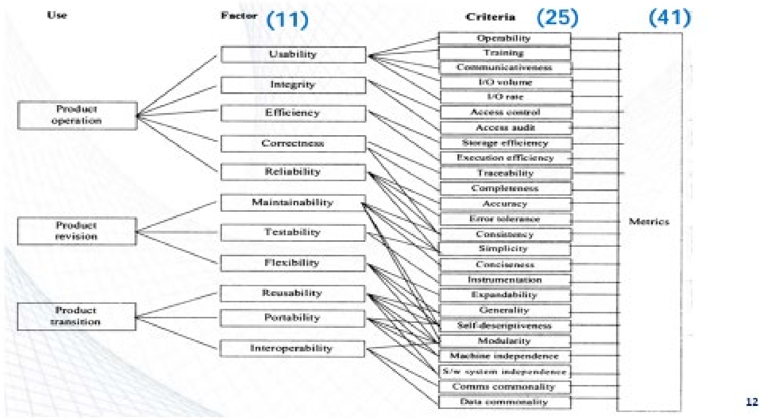
SE 433/333 Software Testing & Quality Assurance

Code-smells and bad-design practices

Last Week

McCall's Model

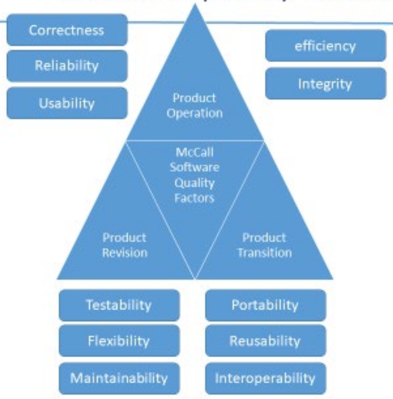


Chidamber and Kemerer OO Metrics (CK - 1994)

- Weighted methods per class (MWC)
- Depth of inheritance tree (DIT)
- Number of children (NOC)
- Coupling between object classes (CBO)
- Response for class (RFC)
- Lack of cohesion metric (LCOM)

