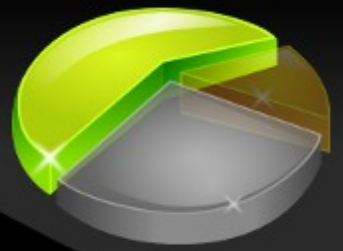
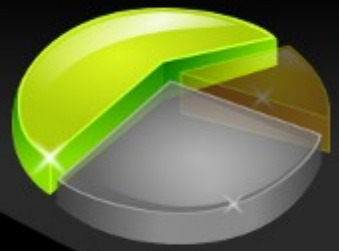
Softwaremetriken verstehen und nutzen



Kore Nordmann <kore@php.net>,
Manuel Pichler <mapi@pdepend.org>

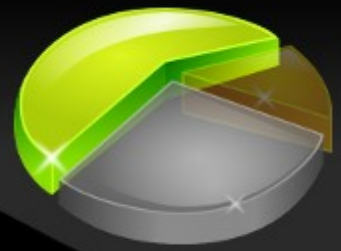
17. November 2009

Über uns



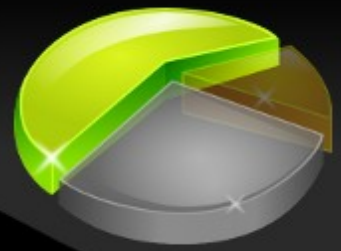
- Kore Nordmann <kore@php.net>
  - Langzeit PHP Entwickler
  - Speaker auf diversen Konferenzen, Autor etc.
  - Entwickler diverser Open-Source-Projekte:
    - EZ Components, Arbit, PHPUnit, PHPillow
- Manuel Pichler <mapi@pdepend.org>
  - Jahrgang 1978
  - Diplom Informatiker, Softwarearchitekt
  - Entwickler diverser Projekte:
    - PHP_Depend, phpUnderControl, PHPMD

Agenda



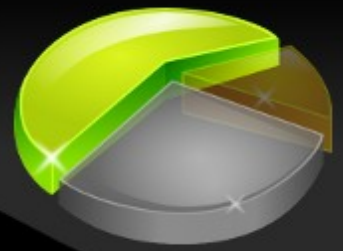
- Was sind Metriken?
- Welche Arten an Metriken gibt es?
- Klassische Softwaremetriken
  - Umfang
  - Komplexität
- Objektorientierte Softwaremetriken
  - Kopplung
  - Abstraktion

Softwaremetriken



- Eine Softwaremetrik ist eine Maßzahl für ein Qualitätsmerkmal von Software
  - „Funktionen zur Ermittlung von Kennzahlen eines Softwareartefakts“ (Wikipedia)
  - „You cannot control what you cannot measure.“ (Tom DeMarco)
    - Auch wenn er diese Aussage bereits relativierte
  - „Ohne Messung können Auswirkungen durch Änderungen nicht bewertet werden“ (A. Fleischer)

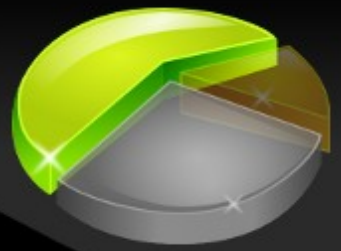
Arten an Metriken



- Prozessmetrik
- Aufwandsmetrik
- Projektlaufzeitmetrik
- Anwendungsmetrik

Quelle: Wikipedia

Vergesst das aber alles

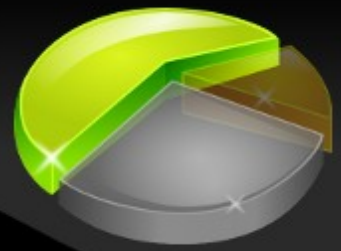


wir beschäftigen uns ausschließlich

mit

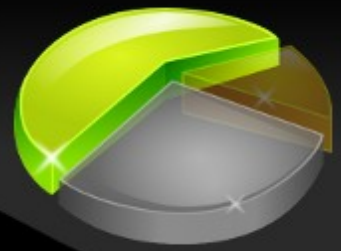
Produktmetriken

Produktmetriken



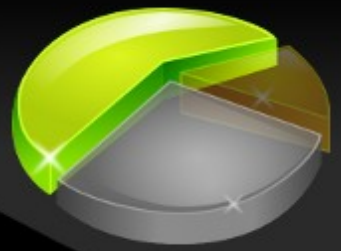
- Umfang
- Komplexität
- Kopplung & Abstraktion
- Lesbarkeit

Umfangsmetriken



- Summen über Softwareartefakte
  - Lines Of *
    - LOC - Lines Of Code
    - ELOC - Executable Lines Of Code
    - CLOC - Comment Lines Of Code
    - NCLOC - None Comment Lines Of Code
  - Number Of *
    - NOC - Number Of Classes
    - NOM - Number Of Methods
    - NOP - Number Of Packages

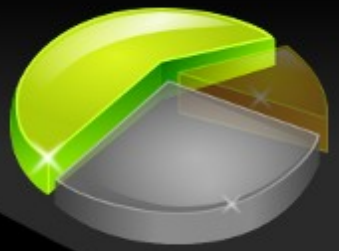
Lines Of *, Number Of *



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

abstract class FooBar {
    abstract function bar();
}

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

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

- Lines Of *
  - LOC =
  - ELOC =
  - CLOC =
  - NCLOC =
- Number Of *
  - NOC =
  - NOM =
  - NOP =

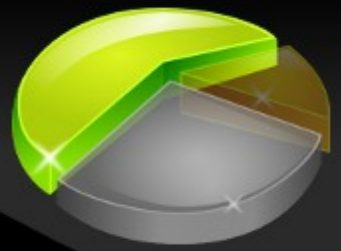
Lines Of *, Number Of *



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

abstract class FooBar {
    abstract function bar();
}

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

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

- Lines Of *
  - LOC = 16
  - ELOC =
  - CLOC =
  - NCLOC =
- Number Of *
  - NOC =
  - NOM =
  - NOP =

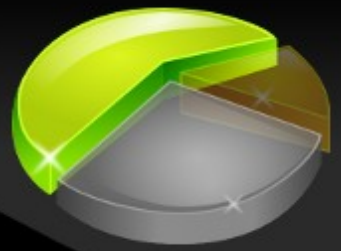
Lines Of *, Number Of *



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

abstract class FooBar {
    abstract function bar();
}

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

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

- Lines Of *
  - LOC = 16
  - ELOC = 3
  - CLOC =
  - NCLOC =
- Number Of *
  - NOC =
  - NOM =
  - NOP =

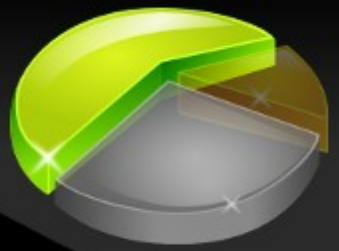
Lines Of *, Number Of *



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

abstract class FooBar {
    abstract function bar();
}

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

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

- Lines Of *
  - LOC = 16
  - ELOC = 3
  - CLOC = 2
  - NCLOC =
- Number Of *
  - NOC =
  - NOM =
  - NOP =

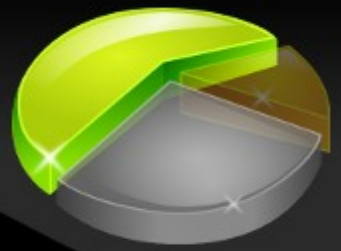
Lines Of *, Number Of *



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

abstract class FooBar {
    abstract function bar();
}

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

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

- Lines Of *
  - LOC = 16
  - ELOC = 3
  - CLOC = 2
  - NCLOC = 14
- Number Of *
  - NOC =
  - NOM =
  - NOP =

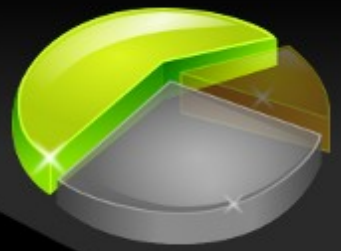
Lines Of *, Number Of *



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

abstract class FooBar {
    abstract function bar();
}

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

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

- Lines Of *
  - LOC = 16
  - ELOC = 3
  - CLOC = 2
  - NCLOC = 14
- Number Of *
  - NOC = 3
  - NOM =
  - NOP =

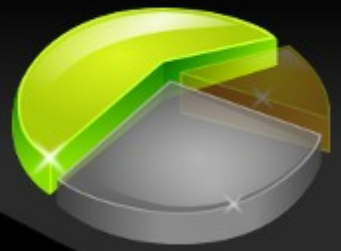
Lines Of *, Number Of *



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

abstract class FooBar {
    abstract function bar();
}

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

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

- Lines Of *
  - LOC = 16
  - ELOC = 3
  - CLOC = 2
  - NCLOC = 14
- Number Of *
  - NOC = 3
  - NOM = 4
  - NOP =

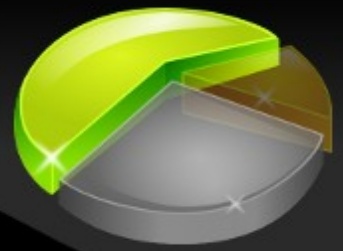
Lines Of *, Number Of *



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

abstract class FooBar {
    abstract function bar();
}

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

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

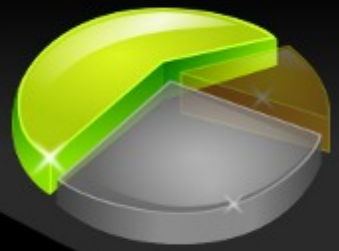
- Lines Of *
  - LOC = 16
  - ELOC = 3
  - CLOC = 2
  - NCLOC = 14
- Number Of *
  - NOC = 3
  - NOM = 4
  - NOP = 1

Beispiel



Ein Beispiel

Komplexitätsmetriken



- Maß für Komplexität sind Kontrollstrukturen
  - if, elseif, for, while, foreach, catch, case, xor, and, or, &&, ||, ?
- Cyclomatic Complexity (CCN)
  - Anzahl der *Verzweigungen*
- NPath Complexity
  - Anzahl der *Ausführungspfade*
  - Berücksichtigt die Struktur von Blöcken

