

A Bayesian Approach for the Detection of Code and Design Smells

Foutse Khomh, Stéphane Vaucher,
Yann-Gaël Guéhéneuc, and Houari Sahraoui

QSIC'09

Jeju-do

2009/08/25



Ptidej Team – Pattern-based Quality Evaluation and Enhancement
Department of Computer Engineering and Software Engineering
École Polytechnique de Montréal, Québec, Canada

© Khomh, 2009



Outline

- Motivation
- Context
- Problems and Objectives
- Previous Work
- Our Solution
- Bayesian Belief Networks
- Experiments
- Discussions
- Conclusion



Motivation

- Independent IV&V team reviews of object-oriented software programs
 - Assess the quality of programs
 - Identify potential problems
 - Defects
 - Anomalies
 - **“Bad Smells”**



Context

- Code smells and antipatterns
- **Blob**: also called God class [23], is a class that centralises functionality and has too many responsibilities. Brown et al. [4] characterise its structure as a large controller class that depends on data stored in several surrounding data classes



Problems and Objectives

- **Problem 1:** Sizes of programs can be large
 - Automate the evaluation process
- **Problem 2:** Resources are limited
 - Prioritise manual reviews
- **Problem 3:** Quality assessment is subjective
 - Consider uncertainty in the process

Previous Work

(1/2)

- Webster's book on "antipatterns" [31]
- Riel's 61 heuristics [23]
- Beck's 22 code smells [11]
- Brown et al. 40 antipatterns [4]
- Travassos et al. manual approach [28]
- Marinescu's detection strategies [17]
(also Munro [19])
- Moha et al. rules cards in **DECOR** [18]

Previous Work

(2/2)

- Either no full automation
 - **Problem 1 is not solved**
- Or strict rules with classes participating or not (binary value) to an antipattern
 - **Problem 2 and 3 are not solved**



Our Solution

(1/2)

- Assign a probability that a class is considered an antipattern by a stakeholder
- Guide a manual inspection according to this probability
- Use Bayesian belief network to build a model of an antipattern and assign a probability to all classes

Our Solution

(2/2)

- Problem 1: Sizes of programs can be large
 - **Assign a probability** to all classes of interest automatically
- Problem 2: Resources are limited
 - Prioritise manual reviews using **probabilities**
- Problem 3: Quality assessment is subjective
 - Uncertainty is naturally considered using **Bayesian beliefs networks**

BBN

(1/3)

- A Bayesian Belief Network is a directed acyclic graph with probability distribution
- Graph structure
 - Nodes = random variables
 - Edges = probabilities dependencies
- Each node depends only on its parents
- **Embody the experts' knowledge**

BBN

(2/3)

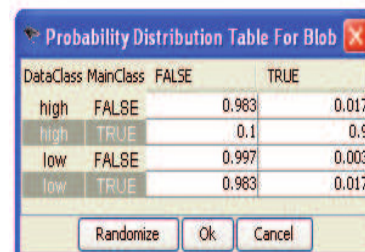
- Classifier
 - $C = \{\text{smell, not smell}\}$
- Input vector describing a class
 - $\langle a_1, \dots, a_n \rangle$
 - $P(A|B) = P(B|A) P(A) / P(B)$

BBN

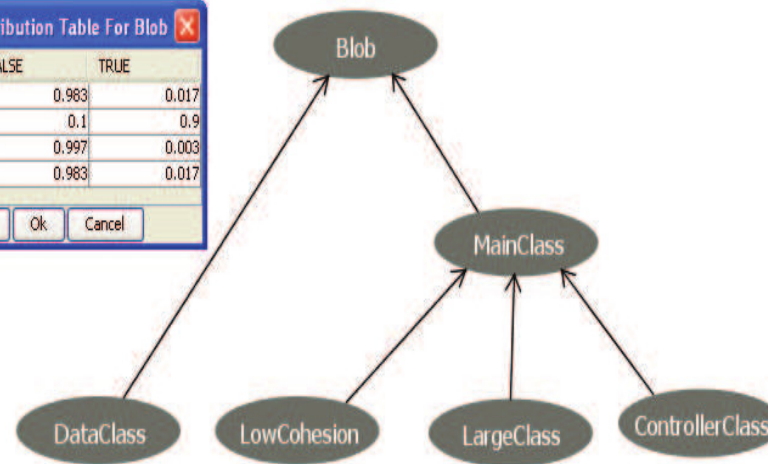
(2/3)