28

McCall's quality factors



6

Functional vs Non-Functional Requirements

Functional Requirements

Functionality
of the program

Non-Functional Requirements

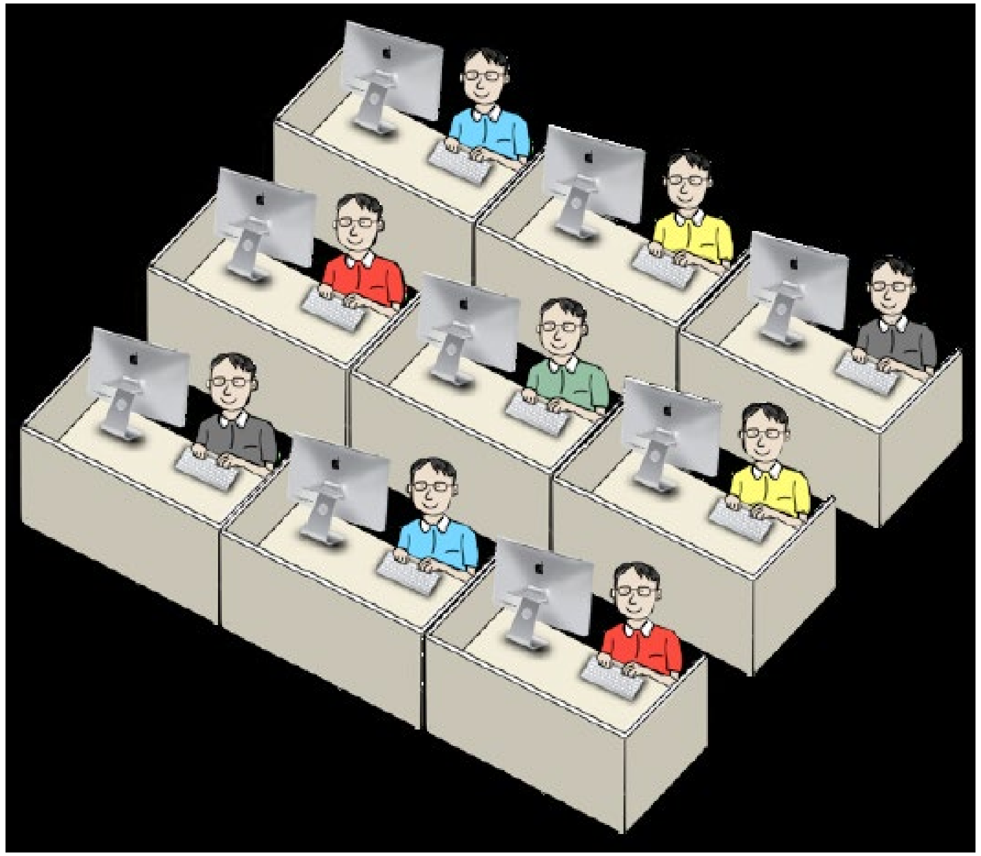
Execution Time

Size

Battery

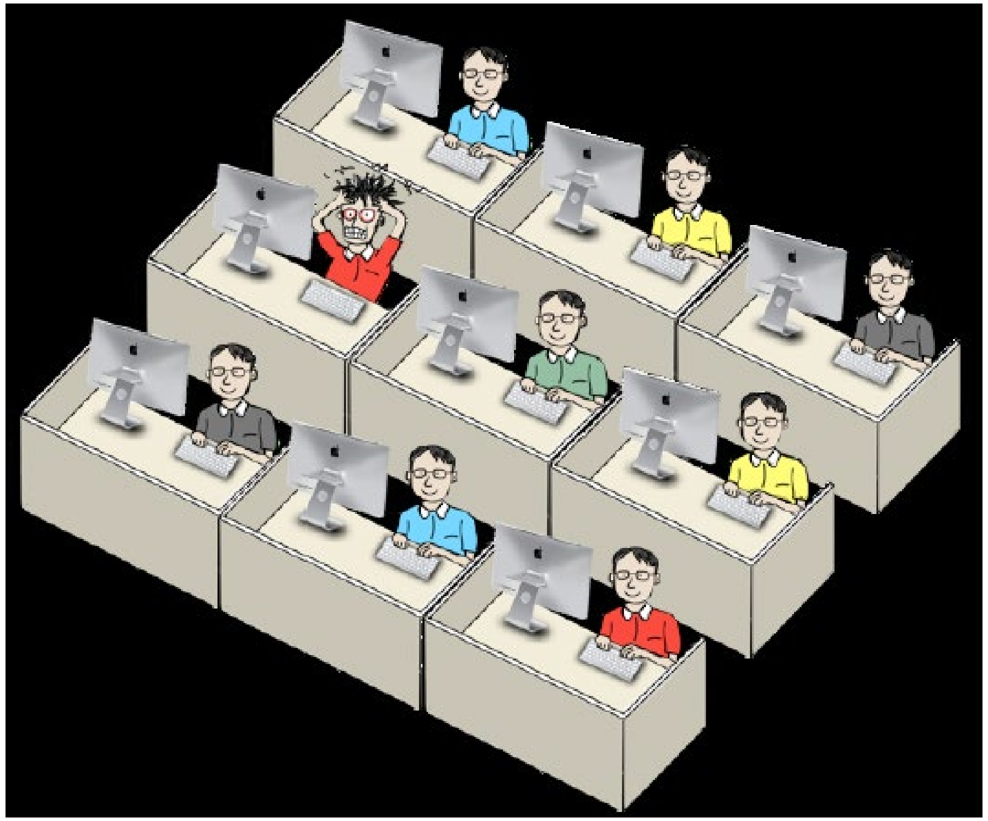
Memory

- Text chat
- Friends list



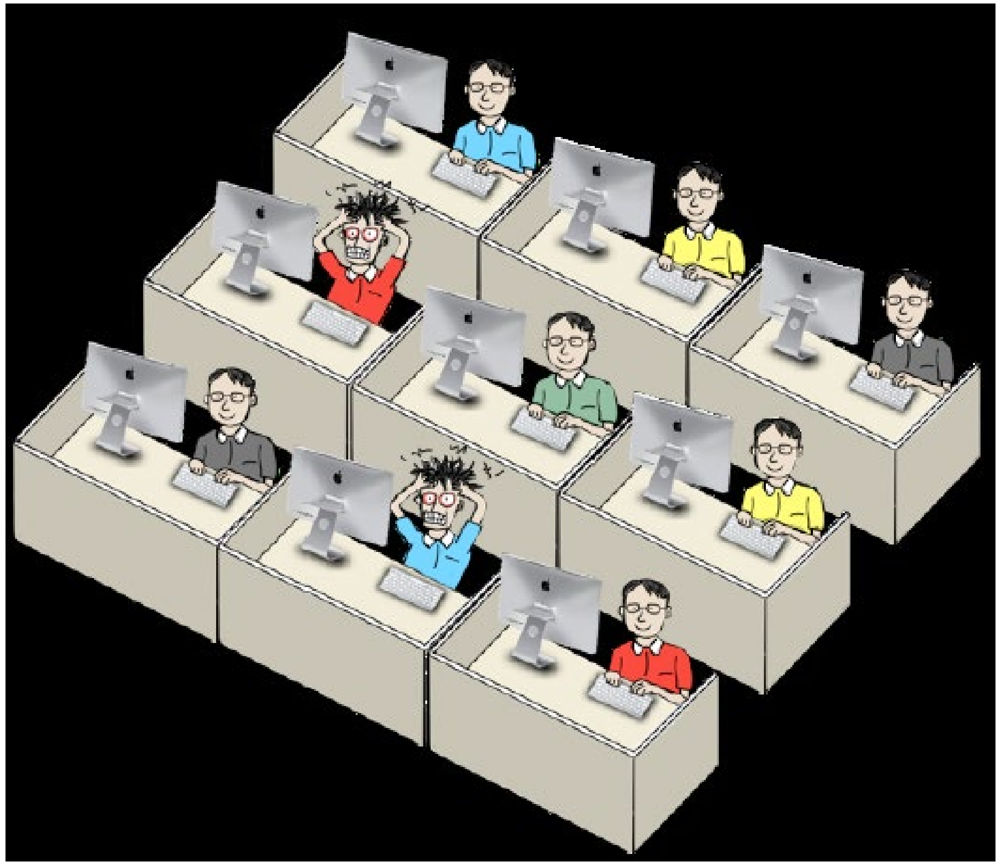
- Text chat
- Friends list

- Voice Communication



- Text chat
- Friends list

- Voice Communication



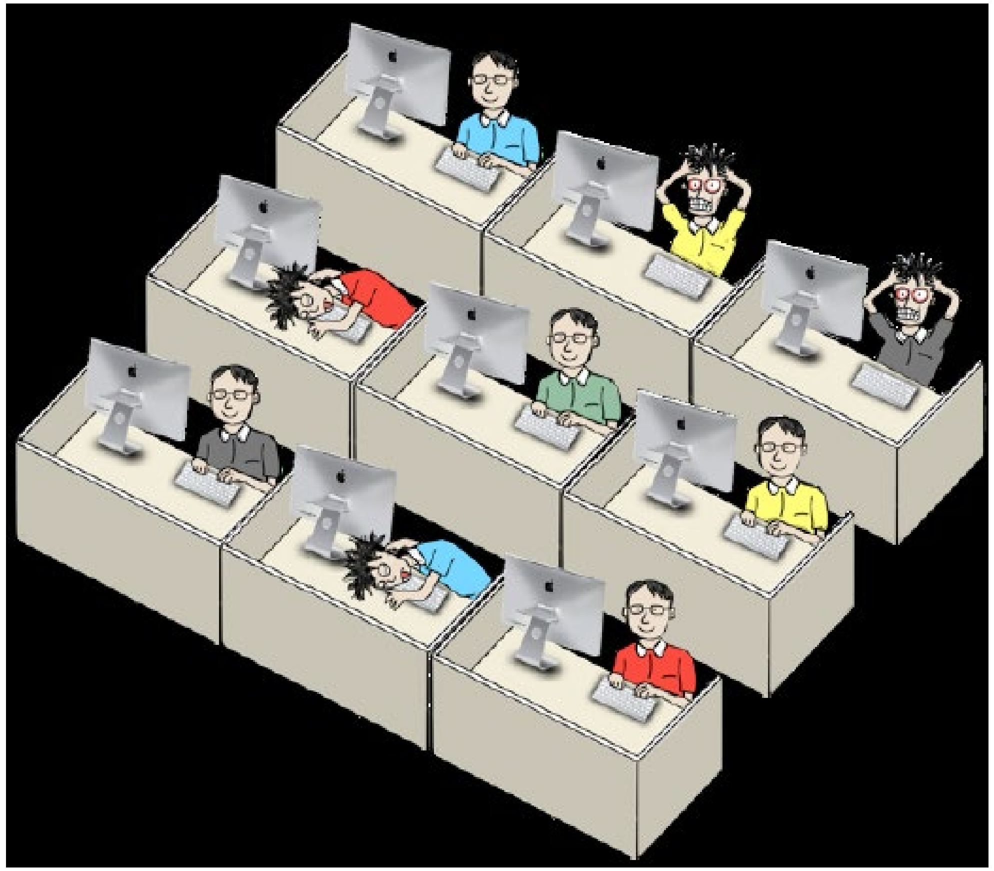
- Text chat
- Friends list

- Voice Communication

-Battery consumption

-Execution time

-Size



- Text chat
- Friends list

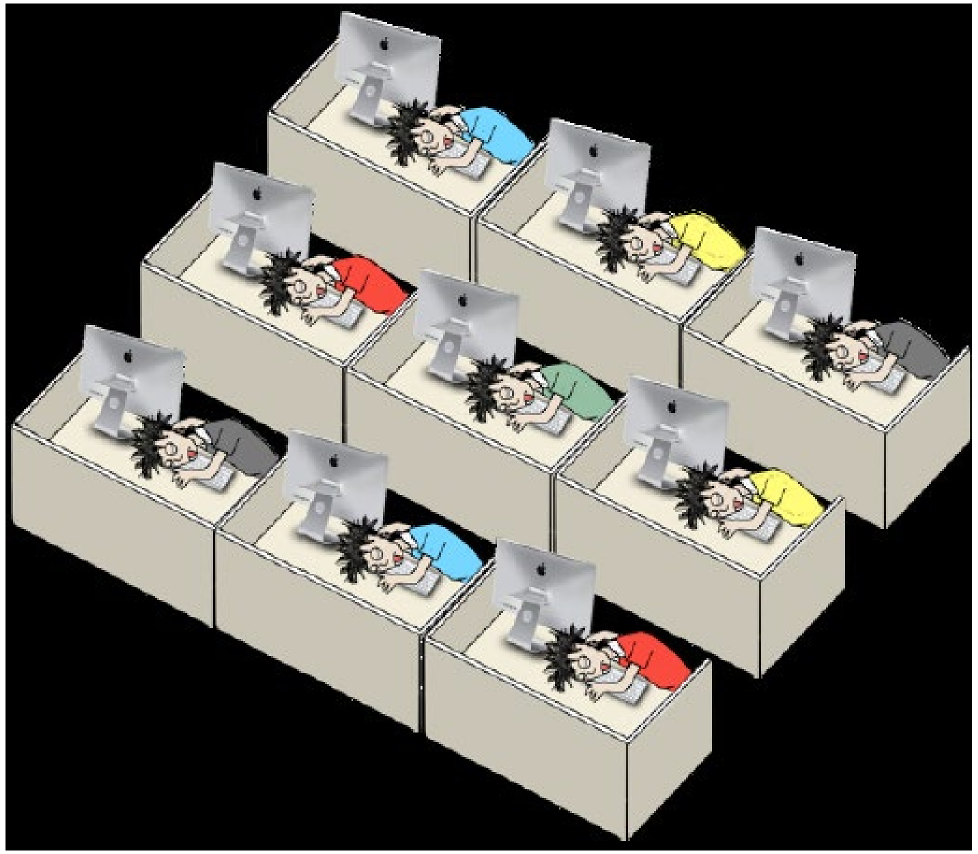
- Voice Communication

-Execution time

-Battery consumption

-Size

-Memory leak



Functional Requirements

humans have to define these

Non-Functional Requirements

a machine can optimize these

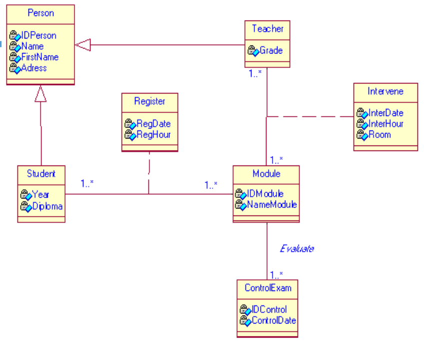
Functional Requirements

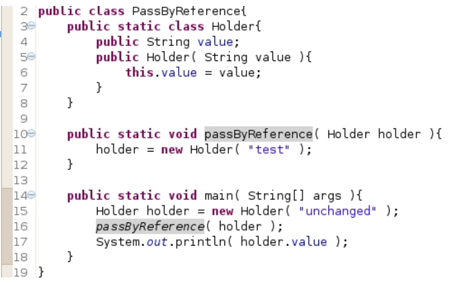
humans have to define these

Non-Functional Requirements

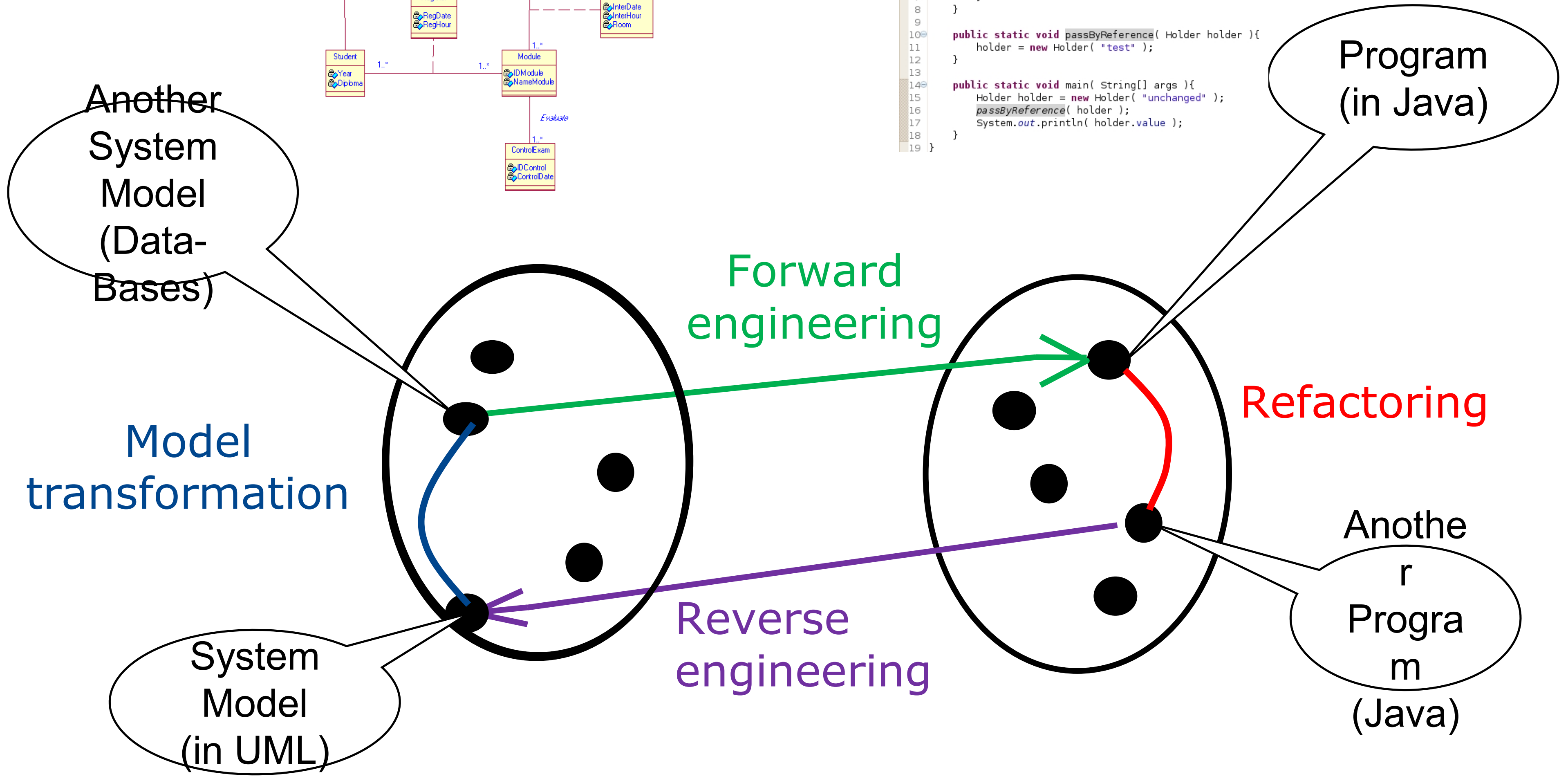
a machine can optimize these

- Software Refactoring:
  - Antipatterns Detection
  - Antipatterns Correction (Refactoring Recommendation)





```java
public class PassByReference{
    public static class Holder{
        public String value;
        public Holder( String value ){
            this.value = value;
        }
    }

    public static void passByReference( Holder holder ){
        holder = new Holder( "test" );
    }

    public static void main( String[] args ){
        Holder holder = new Holder( "unchanged" );
        passByReference( holder );
        System.out.println( holder.value );
    }
}
```



