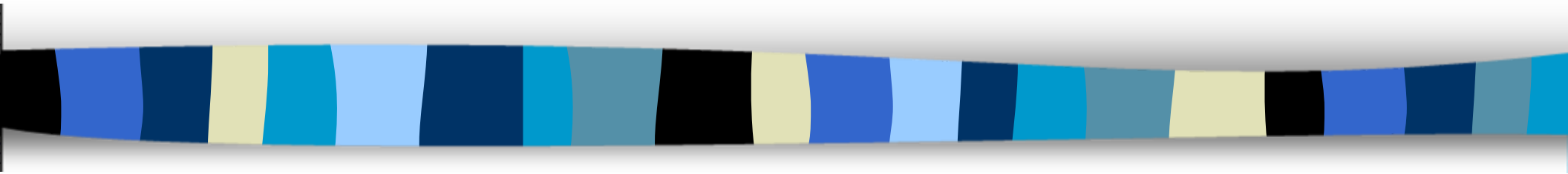
Patterns and Quality of Object-oriented Software Systems



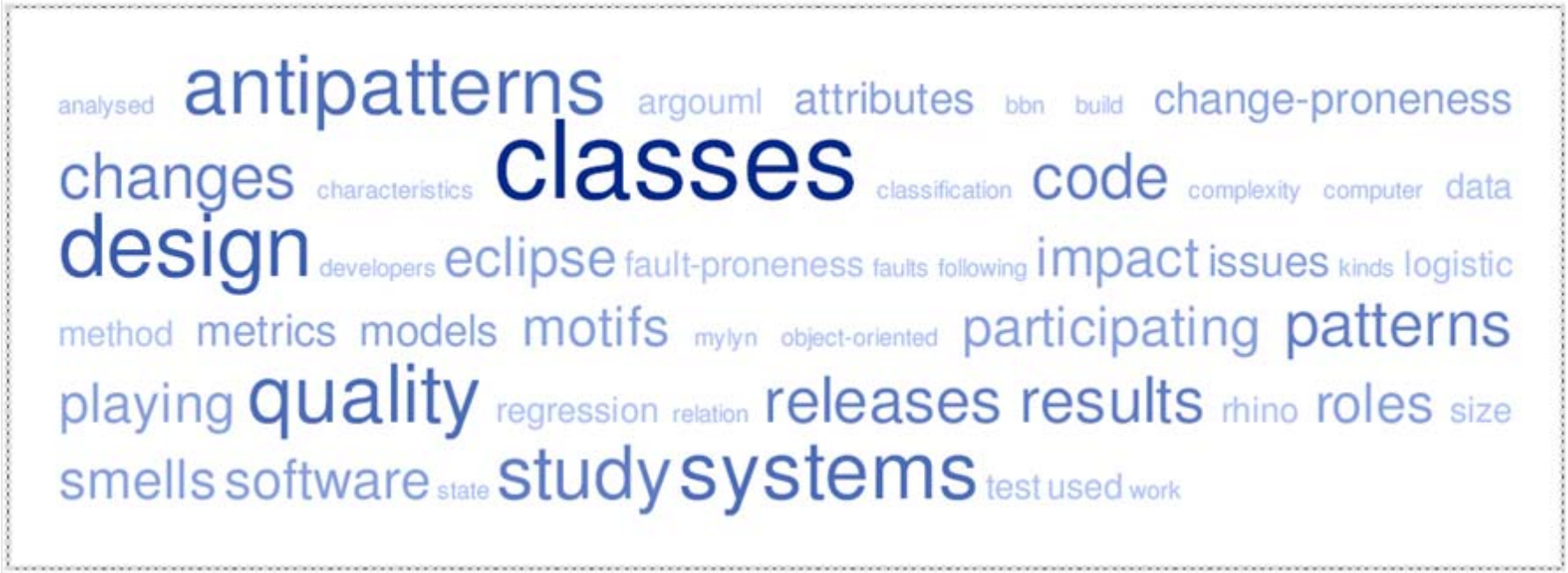
Foutse Khomh

Ph.D. Defense

Department of Computer Science and Operations Research

2010/08/31

Context


analysed antipatterns argouml attributes bbn build change-proneness
changes characteristics classes classification code complexity computer data
design developers eclipse fault-proneness faults following impact issues kinds logistic
method metrics models motifs mylyn object-oriented participating patterns
playing quality regression relation releases results rhino roles size
smells software state study systems test used work

Outline

- Introduction
- Related Work and Contributions
- Experimentations
- Quality Models and Implementation
- Threats to the Validity
- Conclusion and Future Work

Outline

- Introduction
- Related Work and Contributions
- Experimentations
- Quality Models and Implementation
- Threats to the Validity
- Conclusion and Future Work

Introduction

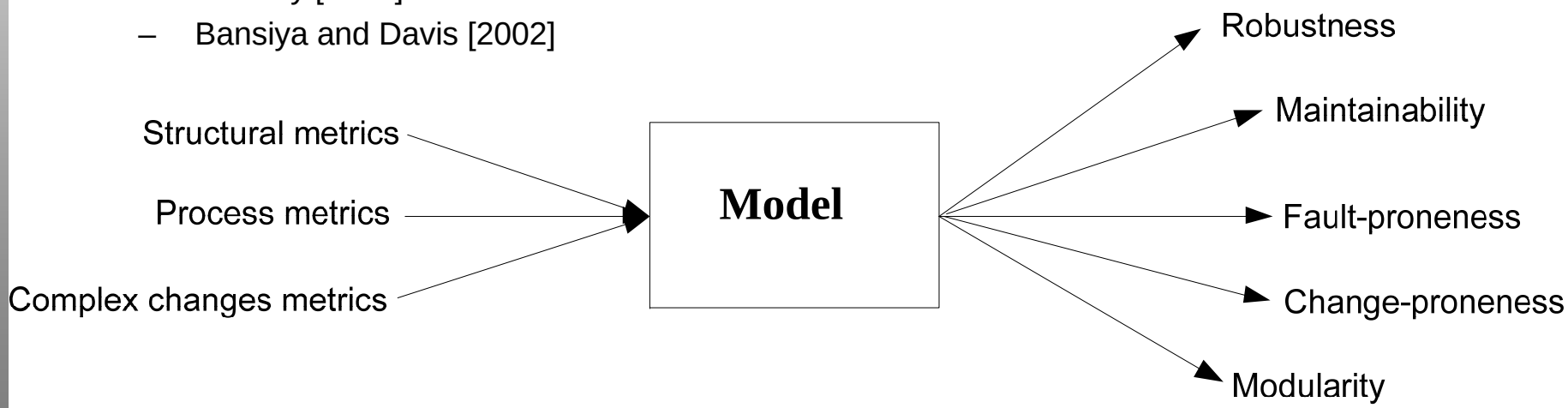
- Maintenance costs during the past decade have reached more than 70% of the overall costs of object-oriented systems
  - Changing software environments
  - Changing users' requirements
  - Overall quality of systems

Introduction

- Many quality models exist [Briand and Wust 2002]
  - Boehm [1976]
  - McCall et al. [1977]
  - ISO 9126 [1991]
  - Dromey [1995]
  - Bansiya and Davis [2002]

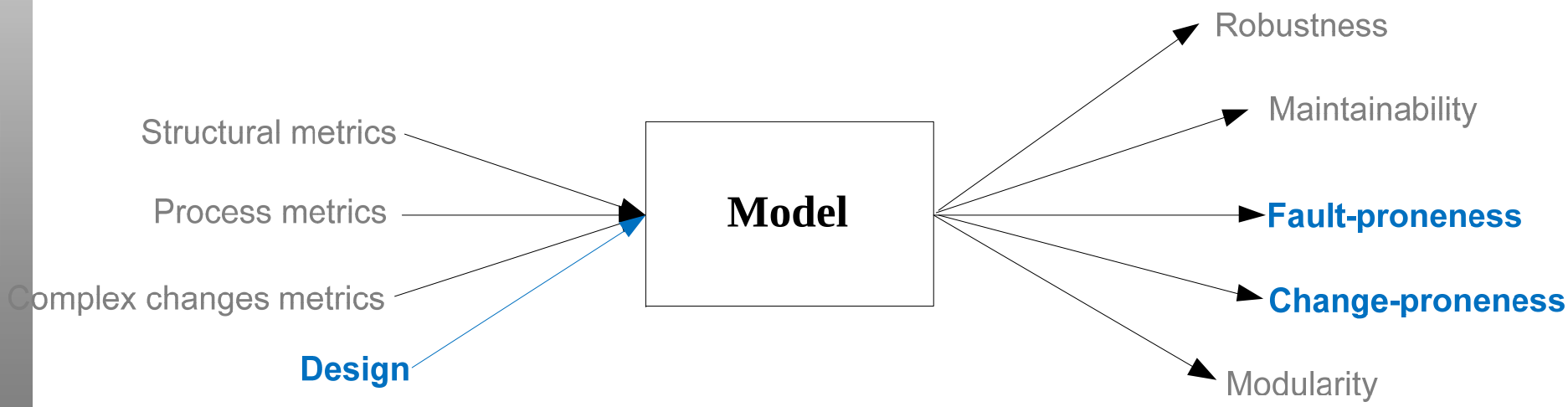
Introduction

- Many quality models exist [Briand and Wust 2002]
  - Boehm [1976]
  - McCall et al. [1977]
  - ISO 9126 [1991]
  - Dromey [1995]
  - Bansiya and Davis [2002]



Introduction

- Yet, the design of a system is the first thing that maintainers see and must master



Introduction

Fault-proneness: removing faults from systems is hard and costly:

it's important to identify them early

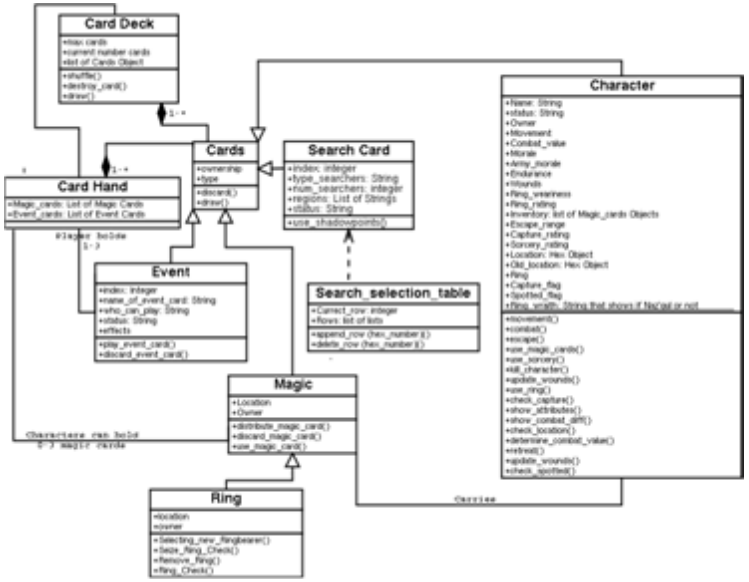
Change-proneness: changing classes requires effort, no matter the reasons of the changes

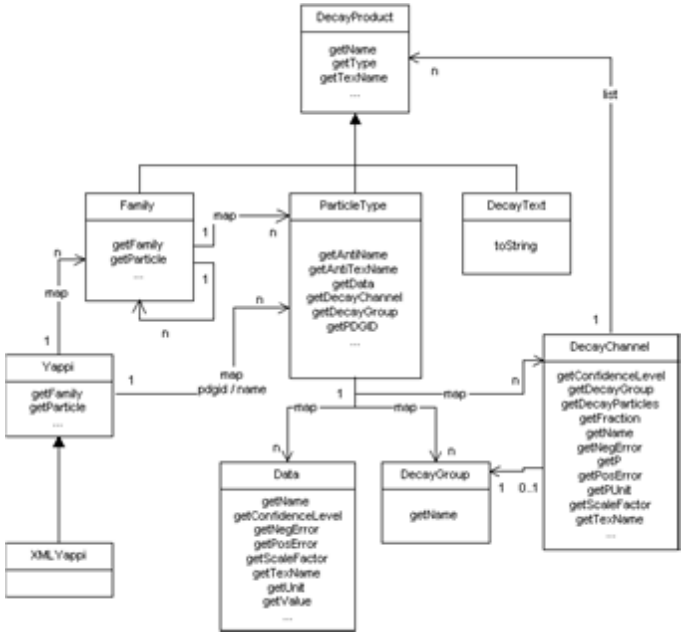
it's important to identify them early

Introduction

- Quality in terms of speed?
  - Their designs → Affect their aerodynamics → Affect their speed

Introduction





- Quality in terms of changes/faults?
  - Their designs → Affect their clarity → Affect their changeability and fault-proneness

Introduction

- Thesis

  “By considering system design; in particular the presence of design patterns and antipatterns, it is possible to build better quality models than simply by considering the internal attributes of classes”

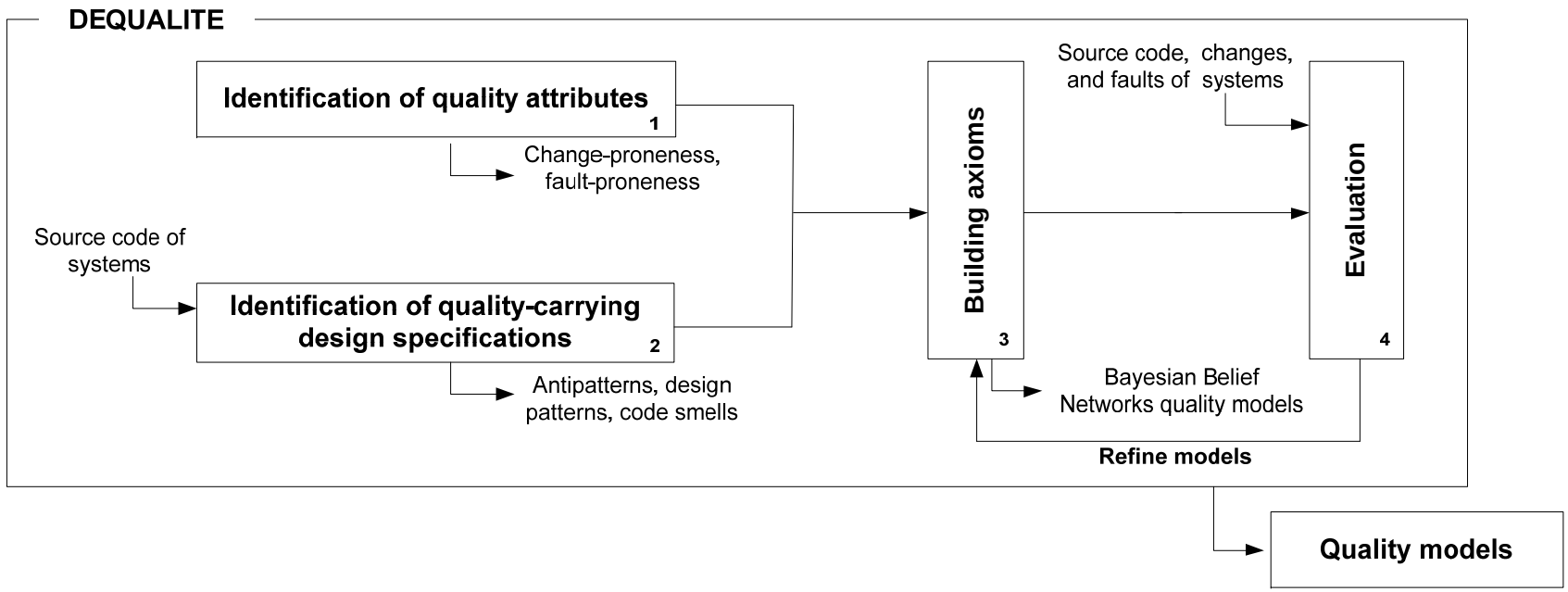
Introduction

- To take the design into account in quality models, we should quantitatively assess their impact on quality attributes

