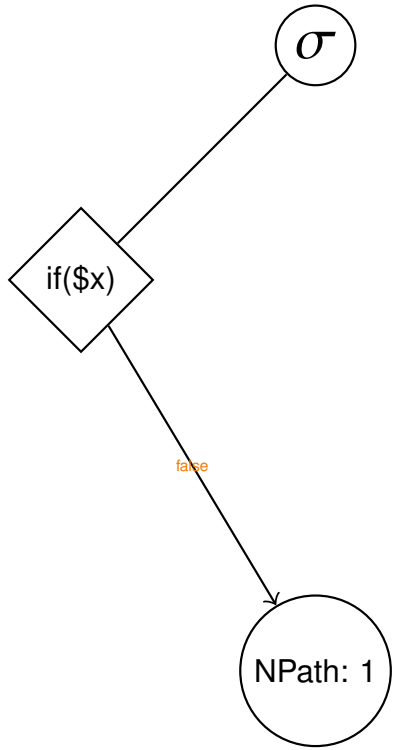


NPath Complexity


```php
<?php
class Foo {
    public function foo() {
        if ($x) {
            if ($y) {
                if ($z) { }
            }
        }
        return $x;
    }
}
```

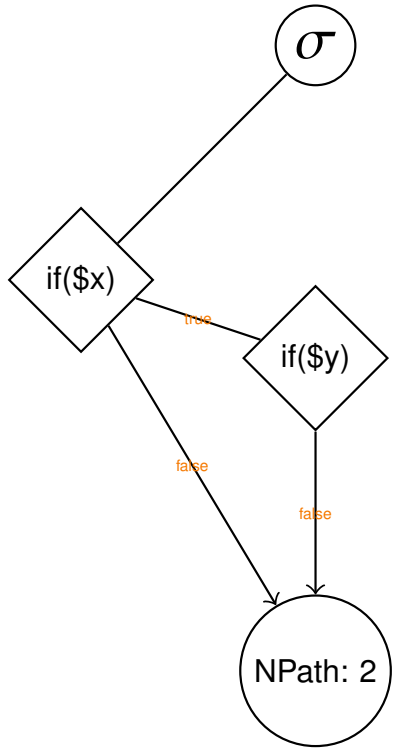
NPath Complexity

```php
<?php
class Foo {
    public function foo() {
        if ($x) {
            if ($y) {
                if ($z) { }
            }
        }
        return $x;
    }
}
```



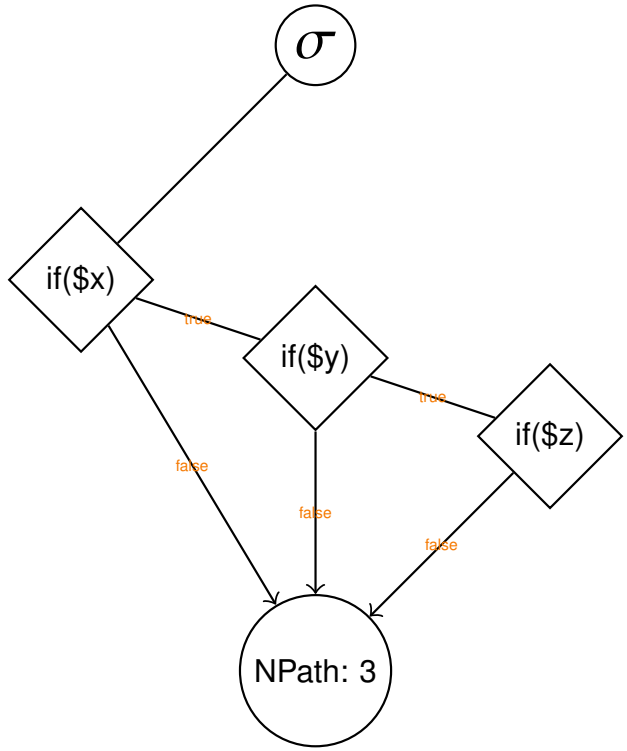
NPath Complexity

```
1   <?php
2   class Foo {
3       public function foo() {
4           if ($x) {
5               if ($y) {
6                   if ($z) { }
7               }
8           }
9           return $x;
10      }
11  }
```



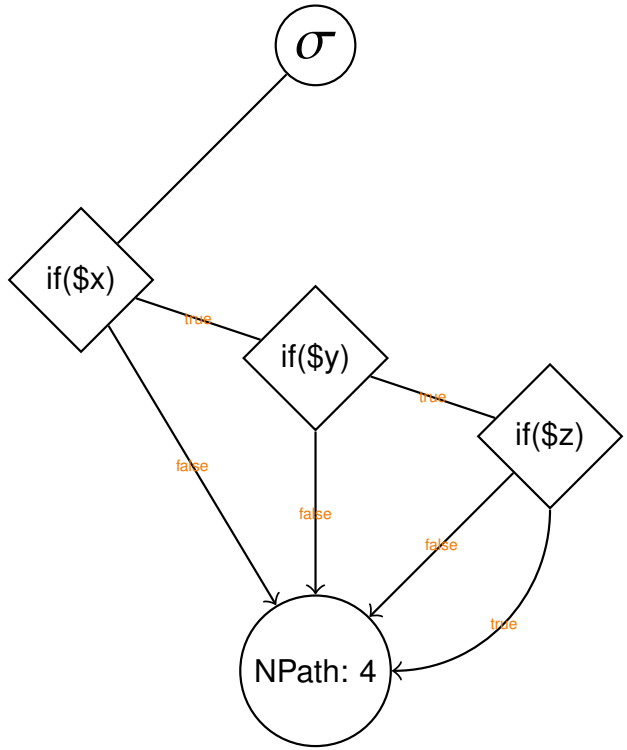
NPath Complexity

```
1   <?php
2   class Foo {
3       public function foo() {
4           if ($x) {
5               if ($y) {
6                   if ($z) { }
7               }
8           }
9           return $x;
10      }
11  }
```



NPath Complexity

```
1   <?php
2   class Foo {
3       public function foo() {
4           if ($x) {
5               if ($y) {
6                   if ($z) { }
7               }
8           }
9           return $x;
10      }
11  }
```



What do you like more?

```php
<?php
class Foo {
    public function foo() {
        if ($x) {
            if ($y) {
                if ($z) { }
            }
        }
        return $x;
    }
}
```

```php
<?php
class Foo {
    public function foo() {
        if ($x) { }
        if ($y) { }
        if ($z) { }
        return $x;
    }
}
```

Sensible limits

- Numbers do not tell anything by themselves
- To judge you need limiting values
  - Cyclomatic Complexity
    - 1-4: low, 5-7: medium, 8-10: high, 11+: hell
  - NPath Complexity
    - 200: critical mass

Sensible limits

- Numbers do not tell anything by themselves
- To judge you need limiting values
  - Cyclomatic Complexity
    - 1-4: low, 5-7: medium, 8-10: high, 11+: hell
  - NPath Complexity
    - 200: critical mass
- Limiting values are at your discretion

Code Coverage

- How many tests do I need?

Code Coverage

- How many tests do I need?
- Line Converage (supported by PHP + XDebug)
  - Shows which lines have been executed (by tests)

Code Coverage

- How many tests do I need?
- Line Converage (supported by PHP + XDebug)
  - Shows which lines have been executed (by tests)
- Path Converage (been worked on)
  - Shows which execution paths have been covered

Code Coverage

- How many tests do I need?
- Line Converage (supported by PHP + XDebug)
  - Shows which lines have been executed (by tests)
- Path Converage (been worked on)
  - Shows which execution paths have been covered
  - Write at least $nPath tests for every method

Code Coverage

- How many tests do I need?
- Line Converage (supported by PHP + XDebug)
  - Shows which lines have been executed (by tests)
- Path Converage (been worked on)
  - Shows which execution paths have been covered
  - Write at least $nPath tests for every method
- Parameter Value Coverage
  - Test all execution paths with sane boundary values for every parameter