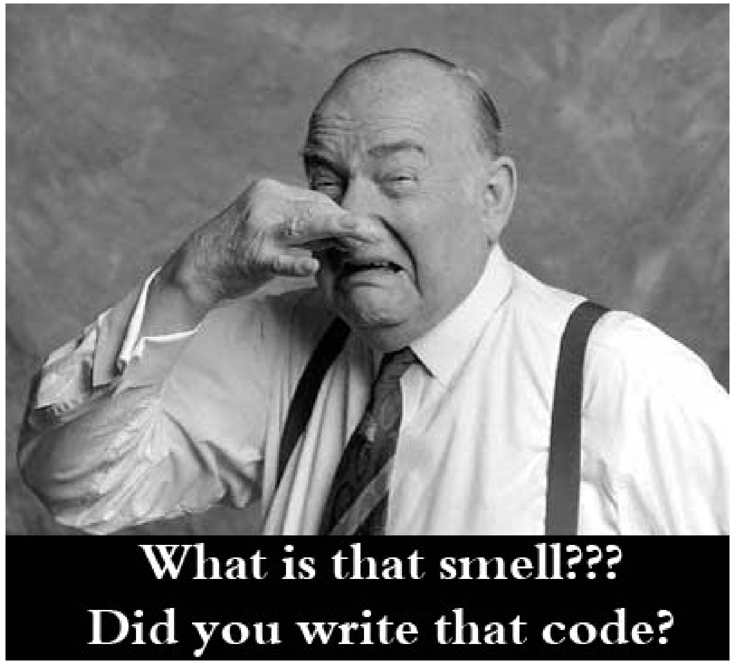
Model space **Source code space**

- Refactoring
  - The process of improving a code after it has been written by changing its internal structure without changing the external behavior (Fowler et al., ‘99)
  - Examples: *Move method, extract class, move attribute, ...*

- Two main refactoring steps
  1. detection of code fragments to improve (e.g., Anti pattern)
  2. identification of refactoring solutions

Learning objectives

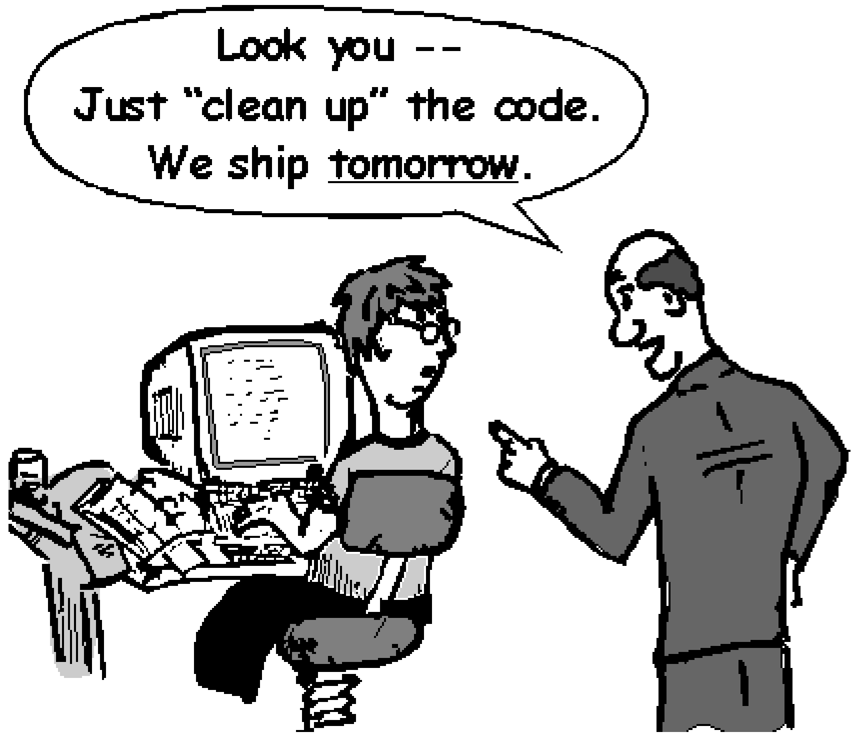
- Anti-patterns: Overview
- Examples of anti-patterns
- How to detect anti-patterns using quality metrics?



Design defects/antipatterns

- Design defects/antipatterns are poor coding and design choices introduced during different phases of software development
  - Anomalies, code smells , bad smells…
  - Make the design harder to understand, to change
  - Design situations that adversely affect the development of a software (**not bugs**)

Management Antipatterns



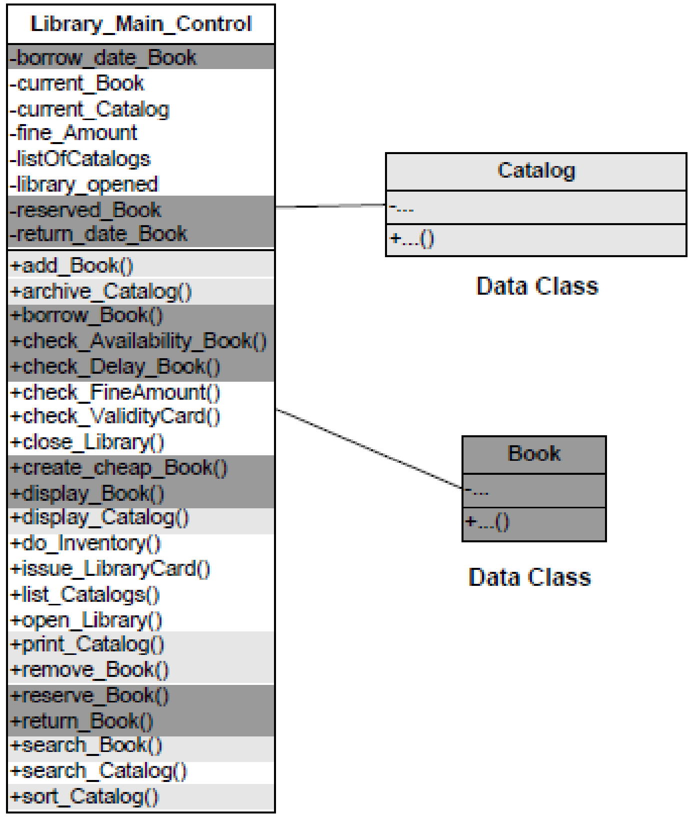
Examples of antipatterns

- Duplicated code
- Long method
- God class
- Long parameter list
- Message chain
- Data class
- Functional decomposition
- …

The Blob (God Class)

- Definition
  - Procedural-style design leads to one object with numerous responsibilities and most other objects only holding data or executing simple processes.

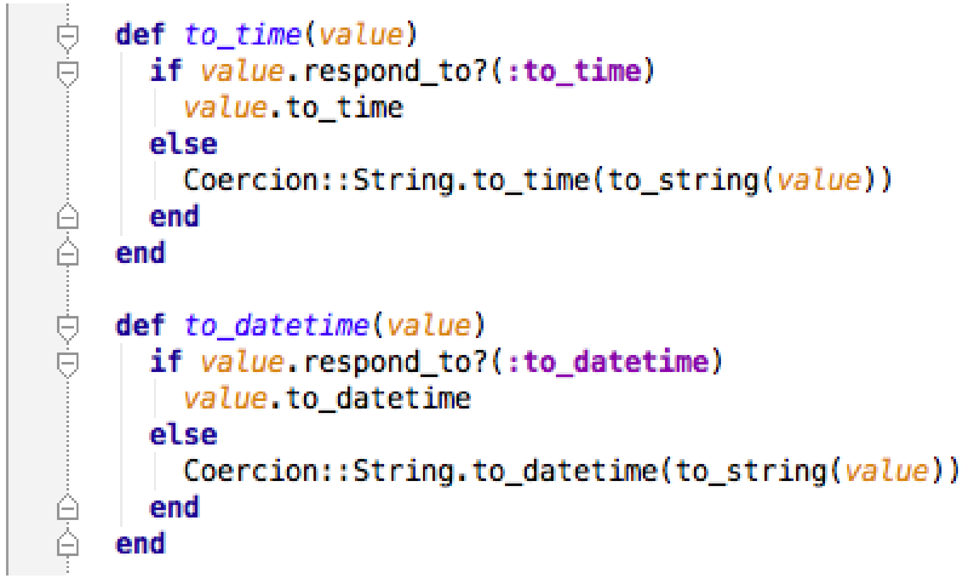
- Symptoms
  - A Blob is a *controller* class, *abnormally large*, with *almost* no parents and no children. It *mainly uses* data classes, i.e. *very small* classes with *almost* no parents and no children (Brown et al. '98).



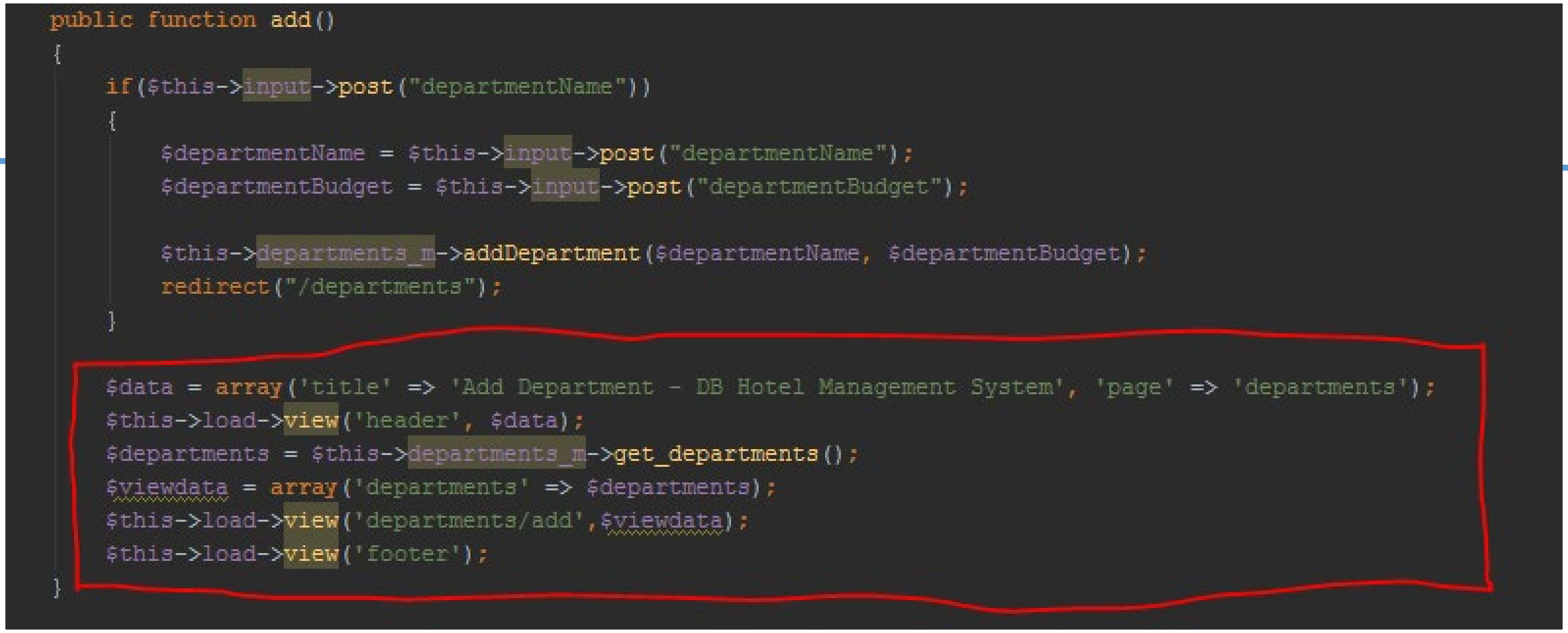
- Causes
  - *Lack of an object-oriented architecture.*
    - *Inadequate understanding of OO principles*
  - *Lack of (any) architecture*
    - *Design, interaction between object etc.*
  - *Too limited intervention*