- We propose:
  - A method DEQUALITE to build quality models systematically
  - We perform three empirical studies on the impact of design patterns and antipatterns on change- and fault-proneness

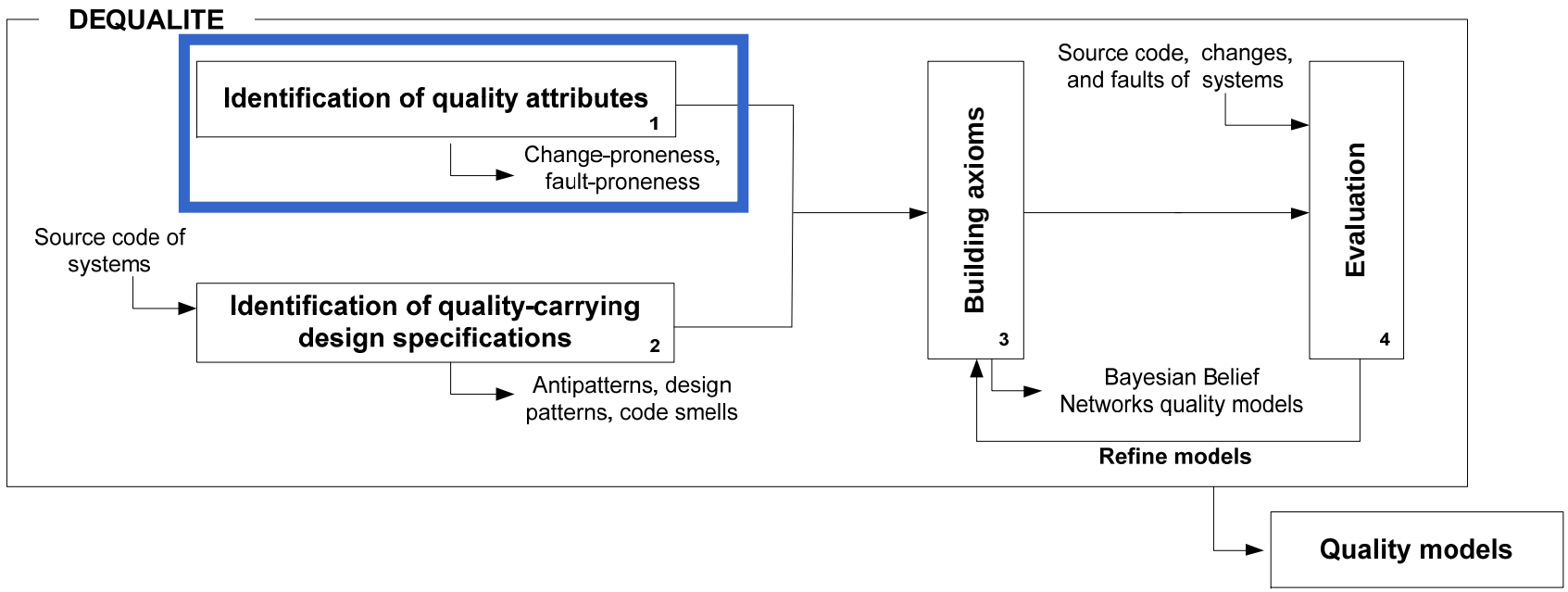
DEQUALITE (1/2)

- Design Enhanced QUALITy Evaluation
  - A method in four steps



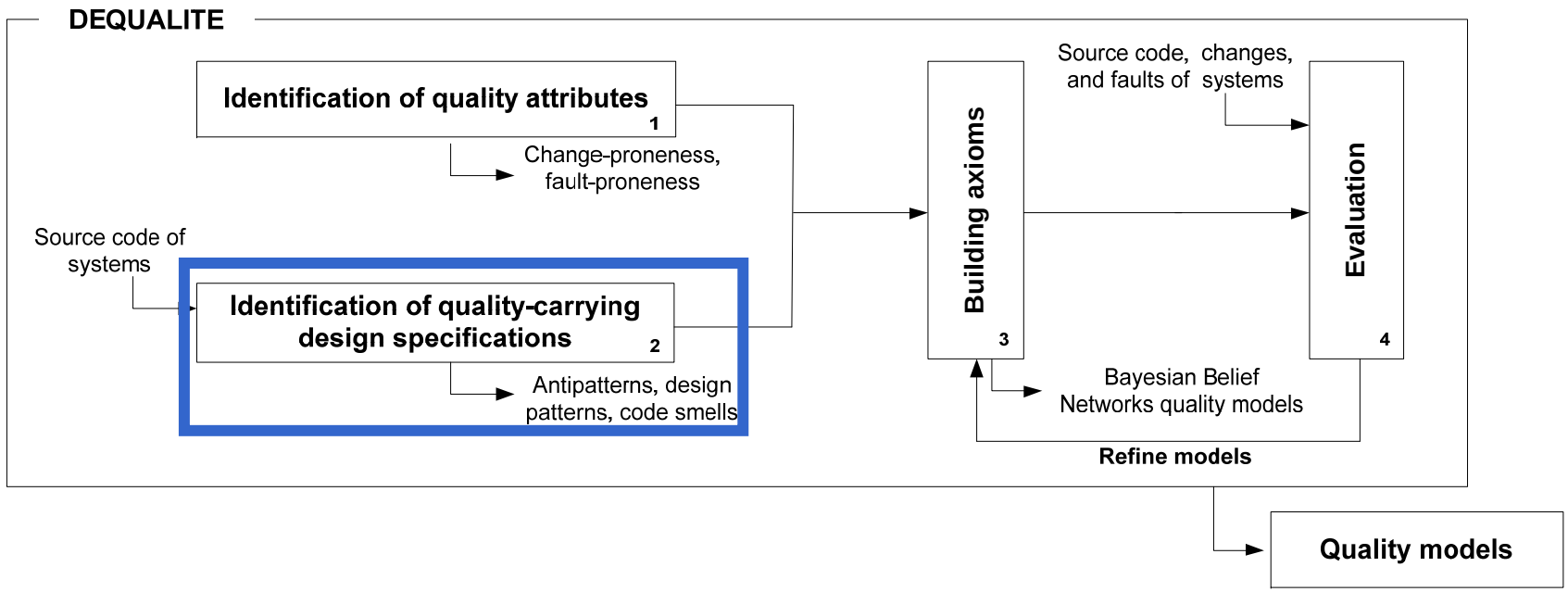
DEQUALITE (1/2)

- Design Enhanced QUALITy Evaluation
  - A method in four steps



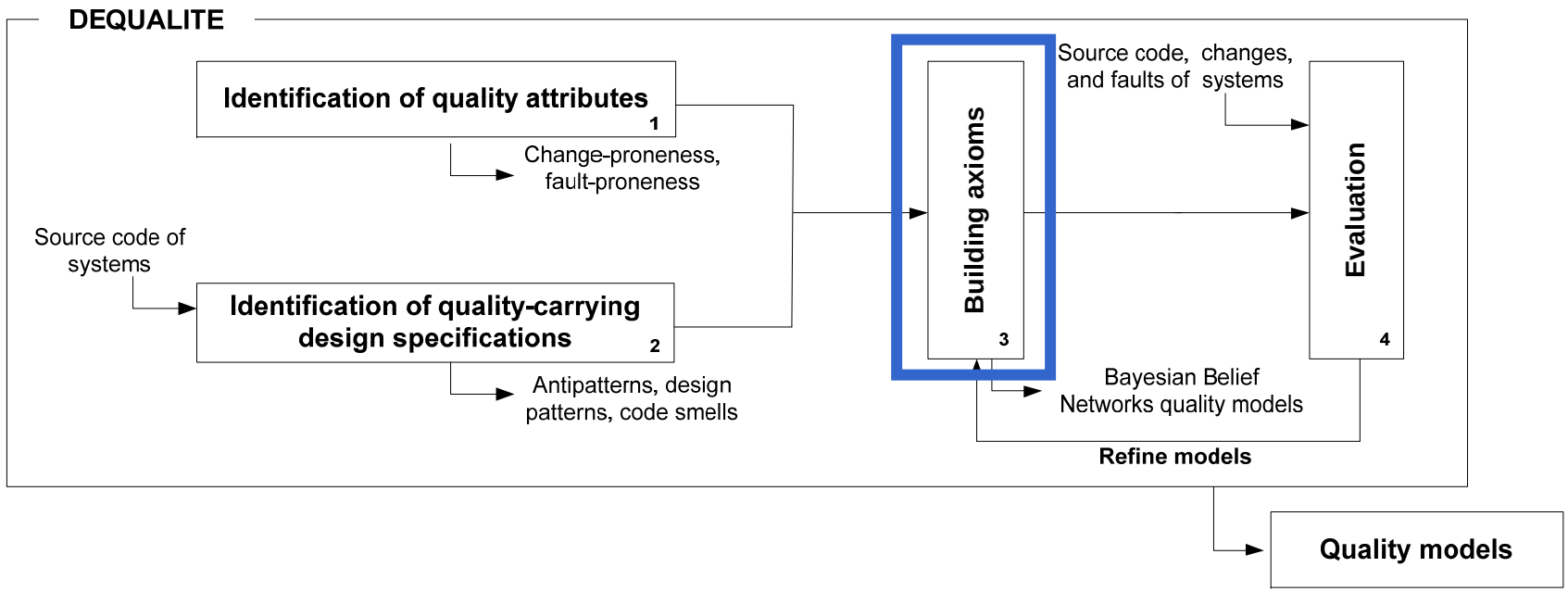
DEQUALITE (1/2)

- Design Enhanced QUALITy Evaluation
  - A method in four steps



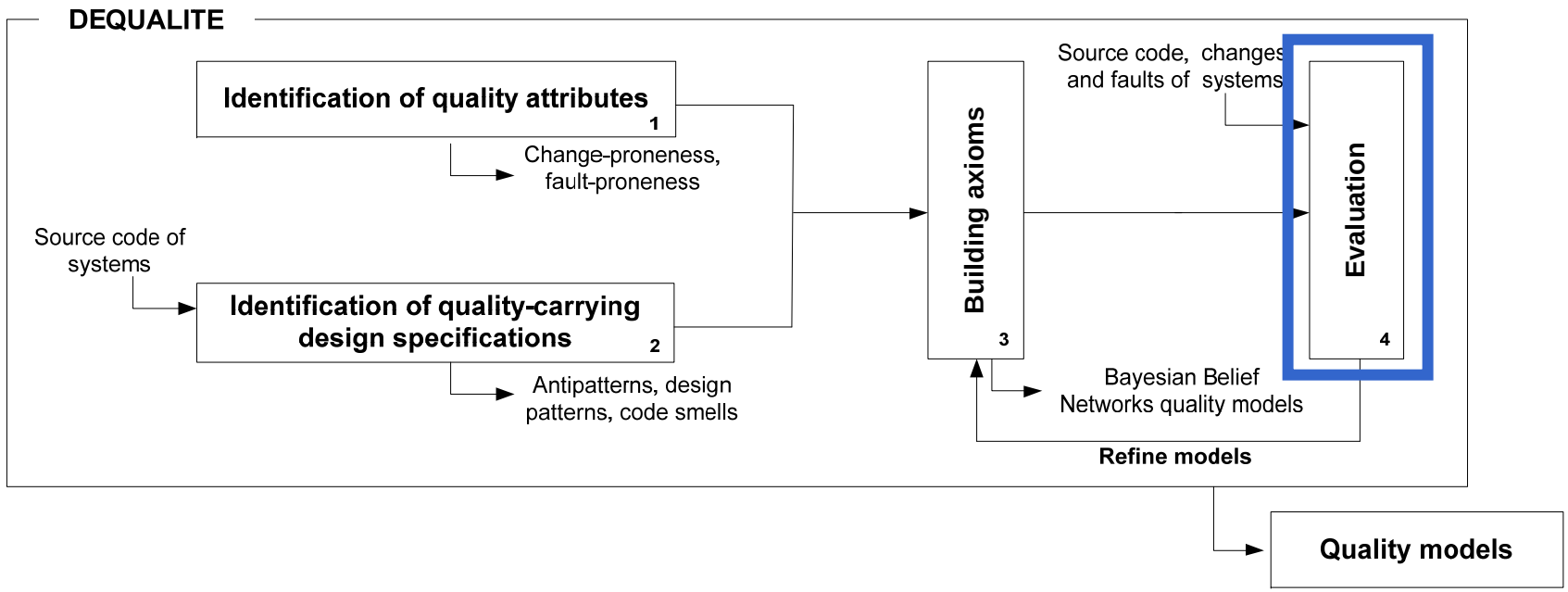
DEQUALITE (1/2)

- Design Enhanced QUALITy Evaluation
  - A method in four steps



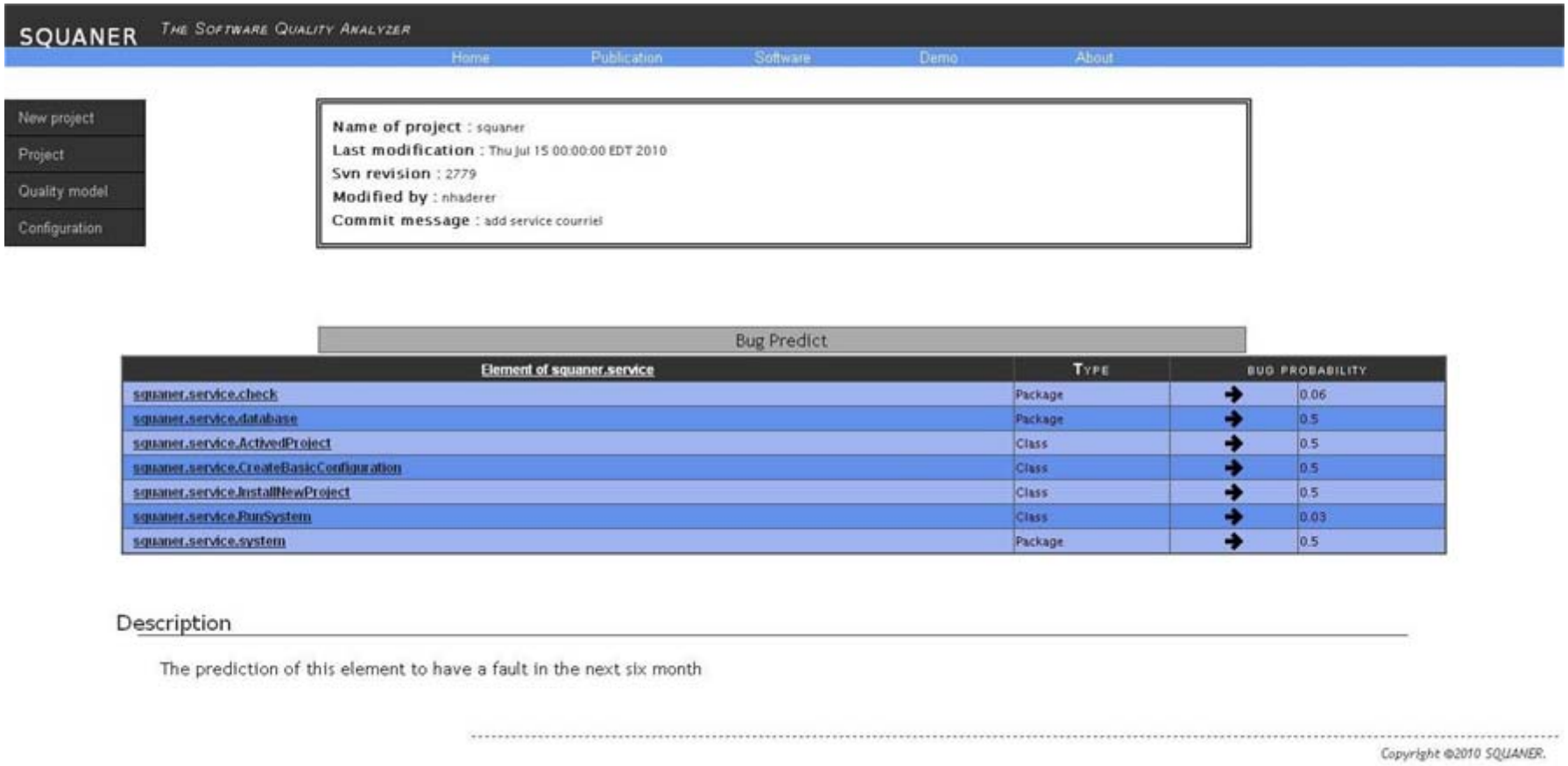
DEQUALITE (1/2)