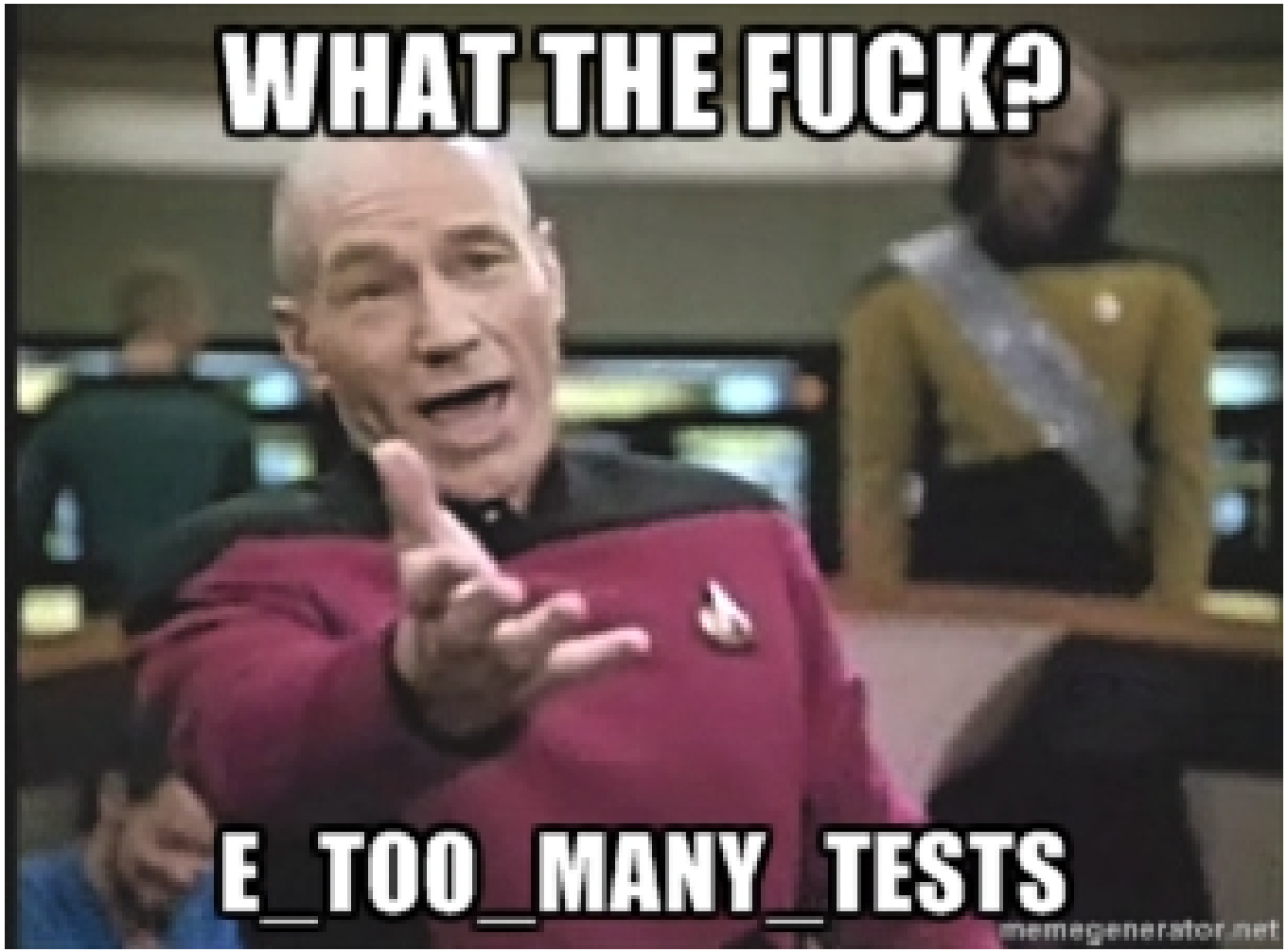
Code Coverage

- How many tests do I need?
- Line Converage (supported by PHP + XDebug)
  - Shows which lines have been executed (by tests)
- Path Converage (been worked on)
  - Shows which execution paths have been covered
  - Write at least $nPath tests for every method
- Parameter Value Coverage
  - Test all execution paths with sane boundary values for every parameter
  - Write at least $*nPath* $*parameterCount* $*boundaries* tests per method

Code Coverage

- How many tests do I need?
- Line Converage (supported by PHP + XDebug)
  - Shows which lines have been executed (by tests)
- Path Converage (been worked on)
  - Shows which execution paths have been covered
  - Write at least $nPath tests for every method
- Parameter Value Coverage
  - Test all execution paths with sane boundary values for every parameter
  - Write at least $*nPath* $*parameterCount* $*boundaries* tests per method
  - Common integer boundaries: $-2^{63}, -2^{31}, -1, 0, 1, 2^{31}, 2^{63}$

Are you kidding me?



Combine metrics

- Combined metrics allow interesting observations

Combine metrics

- Combined metrics allow interesting observations
  - ELOC / NOC
    - Average class length

Combine metrics

- Combined metrics allow interesting observations
  - ELOC / NOC
    - Average class length
  - ELOC / NOM
    - Average method length

Combine metrics

- Combined metrics allow interesting observations
  - ELOC / NOC
    - Average class length
  - ELOC / NOM
    - Average method length
  - CCN / NOM
    - Average method complexity

Combine metrics: CRAP

Is your code CRAP?

Combine metrics: CRAP

Is your code CRAP?

$$CRAP(m) = \begin{cases} ccn(m)^2 + ccn(m), & \text{if } cov(m) = 0 \\ ccn(m), & \text{if } cov(m) \geq .95 \\ ccn(m)^2 * (1 - cov(m))^3 + ccn(m), & \text{else} \end{cases}$$

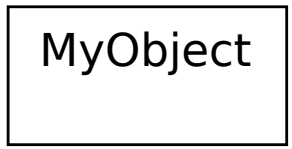
- Change Risk Anti Patterns
  - $ccn(m)$ – Cyclomatic complexity of a method
  - $cov(m)$ – Line coverage of a method

Outline

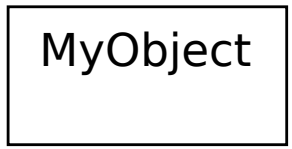
Inheritance

Is inheritance used correctly?

Object Oriented Systems

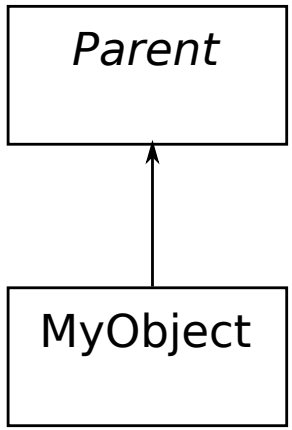


Object Oriented Systems



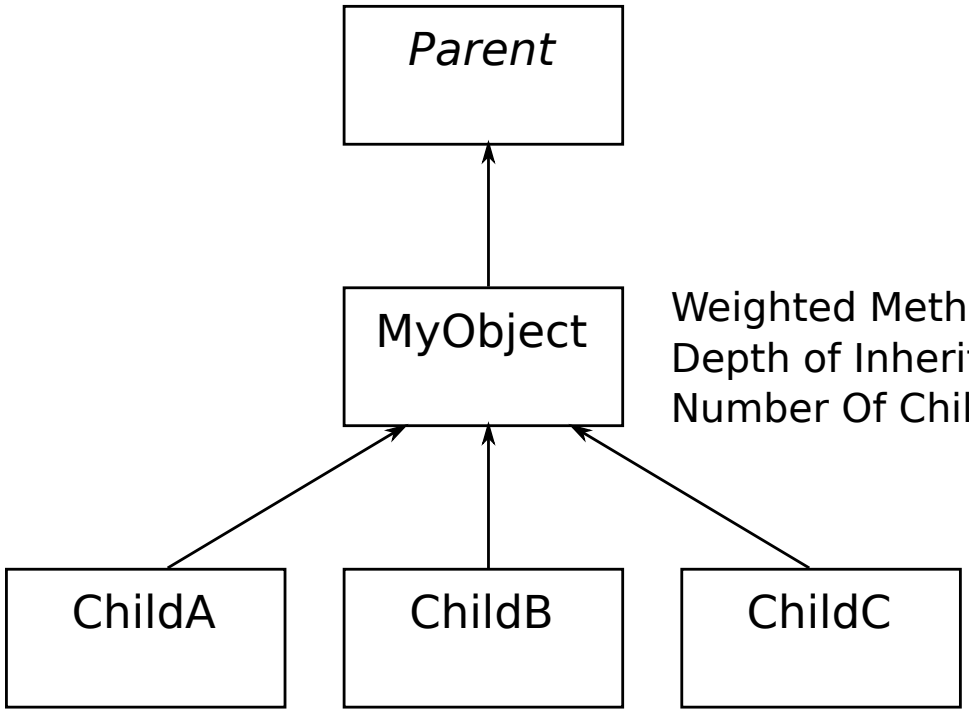
Weighted Method per Class (WMC)

Object Oriented Systems



Weighted Method per Class (WMC)
Depth of Inheritance Tree (DIT): **1**

Object Oriented Systems



Weighted Method per Class (WMC)
Depth of Inheritance Tree (DIT): **1**
Number Of Children (NOC): **3**

Chidamber & Kemerer

- A Metrics Suite for Object Oriented Design

Chidamber & Kemerer

- A Metrics Suite for Object Oriented Design
  - Weighted Methods per Class (WMC)
    - Sum of method complexities
    - Limiting value: 20 - 50

Chidamber & Kemerer

- A Metrics Suite for Object Oriented Design
  - Weighted Methods per Class (WMC)
    - Sum of method complexities
    - Limiting value: 20 - 50
  - Number Of Children (NOC)
    - Number of class extension
    - Indicator for wrong use of abstraction / inheritance

Chidamber & Kemerer

- A Metrics Suite for Object Oriented Design
  - Weighted Methods per Class (WMC)
    - Sum of method complexities
    - Limiting value: 20 - 50
  - Number Of Children (NOC)
    - Number of class extension
    - Indicator for wrong use of abstraction / inheritance
  - Depth of Inheritance Tree (DIT)
    - Inheritance can increase software complexity
    - Limiting value: ≤ 5
    - Commonly limited at component boundary

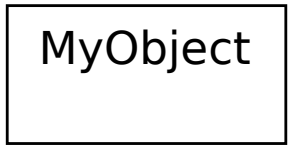
Chidamber & Kemerer

- A Metrics Suite for Object Oriented Design
  - Weighted Methods per Class (WMC)
    - Sum of method complexities
    - Limiting value: 20 - 50
  - Number Of Children (NOC)
    - Number of class extension
    - Indicator for wrong use of abstraction / inheritance
  - Depth of Inheritance Tree (DIT)
    - Inheritance can increase software complexity
    - Limiting value: $\leq$ ~~5~~ 1
    - Commonly limited at component boundary