■ Building a BBN

1. Define its structure
2. Assign/learn its probability tables



DataClass	MainClass	FALSE	TRUE
high	FALSE	0.983	0.017
high	TRUE	0.1	0.9
low	FALSE	0.997	0.003
low	TRUE	0.983	0.017





1. Defining the BBN Structure

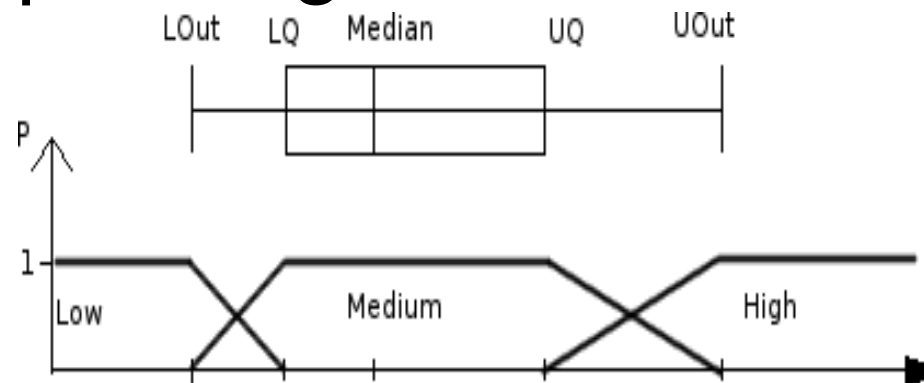
- Based on Moha et al.'s **rule cards**
 - Detection relies on metrics, binary class relations, and linguistic analysis
 - Values are combined using set operators
 - Thresholds are statistically defined (using a box plot)

Rule Card of Blob

```
RULE_CARD : Blob {  
  RULE : Blob { ASSOC : associated FROM : mainClass ONE TO : DataClass MANY }  
  ;  
  RULE : MainClass { UNION LargeClassLowCohesion ControllerClass } ;  
  RULE : LargeClassLowCohesion { UNION LargeClassLowCohesion } ;  
  RULE : LargeClass { ( METRIC : NMD + NAD, VERY_HIGH , 0 ) } ;  
  RULE : LowCohesion { ( METRIC : LCOM5, VERY_HIGH , 20 ) } ;  
  RULE : ControllerClass { UNION  
    ( SEMANTIC : METHODNAME , { Process, Control, Ctrl, Command, Cmd,  
      Proc, UI, Manage, Drive } )  
    ( SEMANTIC : CLASSNAME , { Process, Control, Ctrl, Command, Cmd,  
      Proc, UI, Manage, Drive, System, Subsystem } ) } ;  
  RULE : DataClass { ( STRUCT : METHOD_ACCESSOR , 90 ) } ;  
};
```