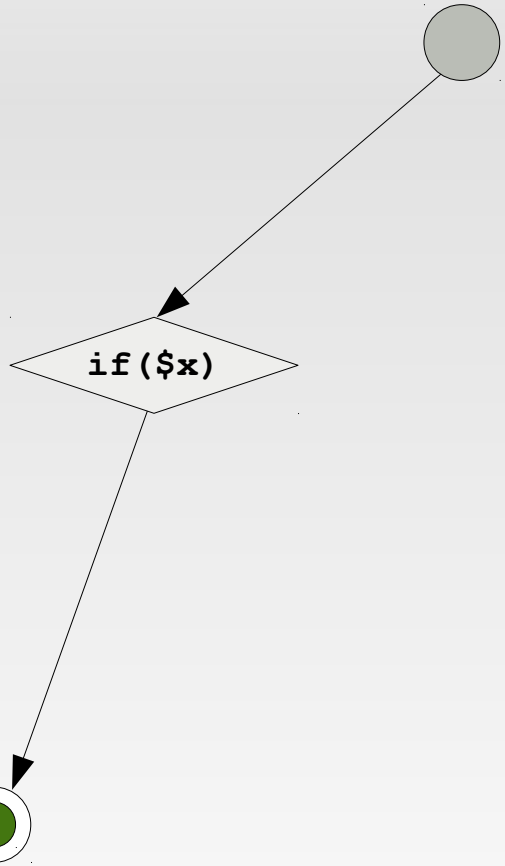
Beispiel 2: Cyclomatic Complexity

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

CCN = 0 

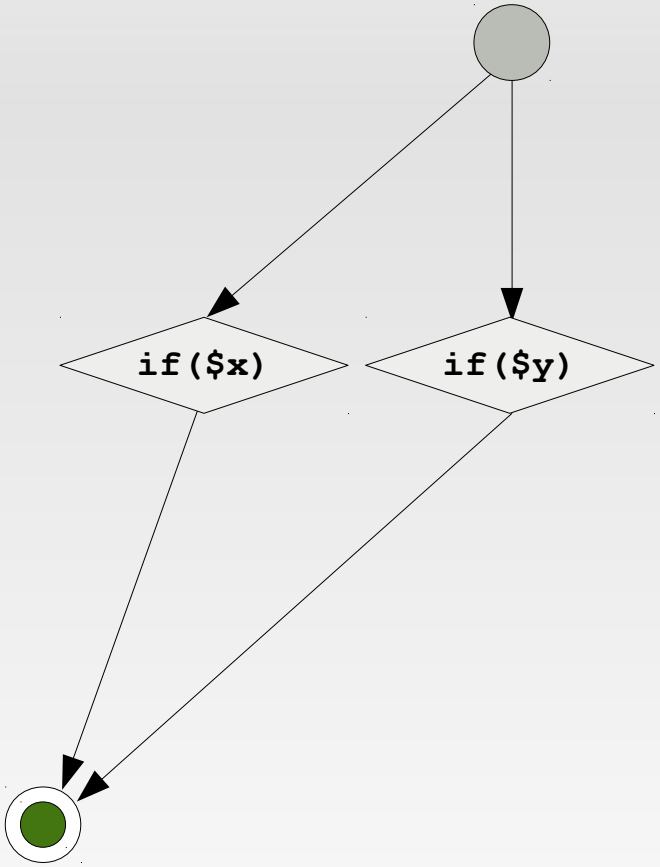
Beispiel 2: Cyclomatic Complexity

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



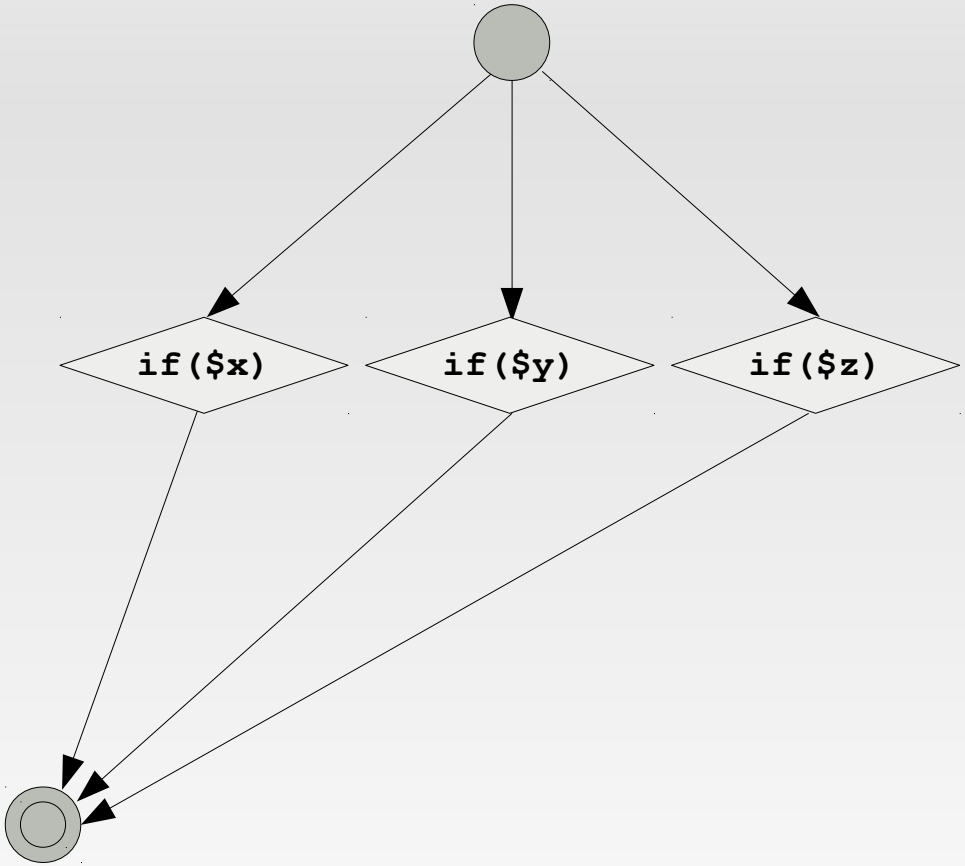
Beispiel 2: Cyclomatic Complexity

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



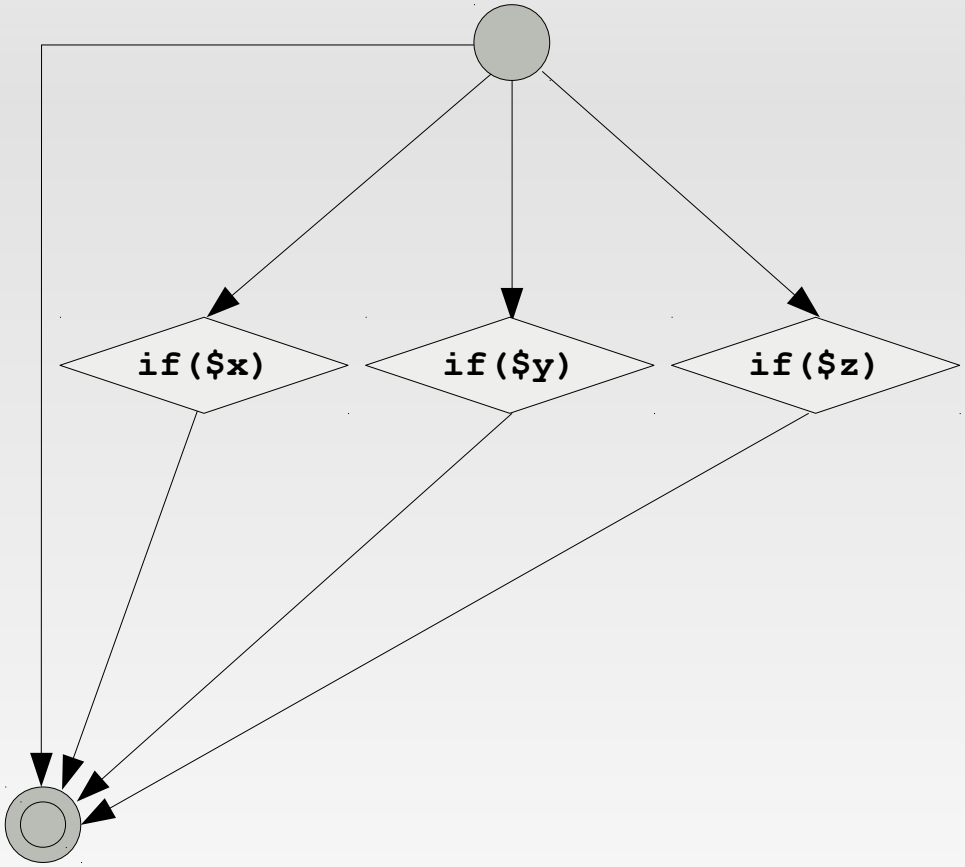
Beispiel 2: Cyclomatic Complexity

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

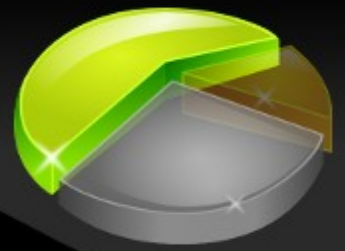


Beispiel 2: Cyclomatic Complexity

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



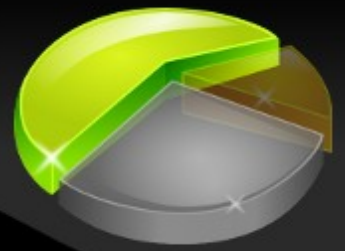
Beispiel 2: NPath Complexity



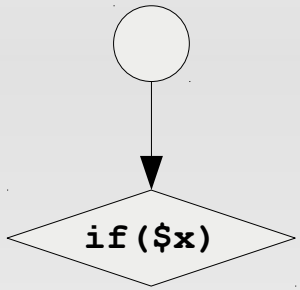
```php
<?php
class Foo {
    public function foo() {
        if ($x) { }
        if ($y) { }
        if ($z) { }
        return $x;
    }
}
```

NPath = 0

Beispiel 2: NPath Complexity

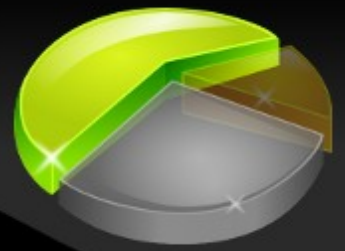


```php
<?php
class Foo {
    public function foo() {
        if ($x) { }
        if ($y) { }
        if ($z) { }
        return $x;
    }
}
```