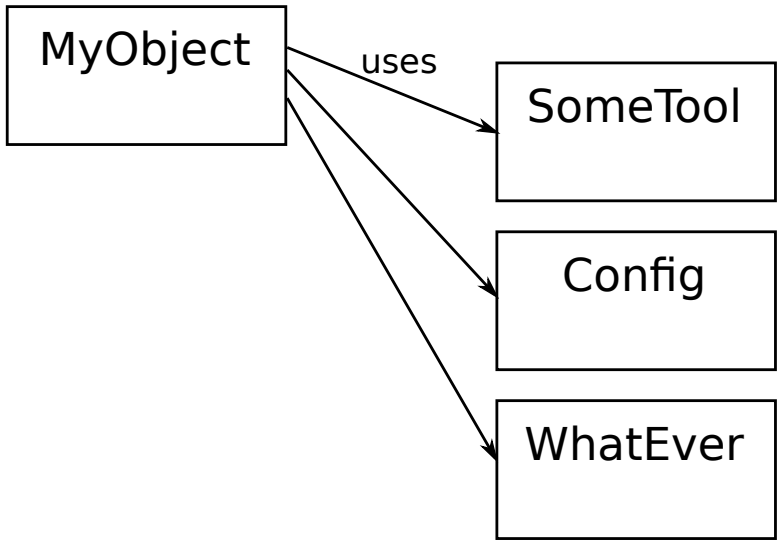
Composition

Are there any evil entities?