- Design Enhanced QUALITy Evaluation
  - A method in four steps



DEQUALITE (2/2)

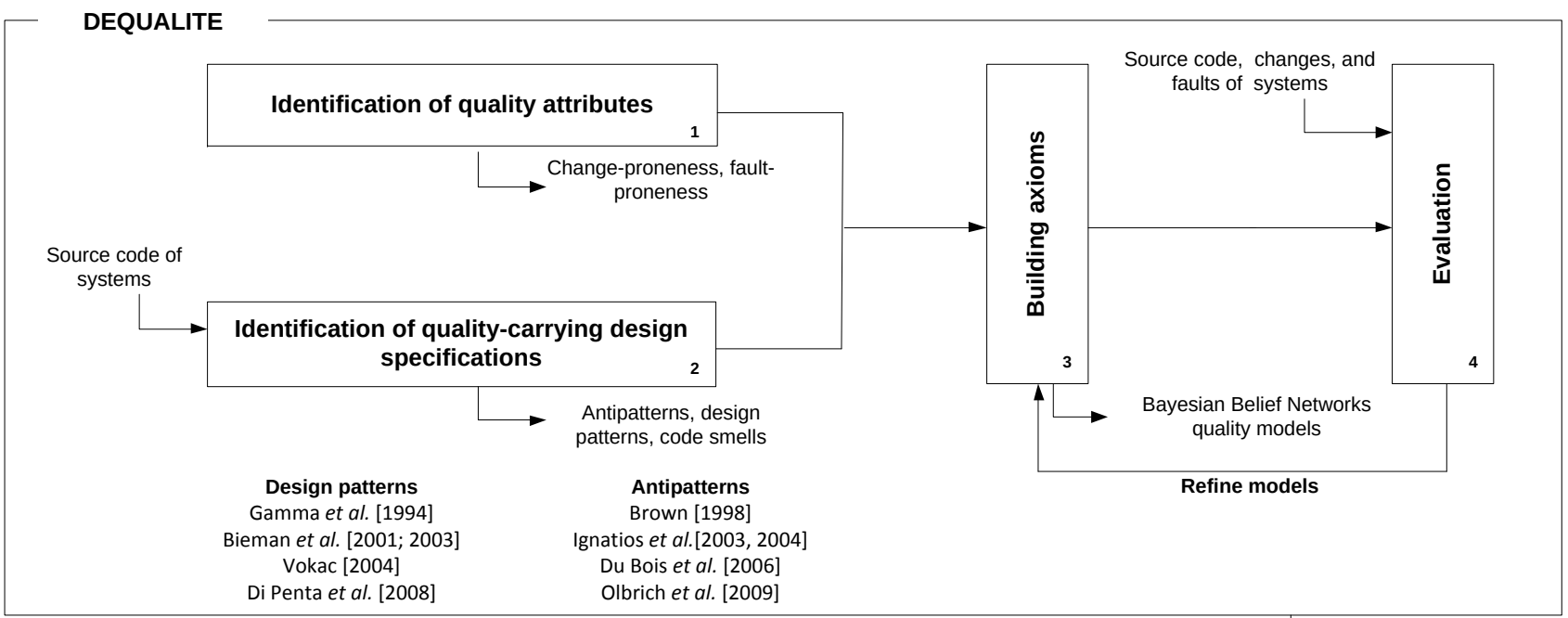
- End Result



Outline

- Introduction
- Related Work and Contributions
- Experimentations
- Quality Models and Implementation
- Threats to the Validity
- Conclusion and Future Work

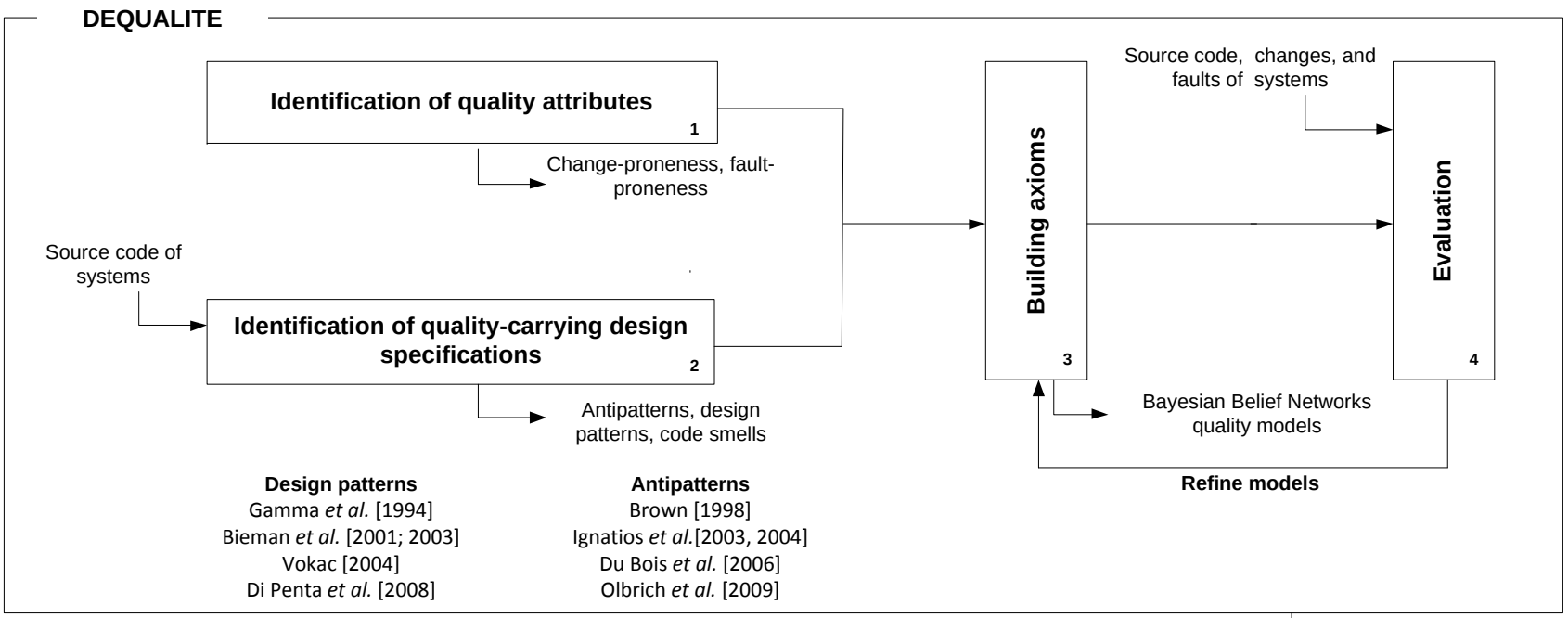
Related Work and Contributions



Quality models

Boehm [1976]
McCall et al. [1977]
ISO 9126 [1991]
Dromey [1995]
Bansiya and Davis [2002]

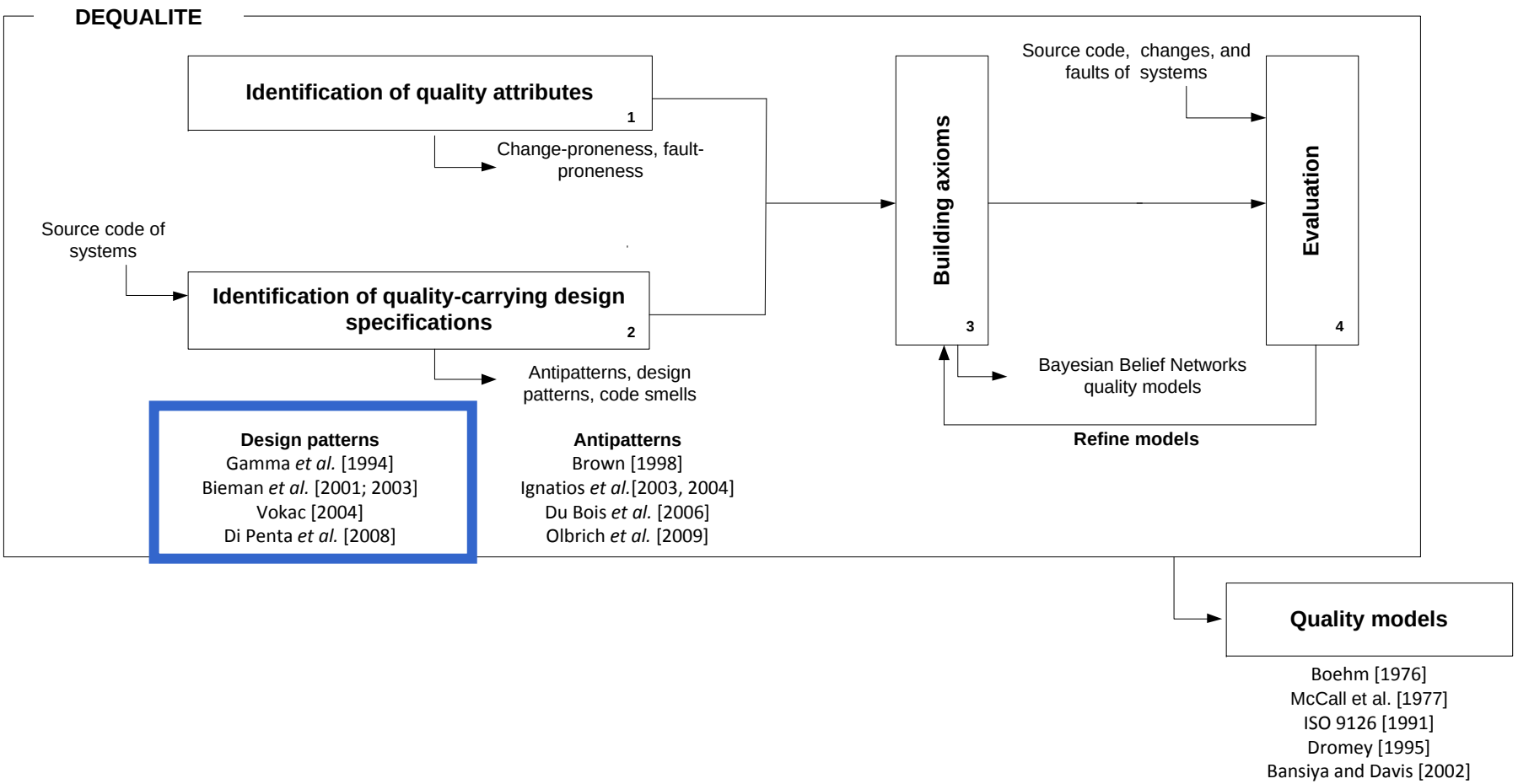
Related Work and Contributions



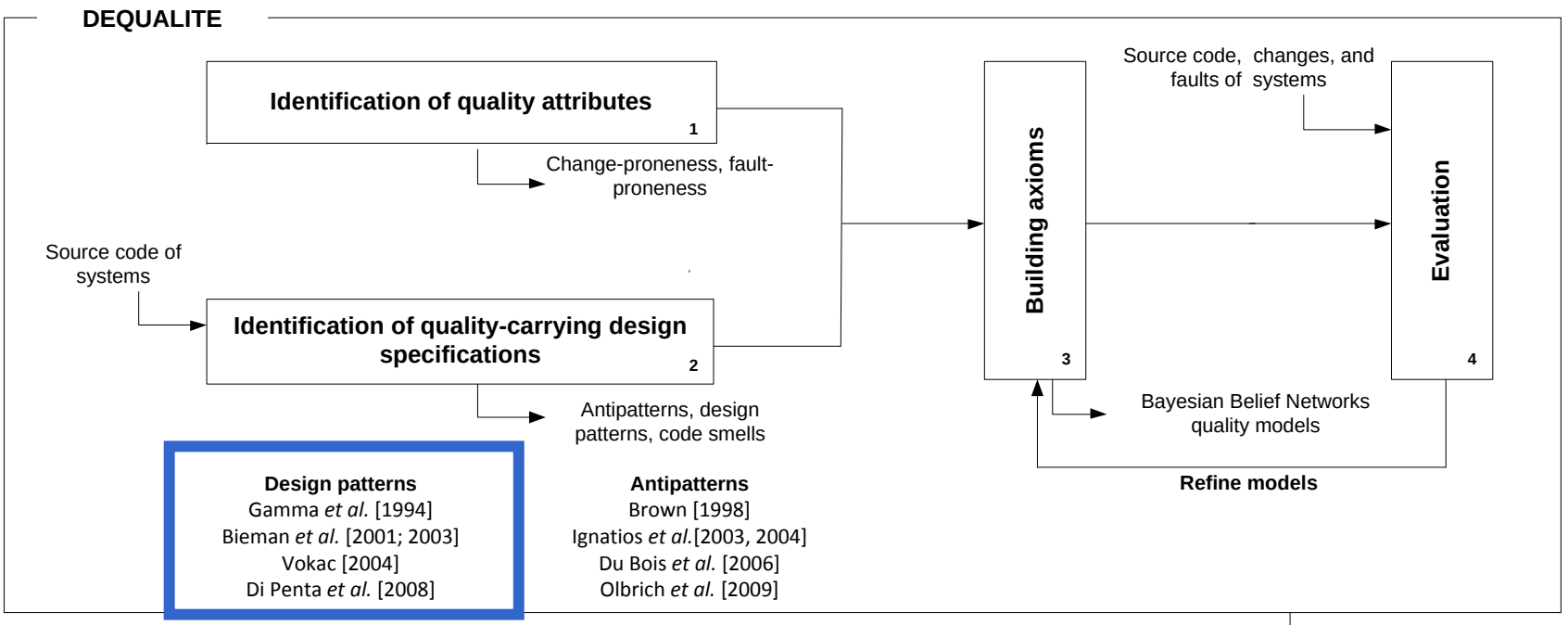
- DEQUALITE, a method to systematically build quality models that take into account both the internal attributes of the systems and their designs

This method allows us to build quality models that outperform state-of-the-art models built with class metrics only

