

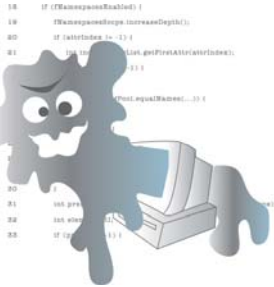
A Systematic Method for the Detection and Correction of Design Defects

Naouel Moha

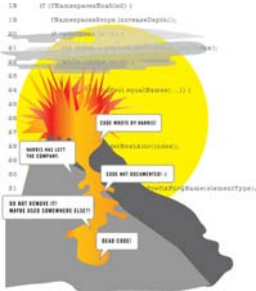
Ptidej Team, GEODES, University of Montreal, Canada
LIFL, INRIA project Jacquard, University of Lille, France

mohanaou@iro.umontreal.ca

<http://ptidej.iro.umontreal.ca/Members/mohanaou>



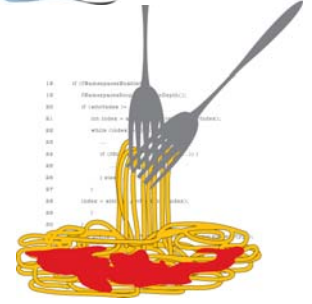
(a) Blob



(b) Lava Flow



(c) Poltergeist



(d) Spaghetti Code

Design Defects (DD) are poor design choices that lessen the quality of object-oriented architectures and impede their evolution and their maintenance.

Motivation

- A good software architecture without DD is easier to understand, change, and thus maintain.
- However, the detection and correction of DD
 - are difficult and remain manual activities because of the lack of precise specifications and tools,
 - are highly time-, resource-consuming, and error-prone activities.

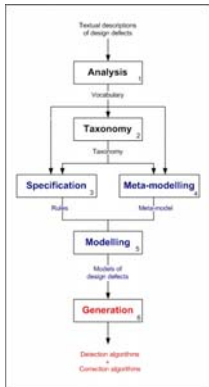
Goal

Our goal is to provide a **systematic method** to specify DD consistently and precisely and to **generate detection and correction algorithms** from their specifications semi-automatically.

Approach

We define a **systematic method** based on a **meta-model** and a **language** to describe DD precisely to **reify, detect, and correct** these defects.

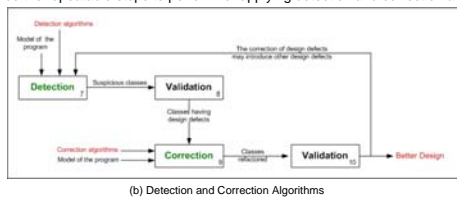
Our method is divided in **10 steps**. The first six steps describe the generic steps for the generation of detection and correction algorithms:



(a) Generation of Detection and Correction Algorithms

- Analysis.** We extract and unify all the key concepts from the textual descriptions of DD. Key concepts form a consistent vocabulary of reusable concepts to describe, detect and correct DD.
- Taxonomy.** We define DD using their vocabulary and classify them to limit misinterpretations and allow the consistent specification of DD.
- Specification.** We specify the detection and correction of DD as sets of **rules** using the vocabulary and the taxonomy.
- Meta-modelling.** We design a **meta-model** to instantiate rules for the detection and correction of DD.
- Modelling.** We instantiate the rules using the constituents of the meta-model. These models are the basis for the automated generation of detection and correction algorithms.
- Generation.** We generate algorithms to detect and correct the DD by visiting the constituents of the models.

The last four steps describe the repeatable steps to perform for applying detection and correction algorithms on programs:



(b) Detection and Correction Algorithms

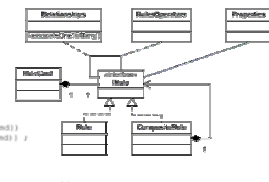
- Detection.** We build a model of the program and apply detection algorithms on this model. The detection algorithms return the list of suspicious classes.
- Validation.** We validate the results of the detection algorithms by analysing the suspicious classes with the complete source code of the program. We obtain the list of classes having real design defects.
- Correction.** We apply the correction algorithms on the model of the program and perform the refactorings suggested by the correction algorithms on the source code.
- Validation.** We perform tests to ensure that the behavior of the program has not changed by applying the refactorings.