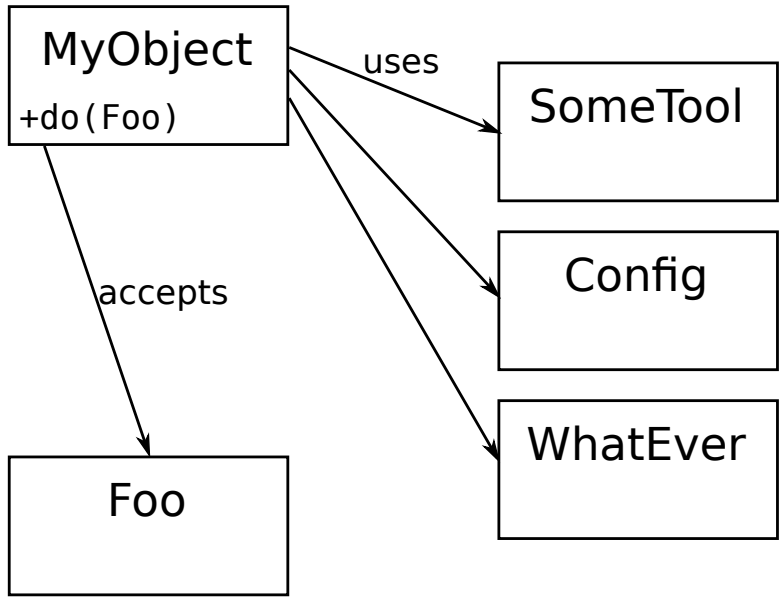
Object Oriented Systems



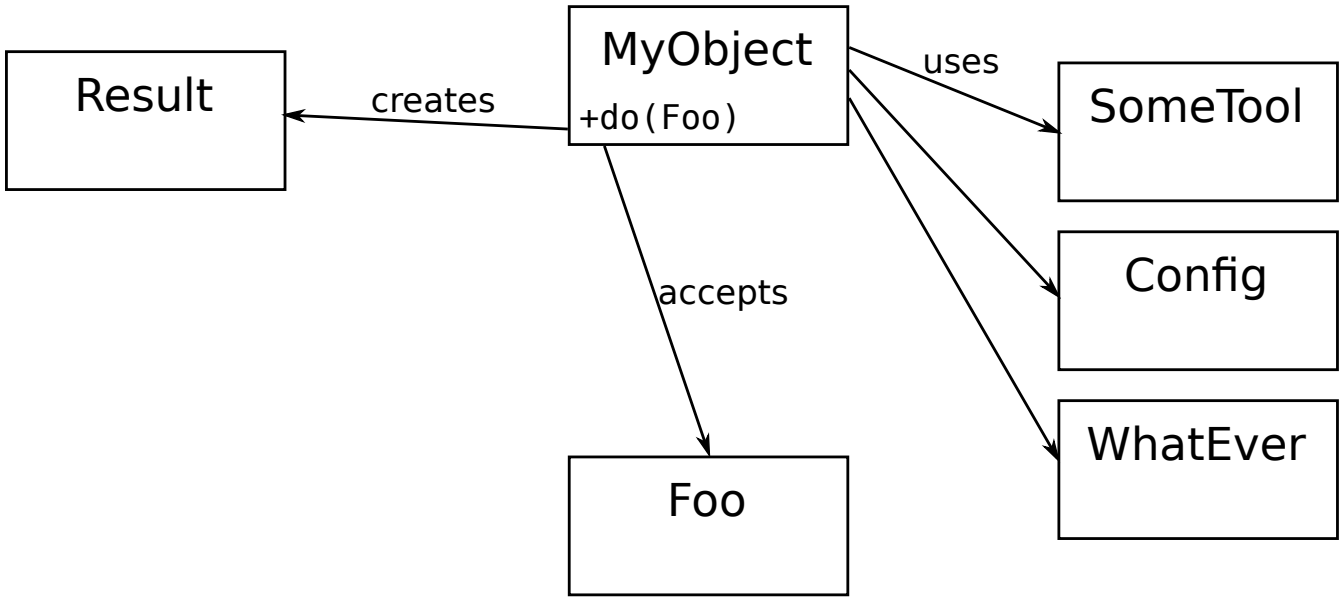
Object Oriented Systems



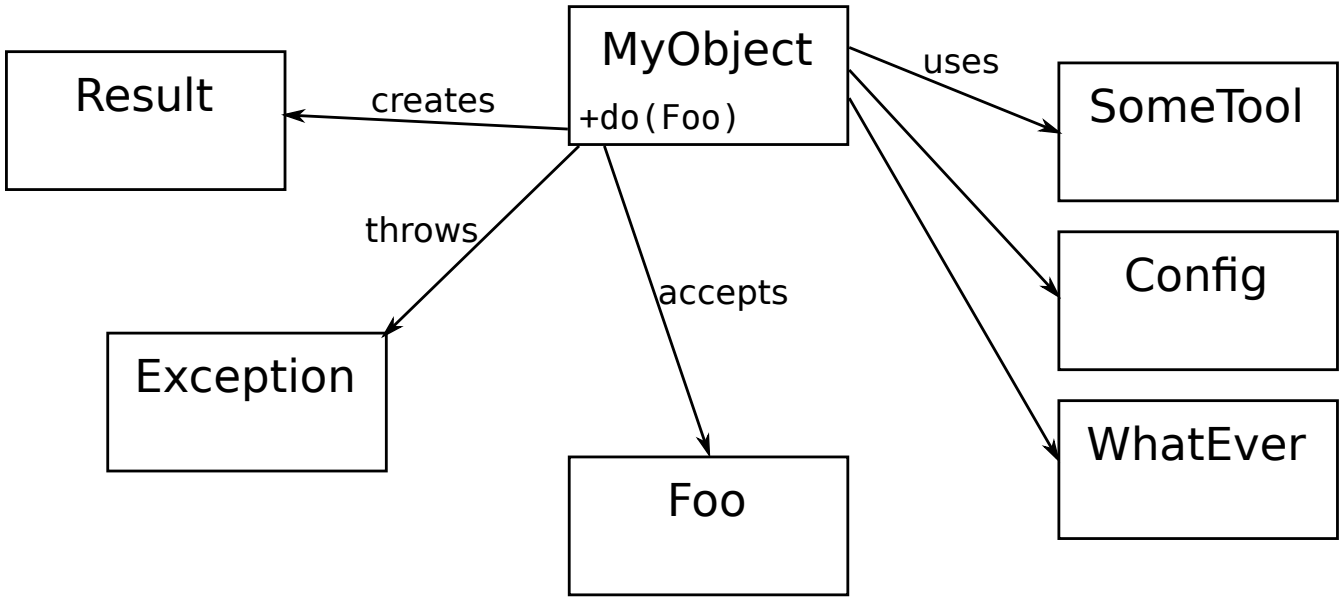
Object Oriented Systems



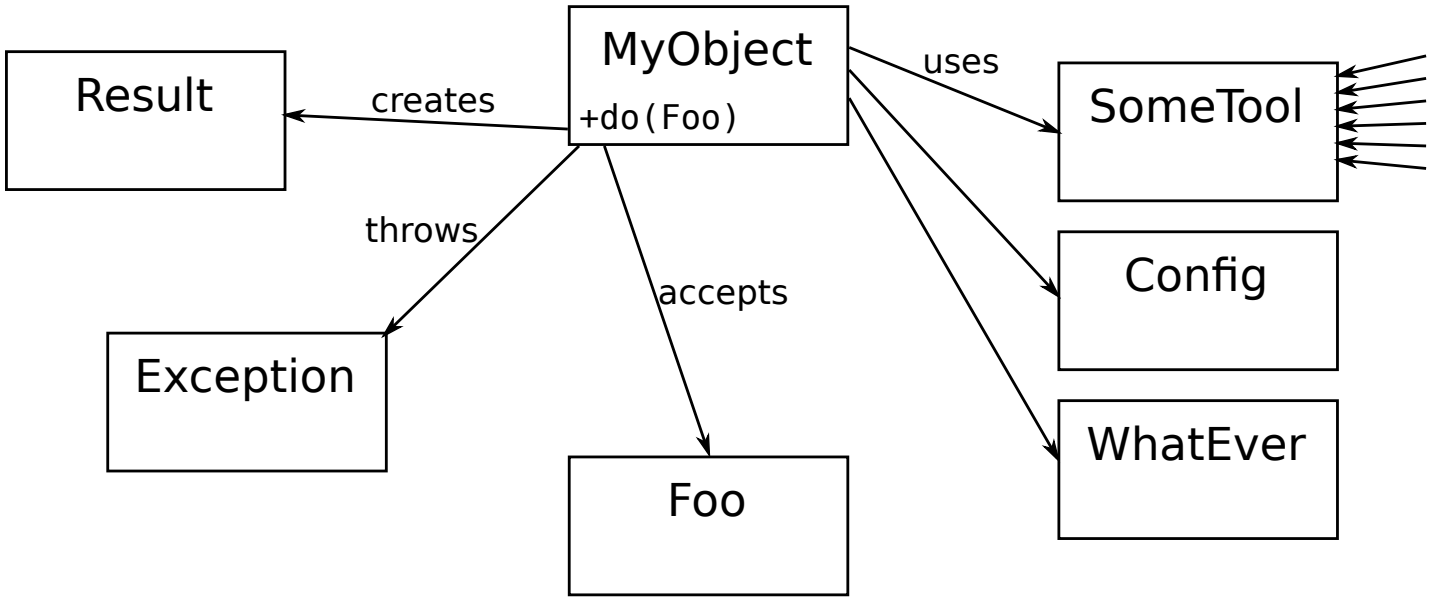
Object Oriented Systems



Object Oriented Systems



Object Oriented Systems



Coupling

- Excessive coupling is one of the key problems

Coupling

- Excessive coupling is one of the key problems
  - Dependencies between artifacts are established by:
    - Object instantiations
    - Static method calls
    - Method parameters
    - Thrown and catched exceptions

Coupling

- Excessive coupling is one of the key problems
  - Dependencies between artifacts are established by:
    - Object instantiations
    - Static method calls
    - Method parameters
    - Thrown and catched exceptions
- (High) Efferent Coupling C_E (outgoing dependencies)
  - Artifact relies on a lot of code
  - Artifact tends to be unstable

Coupling

- Excessive coupling is one of the key problems
  - Dependencies between artifacts are established by:
    - Object instantiations
    - Static method calls
    - Method parameters
    - Thrown and catched exceptions
- (High) Efferent Coupling C_E (outgoing dependencies)
  - Artifact relies on a lot of code
  - Artifact tends to be unstable
  - Also called “Coupling Between Objects” (CBO)

Coupling

- Excessive coupling is one of the key problems
  - Dependencies between artifacts are established by:
    - Object instantiations
    - Static method calls
    - Method parameters
    - Thrown and catched exceptions
- (High) Efferent Coupling C_E (outgoing dependencies)
  - Artifact relies on a lot of code
  - Artifact tends to be unstable
  - Also called “Coupling Between Objects” (CBO)
- (High) Afferent Coupling C_A (incoming dependencies)
  - A lot of code relies on artifact
  - Artifact should be really stable

Instability vs. Abstractness

Instability:

$$I = \frac{C_E}{C_E + C_A}$$

- C_E: Efferent Coupling (outgoing)
- C_A: Afferent Coupling (incoming)

Instability vs. Abstractness

Instability:

$$I = \frac{C_E}{C_E + C_A}$$

- C_E: Efferent Coupling (outgoing)
- C_A: Afferent Coupling (incoming)

Abstractness:

$$A = \frac{Abstracts}{Concretes + Abstracts}$$

- *Abstracts*: Abstract sub-artificats
- *Concretes*: Concrete sub-artificats

Instability vs. Abstractness

Instability:

$$I = \frac{C_E}{C_E + C_A}$$

- C_E: Efferent Coupling (outgoing)
- C_A: Afferent Coupling (incoming)

Abstractness:

$$A = \frac{Abstracts}{Concretes + Abstracts}$$

- *Abstracts*: Abstract sub-artificats
- *Concretes*: Concrete sub-artificats

- We have a 100% concrete component, what instability can we expect here?

Instability vs. Abstractness

Instability:

$$I = \frac{C_E}{C_E + C_A}$$

- C_E: Efferent Coupling (outgoing)
- C_A: Afferent Coupling (incoming)

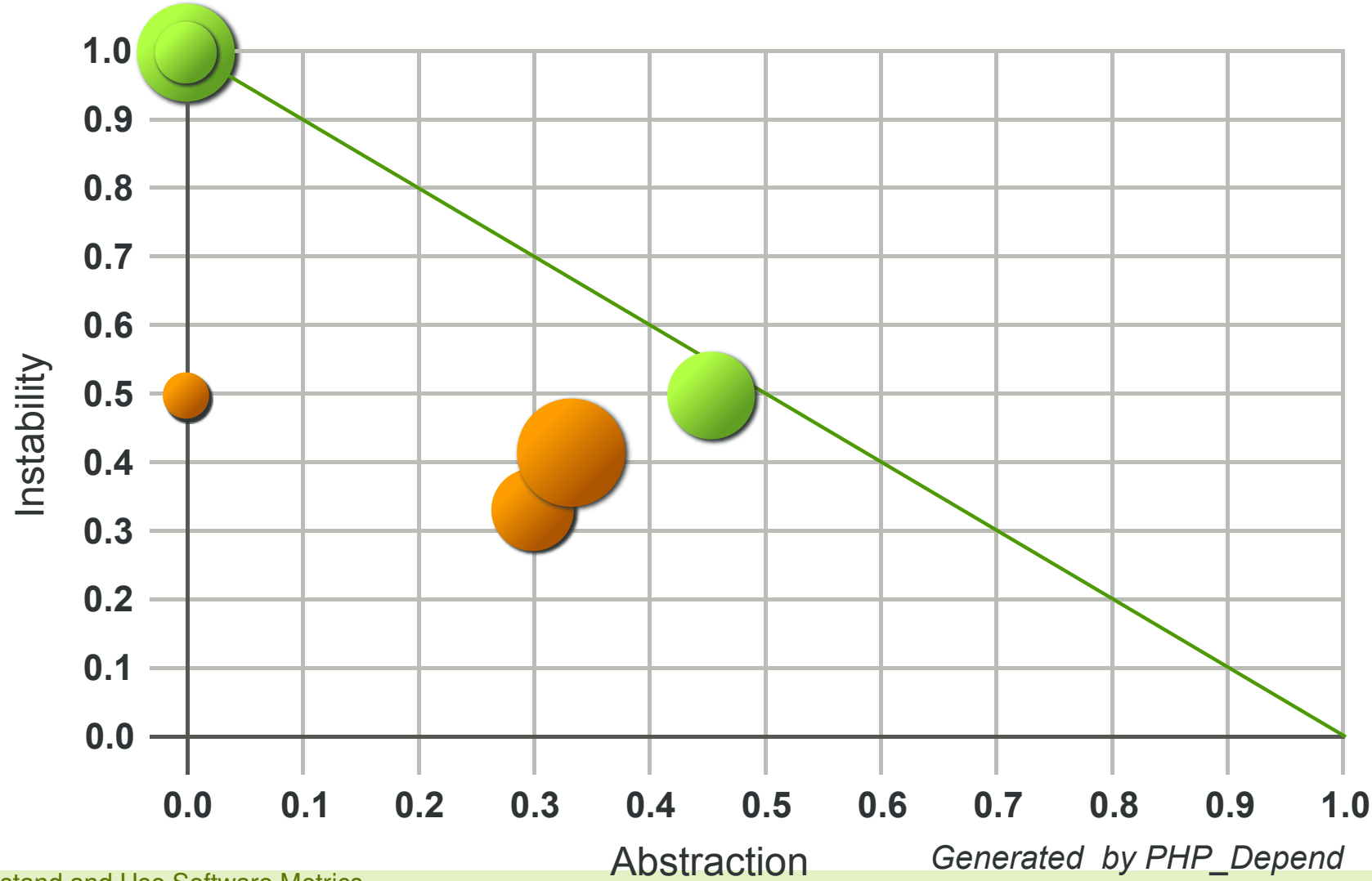
Abstractness:

$$A = \frac{Abstracts}{Concretes + Abstracts}$$

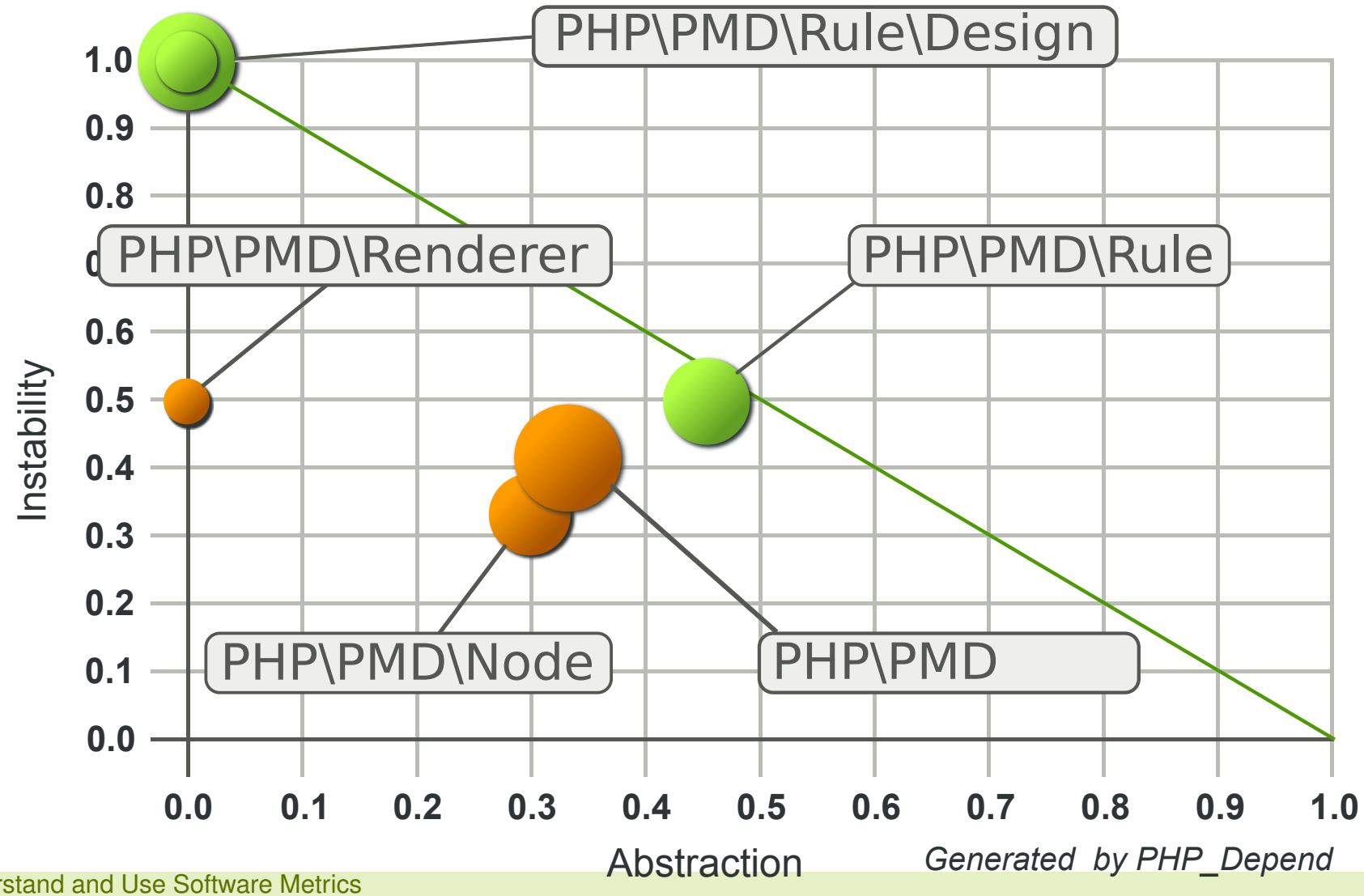
- *Abstracts*: Abstract sub-artificats
- *Concretes*: Concrete sub-artificats

- We have a 100% concrete component, what instability can we expect here?
- What instability could we expect for an abstract class or an interface?