Related Work and Contributions



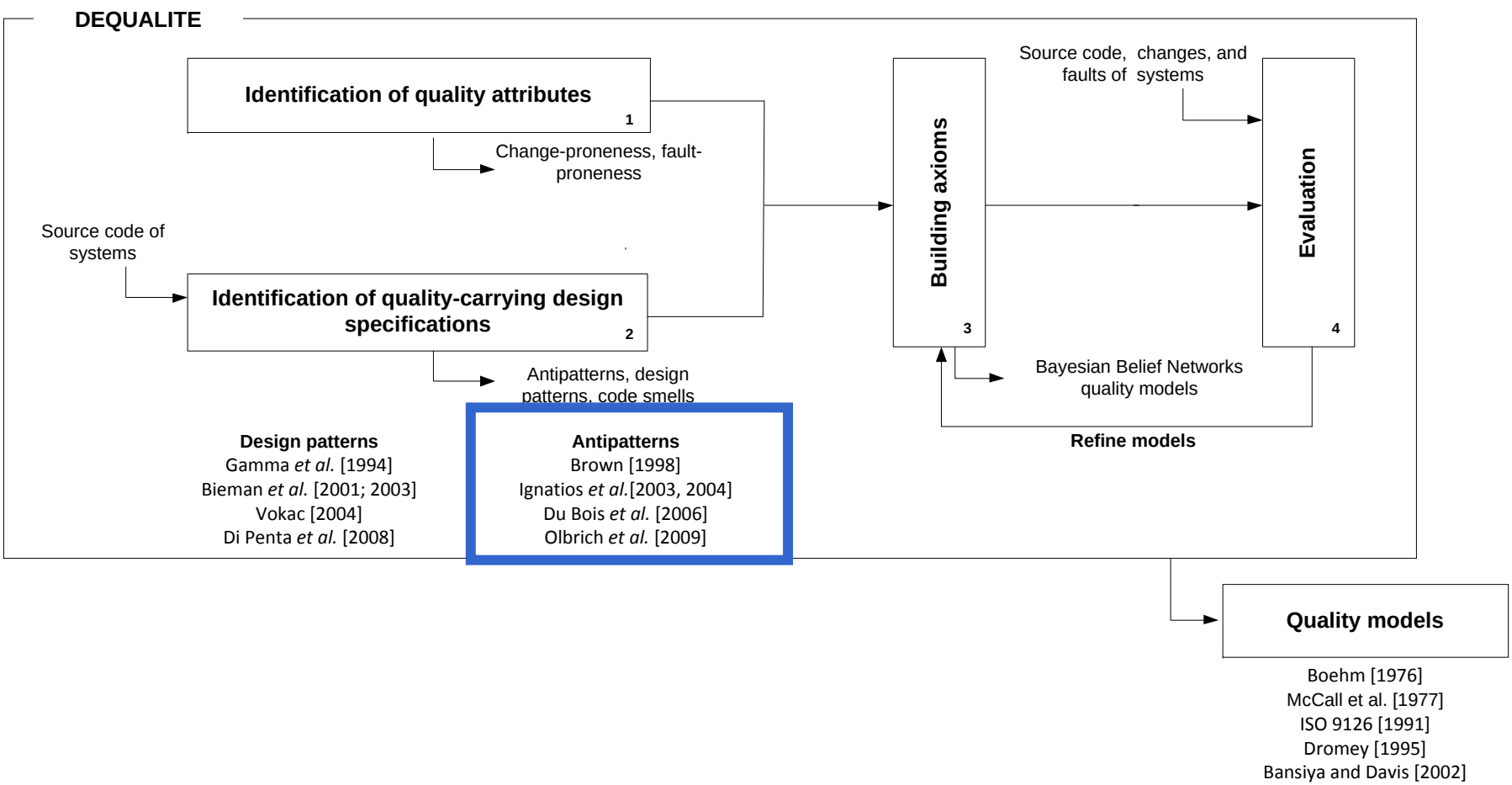
Related Work and Contributions



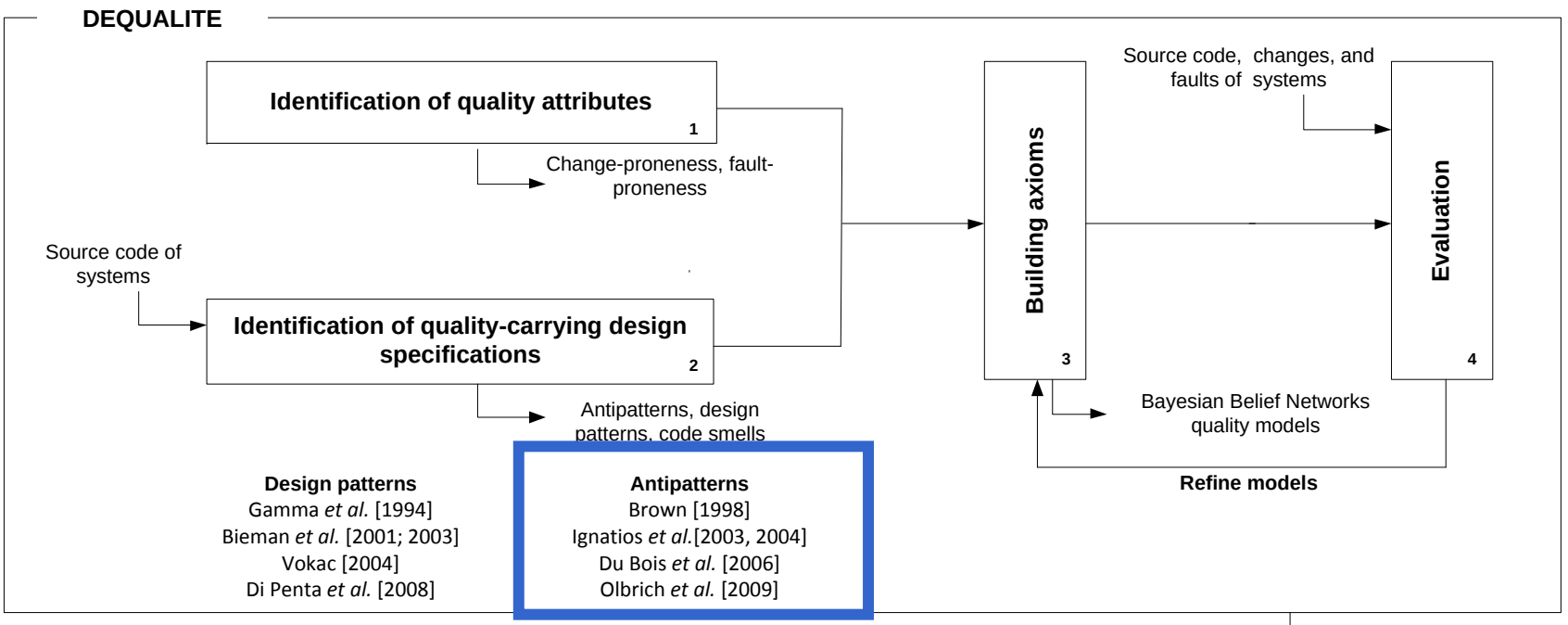
❑ An empirical study of the impact of playing roles in a (some) design pattern(s) for a class, on the internal (class metrics) and external (change- and fault-proneness) characteristics of classes

Roles in design patterns significantly affect the structure of classes as well as their change- and fault-proneness

Related Work and Contributions



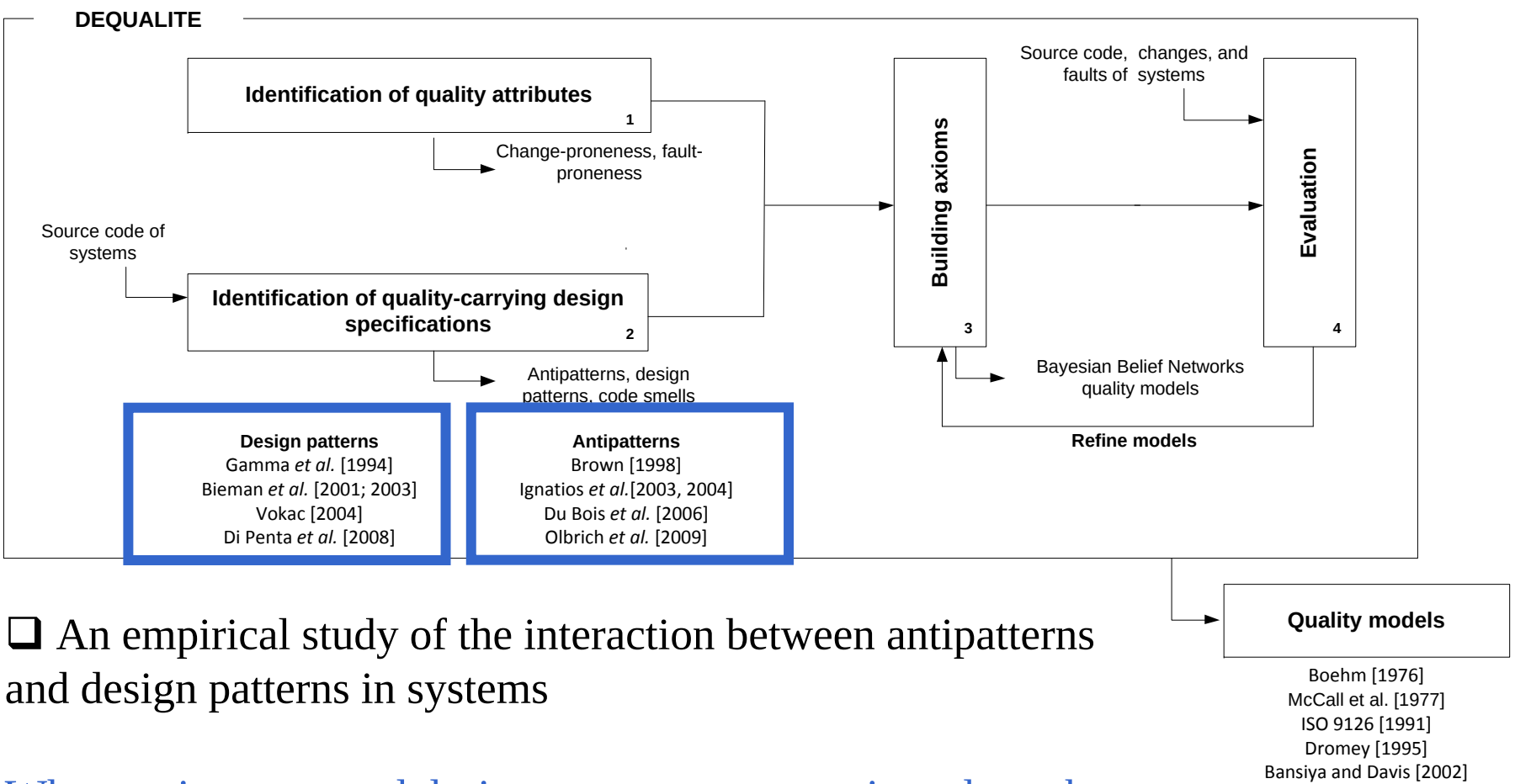
Related Work and Contributions



❑ An empirical study of the impact of antipatterns on class change- and fault-proneness

Classes participating in antipatterns are significantly more likely to be subject to changes and to be involved in fault-fixing issues than other classes

Related Work and Contributions

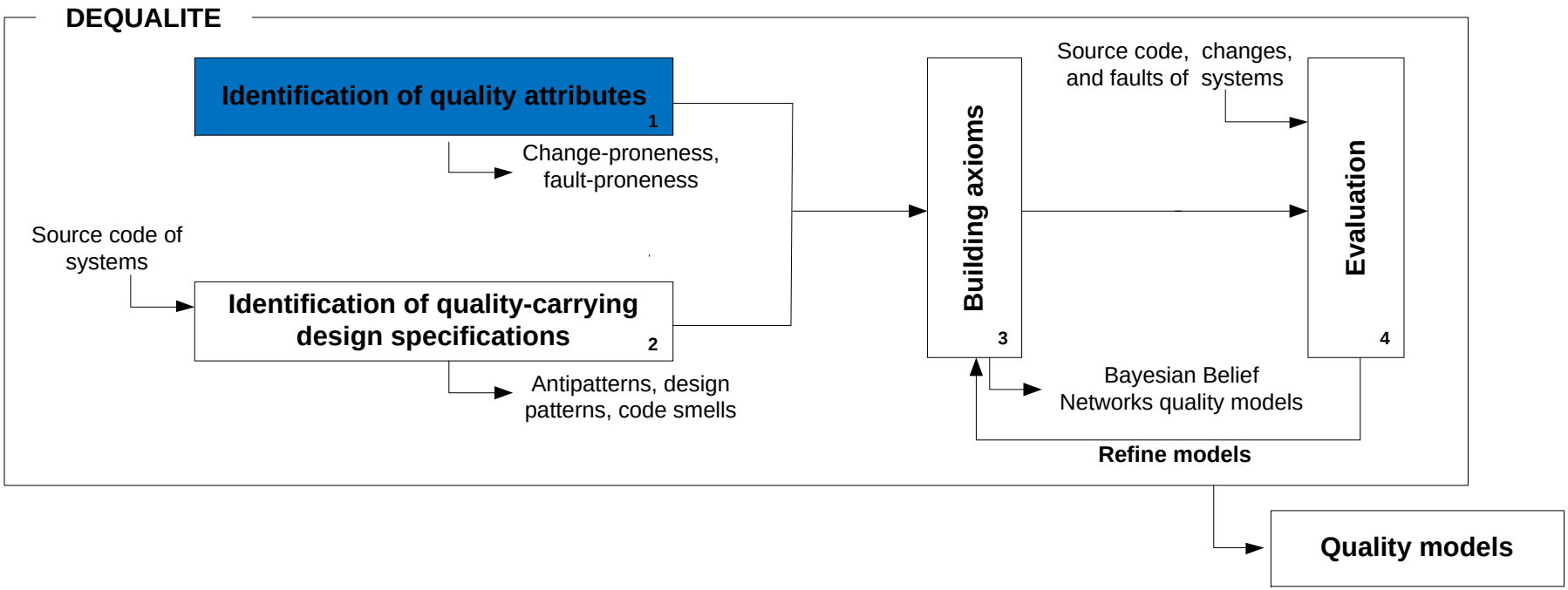


❑ An empirical study of the interaction between antipatterns and design patterns in systems

When antipatterns and design patterns co-occur in a class, the negative effect of antipattern is mitigated