Duplicate Code or Cut and Paste Programming

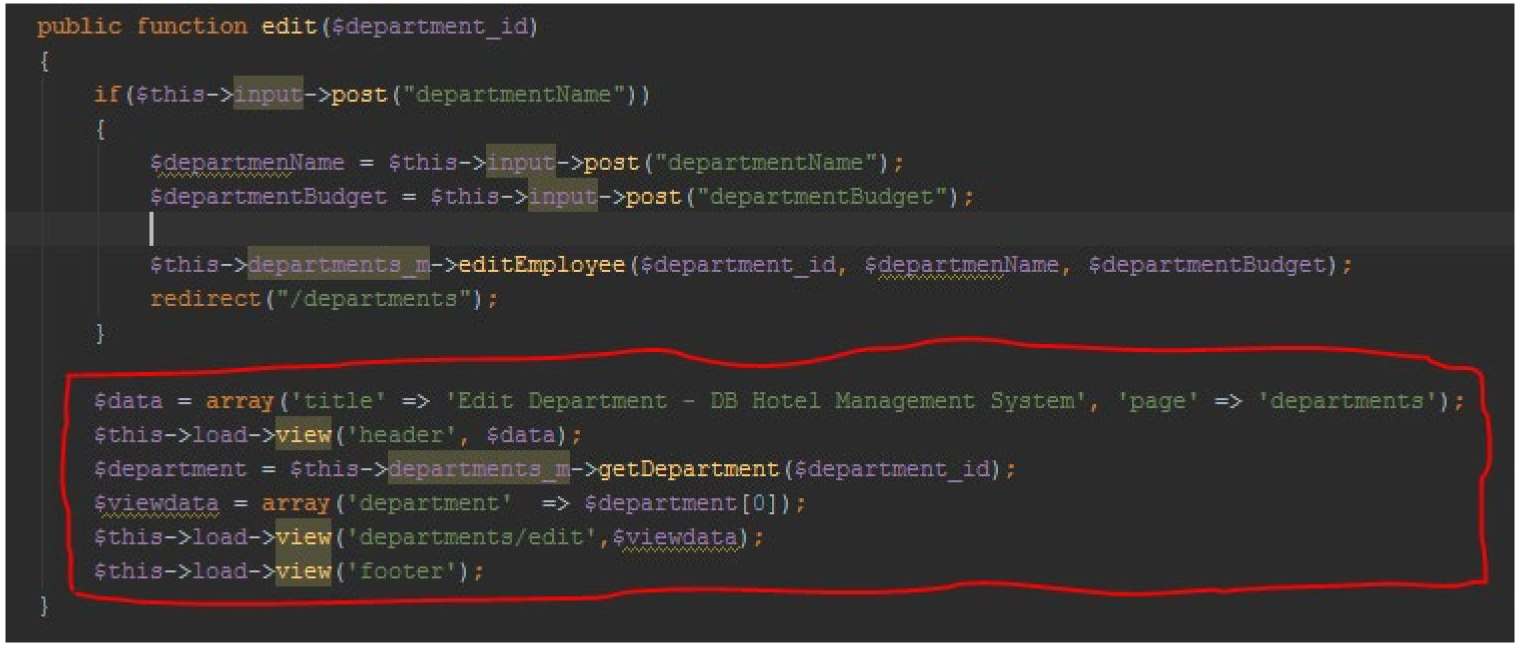
- Code reused by copying source statements leads to significant maintenance problems.
  - Duplicate methods in subclasses
  - Duplicate expressions in same class
  - Duplicate expressions in different classes



```
def to_time(value)
  if value.respond_to?(:to_time)
    value.to_time
  else
    Coercion::String.to_time(to_string(value))
  end
end

def to_datetime(value)
  if value.respond_to?(:to_datetime)
    value.to_datetime
  else
    Coercion::String.to_datetime(to_string(value))
  end
end
```



```
public function add()
{
    if($this->input->post("departmentName"))
    {
        $departmentName = $this->input->post("departmentName");
        $departmentBudget = $this->input->post("departmentBudget");

        $this->departments_m->addDepartment($departmentName, $departmentBudget);
        redirect("/departments");
    }

    $data = array('title' => 'Add Department - DB Hotel Management System', 'page' => 'departments');
    $this->load->view('header', $data);
    $departments = $this->departments_m->get_departments();
    $viewdata = array('departments' => $departments);
    $this->load->view('departments/add',$viewdata);
    $this->load->view('footer');
}
```



```
public function edit($department_id)
{
    if($this->input->post("departmentName"))
    {
        $departmenName = $this->input->post("departmentName");
        $departmentBudget = $this->input->post("departmentBudget");

        $this->departments_m->editEmployee($department_id, $departmenName, $departmentBudget);
        redirect("/departments");
    }

    $data = array('title' => 'Edit Department - DB Hotel Management System', 'page' => 'departments');
    $this->load->view('header', $data);
    $department = $this->departments_m->getDepartment($department_id);
    $viewdata = array('department'  => $department[0]);
    $this->load->view('departments/edit',$viewdata);
    $this->load->view('footer');
}
```

Large method

- A method that does more than one thing
    - Many things, sometimes unconnected things
- Problems
    - Could indicates low levels of abstraction, low level of class design, reduced re-usability
    - Harder to test, poor readability

Long Parameter List

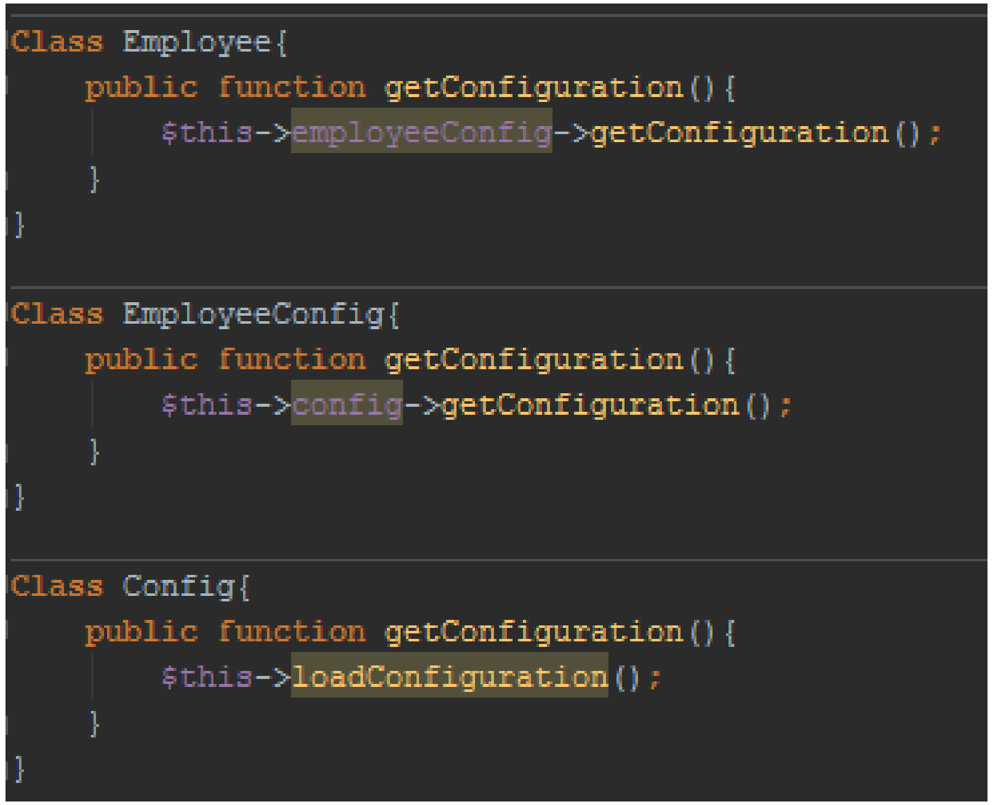
- Introduce parameter object
- Only worthwhile if there are several methods with same parameter list, and they call each other

```
user= userManager.create(USER_NAME,group, USER_NAME,
“test”, Language, false, false, new Date(), “blah”, “new Date())
```

Message Chain

- Long list of method calls:
  - customer.getAddress().getState()
  - window.getBoundingbox().getOrigin().getX()

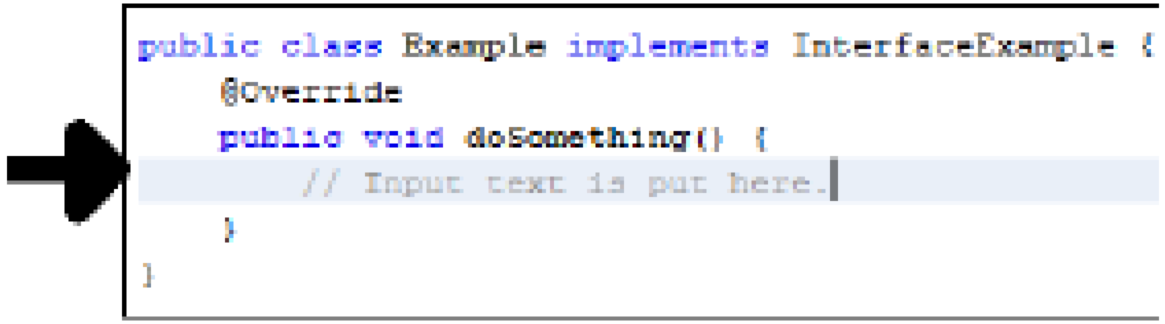
Message Chain



```
Class Employee{
    public function getConfiguration(){
        $this->employeeConfig->getConfiguration();
    }
}

Class EmployeeConfig{
    public function getConfiguration(){
        $this->config->getConfiguration();
    }
}

Class Config{
    public function getConfiguration(){
        $this->loadConfiguration();
    }
}
```

Data Class/Lazy Class

- Class has no methods except for getter and setters



```
public class Example implements InterfaceExample {
    @Override
    public void doSomething() {
        // Input text is put here.
    }
}
```

Switch statements

- Switch statements are very rare in properly designed object-oriented code
  - switch statement is a simple and easily detected “bad smell”
  - Of course, not all uses of switch are bad
  - A switch statement should *not* be used to distinguish between various kinds of object

Example 1, continued

- class Animal {

```
    final int MAMMAL = 0, BIRD = 1, REPTILE = 2;
    int myKind;  // set in constructor
    ...
    String getSkin() {
        switch (myKind) {
            case MAMMAL: return "hair";
            case BIRD: return "feathers";
            case REPTILE: return "scales";
            default: return “skin";
        }
    }
}
```