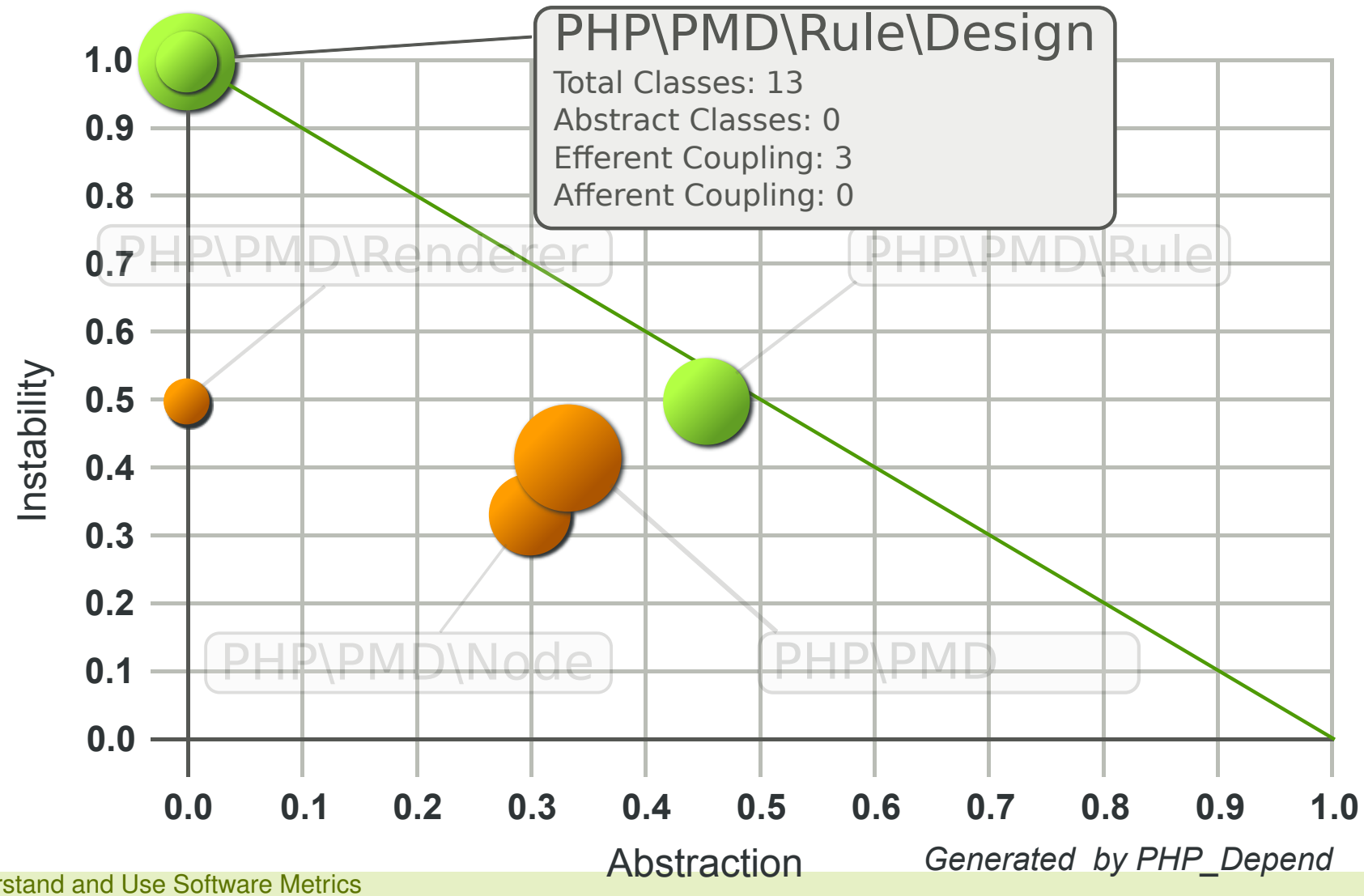
Abstractness & Instability



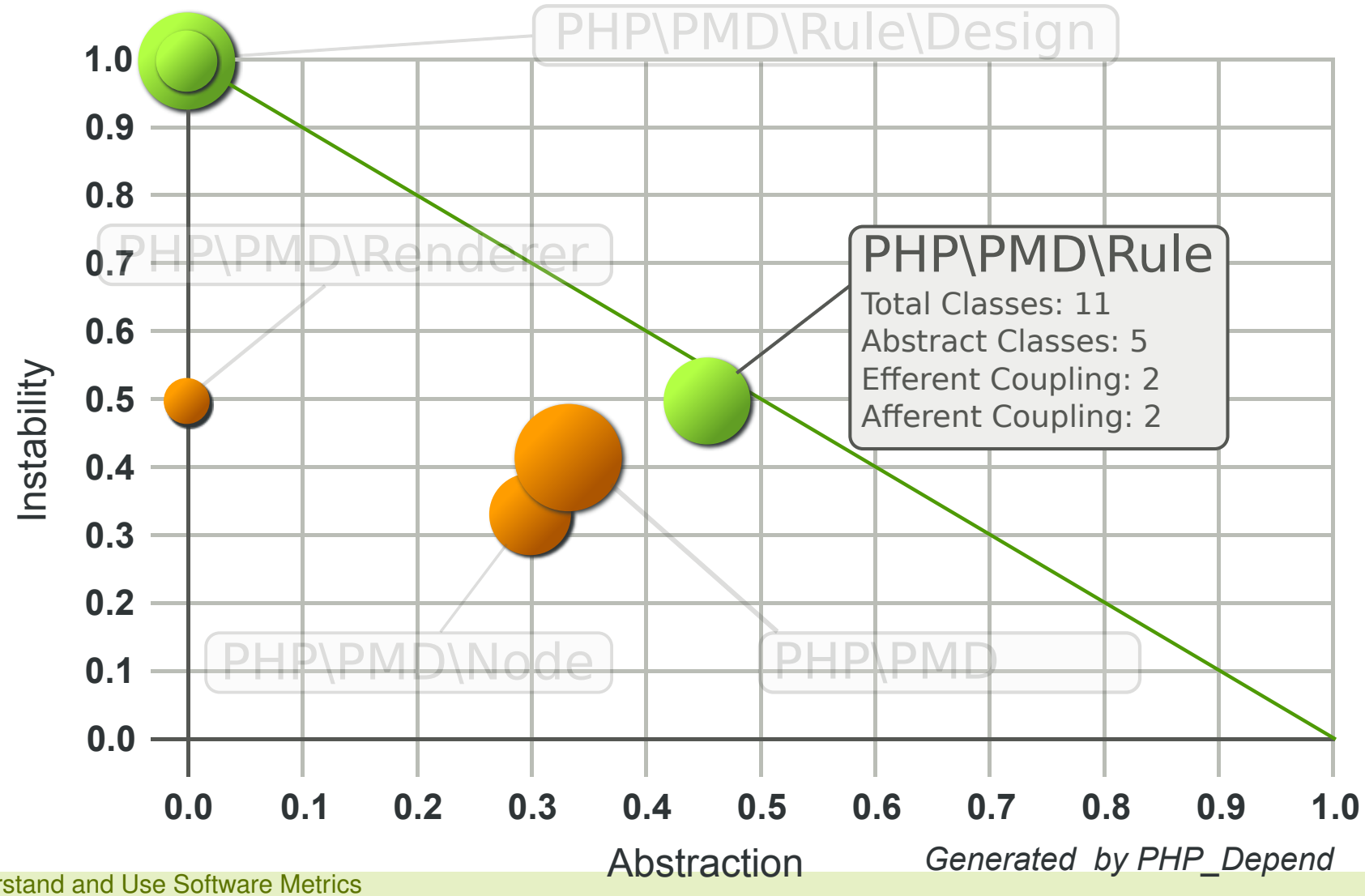
Abstractness & Instability



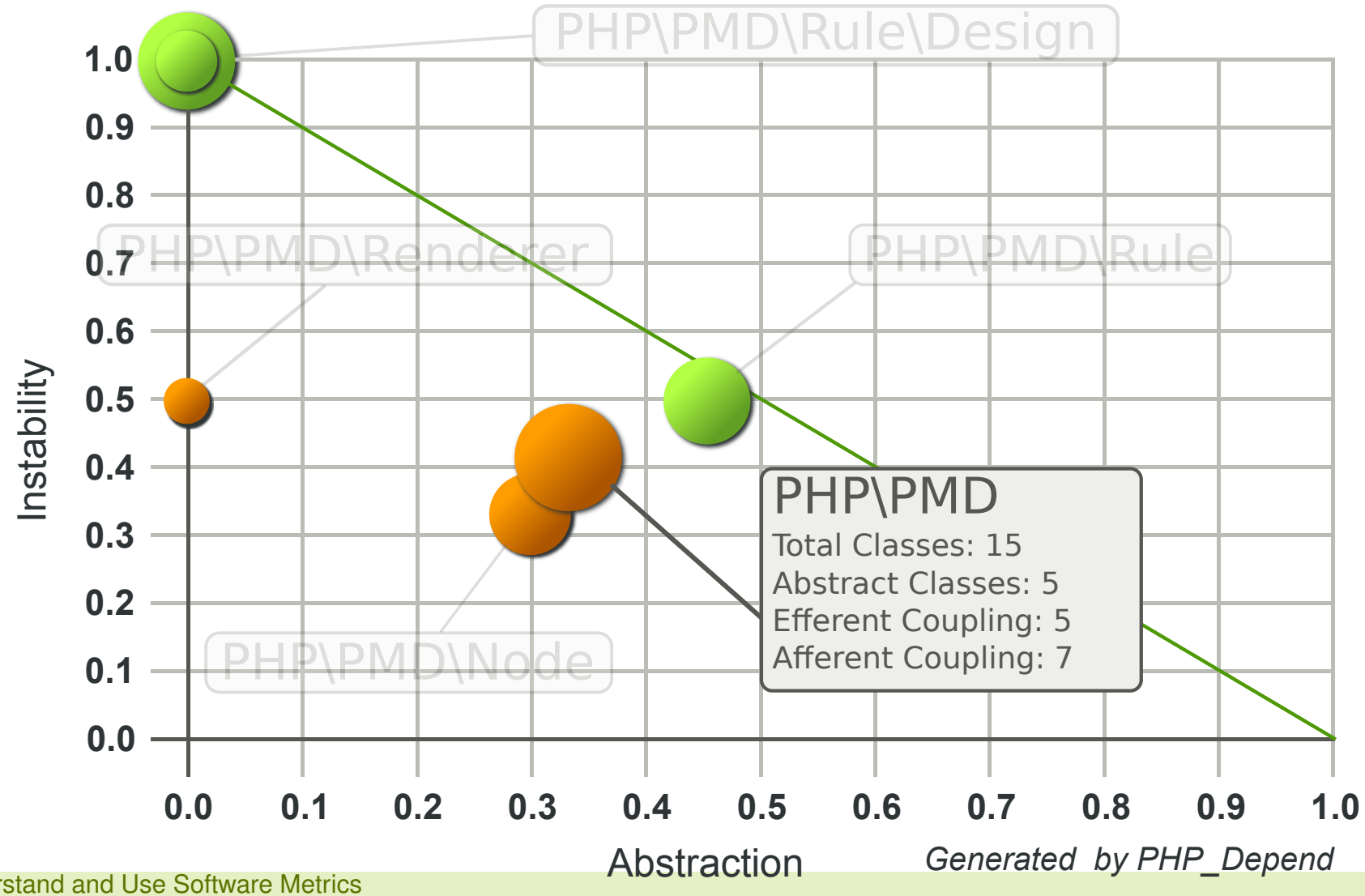
Abstractness & Instability



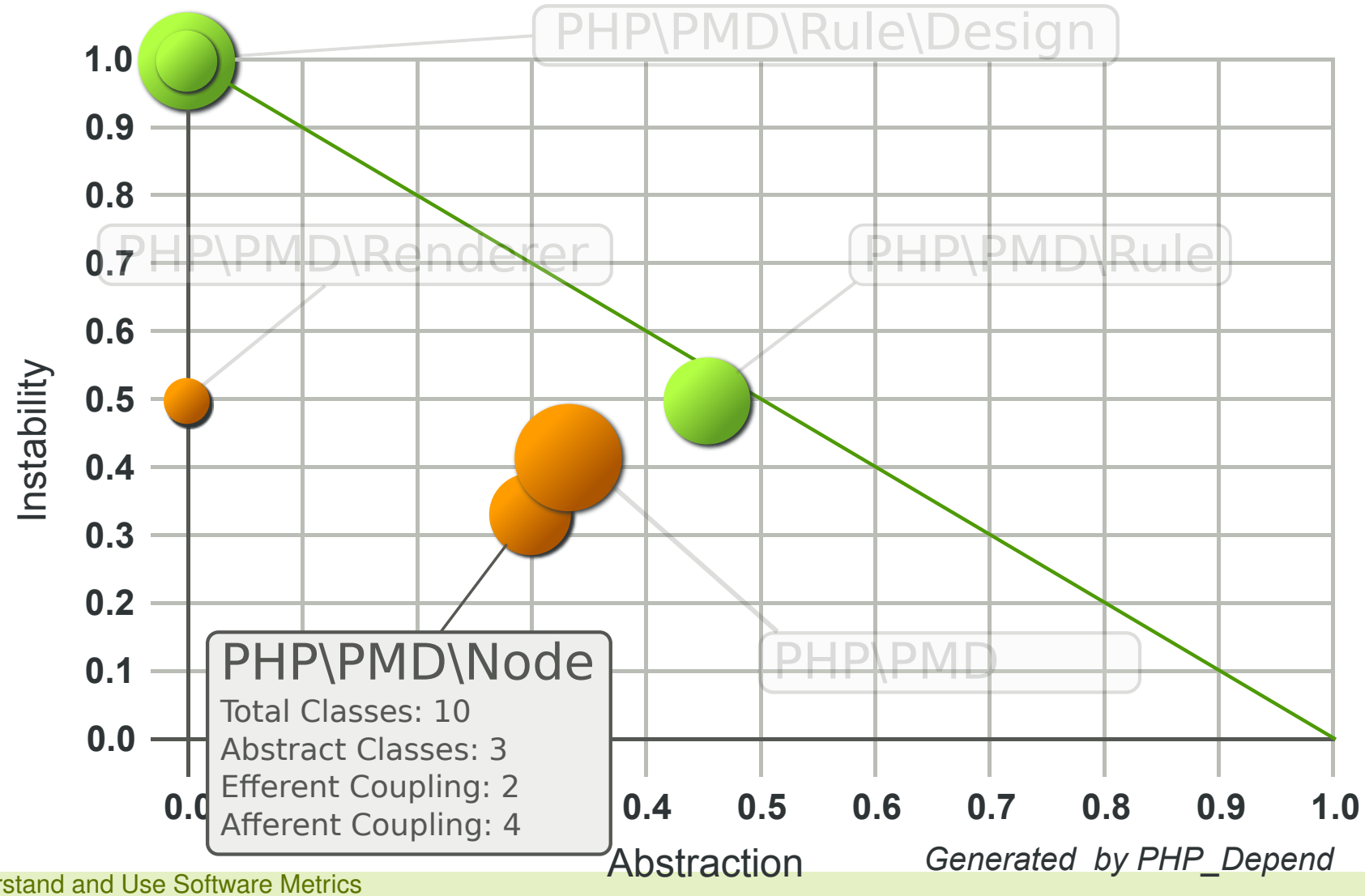
Abstractness & Instability



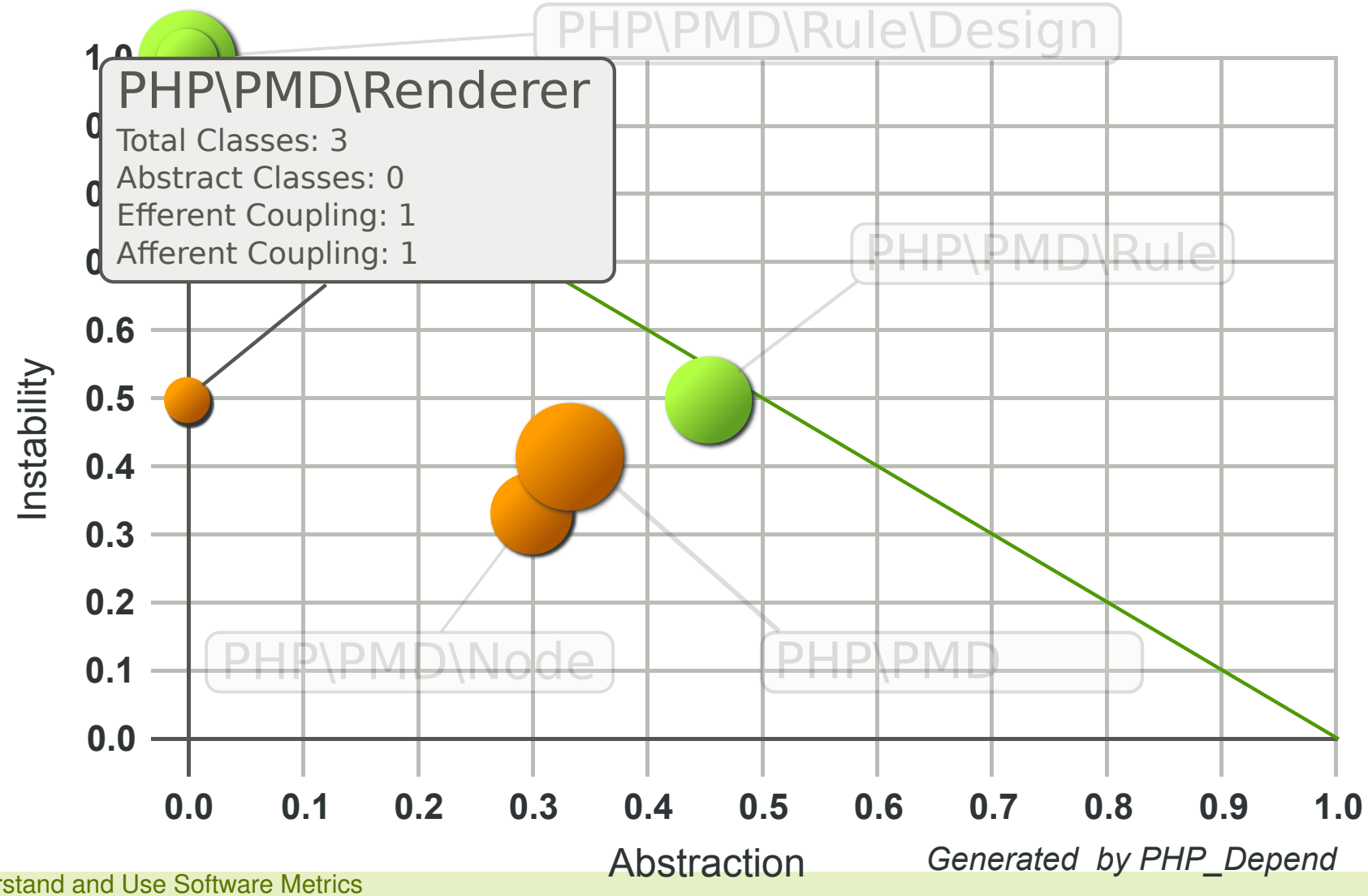
Abstractness & Instability



Abstractness & Instability



Abstractness & Instability



CodeRank

- Googles PageRank^{TM}for classes!

CodeRank

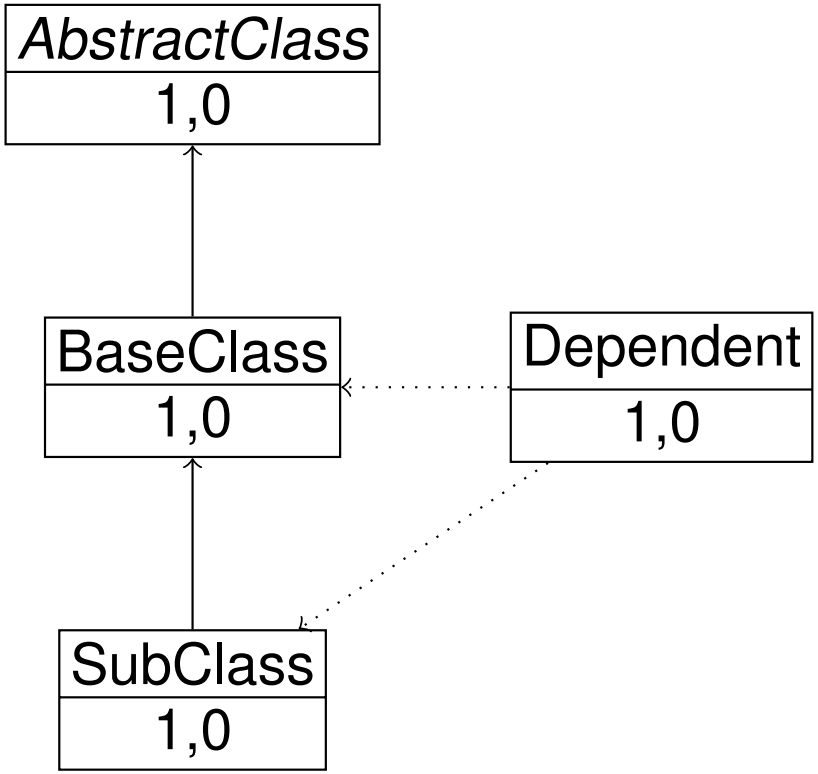
- Googles PageRank^{TM}for classes!