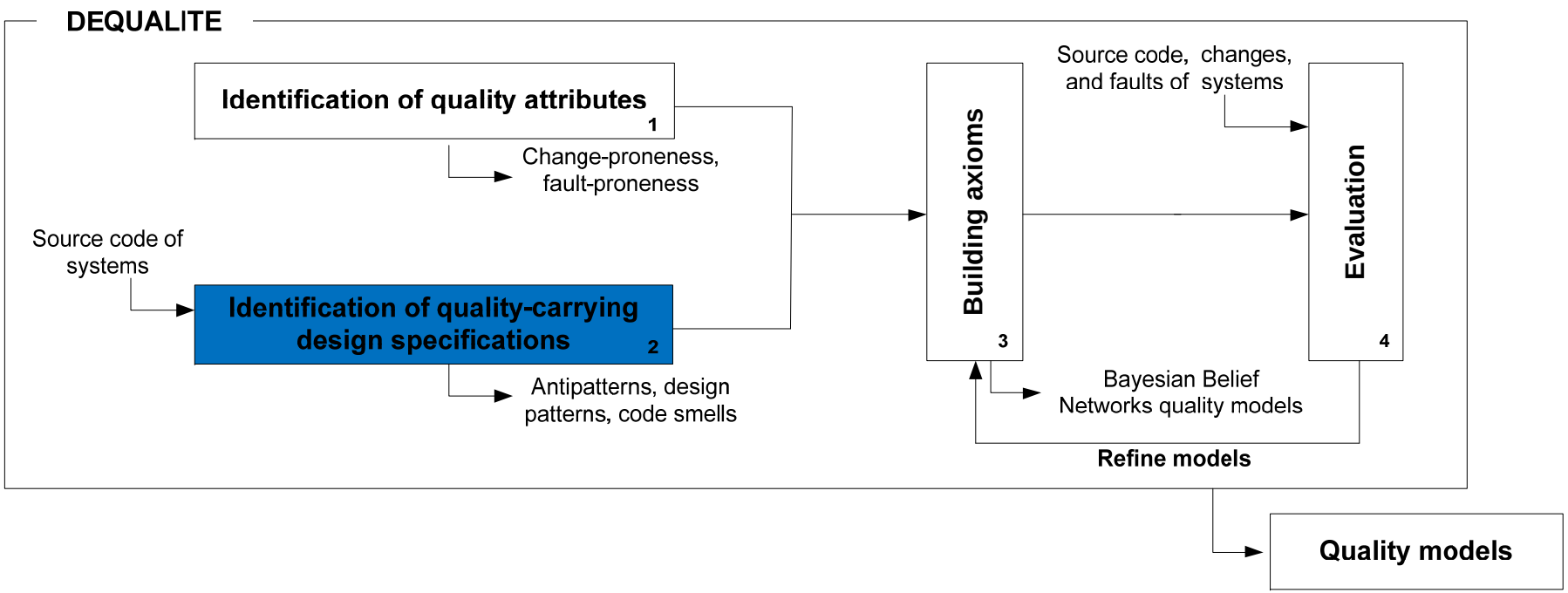
Outline

- Introduction
- Related Work and Contributions
- Experimentations
- Quality Models and Implementation
- Threats to the Validity
- Conclusion and Future Work


DEQUALITE
Identification of quality attributes
1
Change-proneness,
fault-proneness
Source code of
systems
Identification of quality-carrying
design specifications
2
Antipatterns, design
patterns, code smells
Building axioms
3
Source code, changes,
and faults of systems
Evaluation
4
Bayesian Belief
Networks quality models
Refine models
Quality models

Identification of Quality Attributes

- Change-proneness
  - It refers to whether a class underwent at least a change between two given releases
- Fault-proneness
  - It refers to whether a class underwent at least a fault-fixing between two given releases


DEQUALITE
Identification of quality attributes
1
Change-proneness,
fault-proneness
Source code of
systems
Identification of quality-carrying
design specifications
2
Antipatterns, design
patterns, code smells
Building axioms
3
Source code, changes,
and faults of systems
Evaluation
4
Bayesian Belief
Networks quality models
Refine models
Quality models

Design Specifications

- The most popular forms of design implementations in systems are:
  - Design patterns: “good” solutions to design problems
    - Claim to improve the quality of systems
  - Antipatterns: “poor” solutions to design problems
    - Claim to make object-oriented systems harder to maintain
  - Few empirical evidences support these claims

Research Questions

- Design Patterns and Quality
  - What is the impact of design patterns on the change- and fault-proneness of classes?
- Antipatterns and Quality
  - What is the impact of antipatterns on the change- and fault-proneness of classes?
- Relation between Antipatterns and Design Patterns
  - What is the interaction between antipatterns and design patterns and their impact on the change- and fault-proneness of classes?

Method and Needs

- We follow a Goal-Question-Metric methodology
  - Define sub-research questions
  - Formulate null hypotheses
  - Define variables
  - Perform statistical analyses
    - Fisher's exact test
    - Logistic regression model
    - Stepwise regression
    - Wilcoxon rank-sum test
    - We compute Odds ratios (OR)
    - We compute sample sizes
    - We compute effect sizes

Method and Needs

- Needs:
  - A population of systems
  - A list of design patterns
  - A list of antipatterns
  - Data on changes
  - Data on faults

Method and Needs

- A population of systems
  - Eclipse ~ 3,756,164 LOCs
  - JDT Core ~ 528,522 LOCs
  - ArgoUML ~ 316,971 LOCs
  - Mylyn ~ 276,401 LOCs
  - Xalan ~ 259,286 LOCs
  - Xerces ~ 86,814 LOCs
  - Azureus ~ 83,534 LOCs
  - Rhino ~ 79,406 LOCs
  - JHotDraw ~ 44,898 LOCs

Method and Needs

- A list of design patterns
  - Adapter (A)
  - Command (Cmd)
  - Composite (C)
  - Decorator (D)
  - Factory Method (FM)
  - Observer (O)
  - Prototype (P)
  - State (S)
  - Template Method (TM)
  - Visitor (V)

Method and Needs

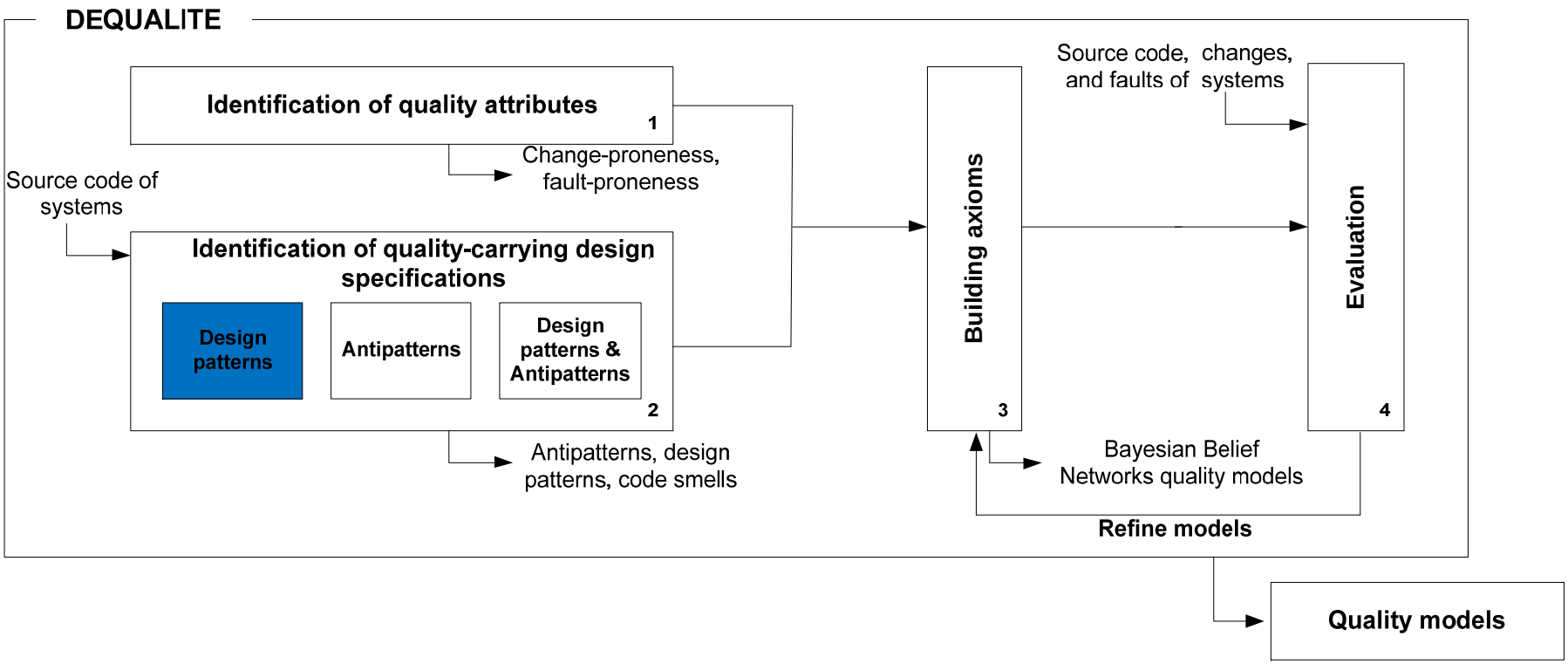
- A list of antipatterns
  - AntiSingleton
  - Blob
  - ClassDataShouldBe Private (CDSBP)
  - ComplexClass
  - LargeClass
  - LazyClass
  - LongMethod
  - LongParameterList (LPL)
  - MessageChains
  - RefusedParentBequest (RPB)
  - SpaghettiCode
  - SpeculativeGenerality (SG)
  - SwissArmyKnife

Method and Needs

- Data on changes
  - We count the number of changes $c_{i,k}$ that a class underwent between two subsequent releases r_k and r_{k+1}
  - Changes are identified, for each class in a system, by looking at commits in the control-version system (CVS or SVN); for each class, we counted, the number of commits related to that class

Faults

- Need: Data on faults
  - We count the number of fault-fixing issues occurring to a class between two subsequent releases r_k and r_{k+1}
  - We considered a set of manually-validated and publicly-available faults for Mylyn and Rhino


DEQUALITE
Identification of quality attributes
1
Change-proneness, fault-proneness
Source code of systems
Identification of quality-carrying design specifications
Design patterns
Antipatterns
Design patterns & Antipatterns
2
Antipatterns, design patterns, code smells
Building axioms
3
Source code, changes, and faults of systems
Evaluation
4
Bayesian Belief Networks quality models
Refine models
Quality models

Design Patterns

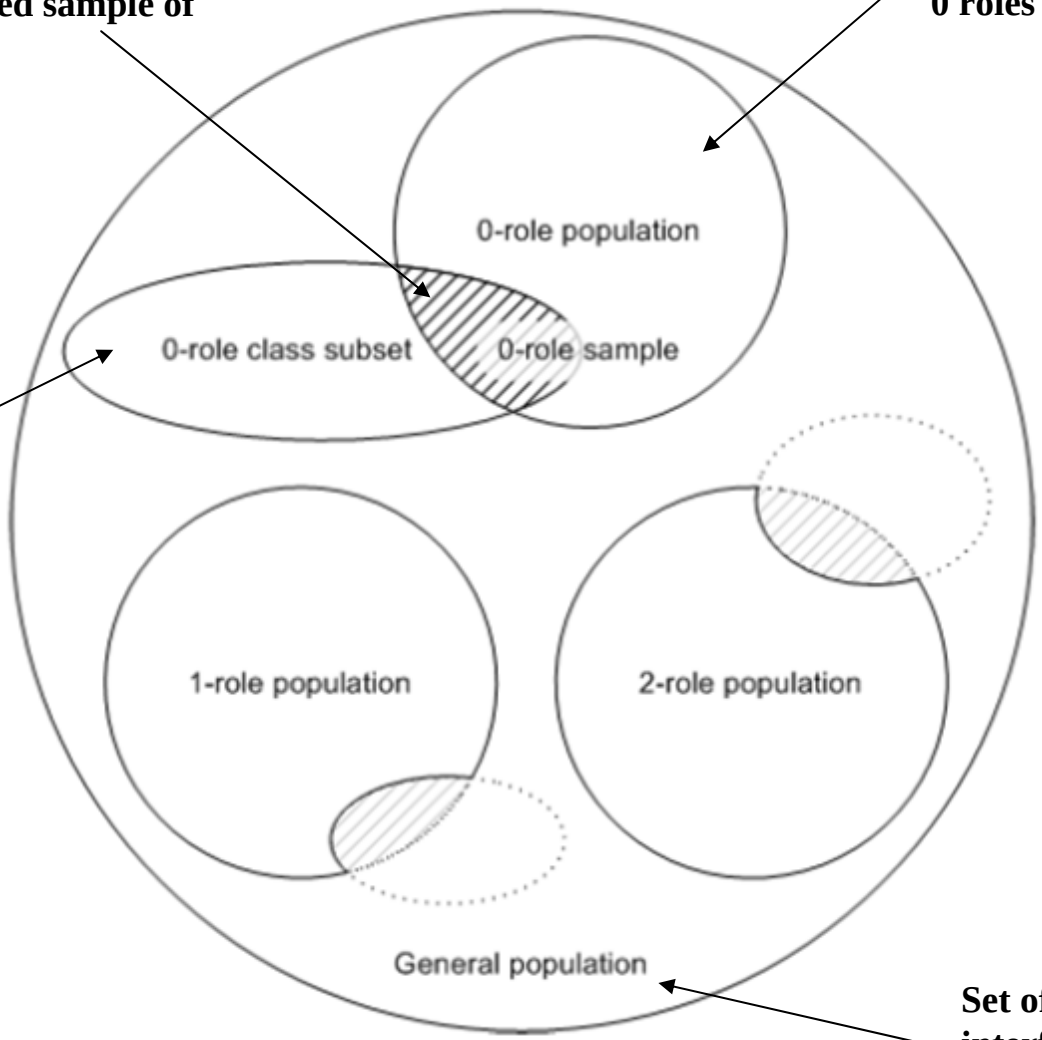
- Sub-research questions:
  - RQ1: What is the proportion of classes playing zero, one, or two roles in some design patterns?
  - RQ2: What are the internal characteristics of a class that are the most impacted by playing one or two roles with respect to playing less roles?
  - RQ3: What are the external characteristics (change- and fault-proneness) of a class that are the most impacted by playing one or two roles with respect to playing less roles?