Example 1, improved

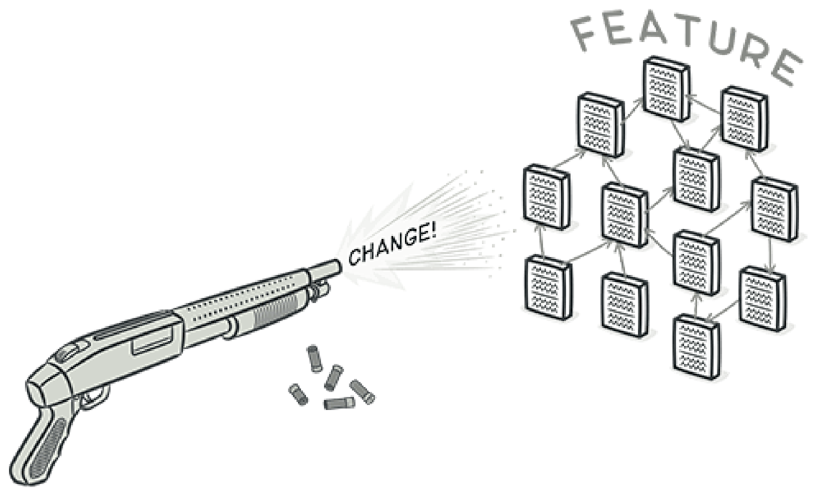
```
 class Animal {
      String getSkin() { return “skin"; }
}
class Mammal extends Animal {
      String getSkin() { return "hair"; }
}
class Bird extends Animal {
      String getSkin() { return "feathers"; }
}
class Reptile extends Animal {
      String getSkin() { return "scales"; }
}
```

Dead Code/ Old Baggage

- Description
  - System contains many classes whose purpose is not known
    - Lava Flow, **Dead Code**
  - Much of the code is left over from previous ideas and no longer has a purpose
    - was once fluid and useful, now is solid lava that you are afraid to remove
- Consequences
  - difficult to maintain, just gets worse

Shotgun Surgery

- This happens when, you want to make some kind of change, your are forced to make a lot of changes to a lot of different classes
- And when changes are all over the place, they are hard to find, and it's easy to miss an important change



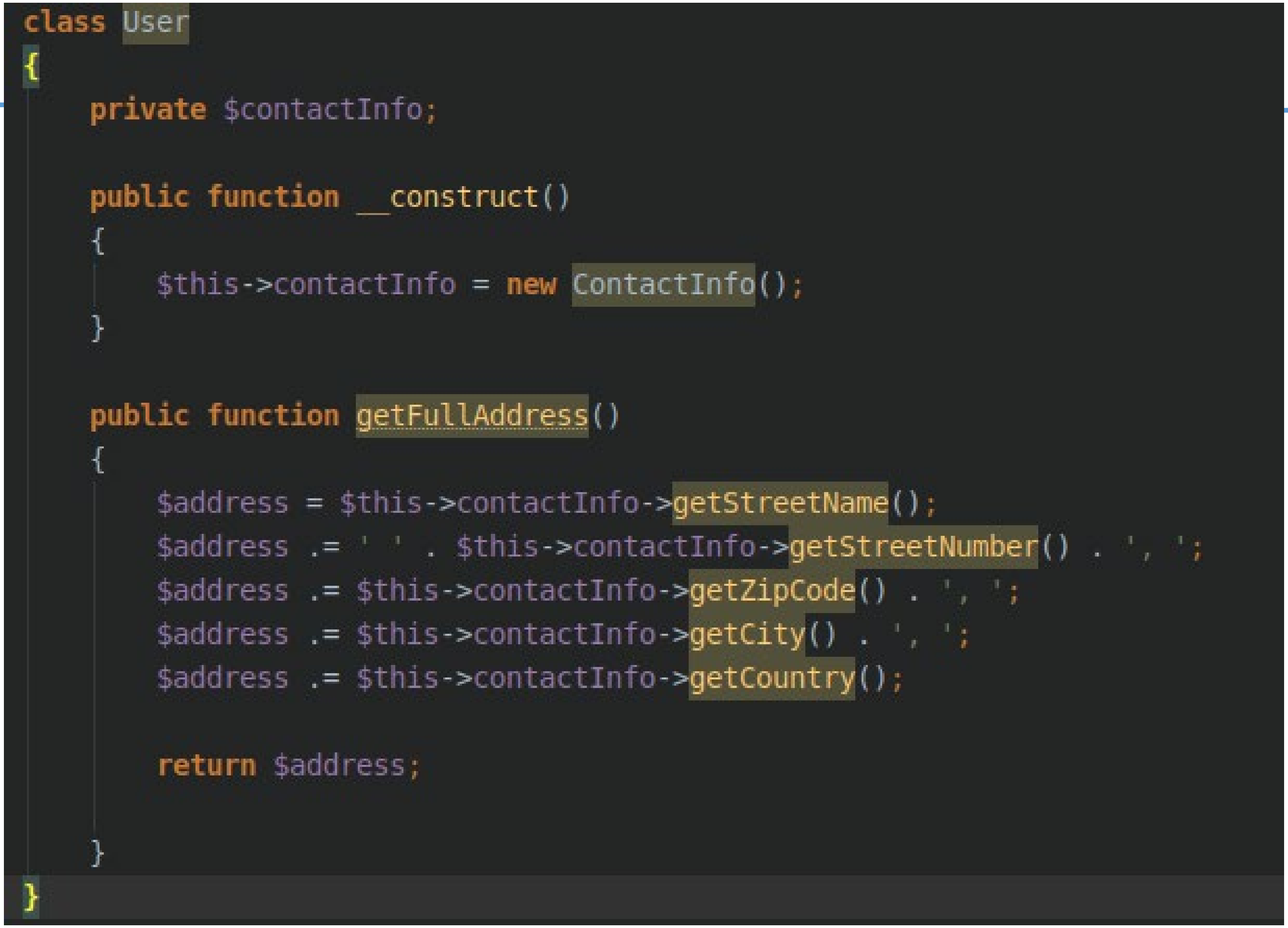
Functional Decomposition

- Description
  - Classes with names ‘’functions”
  - All class attributes are private and are used only inside the class
  - Classes with a single action similar to a procedural function.

- Consequences
  - No O/O benefits such as inheritance and polymorphism
    - Expensive to maintain
    - Complexity of testing software
    - Complexity of reuse of the code

Feature envy

- When a method seems more interesting in a class, other than the one in actually it is
- Example : a method that invokes half a dozen getting methods on another object to calculate some value.



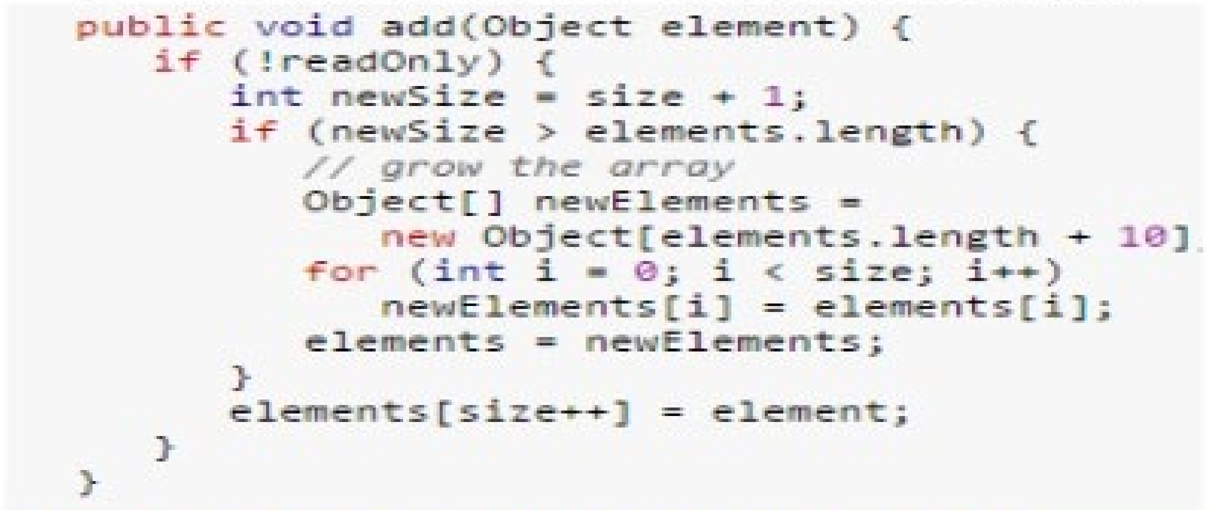
```php
class User
{
    private $contactInfo;

    public function __construct()
    {
        $this->contactInfo = new ContactInfo();
    }

    public function getFullAddress()
    {
        $address = $this->contactInfo->getStreetName();
        $address .= ' ' . $this->contactInfo->getStreetNumber() . ', ';
        $address .= $this->contactInfo->getZipCode() . ', ';
        $address .= $this->contactInfo->getCity() . ', ';
        $address .= $this->contactInfo->getCountry();

        return $address;

    }
}
```

Spaghetti code

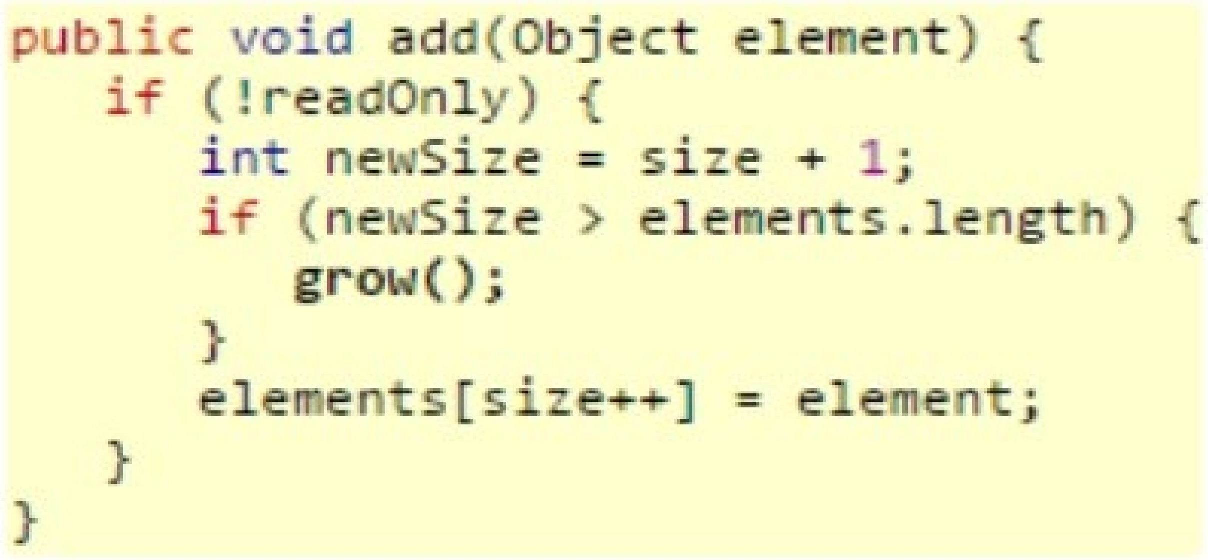
- Description
  - System hard to debug, modify
  - Bunch of code similar in structure to a bowl of spaghetti.
    - Bad coding practices
- Consequences
  - Low readability
  - Impossible to understand how it exactly works

Comments?