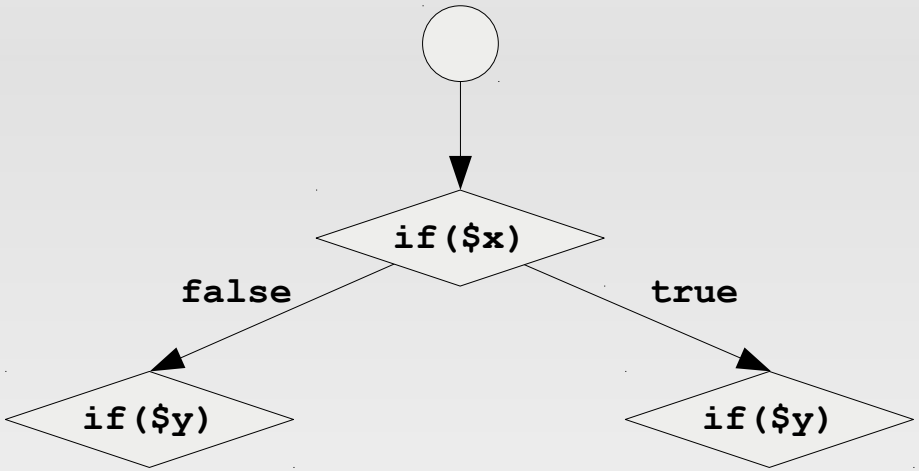


NPath = 0

Beispiel 2: NPath Complexity

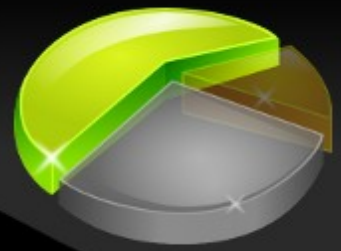


```php
<?php
class Foo {
    public function foo() {
        if ($x) { }
        if ($y) { }
        if ($z) { }
        return $x;
    }
}
```

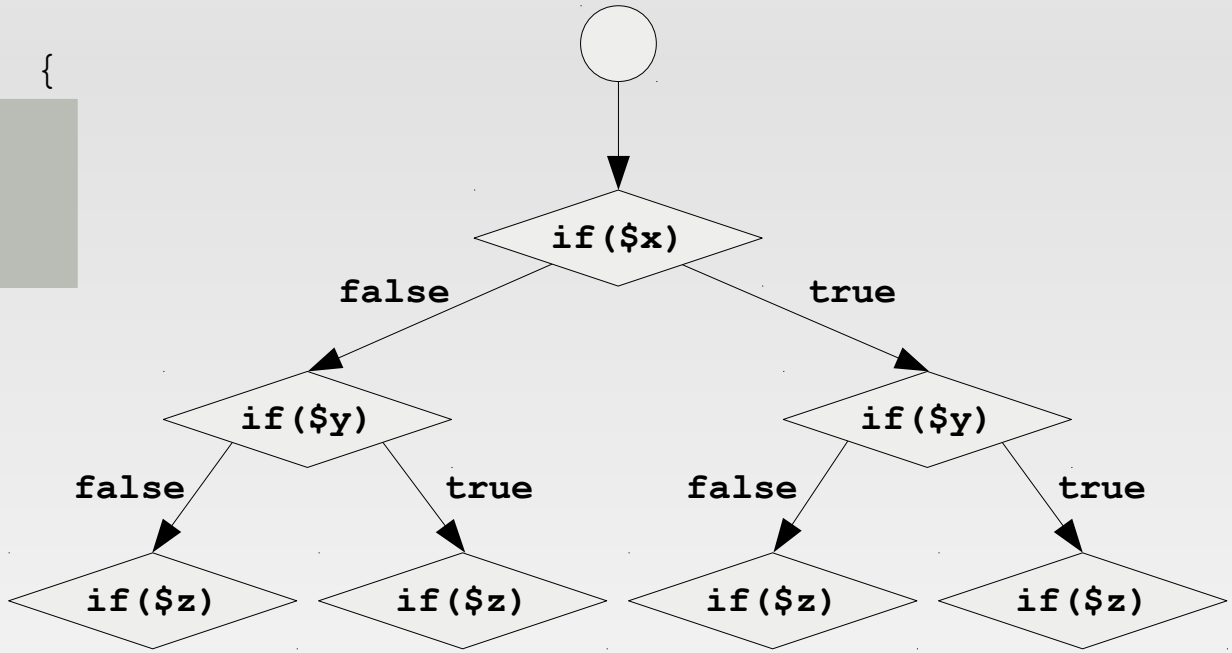


NPath = 0

Beispiel 2: NPath Complexity

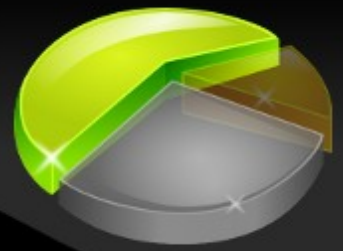


```php
<?php
class Foo {
    public function foo() {
        if ($x) { }
        if ($y) { }
        if ($z) { }
        return $x;
    }
}
```

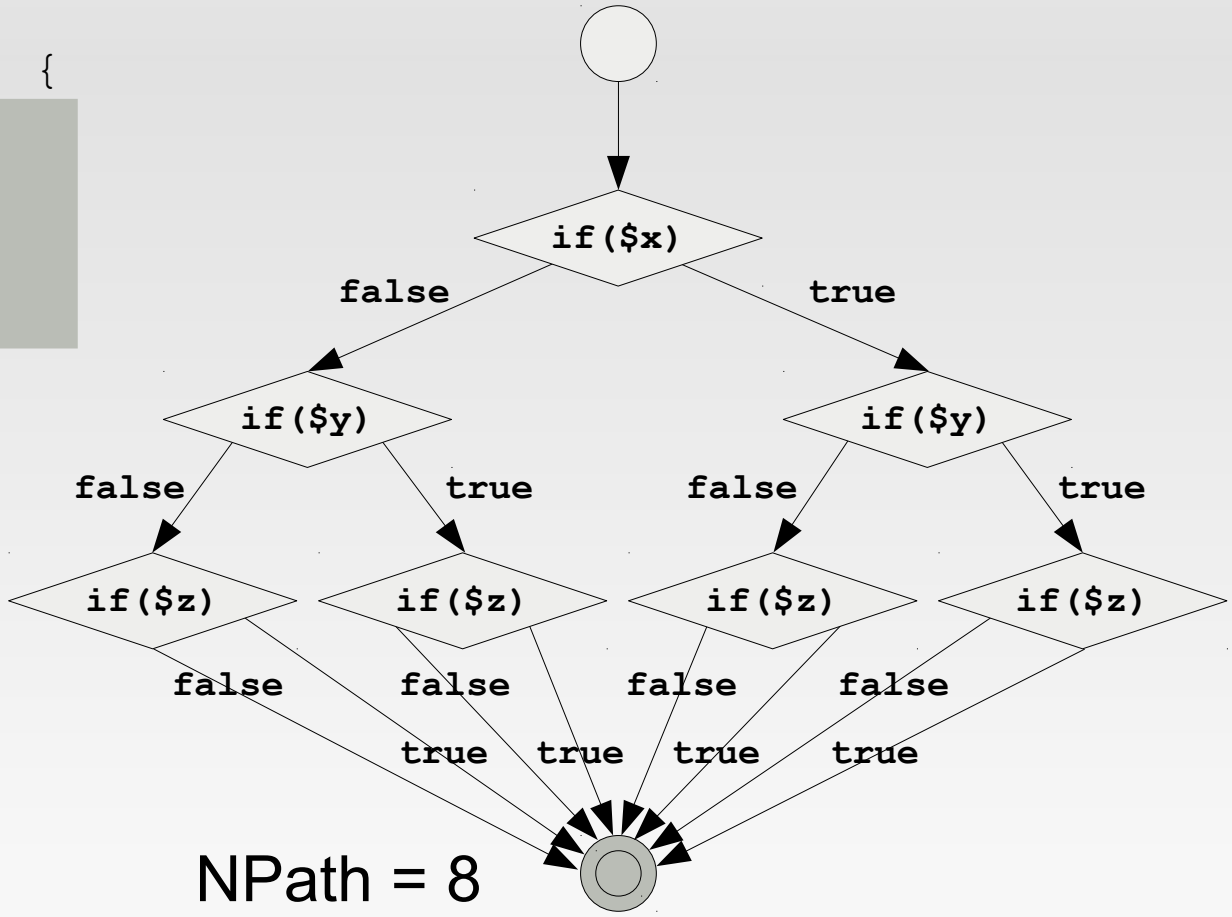


NPath = 0

Beispiel 2: NPath Complexity

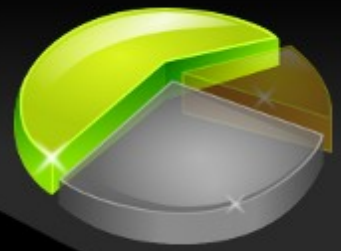


```php
<?php
class Foo {
    public function foo() {
        if ($x) { }
        if ($y) { }
        if ($z) { }
        return $x;
    }
}
```



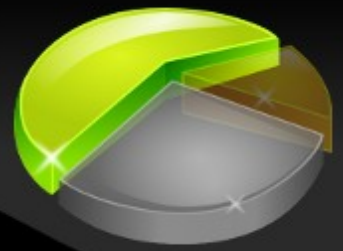
NPath = 8

Grenzwerte



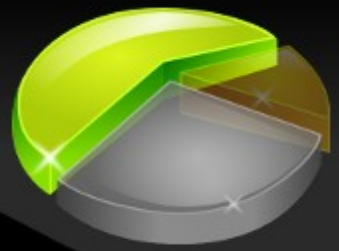
- Zahlen allein sagen noch nichts aus
  - Oder was bedeuten die Werte 4 und 8 ?
- Für eine Beurteilung benötigt man Grenzwerte
  - Cyclomatic Complexity
    - 1-4: low, 5-7: medium, 8-10: high, 11+: hell
  - NPath Complexity
    - 200: critical mass
- Grenzwerte sind immer Ermessenssache

Beispiel



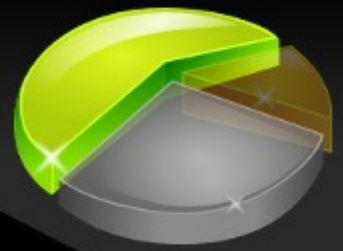
Ein Beispiel

Metriken kombinieren



- Die Kombination von Metriken erlaubt einen sehr tiefen Einblick in ein System.
  - LOC: 300; CCN: 42; NOC: 5; NOM: 15
    - CCN / LOC = 0,14
      - Jede sechste Zeile eine Kontrollstruktur
    - LOC / NOC = 60
      - Primär prozedural oder große Klassen
    - LOC / NOM = 20
      - Große Methoden/Funktionen oder prozedural
    - CCN / NOM = 2,8
      - Übermäßig komplexe Methoden/Funktionen

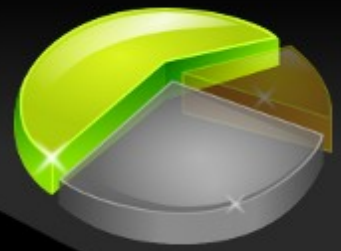
Schluss damit,



beschäftigen wir uns mit den

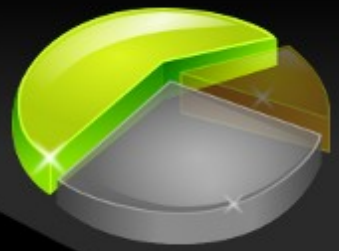
OO-Metriken.