Variables (1/3)

- Independent variables
  - Three samples of classes playing zero, one, and two roles in design motifs
    - We name these samples
      - 0-role sample
      - 1-role sample
      - 2-role sample
    - We use DeMIMA to extract design patterns [Guéhéneuc and Antoniol, 2008]

Variables (2/3)

- Independent variables



We manually validated 238 classes

Variables (3/3)

- Dependent variables
  - 56 different metrics from the literature
    - Coupling metrics
    - Complexity metrics
    - Cohesion metrics
    - Inheritance metrics
    - Polymorphism and size
  - Change proneness
  - Fault proneness

Results (1/4)

- RQ1
  - Classes playing one or two roles do exist in programs and are not negligible

Programs	Total	One Role	Two Roles
ArgoUML v0.18.1	1,267	51	316
	100%	4.02%	24.94%
Azureus v2.1.0.0	591	67	75
	100%	11.33%	[illegible]
JDT Core v2.1.2	669	46	178
	100%	6.88%	26.60%
JHotDraw v5.4b2	413	24	101
	100%	5.81%	24.45%
Xalan v2.7.0	734	36	104
	100%	4.90%	14.16%
Xerces v1.4.4	306	94	56
	100%	30.72%	18.30%
Total	3,980	318	830
	100%	7.99%	20.85%

Results (2/4)

- RQ2

Metric Groups	Metric Names	1 role vs. 0 role		2 role vs. 0 role		2 role vs. 1 role	
		p-values	Trends	*p*-values	Trends	*p*-values	Trends
Cohesion	CAM	0.854		**0.0001996**	↗	**0.0003884**	↗
	cohesionAttributes	0.6881		**0.04051**	↗	**0.0009488**	↗
	LCOM1	**0.01313**	↘	**6.22E-09**	↗	**0.0009946**	↗
	LCOM2	**0.01087**	↘	**1.41E-07**	↗	**0.0017**	↗
	LCOM5	**0.03454**	↗	**3.95E-06**	↗	**0.001383**	↗
Complexity	McCabe	0.2274		**7.85E-07**	↗	**0.00063**	↗
	SIX	**0.004657**	↗	**1.41E-08**	↗	**0.0008183**	↗
	WMC1	**2.09E-05**	↗	**4.00E-08**	↗	**0.0467**	↗
	WMC	**0.01453**	↘	**5.40E-07**	↗	**0.001297**	↗
Coupling	ACAIC	0.1733		**0.03935**	↗	0.5029	
	ACMIC	0.284		**0.002702**	↗	**0.04961**	↗
	CBO	0.5706		**0.0001434**	↗	**0.001948**	↗
	CBOin	0.191		**7.89E-06**	↗	**0.0005939**	↗
	CBOout	0.1055		**5.96E-07**	↗	**0.0001025**	↗
	connectivity	0.5005		0.07963		0.2603	
	CP	0.9802		0.2272		0.1428	
	DCAEC	**9.37E-06**	↗	**0.003612**	↗	0.06724	
	DCC	0.4149		**2.98E-05**	↗	**0.002347**	↗
	DCMEC	**0.0001468**	↗	**0.001024**	↗	0.595	
	PP	0.829		0.1382		0.1468	
	RFP	**0.04845**	↗	**0.01477**	↗	0.6074	
	RRFP	0.0968		**0.02306**	↘	0.5106	
	RRTP	**0.02637**	↘	**0.03722**	↘	0.6952	
	RTP	0.2005		**0.01295**	↗	0.3693	

Not significant (8) Significant 29 48 26

Results (3/4)

- RQ2

Metric Groups	Metric Names	1 role vs. 0 role		2 role vs. 0 role		2 role vs. 1 role	
		p-values	Trends	*p*-values	Trends	*p*-values	Trends
Inheritence	AID	0.126		**0.0001542**	↗	0.1391	
	ANA	0.2958		0.8977		0.2918	
	CLD	**< 2.2e-16**	↗	**7.94E-11**	↗	**0.003298**	↘
	DIT	0.08713		**8.59E-05**	↗	0.2632	
	NCM	**0.00087**	↗	**4.84E-09**	↗	0.07486	
	NOC	**2.22E-16**	↗	**3.55E-11**	↗	0.245	
	NOD	**2.22E-16**	↗	**5.29E-11**	↗	0.07351	
	NOH	0.5644		0.601		0.9663	
	NOP	0.2248		**6.10E-06**	↗	**0.007146**	↗
	ICHClass	**0.03035**	↗	**2.03E-07**	↗	**0.001095**	↗
Polymorphism and Size	CIS	**9.22E-07**	↗	**1.50E-08**	↗	0.1605	
	DAM	0.1285		**1.94E-05**	↗	**0.003362**	↗
	DSC	0.1461		0.2098		0.8725	
	EIC	**0.0002848**	↗	**9.03E-06**	↗	0.5616	
	EIP	**7.26E-13**	↗	**1.43E-09**	↗	0.1039	
	MFA	0.1138		0.7105		0.243	
	MOA	**0.0001883**	↗	**6.44E-10**	↗	**0.01493**	↗
	NAD	0.1349		**5.03E-06**	↗	**0.003884**	↗
	NADExtended	0.1514		**1.14E-05**	↗	**0.005466**	↗
	NCP	**5.39E-06**	↗	**0.01465**	↗	0.1198	
	NMA	**9.34E-06**	↗	**2.30E-06**	↗	0.3157	
	NMD	**2.09E-05**	↗	**4.00E-08**	↗	**0.0467**	↗
	NMDExtended	**3.37E-05**	↗	**1.07E-07**	↗	0.05112	
	NMI	0.1029		**0.0001075**	↗	0.2016	
	NMO	**0.00163**	↗	**3.57E-10**	↗	**0.0005408**	↗
	NOA	0.1868		**7.35E-08**	↗	**0.01153**	↗
	NOM	**2.09E-05**	↗	**4.00E-08**	↗	**0.0467**	↗
	NOParam	**7.81E-06**	↗	**2.38E-08**	↗	0.1551	
	NOPM	**2.89E-14**	↗	**1.93E-10**	↗	0.2793	
	PIIR	**7.00E-05**	↗	**0.01216**	↗	0.2846	
	REIP	**5.94E-10**	↗	**7.54E-08**	↗	0.3336	
	RPII	0.1486		0.08605		0.8614	

Results (4/4)

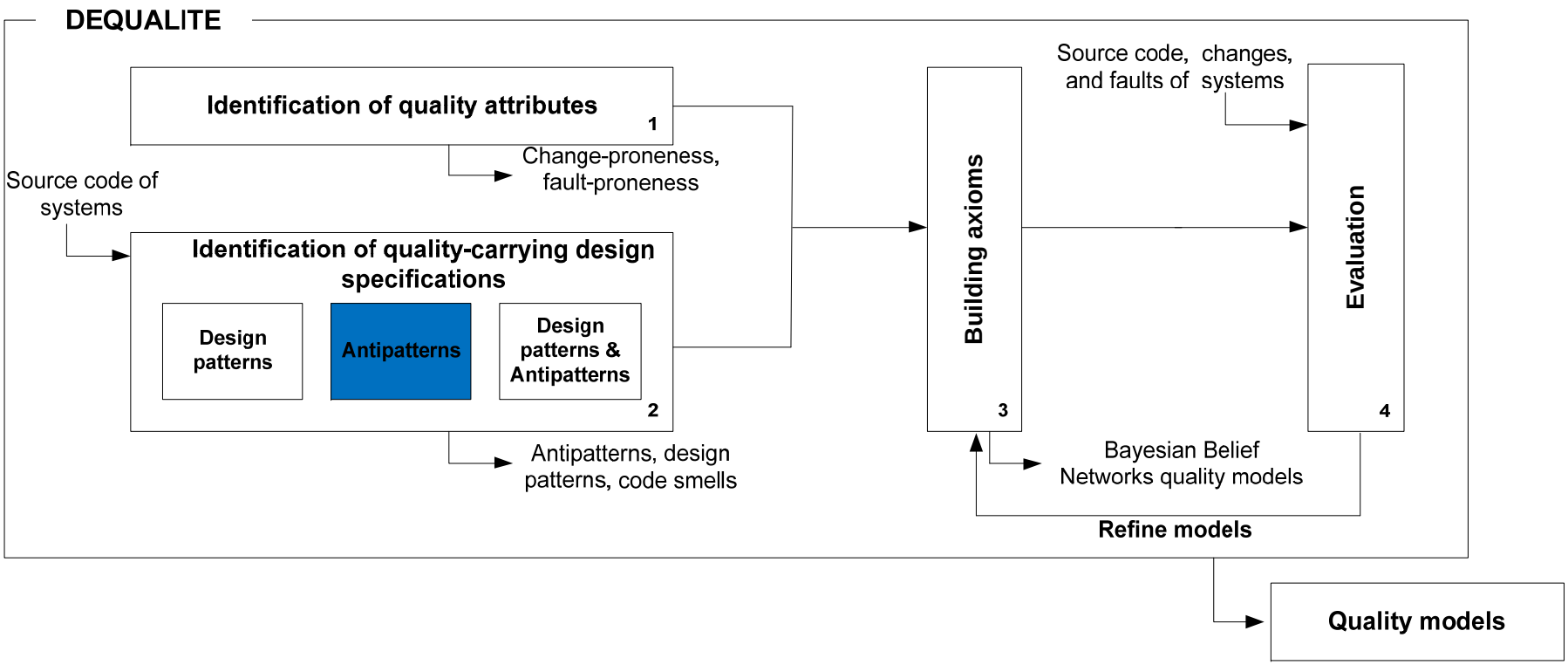
- RQ3

Metric Groups	Metric Names	1 role vs. 0 role		2 role vs. 0 role		2 role vs. 1 role	
		p-values	Trends	*p*-values	Trends	*p*-values	Trends
Changeability	Frequencies of Past Changes	**8.26E-07**	↗	**1.24E-09**	↗	0.08794	
	Frequencies of Future Changes	**0.0001564**	↗	**7.44E-06**	↗	0.5983	
	Numbers of Past Changes	**3.54E-07**	↗	**5.50E-10**	↗	0.06668	
	Numbers of Future Changes	**0.001552**	↗	**9.72E-05**	↗	0.7018	
Issues	Numbers of Issues	**0.0003619**	↗	**0.0003612**	↗	0.6645	

  - Playing roles do impact the number of changes and issues as well as the frequencies of the changes
  - Yet, no significant difference between one/two roles for change- and issues-proneness

Summary on Design Patterns

- In average, 8% of the classes of the six studied programs played 1 role in some design pattern
- In average, 18% of the classes of the six studied programs played 2 roles in some design patterns
- Playing 1 or 2 roles in a design pattern has a significant impact on the structure of classes: coupling, cohesion, inheritance, connectivity, complexity…
- Playing 1 or 2 roles in a design pattern have a significant impact on the change- and issue-proneness of classes


DEQUALITE
Identification of quality attributes
1
Change-proneness, fault-proneness
Source code of systems
Identification of quality-carrying design specifications
Design patterns
Antipatterns
Design patterns & Antipatterns
2
Antipatterns, design patterns, code smells
Building axioms
3
Source code, changes, and faults of systems
Evaluation
4
Bayesian Belief Networks quality models
Refine models
Quality models

Antipatterns

- Sub-research questions
  - RQ1and RQ2: What is the relation between antipatterns and change- and fault- proneness?
  - RQ3 and R4: What is the relation between particular kinds of antipatterns and change- and fault-proneness?
  - RQ5: What kind of changes are performed on classes participating or not in antipatterns?

Variables (1/2)

- Independent variables
  - 13 kinds of antipatterns
    - We counted the number of times a class *i* has an antipattern *j* in a release r_k
    - We use DECOR to extract antipatterns [Moha *et al.*, 2009]

Variables (2/2)

- Dependent variables
  - Class change-proneness
  - Class fault-proneness
  - Kinds of changes
    - We counted as the number of each kind of changes occurring to a class participating in an antipattern in release *k*
      - Structural changes: addition/removal/change of/to attributes, addition/removal of methods, or changes to the method signatures
      - Non-structural changes: changes in method implementation

Results (1/4)

- RQ1 and RQ2: antipatterns and changes/faults

Proneness to Changes								Proneness to Faults/Issues							
ArgoUML		Eclipse		Mylyn		Rhino		ArgoUML		Eclipse		Mylyn		Rhino	
Releases	Odds Ratios	Releases	Odds Ratios	Releases	Odds Ratios	Releases	Odds Ratios	Releases	Odds Ratios	Releases	Odds Ratios	Releases	Odds Ratios	Releases	Odds Ratios
0.10.1	4.17	**1.0**	1.13	1.0.1	10.51	1.4R3	10.41	0.10.1	4.43	1.0	1.32	1.0.1	10.45	1.4R3	6.44
0.12	7.16	2.0	**0.75**	2.0M1	10.37	1.5R1	17.98	0.12	4.87	2.0	1.57	2.0M1	17.70	1.5R1	31.29
0.14	6.22	2.1.1	2.55	2.0M2	7.38	1.5R2	17.37	0.14	17.53	2.1.1	1.70	2.0M2	>>300	1.5R2	–
0.16	15.84	2.1.2	1.42	2.0M3	206.60	1.5R3	15.71	0.16	6.58	2.1.2	2.00	2.0M3	–	1.5R3	13.93
0.18.1	10.00	2.1.3	1.15	2.0	14.17	1.5R4	16.19	0.18.1	5.33	2.1.3	2.03	2.0	–	1.5R4	9.06
0.20	26.54	3.0	**0.88**	2.1	10.89	1.5R41	30.71	0.20	4.95	3.0	2.52	2.1	–	1.5R41	30.05
0.22	8.83	3.0.1	**0.86**	2.2.0	11.10	1.5R5	15.51	0.22	9.42	3.0.1	1.95	2.2.0	–	1.5R5	10.57
0.24	15.40	3.0.2	**0.89**	2.3.0	9.83	1.6R1	24.73	0.24	2.25	3.0.2	1.86	2.3.0	–	1.6R1	29.26
0.26	3.98	3.2	2.15	2.3.1	7.66	1.6R2	12.69	0.26	8.08	3.2	2.72	2.3.1	–	1.6R2	–
0.26.2	6.75	3.2.1	1.94	2.3.2	24.38	1.6R3	19.95	0.26.2	9.73	3.2.1	2.19	2.3.2	–	1.6R3	–
		3.2.2	1.47	3.0.0	9.45	1.6R4	33.05			3.2.2	2.05	3.0.0	–	1.6R4	23.00
		3.3	2.43	3.0.1	9.85	1.6R5	19.97			3.3	3.18	3.0.1	–	1.6R5	13.29
		3.3.1	1.42	3.0.2	5.31	1.6R6	20.56			3.3.1	1.23	3.0.2	–	1.6R6	–
				3.0.3	8.18							3.0.3	–		
				3.0.4	3.77							3.0.4	–		
				3.0.5	4.96							3.0.5	–		
				3.1.0	10.53							3.1.0	–		
				3.1.1	5.59							3.1.1	–		

Classes with antipatterns are more change/faults-prone than others, few exceptions for Eclipse

Results (2/4)

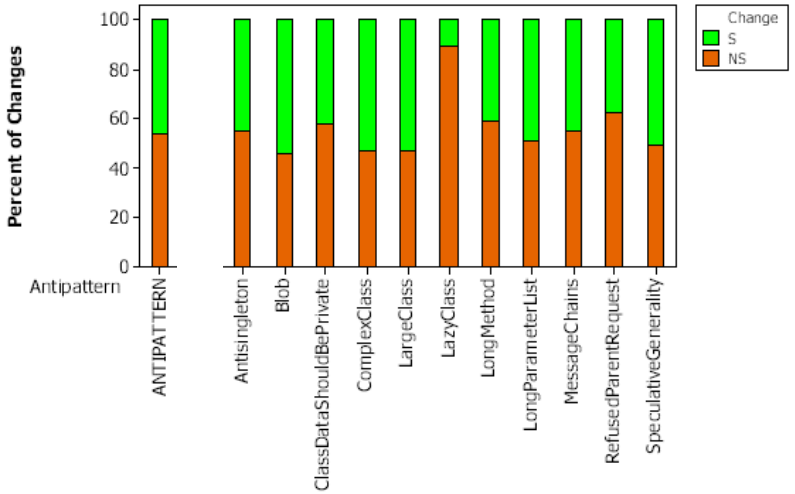
- RQ3 and RQ4: kinds of antipatterns and changes/faults

Antipatterns	Proneness to Changes				Proneness to Faults/Issues			
	ArgoUML	Eclipse	Mylyn	Rhino	ArgoUML	Eclipse	Mylyn	Rhino
AntiSingleton	**8 (80%)**	5 (38%)	7 (39%)	–	5 (50%)	**13 (100%)**	–	–
Blob	2 (20%)	8 (62%)	9 (50%)	–	1 (10%)	7 (54%)	–	–
CDSBP	3 (30%)	7 (54%)	9 (50%)	6 (46%)	2 (20%)	7 (54%)	2 (66%)	3 (33%)
ComplexClass	2 (20%)	**12 (92%)**	2 (11%)	–	–	**13 (100%)**	1 (33%)	–
LargeClass	2 (20%)	–	4 (22%)	4 (31%)	3 (30%)	–	–	3 (33%)
LazyClass	5 (50%)	**12 (92%)**	3 (17%)	1 (8%)	–	**12 (92%)**	–	2 (22%)
LongMethod	**10 (100%)**	**12 (92%)**	**17 (94%)**	5 (38%)	1 (10%)	**13 (100%)**	–	3 (33%)
LPL	**9 (90%)**	**10 (77%)**	7 (39%)	3 (23%)	5 (50%)	9 (60%)	2 (66%)	3 (33%)
MessageChains	**10 (100%)**	**12 (92%)**	**18 (100%)**	**13 (100%)**	7 (70%)	**10 (77%)**	1 (33%)	**7 (78%)**
RFB	**9 (90%)**	6 (46%)	10 (56%)	5 (38%)	4 (40%)	4 (31%)	1 (33%)	–
SpaghettiCode	–	–	–	–	–	–	–	–
SG	–	3 (23%)	6 (33%)	1 (8%)	–	4 (31%)	–	1 (11%)
SwissArmyKnife	–	6 (46%)	–	–	–	1 (8%)	–	–