- The purpose of comments should be only “why you are doing something (to help future modifiers_ rather than “what code is doing”
- Whenever possible make your code express the intent of the comment and remove the comment.
- Comments are to provide intent that is not expressible in code
- Any comment that duplicates what the code says should be deleted

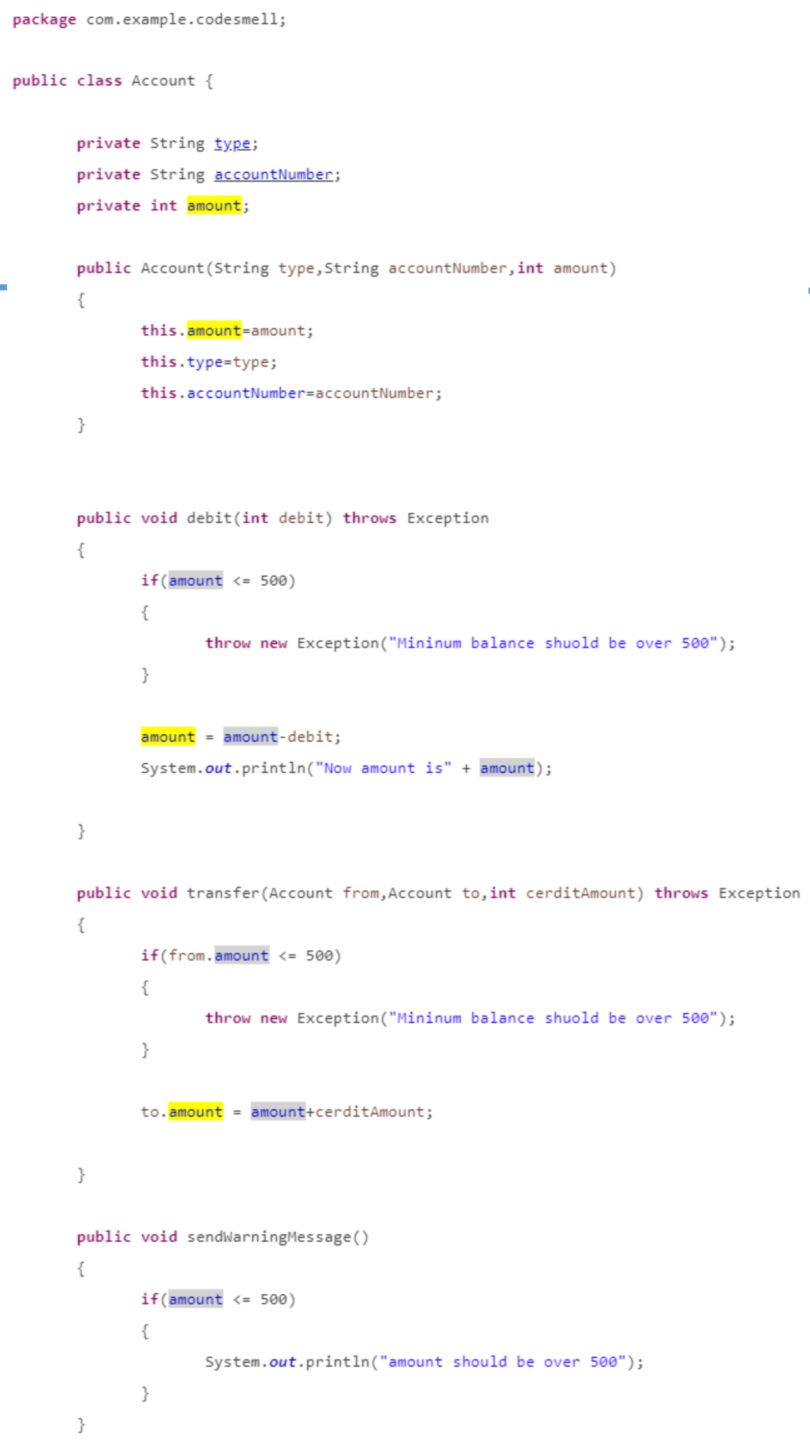


```java
public void add(Object element) {
    if (!readOnly) {
        int newSize = size + 1;
        if (newSize > elements.length) {
            // grow the array
            Object[] newElements =
                new Object[elements.length + 10]
            for (int i = 0; i < size; i++)
                newElements[i] = elements[i];
            elements = newElements;
        }
        elements[size++] = element;
    }
}
```



```java
public void add(Object element) {
    if (!readOnly) {
        int newSize = size + 1;
        if (newSize > elements.length) {
            grow();
        }
        elements[size++] = element;
    }
}
```

- Code smell example:



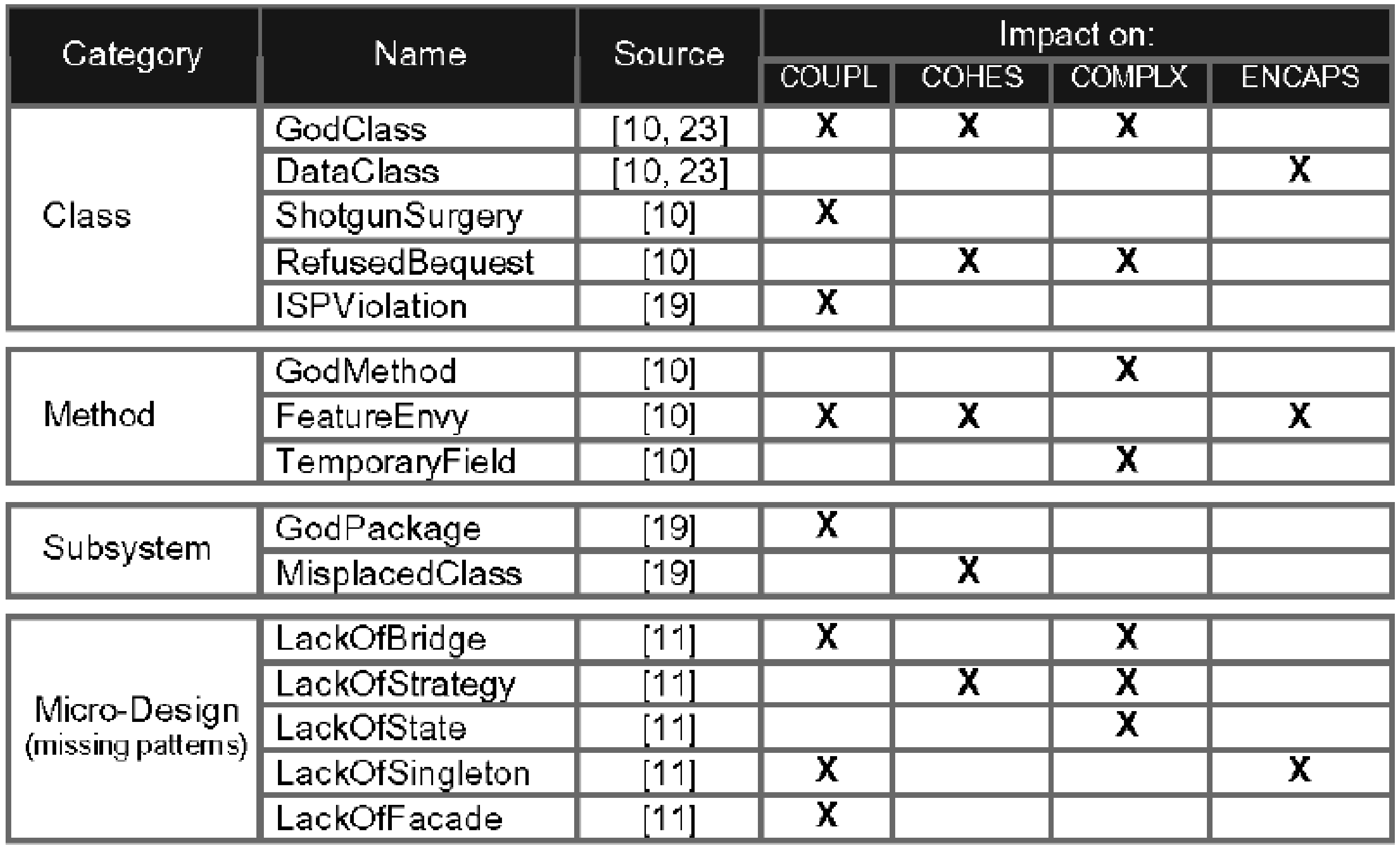
```java
package com.example.codesmell;

public class Account {

	private String type;
	private String accountNumber;
	private int amount;

	public Account(String type,String accountNumber,int amount)
	{
		this.amount=amount;
		this.type=type;
		this.accountNumber=accountNumber;
	}


	public void debit(int debit) throws Exception
	{
		if(amount <= 500)
		{
			throw new Exception("Mininum balance shuold be over 500");
		}

		amount = amount-debit;
		System.out.println("Now amount is" + amount);

	}

	public void transfer(Account from,Account to,int cerditAmount) throws Exception
	{
		if(from.amount <= 500)
		{
			throw new Exception("Mininum balance shuold be over 500");
		}

		to.amount = amount+cerditAmount;

	}

	public void sendWarningMessage()
	{
		if(amount <= 500)
		{
			System.out.println("amount should be over 500");
		}
	}
```