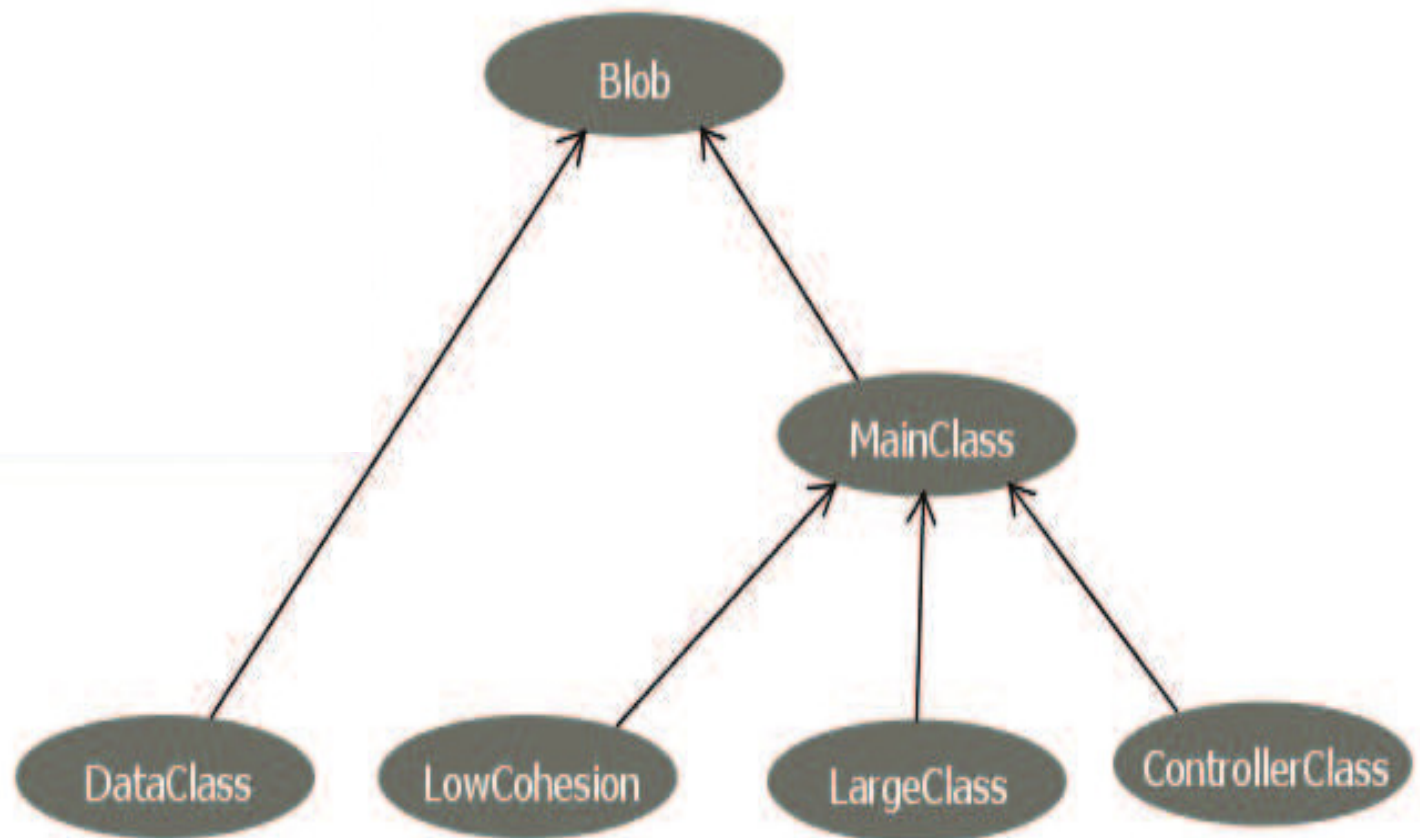
Conversion into BBN

- Set operators (OR, AND) are learned using conditional probabilities
- Hard thresholds and rules are smoothed by interpolating values