Objektorientierte Metriken



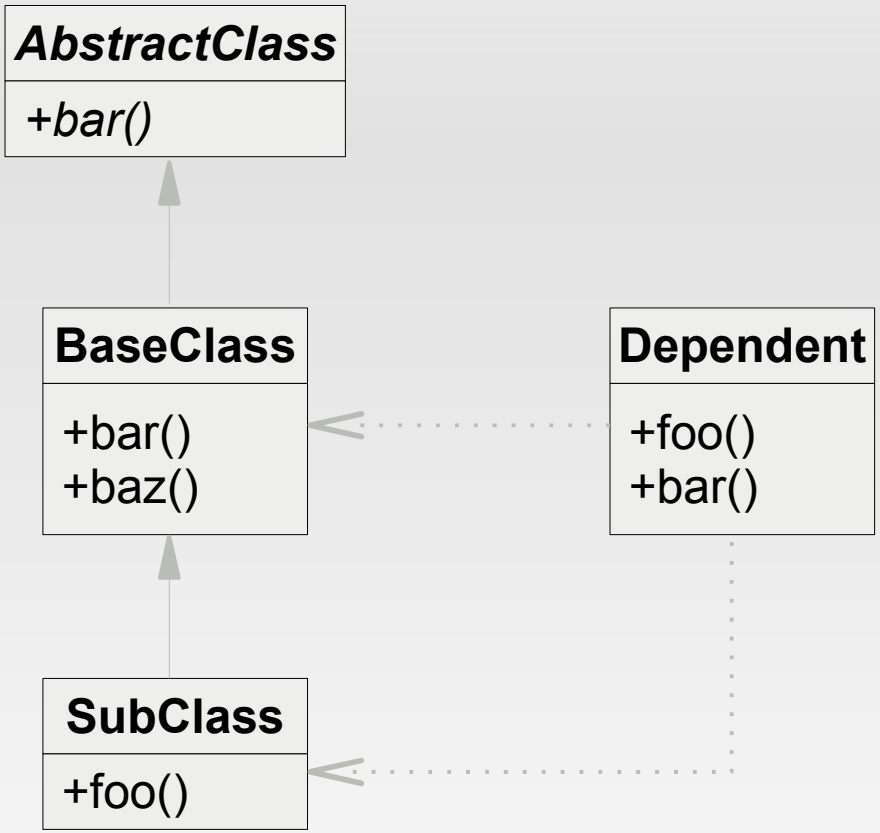
- Chidamber & Kemerer OO-Metriken
- Afferent Coupling / Efferent Coupling
- Abstraction, Instability, Distance
  - Modell für eine ausgewogene Architektur
- Code Rank
  - (In)direkte Abhängigkeiten zwischen Komponenten

Chidamber & Kemerer



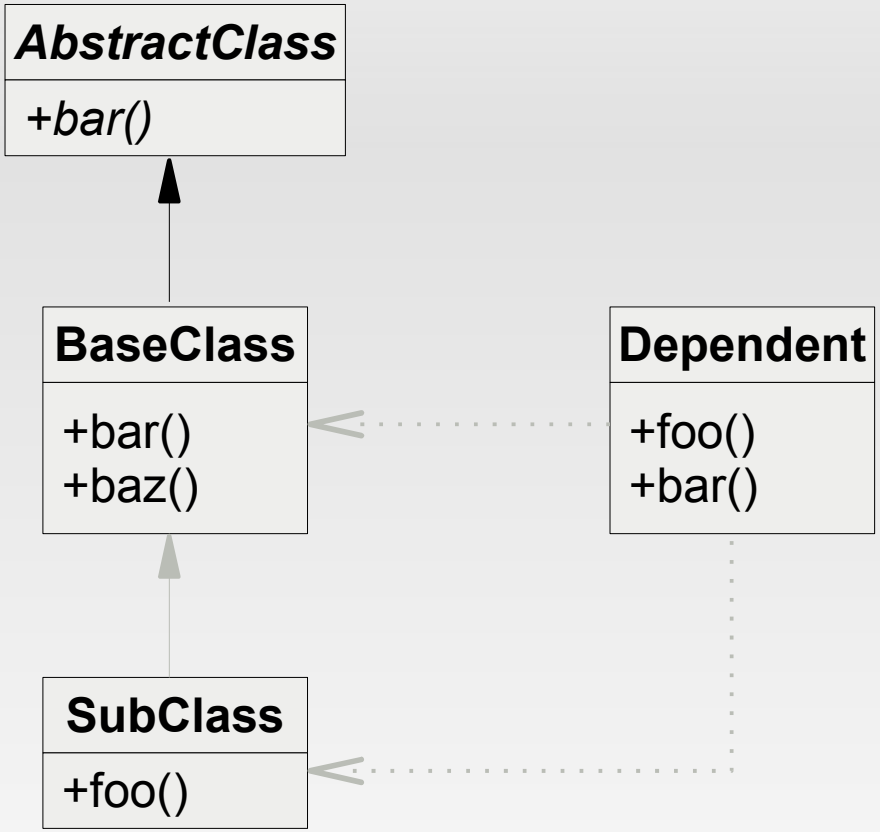
- Weighted Methods per Class (WMC)
  - Summe der Komplexität aller Methoden
  - Grenzwert 20 - 50
- Number Of Children (NOC)
  - Anzahl der direkten Ableitungen
  - Falsch gewählte Abstraktion
- Depth of Inheritance Tree (DIT)
  - Vererbungsstrukturen erhöhen die Komplexität
  - Grenzwert <= 5

CK OO-Metriken



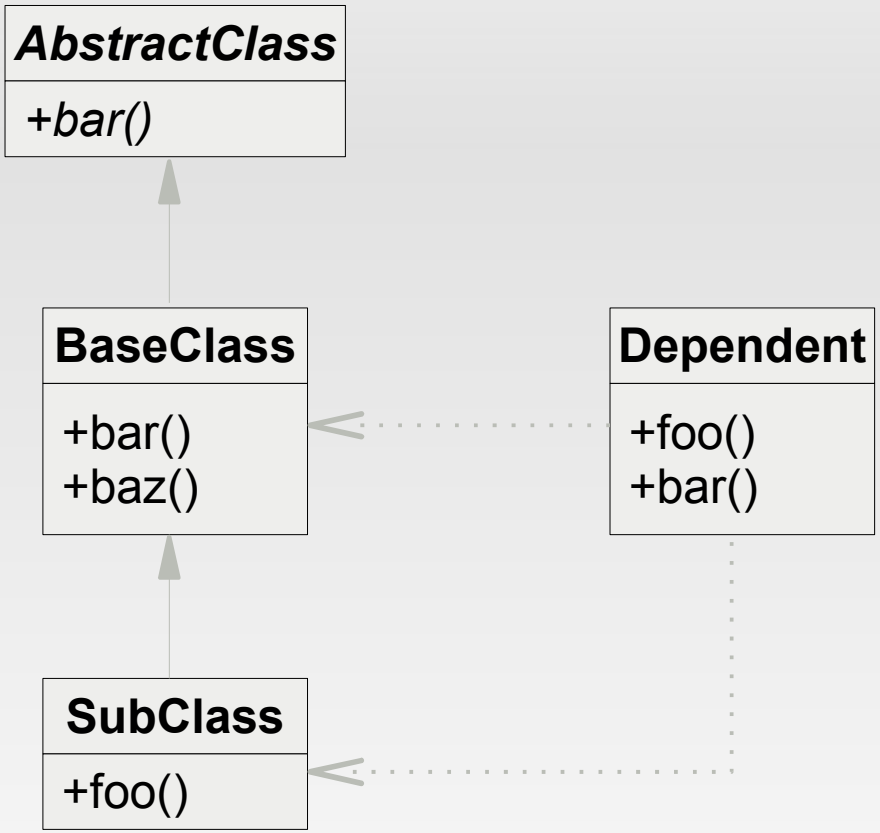
- *AbstractClass*
  - NOC = ; DIT =
- BaseClass
  - NOC = ; DIT =
- SubClass
  - NOC = ; DIT =
- Dependent
  - NOC = ; DIT =

CK OO-Metriken



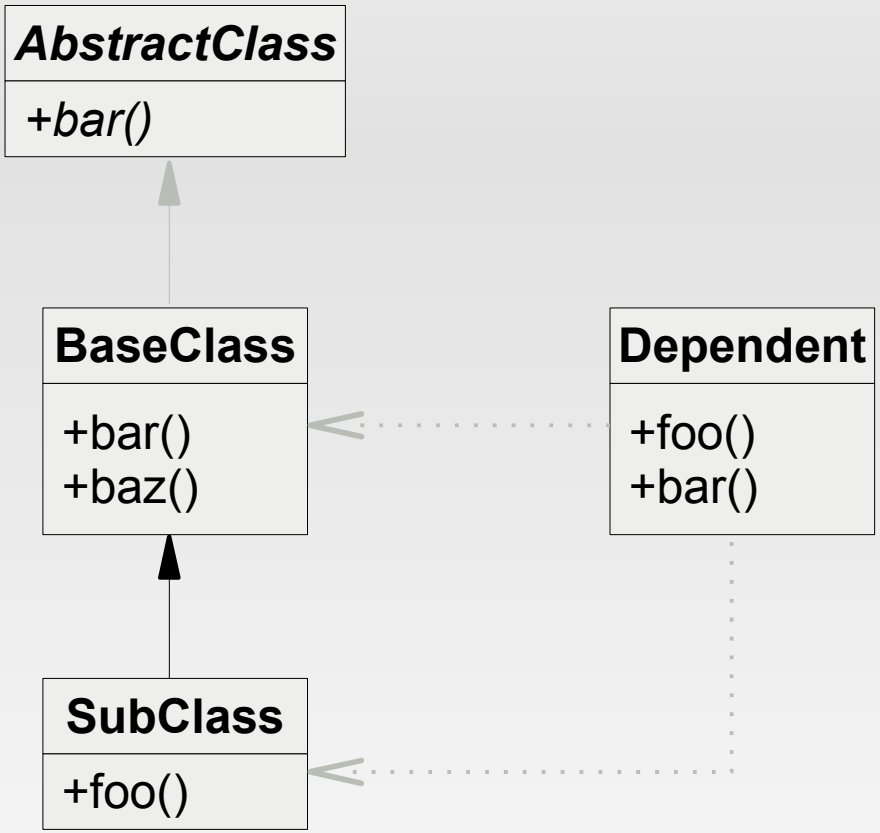
- *AbstractClass*
  - NOC = 1; DIT = 0
- BaseClass
  - NOC = ; DIT =
- SubClass
  - NOC = ; DIT =
- Dependent
  - NOC = ; DIT =