Class Activity

See additional file added to d2l

Some existing approaches for code smells detection

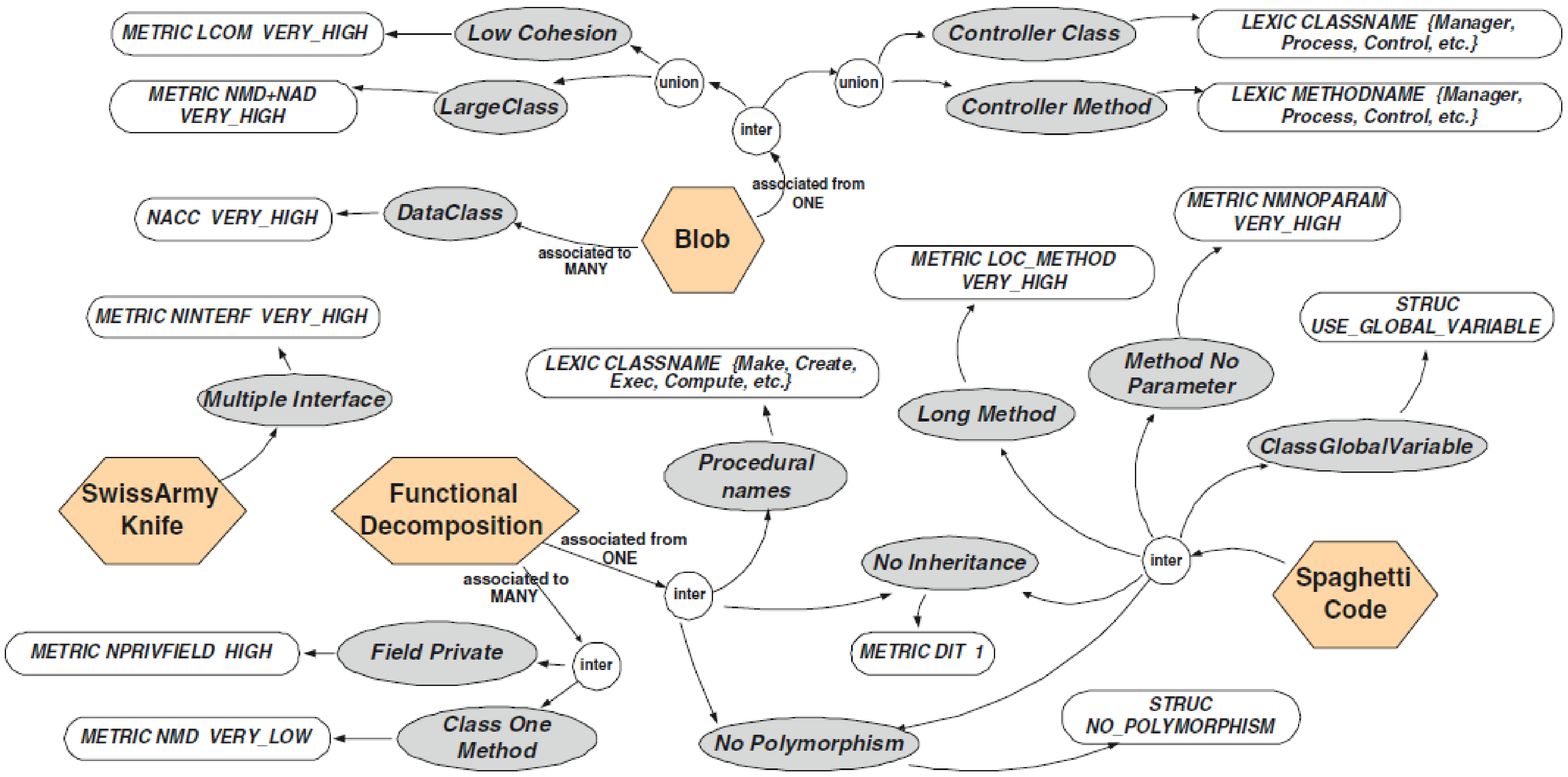
Defects summary by Marienscu (2003)



Category	Name	Source	Impact on:			
			COUPL	COHES	COMPLX	ENCAPS
Class	GodClass	[10, 23]	X	X	X	
	DataClass	[10, 23]				X
	ShotgunSurgery	[10]	X			
	RefusedBequest	[10]		X	X	
	ISPViolation	[19]	X			
Method	GodMethod	[10]			X	
	FeatureEnvy	[10]	X	X		X
	TemporaryField	[10]			X	
Subsystem	GodPackage	[19]	X			
	MisplacedClass	[19]		X		
Micro-Design (missing patterns)	LackOfBridge	[11]	X		X	
	LackOfStrategy	[11]		X	X	
	LackOfState	[11]			X	
	LackOfSingleton	[11]	X			X
	LackOfFacade	[11]	X			

Overview of design flaws

Moha's smells classification (2009)



Moha's smells classification (2009)

Software Quality Metrics

- Some examples…
  - Number of methods per class
  - Number of classes
  - Number of lines of code
  - Number of packages
  - Cohesion
  - Coupling
  - …

Detection strategy

- A detection strategy is a *metrics-based predicate* to identify *candidate* software artifacts that *conform to* (or violate) a particular *design rule.*

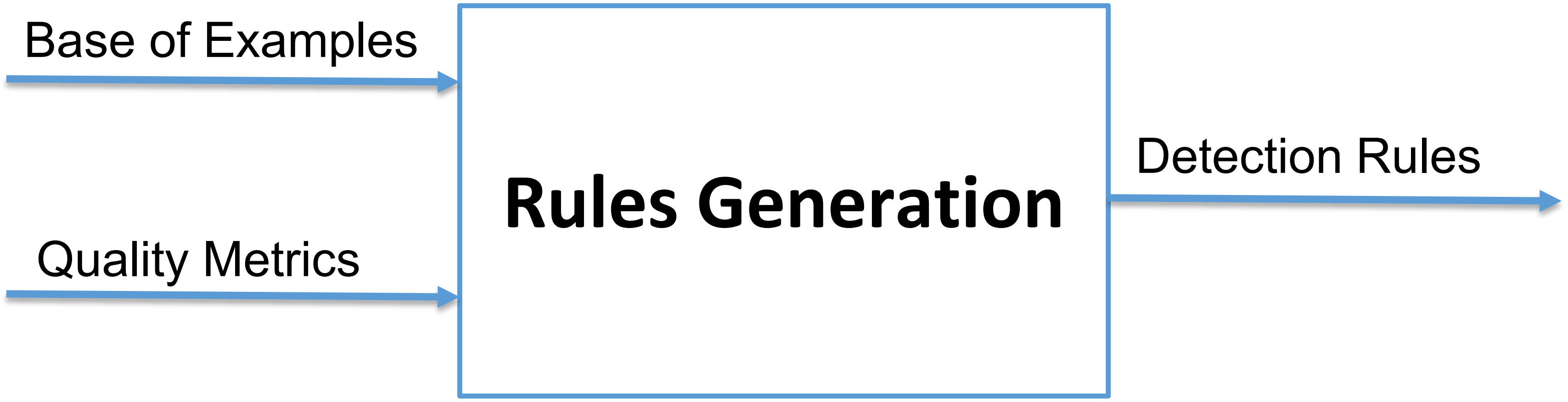
Example of research study: code smells detection

Example of code smells detection

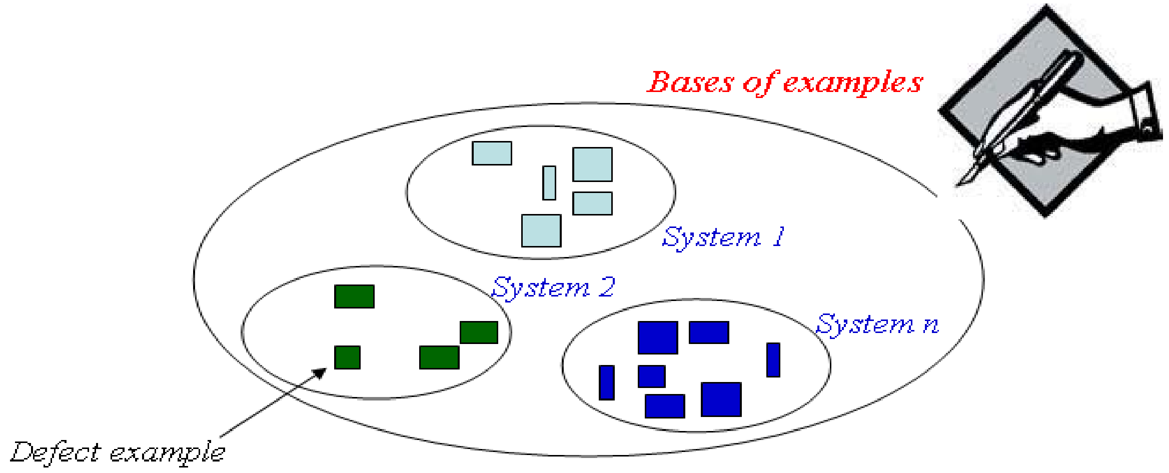
Kessentini et al. : Design Defects Detection and Correction by Example

http://www-ens.iro.umontreal.ca/~kessentw/PDF/ICPC.pdf

Example of research study: code smells detection



Example of research study: code smells detection



- Defect example :
  - A class that has at least one design defect (blob, spaghetti code, functional decomposition)

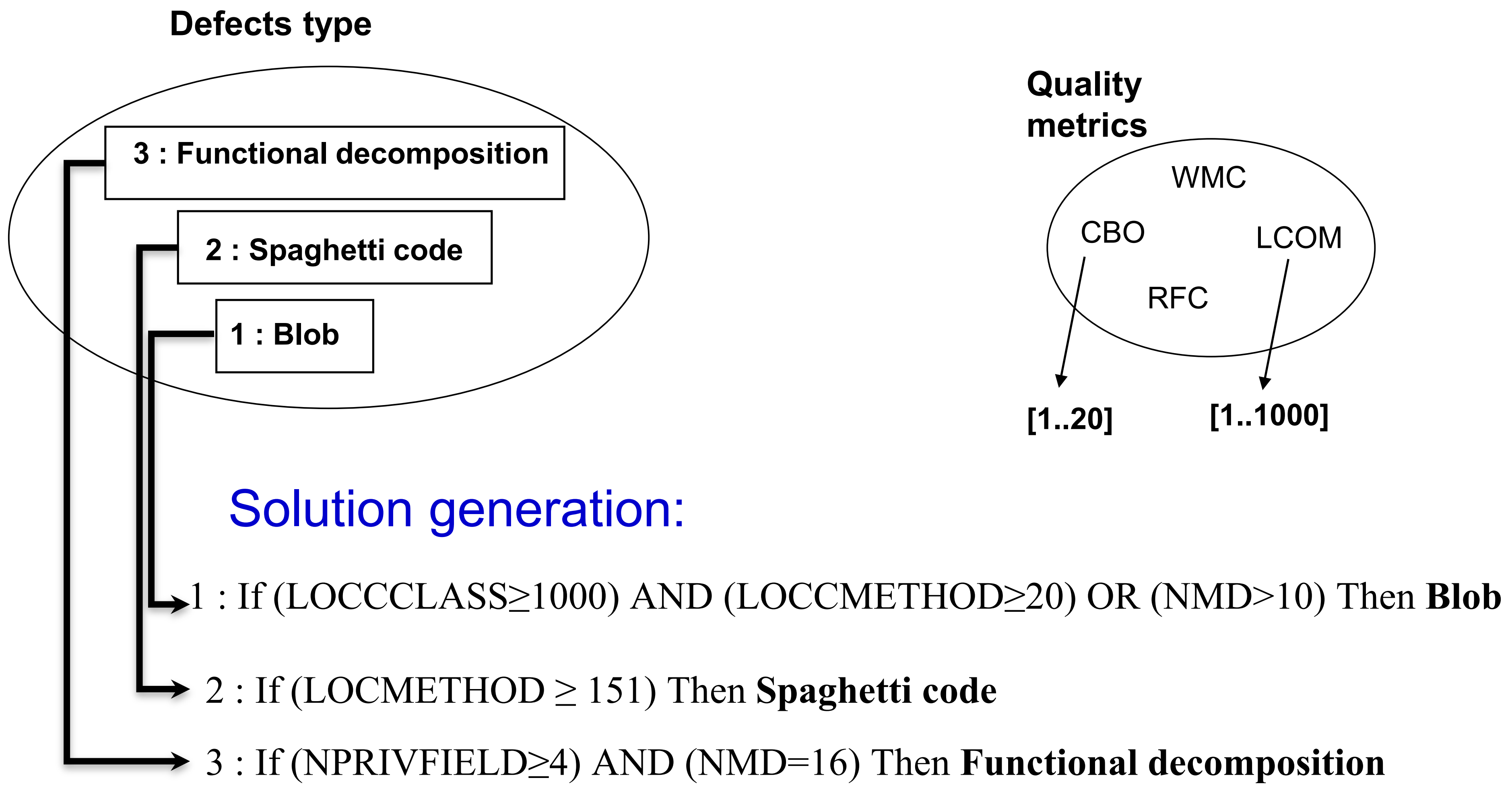
Example of research study: code smells detection