- Binary class relations are counted and treated as metrics

BBN Structure for the Blob

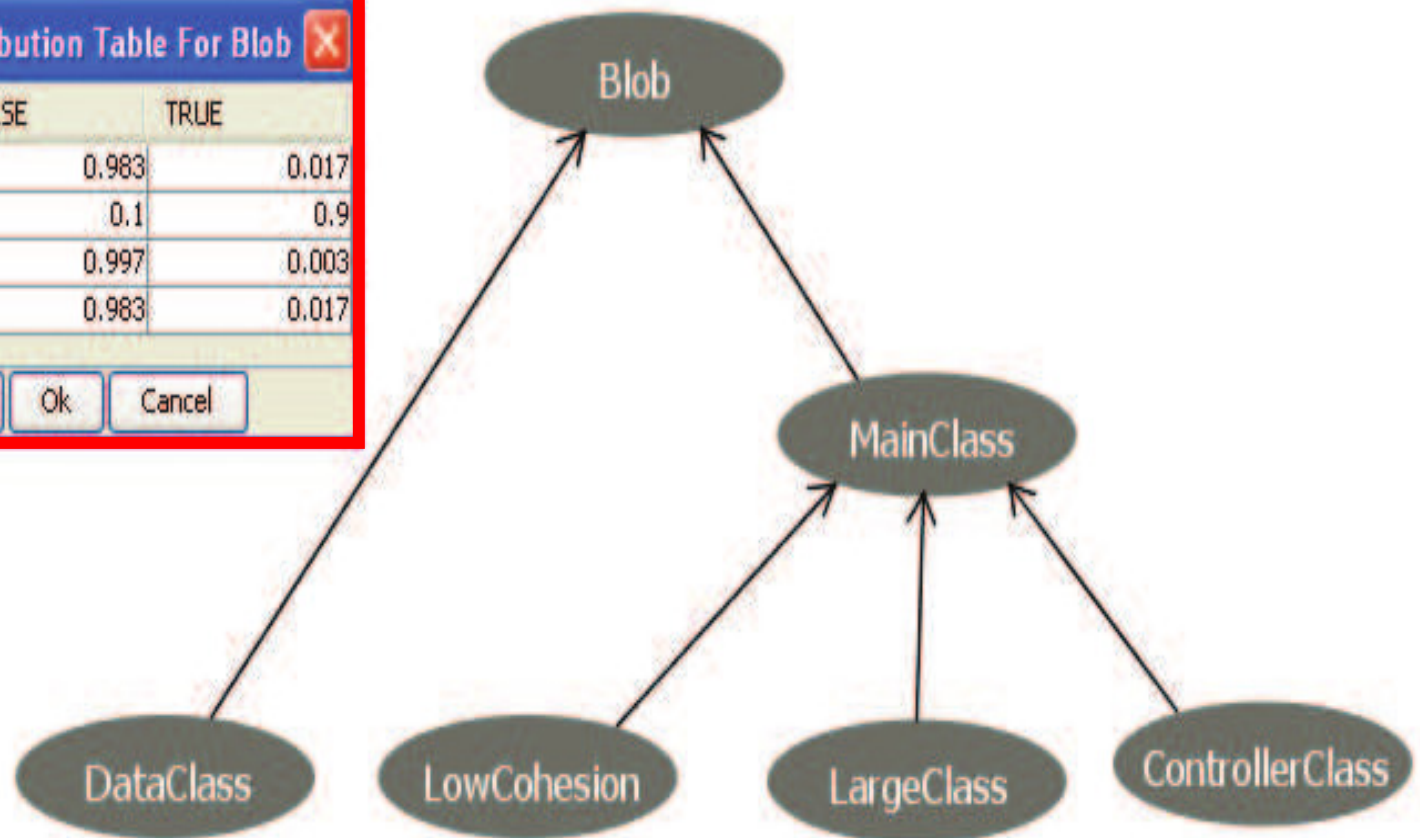


2. Resulting BBN for the Blob

Probability Distribution Table For Blob

DataClass	MainClass	FALSE	TRUE
high	FALSE	0.983	0.017
high	TRUE	0.1	0.9
low	FALSE	0.997	0.003
low	TRUE	0.983	0.017

Randomize Ok Cancel





Experiments

- **RQ1**: To what extent a model built with a BBN based on an existing rule-based model is able detect smells in a program?
- **RQ2**: Is a model built with a BBN better than a state-of-the-art approach, DECOR?