Implementation

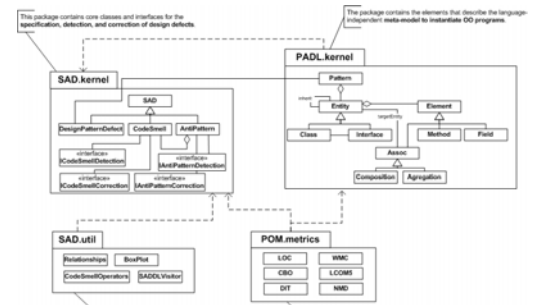
```

RULE_CARD : Blob {
  ROLE : Blob (ASSOC: associated FROM: MainClass ONE TO: DataClass MANY) ;
  ROLE : MainClass (UNION LargeClassLowCohesion ControllerClass) ;
  ROLE : LargeClassLowCohesion (UNION LargeClass LowCohesion) ;
  ROLE : LargeClass (ONESTRUC: NND * NAD, VERY_HIGH) ;
  ROLE : LowCohesion (ONESTRUC: LCOM, VERY_HIGH) ;
  ROLE : ControllerClass (UNION (ONESTRUC: METROGRAM, (Process, Control, Ctrl, Command, Cmd))
    (ONESTRUC: CLASSNAME, (Process, Control, Ctrl, Command, Cmd)) ;
  ROLE : DataClass ( (STRUCT: METHOD, Accessor) ) ;
}
    
```

(c) Set of rules for the Blob design defect



(d) Meta-model to instantiate rules



(e) SAD architecture of the framework that supports the detection and the correction of Design Defects

Preliminary Results

Steps 1-8 : We are able to specify and detect DD in several **open-source programs**, and thus show the usefulness of our systematic method.

Blob						
	ArgoUML	Azureus	GanttProject	PMD	QuickUML	Total
	113 KLOC 1230 classes	192 KLOC 1449 classes	192 KLOC 1449 classes	42 KLOC 423 classes	9 KLOC 142 classes	
Effectifs	91	143	19	15	3	271
Precisions	70/91 = 76.9%	82/143 = 57.3%	10/19 = 52.6%	3/15 = 20%	1/3 = 33.3%	166/271 = 61.5%
Time	40.5s	2m	5.6s	31.6s	2.1s	3m 19.8s

Spaghetti Code						
	ArgoUML	Azureus	GanttProject	PMD	QuickUML	Total
Effectifs	26	35	8	11	1	81
Precisions	21/26 = 80.7%	29/35 = 82.8%	6/8 = 75%	6/11 = 54.5%	0/1 = 0%	62/81 = 76.6%
Time	39s	1m55s	5.4s	31s	1.9s	3m 12.3s

Table. Results of the detection of DD. *Effectifs* correspond to the numbers of suspicious classes, *precisions* correspond to the numbers and percentages of true positives, and *Time* corresponds to the computation time in minutes and seconds.

Conclusion

- Our **systematic method** offers a greater flexibility to specify, detect, and -ultimately- correct DD semi-automatically.
- Our method ensures via rules the **traceability** between the specifications of DD and detected suspicious classes.
- We proposed a **meta-model** to instantiate DD and their related rules for their detection and correction.
- We proposed a **framework** to generate the detection and correction algorithms.
- We validated our method on open-source OO programs and showed that the **detection algorithms are reasonably efficient and precise**.

Future Work

- Specification of rules for the **correction of DD** and generation of correction algorithms.
- **More defects and more case studies** including commercial programs.
- **Dynamic analyses** to specify and to detect behavioural DD.

References

- [1] N. Moha, D.L. Huynh, Y.G. Guéhéneuc: *Une taxonomie et un métamodèle pour la détection des défauts de conception*, LMO'06.
- [2] N. Moha, Y.G. Guéhéneuc: *On the Automatic Detection and Correction of Software Architectural Defects in Object-Oriented Designs*, WOOR'05.
- [3] W.J. Brown et al.: *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*, John Wiley and Sons, 1st edition, March 1998.
- [4] M. Fowler: *Refactoring Improving the Design of Existing Code*, Addison-Wesley, 1st edition, June 1999.
- [5] Tool suite Ptidej: <http://ptidej.iro.umontreal.ca/>