- Maps software to a graph
  - A node (π) for each software artifact
    - Package, Class, Method
  - An edge (ρ) for each relation
    - Inheritance, Call, Parameter, Exceptions

CodeRank

- Googles PageRank^{TM}for classes!
- Maps software to a graph
  - A node (π) for each software artifact
    - Package, Class, Method
  - An edge (ρ) for each relation
    - Inheritance, Call, Parameter, Exceptions
- CodeRank:

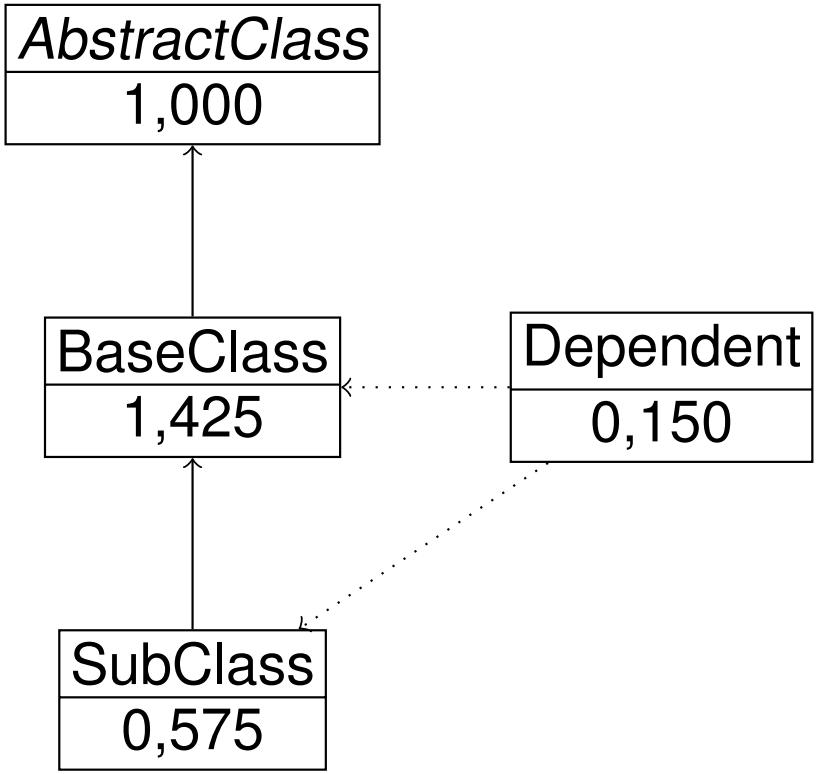
$$CR(\pi_i) = \sum_r r((1-d) + d \sum_r r(CR(\pi_r)/\rho_r))$$