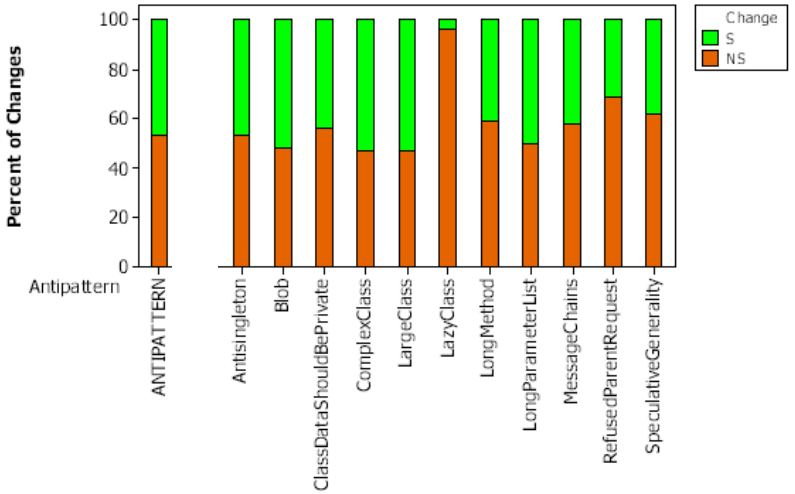
MessageChains are consistently and significantly correlated to more changes/faults

Results (3/4)

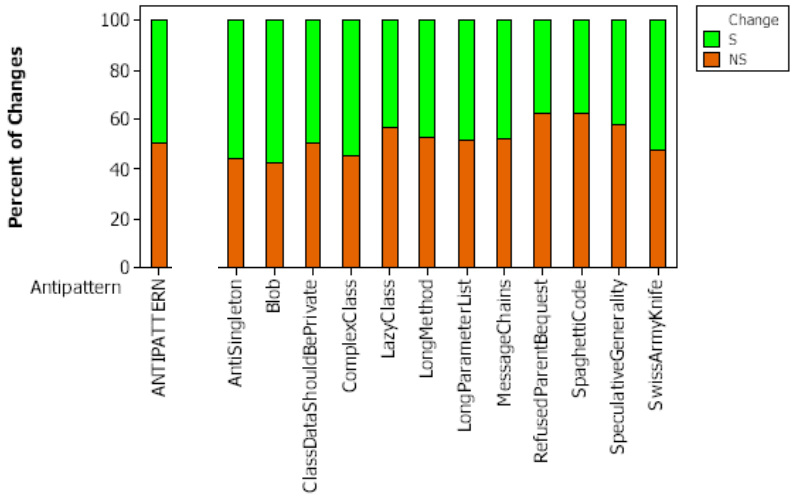
- RQ5: kinds of changes and antipatterns



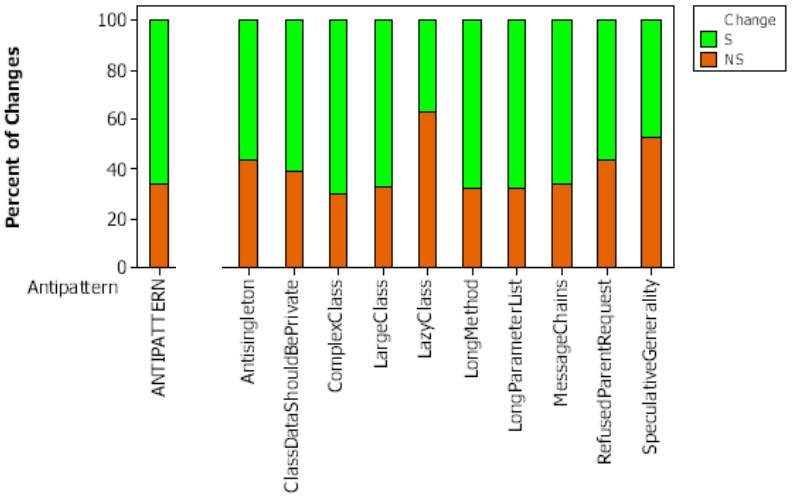
(a) ArgoUML



(b) Mylyn



(c) Eclipse



(d) Rhino

Results (4/4)

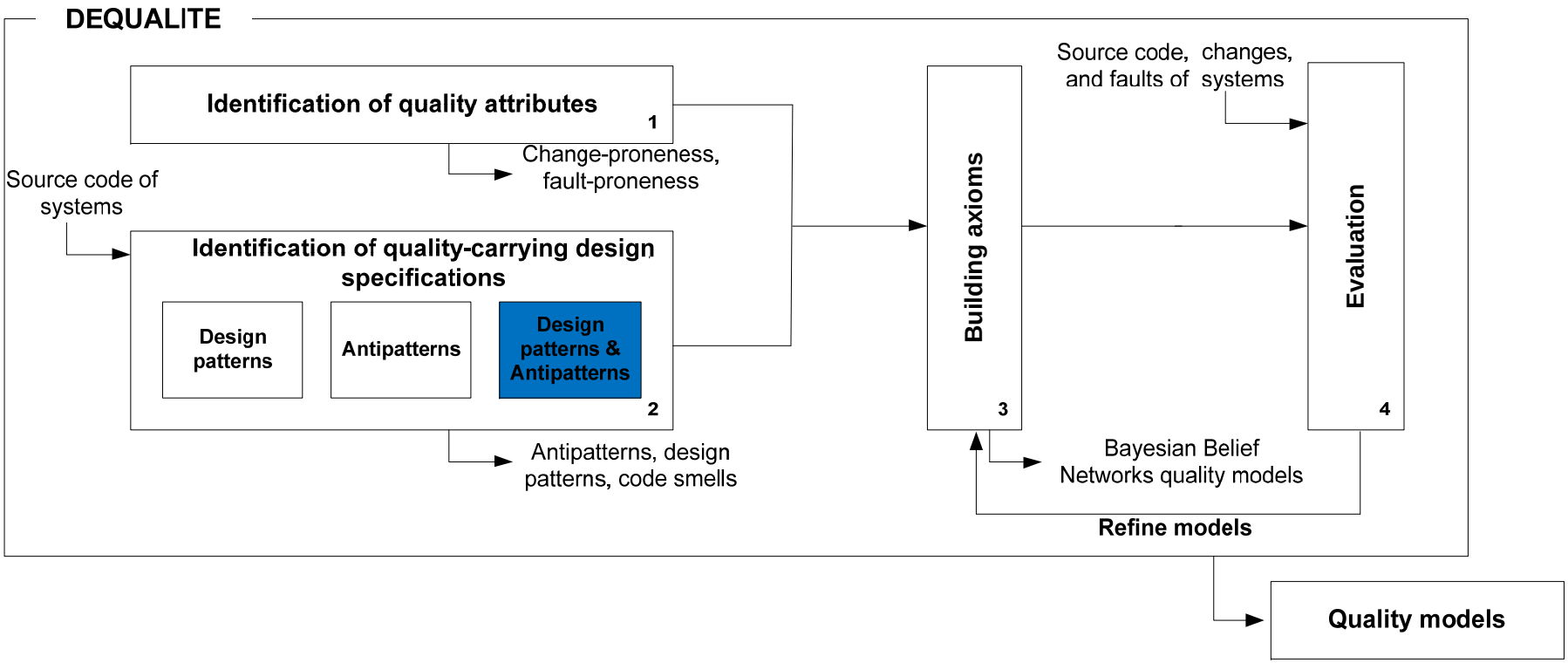
- RQ5: kinds of changes and antipatterns

Systems	p-values	ORs
ArgoUML	< 0.01	1.22
Eclipse	< 0.01	1.03
Mylyn	< 0.01	1.19
Rhino	0.08	1.04

- Structural changes occur more often on classes belonging to antipatterns than other kinds of changes

Summary on Antipatterns

- Classes with antipatterns are more change/fault-prone, with high odds ratios
- MessageChains are consistently and significantly correlated to more changes/faults
- Structural changes occur more often on classes belonging to antipatterns than other kinds of changes. However the effect of this relation is small


DEQUALITE
Identification of quality attributes
1
Change-proneness, fault-proneness
Source code of systems
Identification of quality-carrying design specifications
Design patterns
Antipatterns
Design patterns & Antipatterns
2
Antipatterns, design patterns, code smells
Building axioms
3
Source code, changes, and faults of systems
Evaluation
4
Bayesian Belief Networks quality models
Refine models
Quality models

Design Patterns & Antipatterns

- Research questions
  - RQ1: What is the number of classes participating in antipatterns and design patterns?
  - RQ2: What is the impact on change-proneness for a class to participate both in some antipatterns and design patterns?
  - RQ3: What is the impact of playing roles in particular kinds of antipatterns and design patterns with respect to change-proneness?

Variables (1/2)

- Independent variables
  - 13 kinds of antipatterns
    - We counted the number of times a class i has an antipattern j in a release r_l
    - We use DECOR to extract antipatterns [Moha *et al.,* 2009]
  - 10 kinds of design patterns
    - We counted the number of times a class i has an antipattern j and plays a role in a design pattern k in a release r_l
    - We use DeMIMA to extract design patterns [Guéhéneuc and Antoniol, 2008]

Variables (2/2)

- Dependent variables
  - Class change-proneness

Results (1/4)

- RQ1: proportion of co-occurrences

Systems	Classes	Classes APs		Classes DPs		Classes APs+DPs	
ArgoUML	2,834	1,791	(63%)	1,998	(71%)	1,650	(58%)
Eclipse-JDT	3,144	2,709	(86%)	2,495	(79%)	2,141	(68%)
Mylyn	3,437	1,229	(36%)	2,346	(68%)	1,102	(32%)
Rhino	560	160	(29%)	397	(71%)	154	(28%)

Results (2/4)

- RQ2: antipatterns + design patterns and change-proneness

ArgoUML			Eclipse-JDT			Mylyn			Rhino		
Rel.	ORs APs	ORs APs+DPs	Rel.	ORs APs	ORs APs+DPs	Rel.	ORs APs	ORs APs+DPs	Rel.	ORs APs	ORs APs+DPs
0.10.1	14.17	2.08	1.0	1.42	1.50	2.0.0	14.17	9.16	1.4R3	10.41	7.12
0.12	7.16	1.91	2.0	0.72	0.62	2.1	10.89	5.82	1.5R1	17.98	11.37
0.14	5.36	2.33	2.1.1	2.46	2.81	2.2.0	11.10	6.68	1.5R2	17.37	15.00
0.15.6	97.44	22.78	2.1.2	0.89	0.98	2.3.0	9.83	5.52	1.5R3	15.71	8.63
0.16	15.91	4.47	2.1.3	1.88	1.91	2.3.1	7.66	4.61	1.5R4	27.04	16.07
0.17.5	19.81	5.09				2.3.2	24.38	14.95	1.5R5	15.51	8.55
0.18.1	8.60	4.01				3.0.0	9.45	5.94	1.6R1	24.73	13.87
0.19.8	11.45	3.72				3.0.1	9.85	5.63	1.6R2	12.69	9.53
0.20	26.54	12.27				3.0.2	5.31	3.41	1.6R3	19.95	15.85
						3.0.3	8.18	5.31	1.6R4	33.05	17.49
						3.0.4	3.77	2.27	1.6R5	19.97	13.98
						3.0.5	4.96	3.06	1.6R6	20.56	11.78
						3.1.0	10.53	8.39			

Results (3/4)

- RQ3: design patterns/antipatterns “love” relation

Rel.	Design Patterns	Antipatterns	DPs OR	APs OR	Int. OR
Design Patterns “Love” Antipatterns					
ArgoUML					
0.14	S.Concretestate	Blob	7.32	55.01	0.09
0.14	A.Adapter	MessageChain	3.61	9.32	0.12
0.18.1	D.Concretecomponent	Antisingleton	1.29	8.68	0.09
0.18.1	A.Adaptee	LargeClass	6.54	25.15	0.23
0.18.1	FM.ConcreteCreator	MessageChain	9.91	11.12	0.14
Eclipse-JDT					
2.1.1	D.Concretecomponent	LongMethod	1.89	3.14	0.54
2.1.1	FM.ConcreteProduct	MessageChain	1.65	3.41	0.52
2.1.2	C.Leaf	LPL	1.02	1.07	0.38
2.1.3	FM.product	AntiSingleton	0.63	2.20	0.20
2.1.3	Cmd.Concretecommand	LPL	1.01	2.05	0.47
2.1.3	S.Concretestate	MessageChain	0.58	1.81	1.56
Mylyn					
2.3.0	FM.ConcreteCreator	LongMethod	4.21	17.58	0.07
2.3.1	Visitor.Client	LPL	16.84	24.49	0.04
3.0.3	S.Concretestate	CDSBP	3.22	5.63	0.18
3.0.3	S.Context	LongMethod	3.85	10.91	0.19

Results (4/4)