CK OO-Metriken



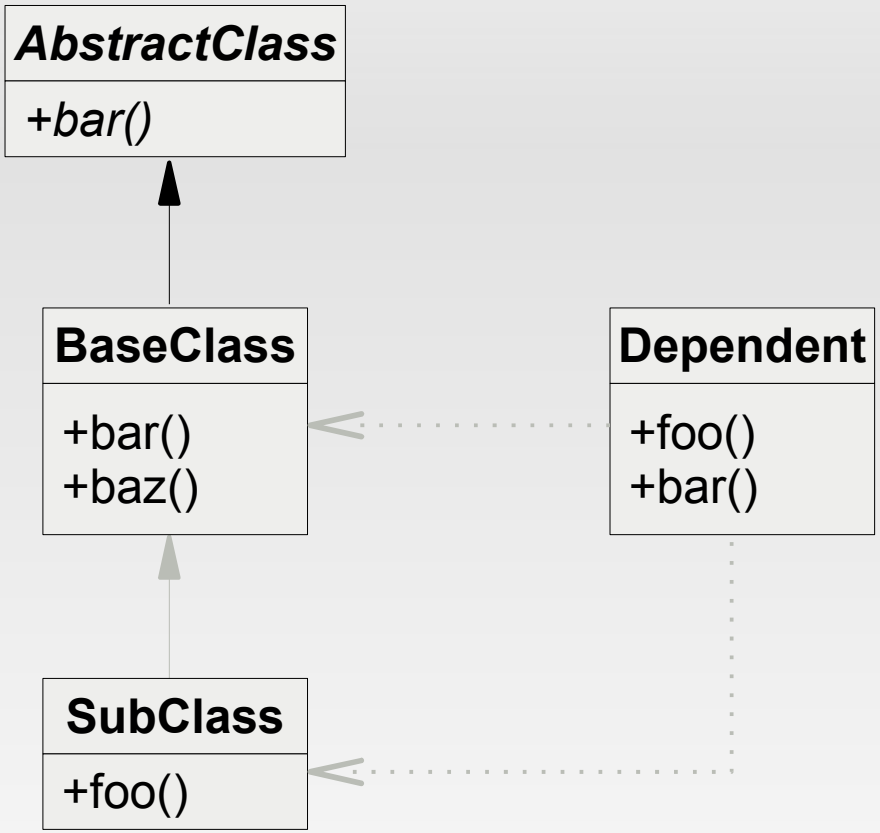
- *AbstractClass*
  - NOC = 1; DIT = 0
- BaseClass
  - NOC = ; DIT =
- SubClass
  - NOC = ; DIT =
- Dependent
  - NOC = ; DIT =

CK OO-Metriken



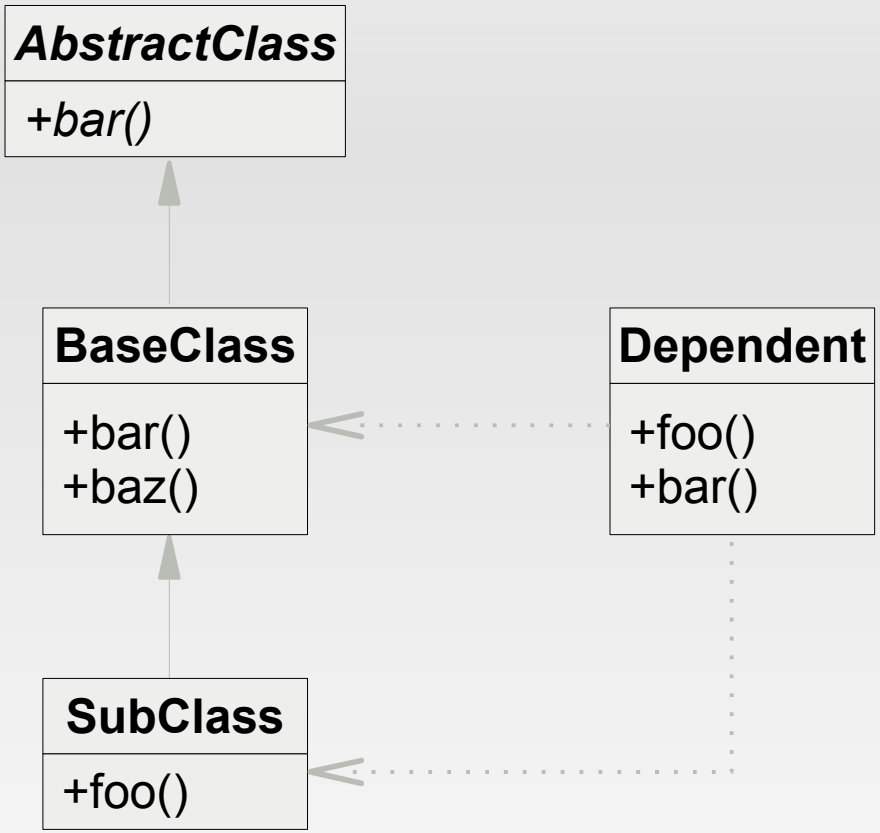
- *AbstractClass*
  - NOC = 1; DIT = 0
- BaseClass
  - NOC = 1; DIT = 1
- SubClass
  - NOC = ; DIT =
- Dependent
  - NOC = ; DIT =

CK OO-Metriken



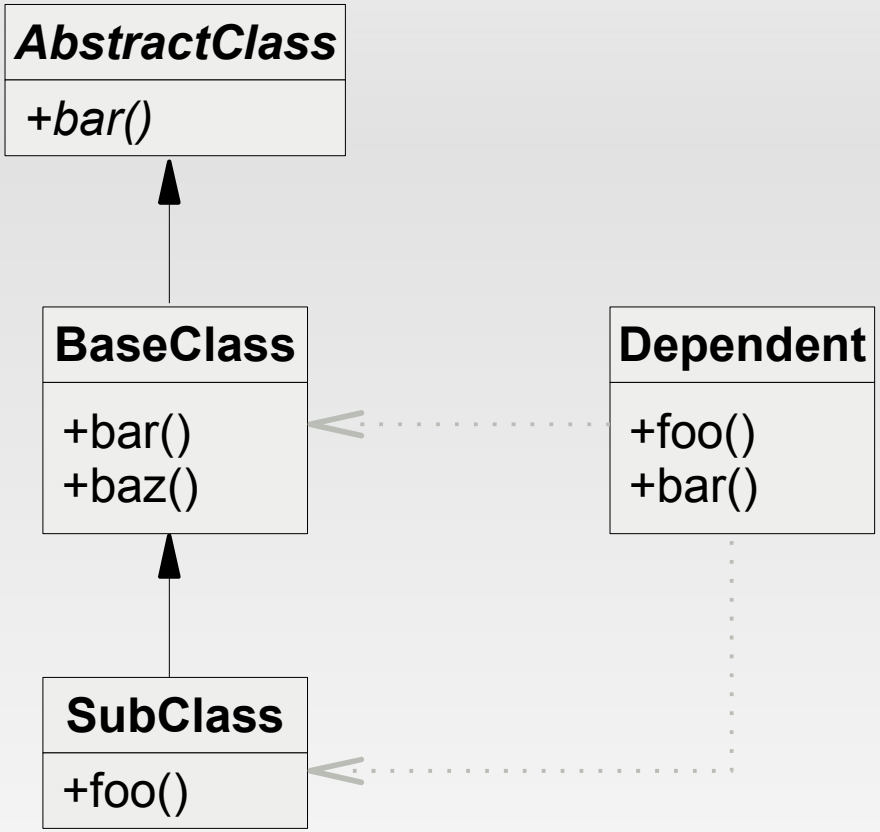
- *AbstractClass*
  - NOC = 1; DIT = 0
- BaseClass
  - NOC = 1; DIT = 1
- SubClass
  - NOC = ; DIT =
- Dependent
  - NOC = ; DIT =

CK OO-Metriken



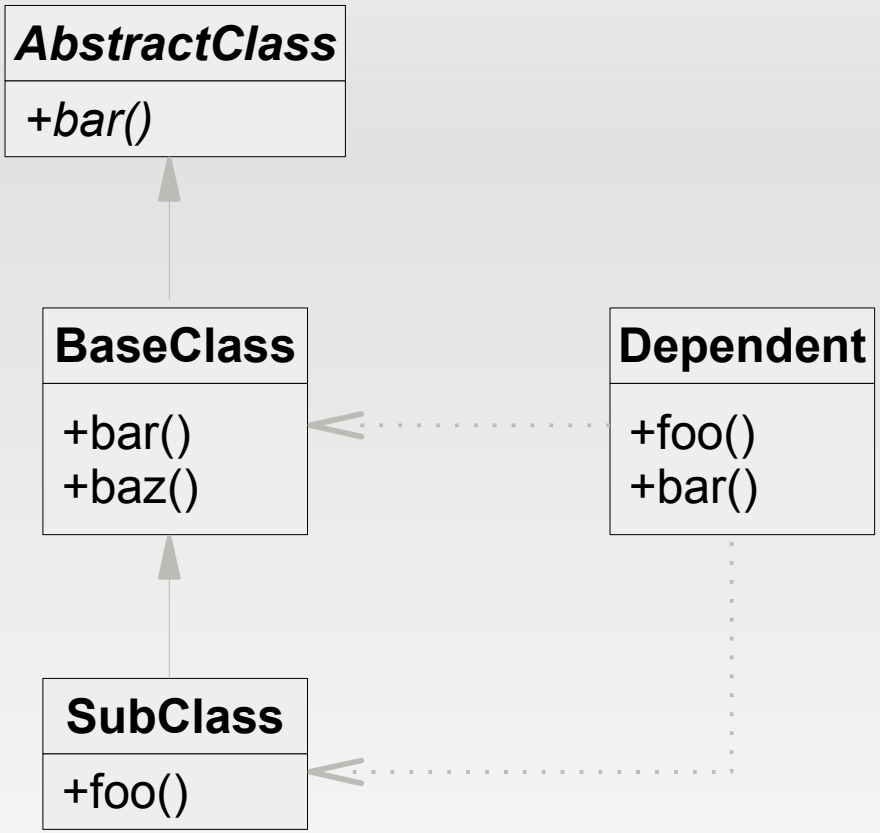
- *AbstractClass*
  - NOC = 1; DIT = 0
- BaseClass
  - NOC = 1; DIT = 1
- SubClass
  - NOC = 0; DIT = 2
- Dependent
  - NOC = ; DIT =