CodeRank



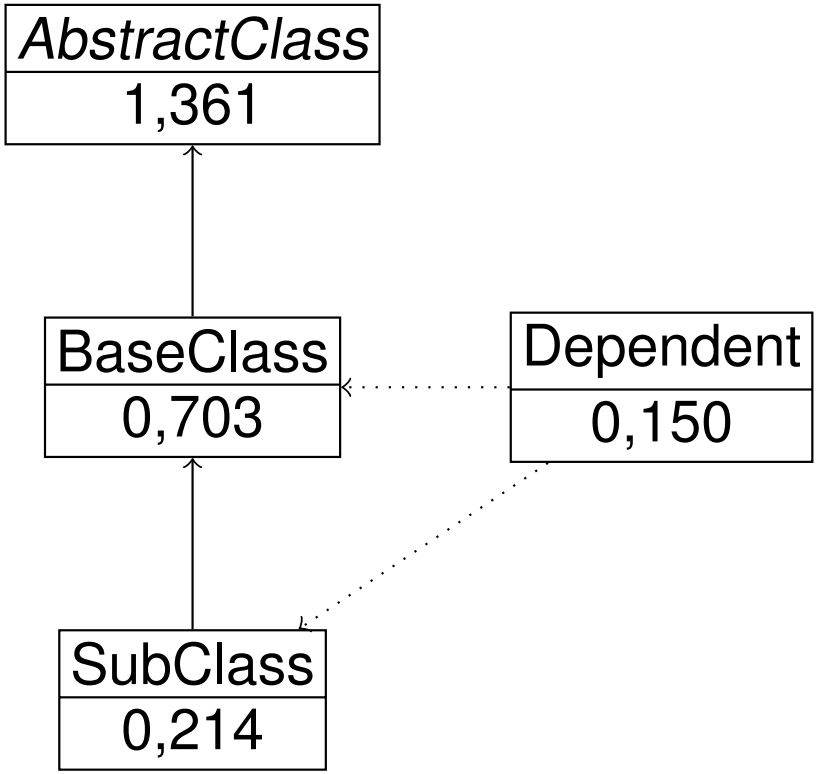
- Iteration: 0

CodeRank



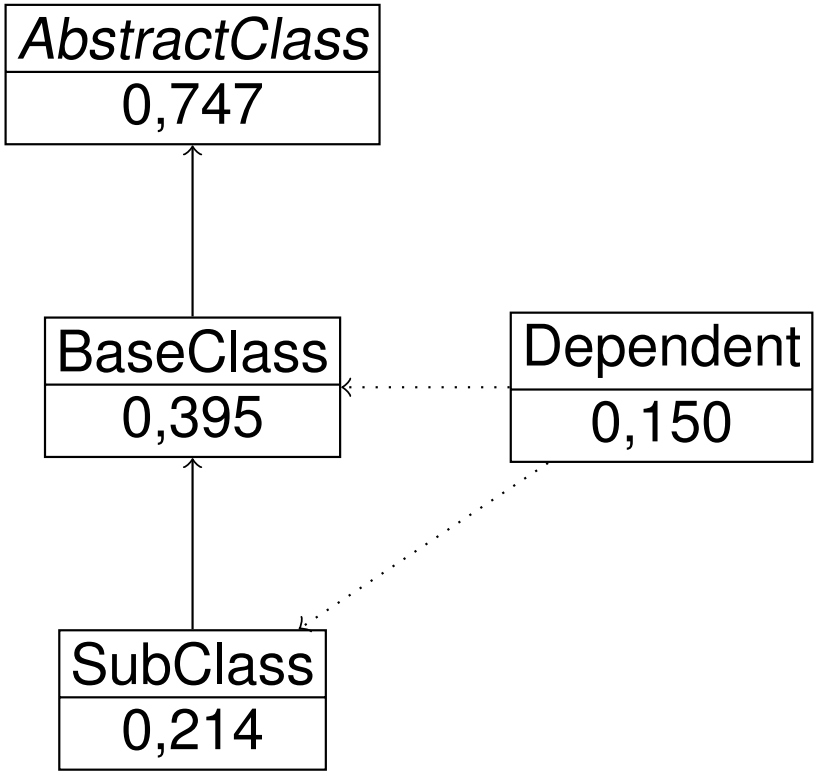
- Iteration: 1

CodeRank



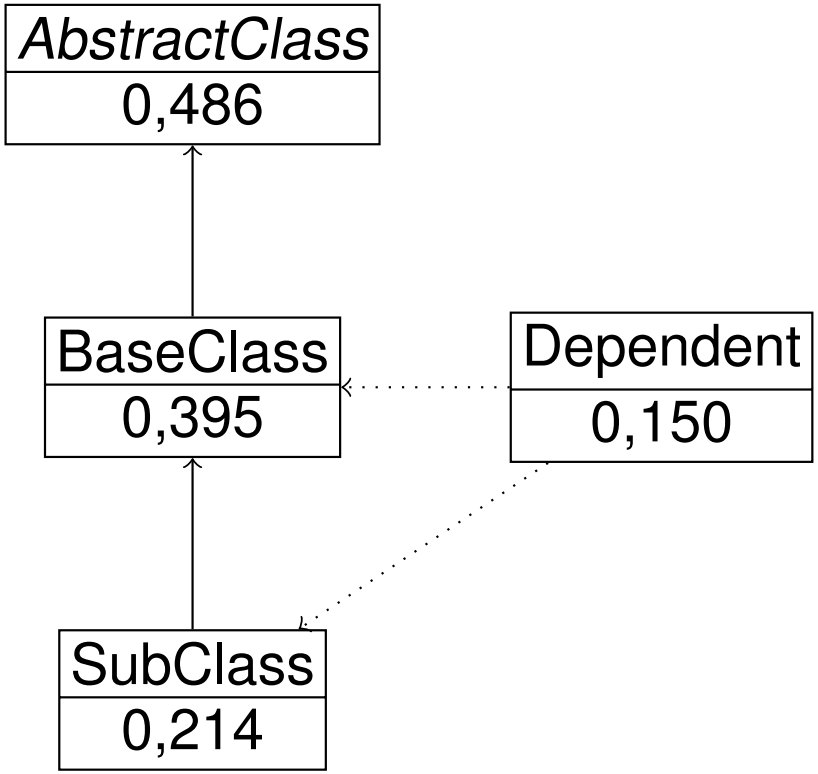
- Iteration: 2

CodeRank



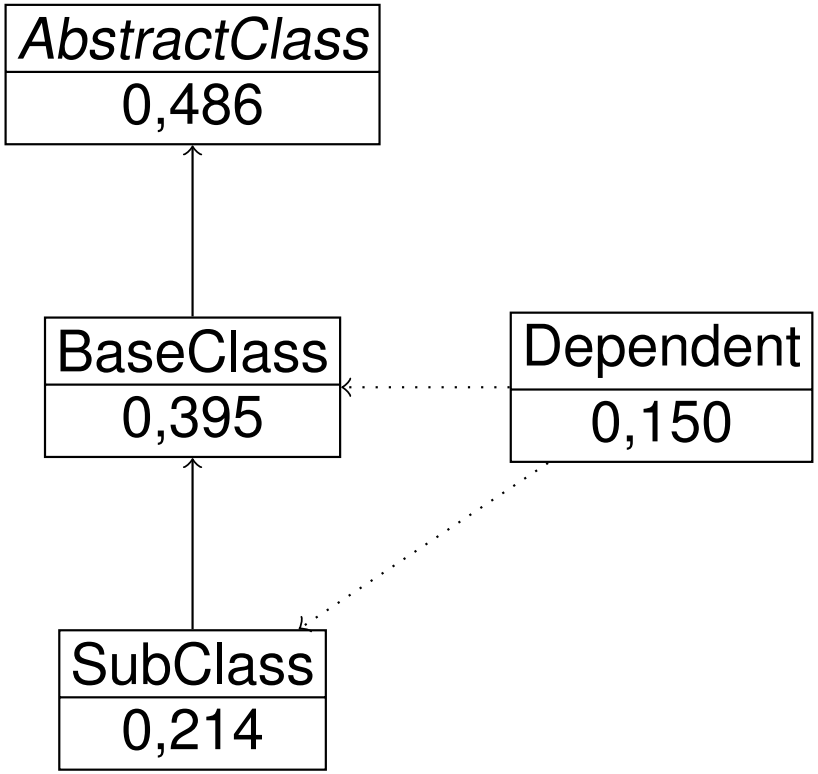
- Iteration: 3

CodeRank



- Iteration: 4

CodeRank



- Iteration: 5

CodeRank

- Incorporates indirect dependencies
- Locates elements with high effect on the whole system
- Reverse CodeRank:
  - Shows dependent components

Combined Metrics

- Important classes to test: $CR * WMC$
- Easy test subjects: $C_E = 0$ (and high CR or WMC)

Outline

Metrics are . . .

- . . . no magic, but simple measured values
- . . . useless without limiting values
- . . . scalable – grow with project growth
- . . . reproducible and automatable
- . . . objective – since calculated by software

Metrics are . . .

- . . . no magic, but simple measured values
- . . . useless without limiting values
- . . . scalable – grow with project growth
- . . . reproducible and automatable
- . . . objective – since calculated by software
- . . . highly interpretable – interpretation depends on viewer

Projects used

- PHPLoc:
  `https://github.com/sebastianbergmann/phploc`
- PDepend:
  `http://pdepend.org/`
- PHP Mess Detector (PHPMD):
  `http://phpmd.org/`
- Qafoo Code Review (CRT):
  `https://github.com/Qafoo/review`

Thanks for Listening

Rate this talk: `https://joind.in/7867`

Thanks for Listening

Rate this talk: `https://joind.in/7867`

Stay in touch

- Kore Nordmann
- kore@qafoo.com
- @koredn / @qafoo

Rent a web quality expert:
`http://qafoo.com`