Settings

- Programs

- Xerces v2.7.0

- Gantt Project v1.10.2

- Oracle: Vote of three groups
(of students) on both programs



Settings

- Calibration (probability tables)
 - Local (same-system calibration)
 - External (calibrated on one system, applied to the other)

Local calibration

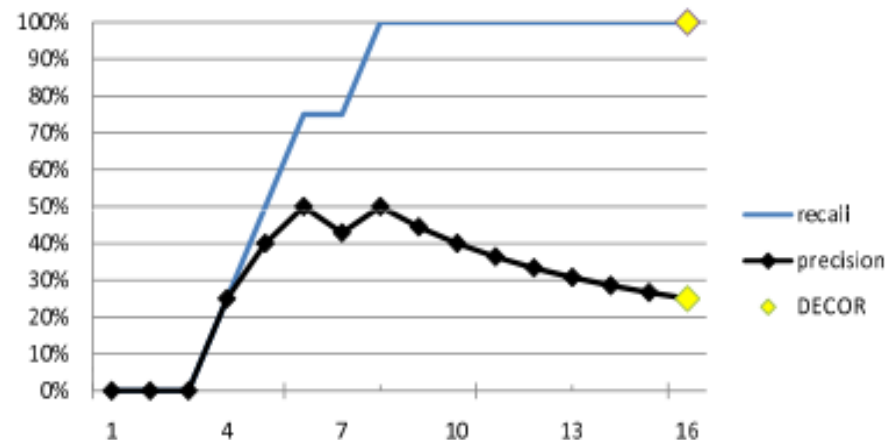
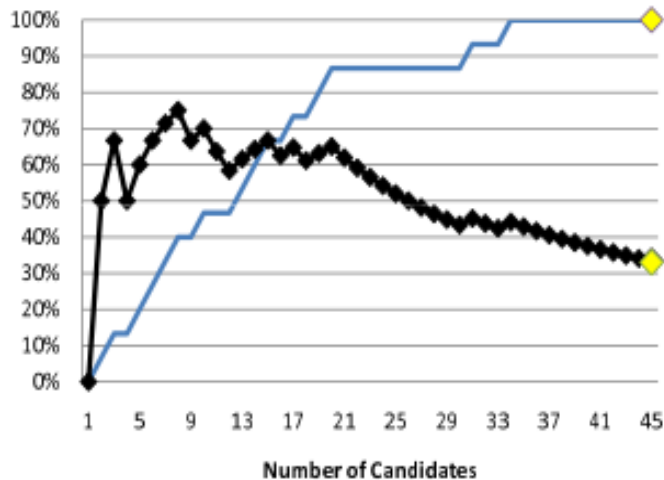
- On Xerces only
- Three-fold cross validation
 - 3 groups with 5 Blobs in each
 - Combination of 2 groups for calibration, tested on the other

	Inspected Classes
Group1	6
Group2	7
Group3	9

Waste of 8% to 44%
of effort (average 32%)

External calibration

- Probabilities learned on one, applied on other
- Precision vs. Recall (per investigation sizes)
- Xerces (left), Gantt (right)





Experiments

- RQ1: To what extent a model built with a BBN based on an existing rule-based model is able detect smells in a program?
- RQ2: Is a model built with a BBN better than a state-of-the-art approach, DECOR?
- **Better than DECOR**

Discussions

(1/2)

- Results of BBN are superior to those of DECOR
- External calibration yields good results even though tweaking is recommended
- A well calibrated model should be able to estimate the number of smells in a program, but our models overestimate by a factor of 2

Discussions

(2/2)

- Used simple techniques to convert rules into probability distributions
- Worked with discrete probabilities, might need to be adapted to continuous cases



Conclusion

- Bayesian models support uncertainty in detection process
- Their performance is better than state-of-the-art rules
- External calibration seems to indicate that this could be used in an industrial context with minimal costs
- Support for automated conversion of rules to support more design smells



Acknowledgements

- Naouel Moha for providing her data and code to compare with DECOR
- Natural Sciences and Engineering Research Council of Canada
- Canada Research Chair in Software Patterns and Patterns of Software

Advertisement

- IEEE Software
- Special issue on Software Evolution
- Deadline: 1st of November, 2009
- Publication: July/August 2010
- Please do not hesitate to contact me! 😊