CK OO-Metriken



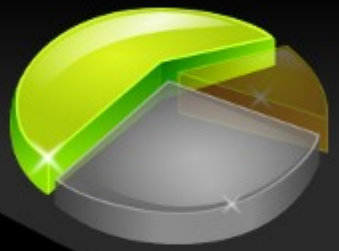
- *AbstractClass*
  - NOC = 1; DIT = 0
- BaseClass
  - NOC = 1; DIT = 1
- SubClass
  - NOC = 0; DIT = 2
- Dependent
  - NOC = ; DIT =

CK OO-Metriken



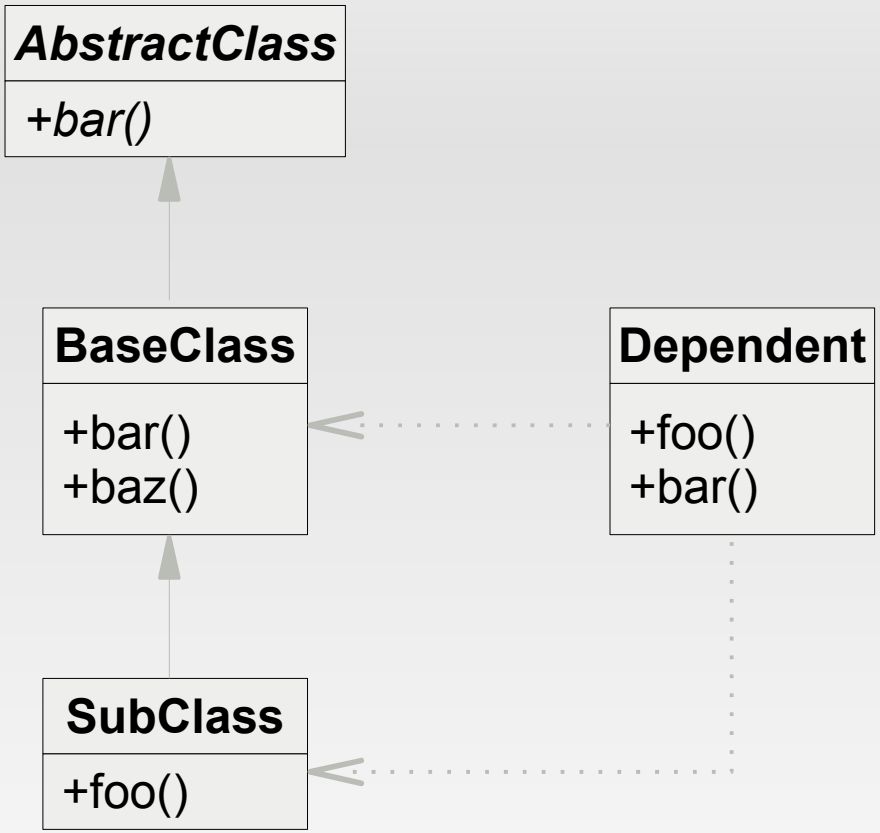
- *AbstractClass*
  - NOC = 1; DIT = 0
- BaseClass
  - NOC = 1; DIT = 1
- SubClass
  - NOC = 0; DIT = 2
- Dependent
  - NOC = 0; DIT = 0

Afferent- & Efferent-Coupling



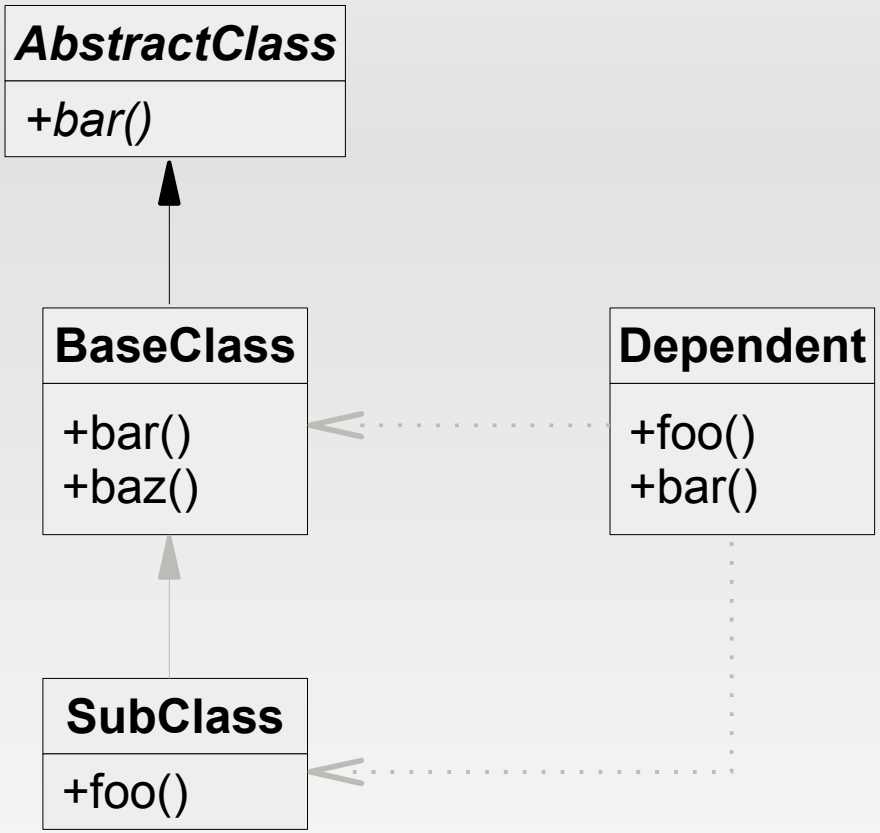
- Afferent Coupling (Ca)
  - Eingehende Abhängigkeiten anderer Komponenten
  - Großer Einfluss aus die Stabilität des Systems
- Efferent Coupling (Ce)
  - Ausgehende Abhängigkeiten:
    - Vererbung, Parameter, Exceptions, Allokationen
  - Abhängigkeit von Stabilität anderer Komponenten

Afferent- & Efferent-Coupling



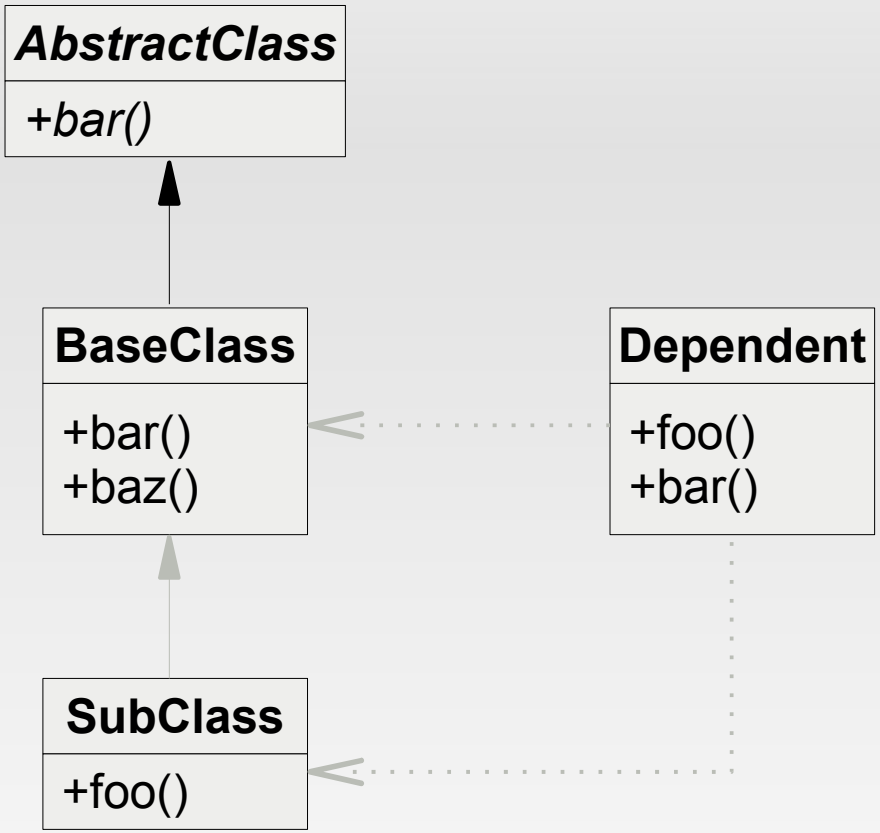
- *AbstractClass*
  - Ce = ; Ca =
- BaseClass
  - Ce = ; Ca =
- SubClass
  - Ce = ; Ca =
- Dependent
  - Ce = ; Ca =

Afferent- & Efferent-Coupling



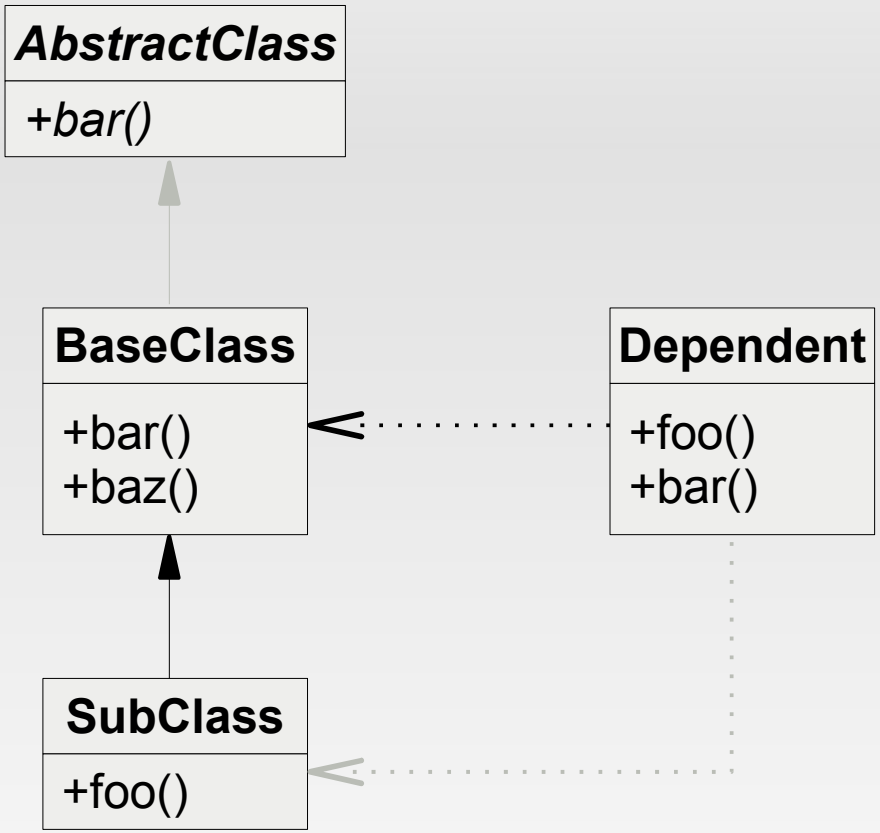
- *AbstractClass*
  - Ce = 0; Ca = 1
- BaseClass
  - Ce = ; Ca =
- SubClass
  - Ce = ; Ca =
- Dependent
  - Ce = ; Ca =