- Software metrics
  - measuring a property of a software code

- Examples

  - Number Of Methods (NOM).
  - Number Of Private Fields(NPRIVFIELD).
  - Lines Of Code in a Class (LOCCLASS).
  - …

Example of research study: code smells detection



Solution generation:

1 : If (LOCCCLASS≥1000) AND (LOCCMETHOD≥20) OR (NMD>10) Then **Blob**

2 : If (LOCMETHOD ≥ 151) Then **Spaghetti code**

3 : If (NPRIVFIELD≥4) AND (NMD=16) Then **Functional decomposition**

Example of research study: code smells detection

Evaluating a Set of Rules

$$f_{norm} = \frac{\frac{\sum_{i=1}^{p} a_i}{t} + \frac{\sum_{i=1}^{p} a_i}{p}}{2}$$

$p :$ number of detected classes

$t :$ number of defects in the base of examples

$$a_i = \begin{cases} 1, & \text{if } c_i \text{ is in the base of examples with the same defect type} \\ 0, & \text{else.} \end{cases}$$

Example of research study: code smells detection

Evaluating a Set of Rules

Defects in the base of examples

Class	Blob	Functional decomposition	Spaghetti code
Student		X	
Person		X	
University		X	
Course	X		
Classroom			X
Administration	X		

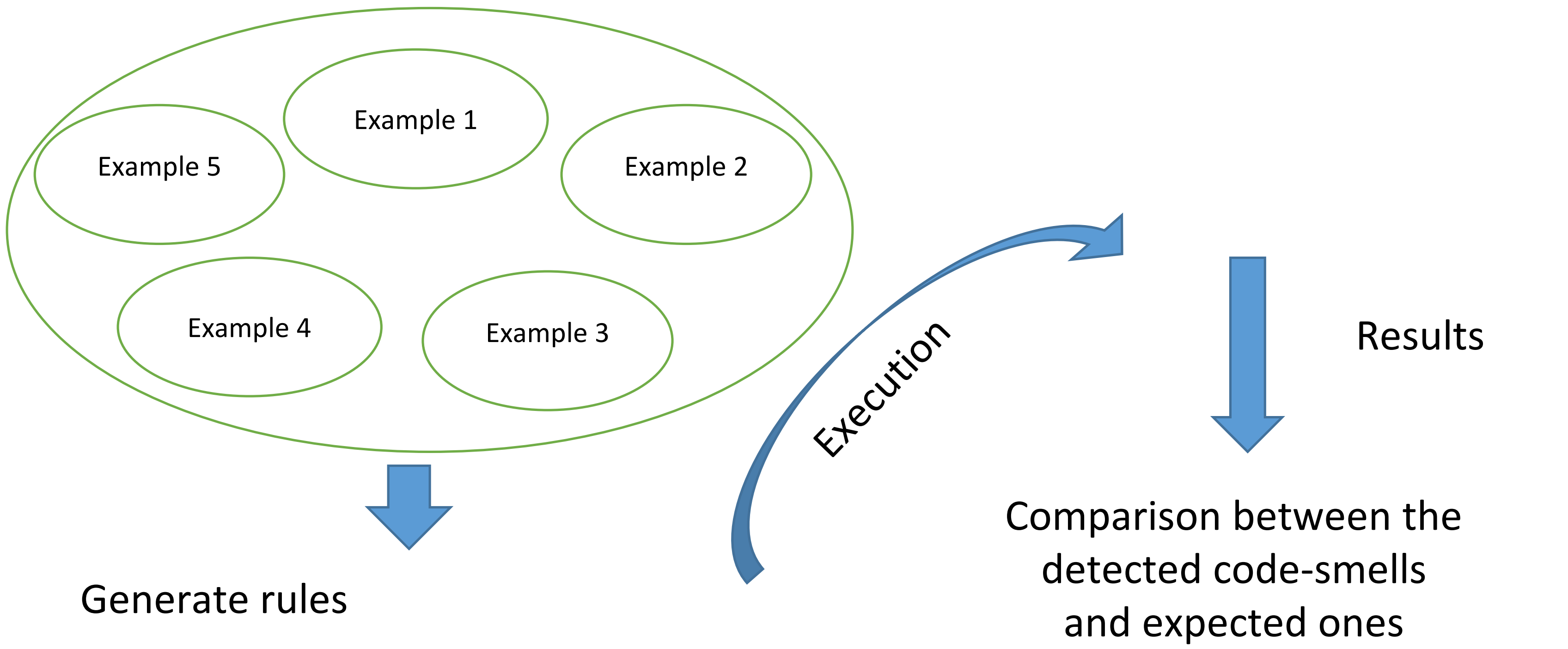
Detection results

Class	Blob	Functional decomposition	Spaghetti code
Person		X	
Classroom	X		
Professor		X	

$$f_{norm} = \frac{\frac{1}{3} + \frac{1}{6}}{2} = 0.25$$

Example of research study: code smells detection

- N-fold cross-validation



Example of research study: code smells detection

- N-fold cross-validation

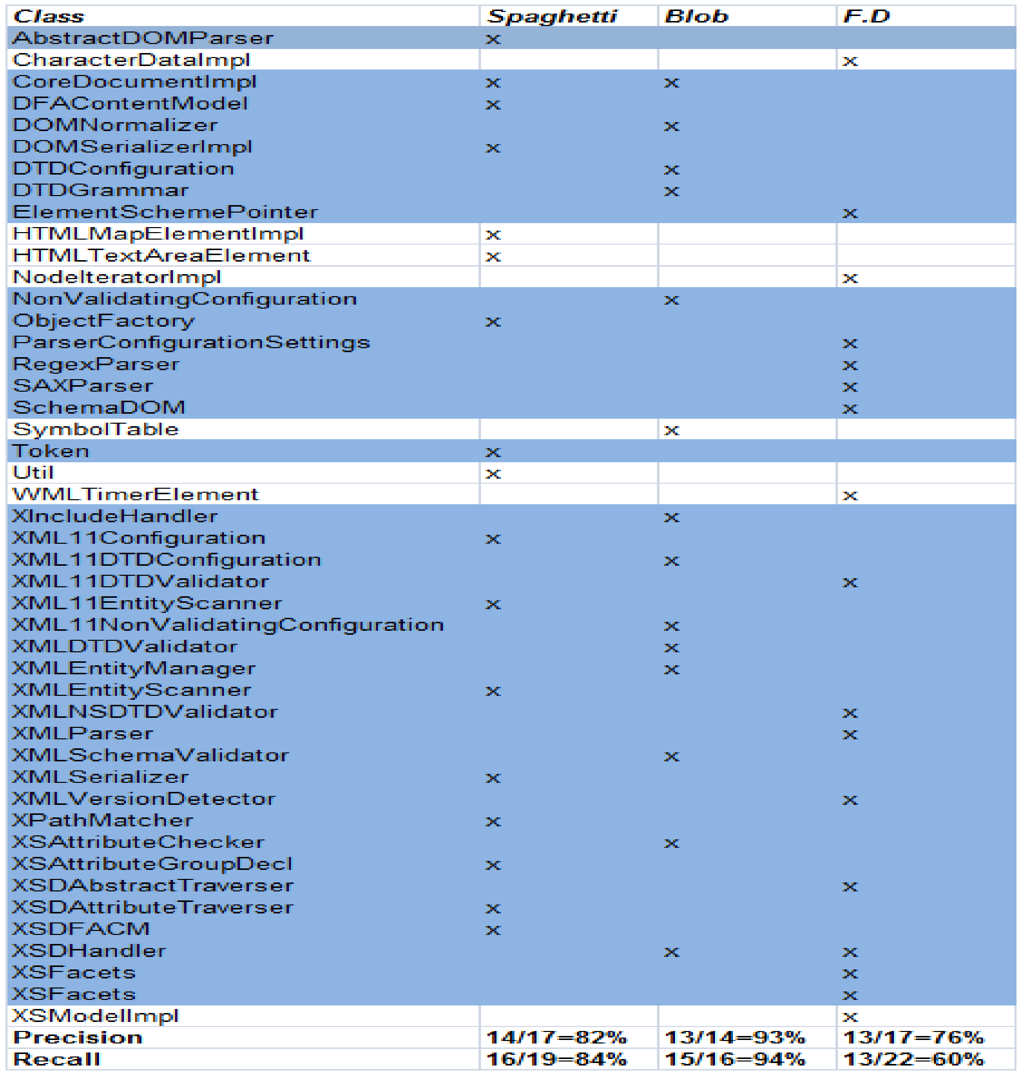
- Measures :

$$\text{Precision} = \frac{\text{correctly detected code - smells}}{\text{the set of all detected code - smells}}$$

$$\text{Recall} = \frac{\text{correctly detected code - smells}}{\text{The set of all manually identified code - smells}}$$

Example of research study: code smells detection

Results



Class	*Spaghetti*	*Blob*	*F.D*
AbstractDOMParser	x		
CharacterDataImpl			x
CoreDocumentImpl	x	x	
DFAContentModel	x		
DOMNormalizer		x	
DOMSerializerImpl	x		
DTDConfiguration		x	
DTDGrammar		x	
ElementSchemePointer			x
HTMLMapElementImpl	x		
HTMLTextAreaElement	x		
NodeIteratorImpl			x
NonValidatingConfiguration		x	
ObjectFactory	x		
ParserConfigurationSettings			x
RegexParser			x
SAXParser			x
SchemaDOM			x
SymbolTable		x	
Token	x		
Util	x		
WMLTimerElement			x
XIncludeHandler		x	
XML11Configuration	x		
XML11DTDConfiguration		x	
XML11DTDValidator			x
XML11EntityScanner	x		
XML11NonValidatingConfiguration		x	
XMLDTDValidator		x	
XMLEntityManager		x	
XMLEntityScanner	x		
XMLNSDTDValidator			x
XMLParser			x
XMLSchemaValidator		x	
XMLSerializer	x		
XMLVersionDetector			x
XPathMatcher	x		
XSAttributeChecker		x	
XSAttributeGroupDecl	x		
XSDAbstractTraverser			x
XSDAttributeTraverser	x		
XSDFACM	x		
XSDHandler		x	x
XSFacets			x
XSFacets			x
XSModelImpl			x
Precision	**14/17=82%**	**13/14=93%**	**13/17=76%**
Recall	**16/19=84%**	**15/16=94%**	**13/22=60%**