Afferent- & Efferent-Coupling



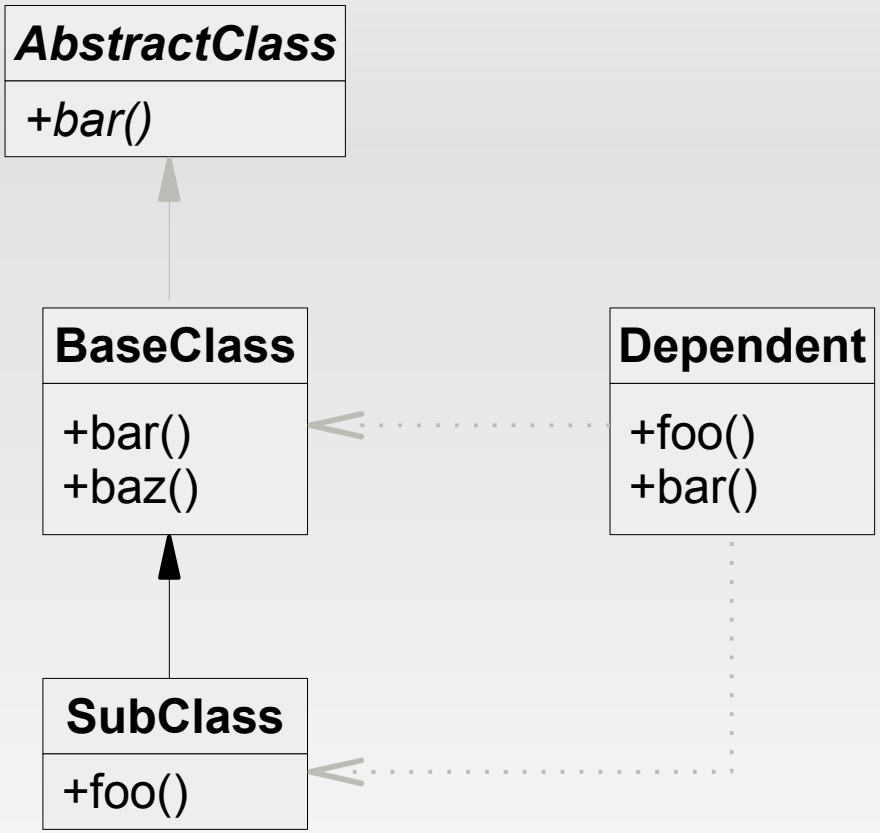
- *AbstractClass*
  - Ce = 0; Ca = 1
- BaseClass
  - Ce = 1; Ca = 2
- SubClass
  - Ce = ; Ca =
- Dependent
  - Ce = ; Ca =

Afferent- & Efferent-Coupling



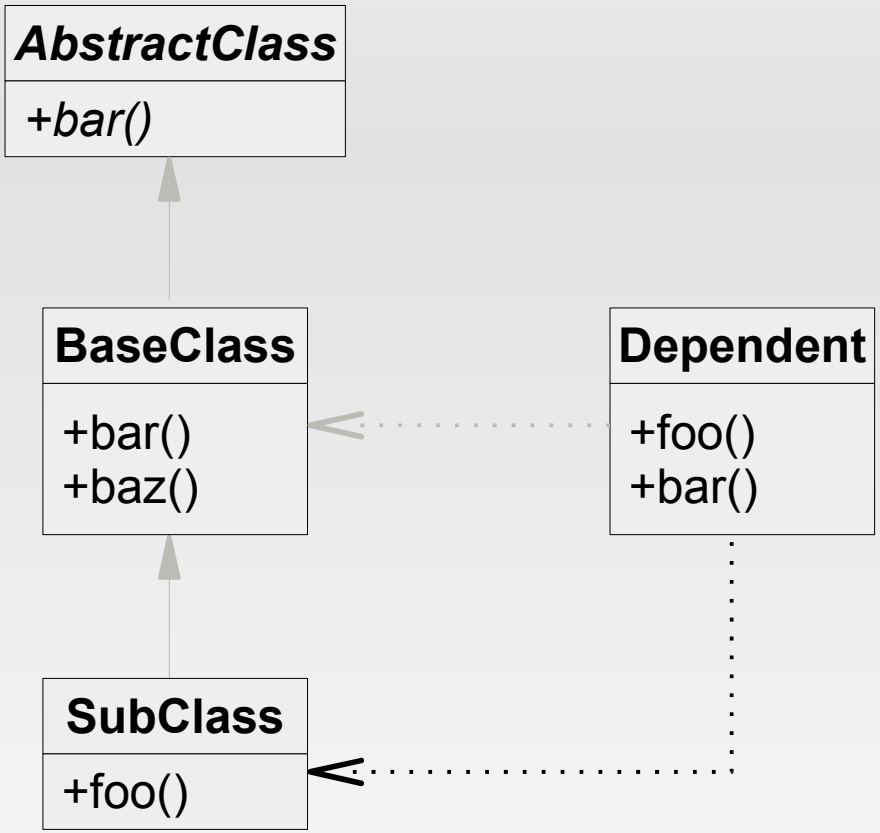
- *AbstractClass*
  - Ce = 0; Ca = 1
- BaseClass
  - Ce = 1; Ca = 2
- SubClass
  - Ce = ; Ca =
- Dependent
  - Ce = ; Ca =

Afferent- & Efferent-Coupling



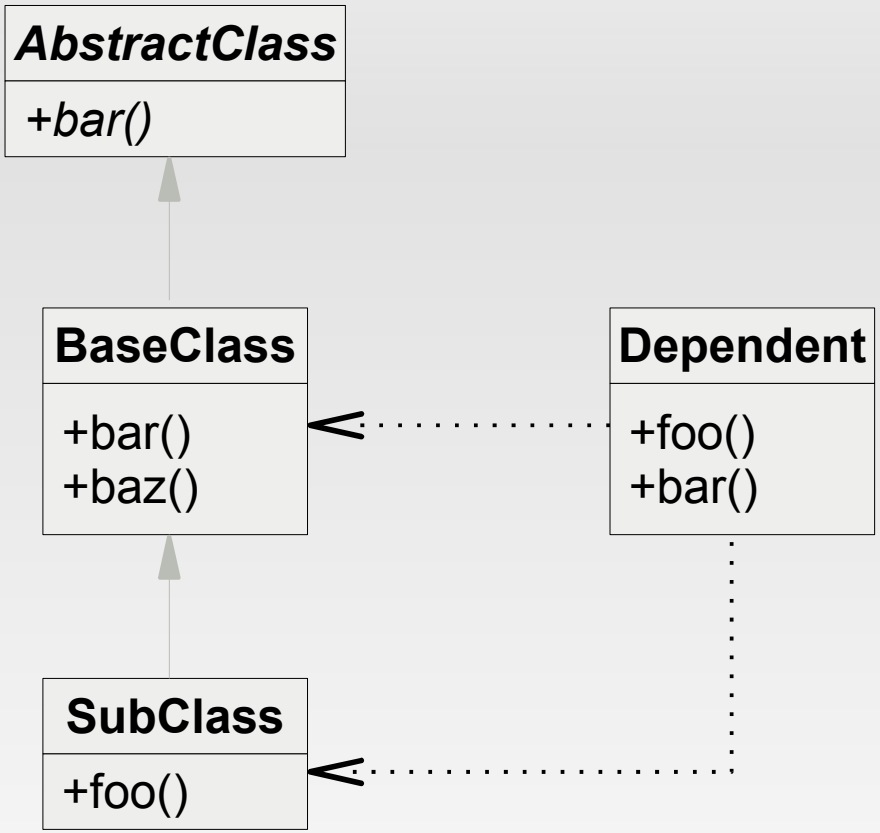
- *AbstractClass*
  - Ce = 0; Ca = 1
- BaseClass
  - Ce = 1; Ca = 2
- SubClass
  - Ce = 1; Ca = 1
- Dependent
  - Ce = ; Ca =

Afferent- & Efferent-Coupling



- *AbstractClass*
  - Ce = 0; Ca = 1
- BaseClass
  - Ce = 1; Ca = 2
- SubClass
  - Ce = 1; Ca = 1
- Dependent
  - Ce = ; Ca =

Afferent- & Efferent-Coupling



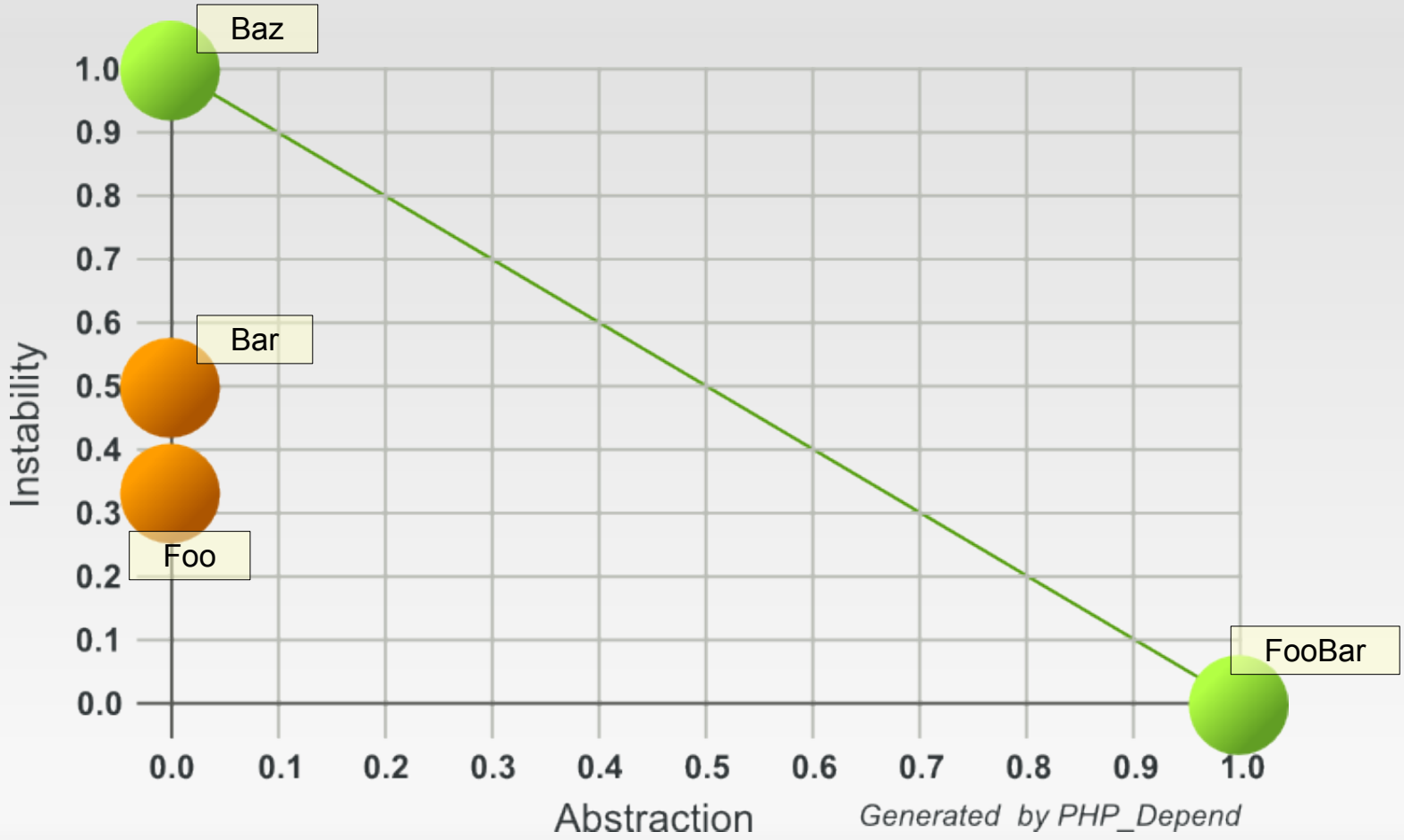
- *AbstractClass*
  - Ce = 0; Ca = 1
- BaseClass
  - Ce = 1; Ca = 2
- SubClass
  - Ce = 1; Ca = 1
- Dependent
  - Ce = 2; Ca = 0

Abstraction, Instability&Distance

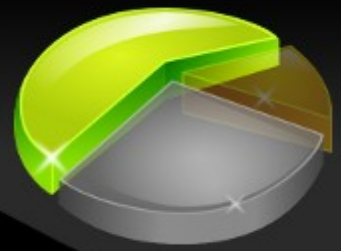
- Modell für eine ausgewogene Architektur
  - Basiert auf Ce, Ca, abstrakten(AC) und konkreten(CC) Softwareartefakten
  - Abstraction
    - $A = AC / (AC + CC)$
  - Instability
    - $I = Ce / (Ce + Ca)$
  - Distance
    - $D = |A + I| - 1$

Abstraction, Instability&Distance

- Unser Beispiel übertragen auf einen Graphen

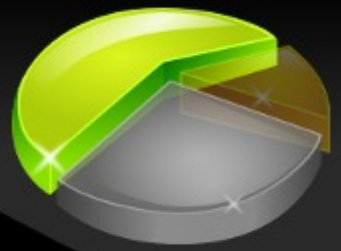


CodeRank



- Basiert auf dem Google PageRank
- Software wird auf eine Graphen abgebildet
  - Knoten(π) je Softwareartefakt
    - Pakete, Klassen, Methoden
  - Kanten(ρ) für jede Beziehung
    - Vererbung, Aufrufe, Parameter, Exceptions
- Für den CodeRank gilt:
  - $CR(\pi_i) = \sum_r((1 - d) + d \sum_r (CR(\pi_r) / \rho_r))$

CodeRank



CR_n(**FooBar**) = (1 -d) + d * CR_{n-1}(**Foo**)

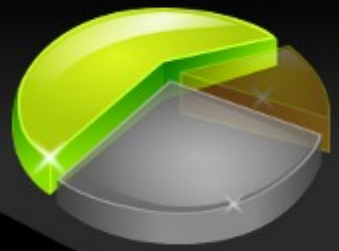
CR_n(**Foo**) = (1 -d) + d * (CR_{n-1}(**Bar**) + 0,5 * CR_{n-1}(**Baz**))

CR_n(**Bar**) = (1 -d) + d * (0,5 * Cr_{n-1}(**Baz**))

CR_n(**Baz**) = (1 -d) + d * 0

#	*FooBar*	Foo	Bar	Baz
0	1	1	1	1

CodeRank



$CR_1(\mathbf{FooBar}) = (1 - d) + d * CR_0(\mathbf{1})$

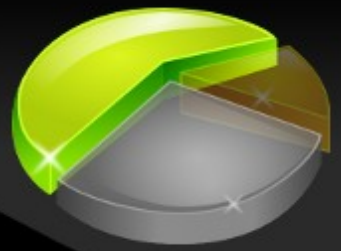
$CR_1(\mathbf{Foo}) = (1 - d) + d * (CR_0(\mathbf{1}) + 0{,}5 * CR_0(\mathbf{1}))$

$CR_1(\mathbf{Bar}) = (1 - d) + d * (0{,}5 * CR_0(\mathbf{1}))$

$CR_1(\mathbf{Baz}) = (1 - d) + d * 0$

#	*FooBar*	Foo	Bar	Baz
0	1	1	1	1
1	1,0000000	1,4250000	0,5750000	0,1500000

CodeRank



CR_2(**FooBar**) = (1 -d) + d * CR_1(**1,425**)

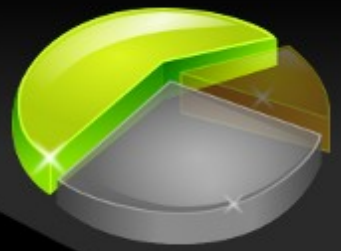
CR_2(**Foo**) = (1 -d) + d * (CR_1(**0,575**) + 0,5 * CR_1(**0,15**))

CR_2(**Bar**) = (1 -d) + d * (0,5 * CR_1(**0,15**))

CR_2(**Baz**) = 0,15

#	*FooBar*	Foo	Bar	Baz
0	1	1	1	1
1	1,0000000	1,4250000	0,5750000	0,1500000
2	1,3612500	0,7025000	0,2137500	0,1500000

CodeRank



CR_3(**FooBar**) = (1 -d) + d * CR_2(**0,7025**)

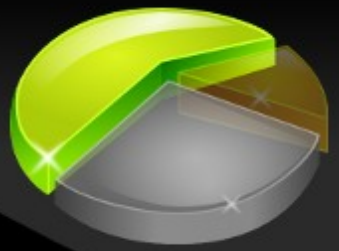
CR_3(**Foo**) = (1 -d) + d * (CR_2(**0,21375**) + 0,5 * CR_2(**0,15**))

CR_3(**Bar**) = (1 -d) + d * (0,5 * CR_2(**0,15**))

CR_3(**Baz**) = 0,15

#	*FooBar*	Foo	Bar	Baz
0	1	1	1	1
1	1,0000000	1,4250000	0,5750000	0,1500000
2	1,3612500	0,7025000	0,2137500	0,1500000
3	0,7471250	0,3954375	0,2137500	0,1500000

CodeRank



CR_4(**FooBar**) = (1 -d) + d * CR_3(**0,3954375**)

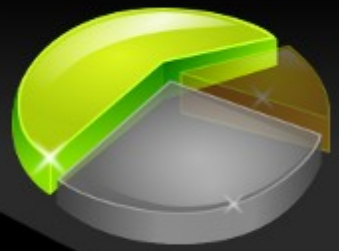
CR_4(**Foo**) = (1 -d) + d * (CR_3(**0,21375**) + 0,5 * CR_3(**0,15**))

CR_4(**Bar**) = (1 -d) + d * (0,5 * CR_3(**0,15**))

CR_4(**Baz**) = 0,15

#	*FooBar*	Foo	Bar	Baz
0	1	1	1	1
1	1,0000000	1,4250000	0,5750000	0,1500000
2	1,3612500	0,7025000	0,2137500	0,1500000
3	0,7471250	0,3954375	0,2137500	0,1500000
4	0,4861218	0,3954375	0,2137500	0,1500000

CodeRank



CR_5(**FooBar**) = (1 -d) + d * CR_4(**0,3954375**)

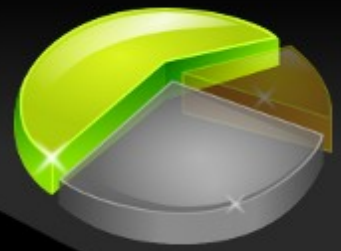
CR_5(**Foo**) = (1 -d) + d * (CR_4(**0,21375**) + 0,5 * CR_4(**0,15**))

CR_5(**Bar**) = (1 -d) + d * (0,5 * CR_4(**0,15**))

CR_5(**Baz**) = 0,15

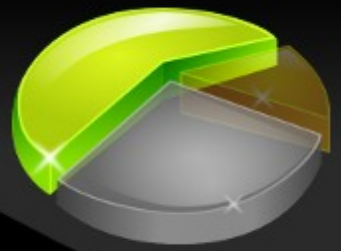
#	*FooBar*	Foo	Bar	Baz
0	1	1	1	1
1	1,0000000	1,4250000	0,5750000	0,1500000
2	1,3612500	0,7025000	0,2137500	0,1500000
3	0,7471250	0,3954375	0,2137500	0,1500000
4	0,4861218	0,3954375	0,2137500	0,1500000
5	0,4861218	0,3954375	0,2137500	0,1500000

CodeRank



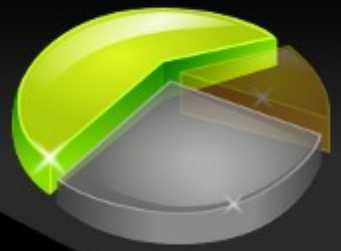
- Mit dem CodeRank können auch indirekte Abhängigkeiten bewertet werden.
- Logik mit Einfluss auf das gesamte System kann schnell gefunden werden
- Äquivalent der Reverse CodeRank
  - Kann mit dem selben Algorithmus berechnet werden
  - Gibt Aufschluss über stark abhängige Komponenten

Metriken sind ...



- ... keine Magie, sondern einfache Maßzahlen
- ... nutzlos, ohne Grenz- oder Referenzwerte
- ... skalierbar, wachsen mit der Projektgröße
- ... automatisierbar und reproduzierbar
- ... objektiv, da Software gestützt
- ... interpretierbar, abhängig vom Betrachter

Done...



...