- RQ3: design patterns/antipatterns “hate” relation

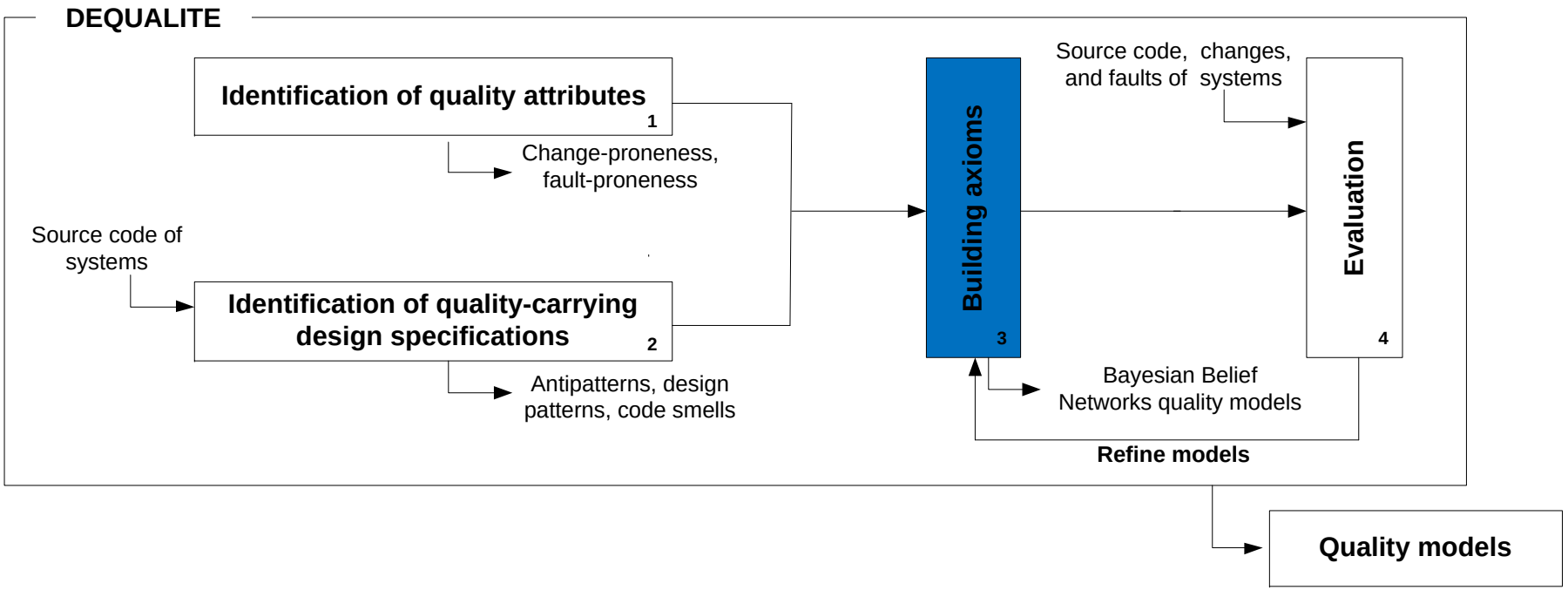
Design Patterns “Hate” Antipatterns					
ArgoUML					
0.14	Cmd.Concretecommand	RPB	3.74	1.60	11.02
Eclipse-JDT					
1.0	S.Context	LazyClass	2.70	1.17	5.54
1.0	FM.ConcreteProduct	LPL	1.95	1.18	4.53
2.0	Visitor.Client	MessageChain	1.04	0.59	19.02
2.1.1	S.Concretestate	ComplexClass	2.29	3.86	4.58
2.1.2	C.Leaf	ComplexClass	0.66	2.19	4.41
2.1.2	FM.ConcreteProduct	LazyClass	1.78	0.43	3.29
2.1.2	O.subject	LazyClass	3.32	0.66	5.43
2.1.2	S.Concretestate	LazyClass	1.48	0.39	2.38
2.1.2	Cmd.Concretecommand	LazyClass	1.60	0.47	1.74
2.1.2	Visitor.Client	LongMethod	0.84	2.01	6.97
2.1.2	O.subject	LPL	2.69	0.93	3.93
2.1.2	Visitor.Client	MessageChain	2.25	1.56	3.00
2.1.2	C.Leaf	MessageChain	0.39	2.30	4.31
2.1.3	S.Concretestate	AntiSingleton	0.78	1.66	1.93
2.1.3	FM.ConcreteCreator	CDSBP	1.97	0.53	2.46
2.1.3	C.Leaf	MessageChain	0.35	1.66	6.39
2.1.3	P.Concreteprototype	RPB	5.92	0.42	13.03

Summary on AP—DP Relation

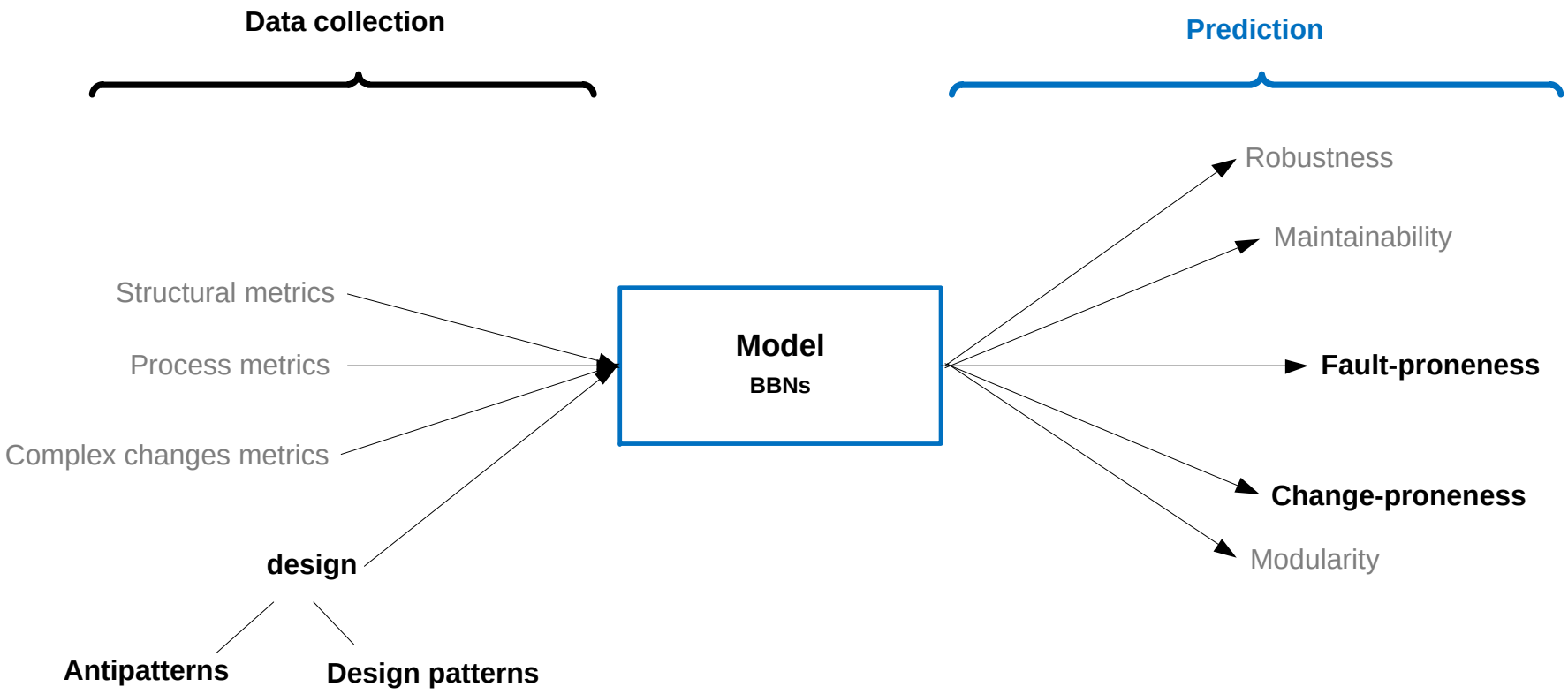
- The percentages of classes that participate in co-occurrences of antipatterns and design patterns range between 28% and 68%
- In all systems but Eclipse-JDT, class change-proneness odds ratios significantly decrease for classes participating in both antipatterns and design patterns with respect to classes participating in antipatterns only
- When a class is properly designed using some design patterns, even if it participates in (or decays towards) antipatterns, the negative effect of the antipatterns is mitigated by the robustness from the design patterns

Outline

- Introduction
- Related Work and Contributions
- Experimentations
- Quality Models and Implementation
- Threats to the Validity
- Conclusion and Future Work


DEQUALITE
Identification of quality attributes
1
Change-proneness,
fault-proneness
Source code of
systems
Identification of quality-carrying
design specifications
2
Antipatterns, design
patterns, code smells
Building axioms
3
Source code, changes,
and faults of systems
Evaluation
4
Bayesian Belief
Networks quality models
Refine models
Quality models

Building Quality Models (1/2)



Building Quality Models (2/2)

- Goal
  - Obtain prediction models to help developers determine where to focus their inspection efforts in systems
  - We use Bayesian Belief Networks (BBNs), which handle uncertainty

BBNs (1/4)

- 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

BBNs (2/4)

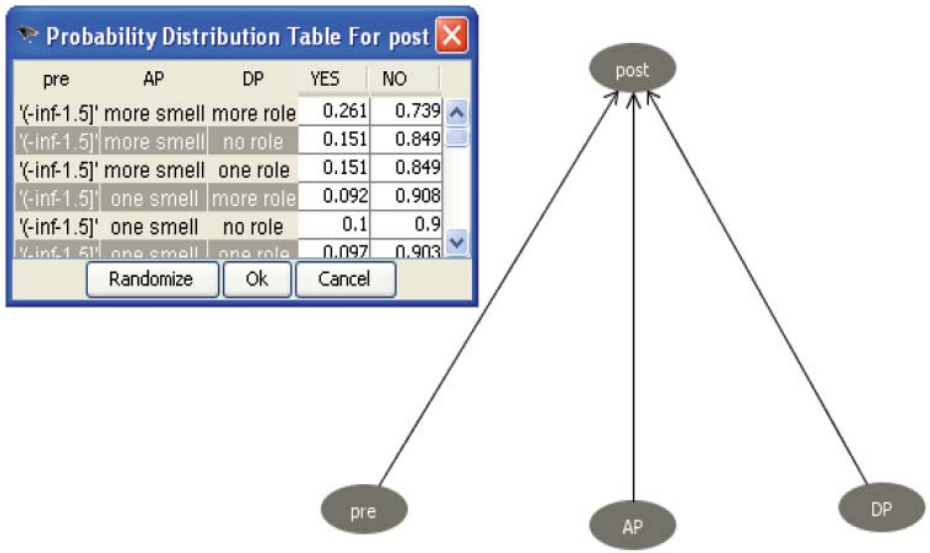
- Classifier
  - C_1 = {change-prone, not change-prone}
  - C_2 = {fault-prone, not fault-prone}
- Input vector describing a class
  - $<a_1, \ldots, a_n>$
  - $P(A|B) = P(B|A)\ P(A) / P(B)$

BBNs (3/4)

- Building a BBN
  - Define its structure

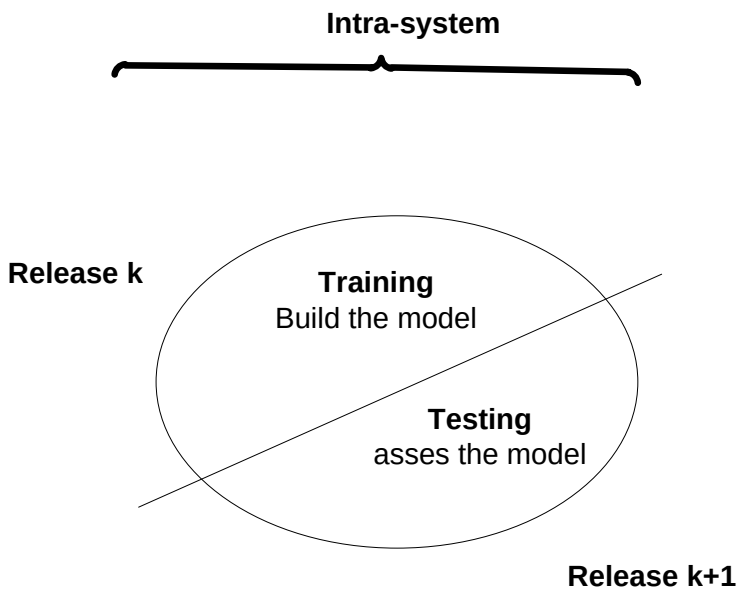
Input Nodes:
characterizations of the design of a class
- Number of roles played in a design patterns
- Number of antipatterns

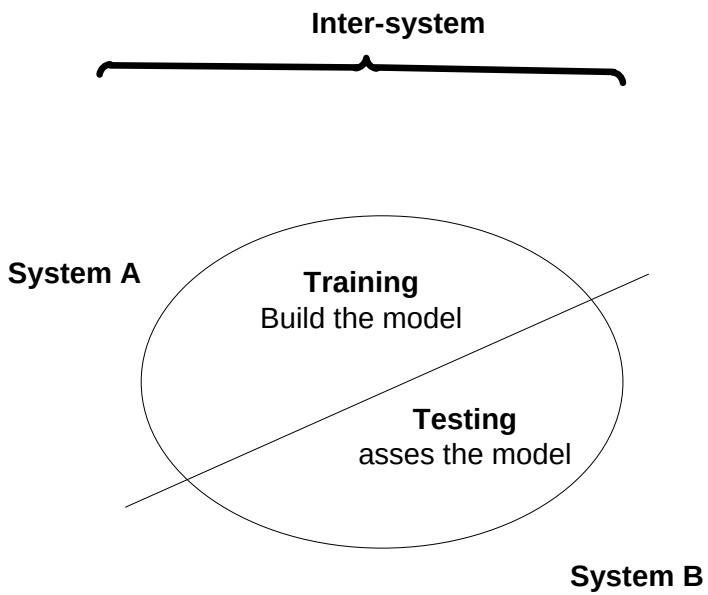
Output Nodes:
probability that the class is change-/fault- prone

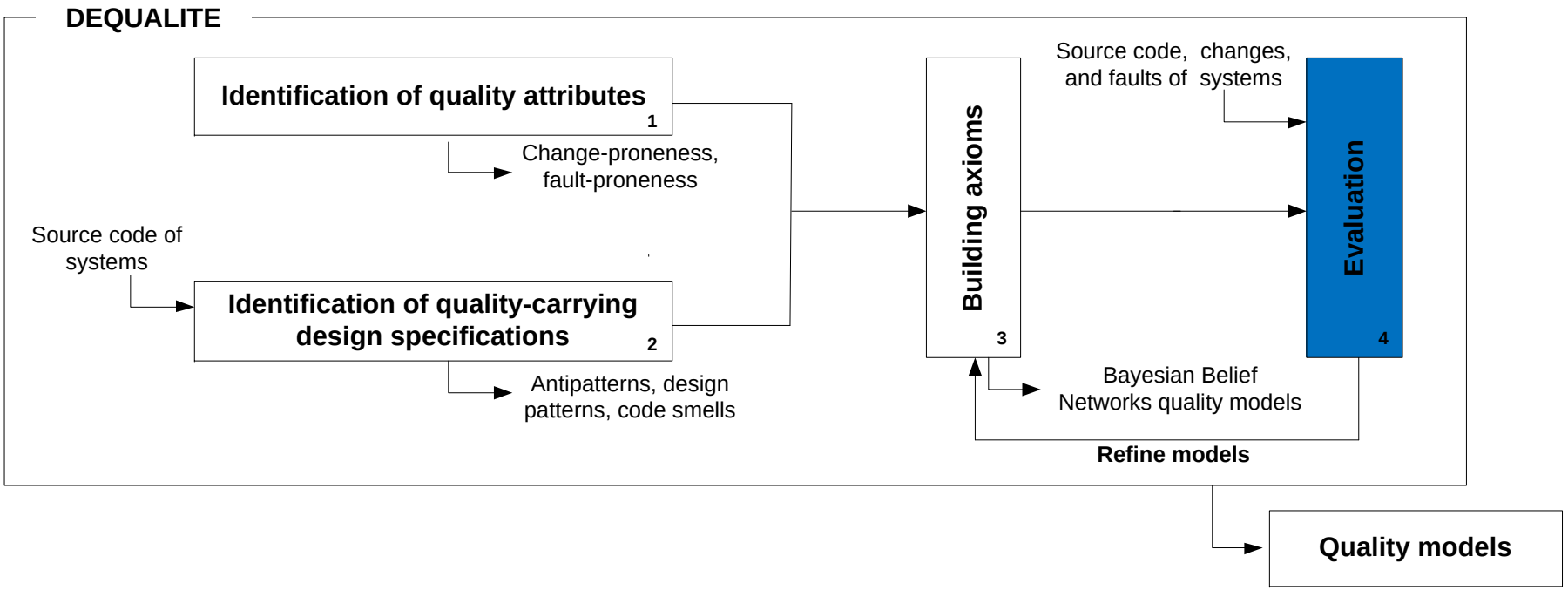


BBNs (4/4)

- Building a BBN
  - Assign/learn its probability tables






DEQUALITE
Identification of quality attributes
1
Change-proneness,
fault-proneness
Source code of
systems
Identification of quality-carrying
design specifications
2
Antipatterns, design
patterns, code smells
Building axioms
3
Source code, changes,
and faults of systems
Evaluation
4
Bayesian Belief
Networks quality models
Refine models
Quality models

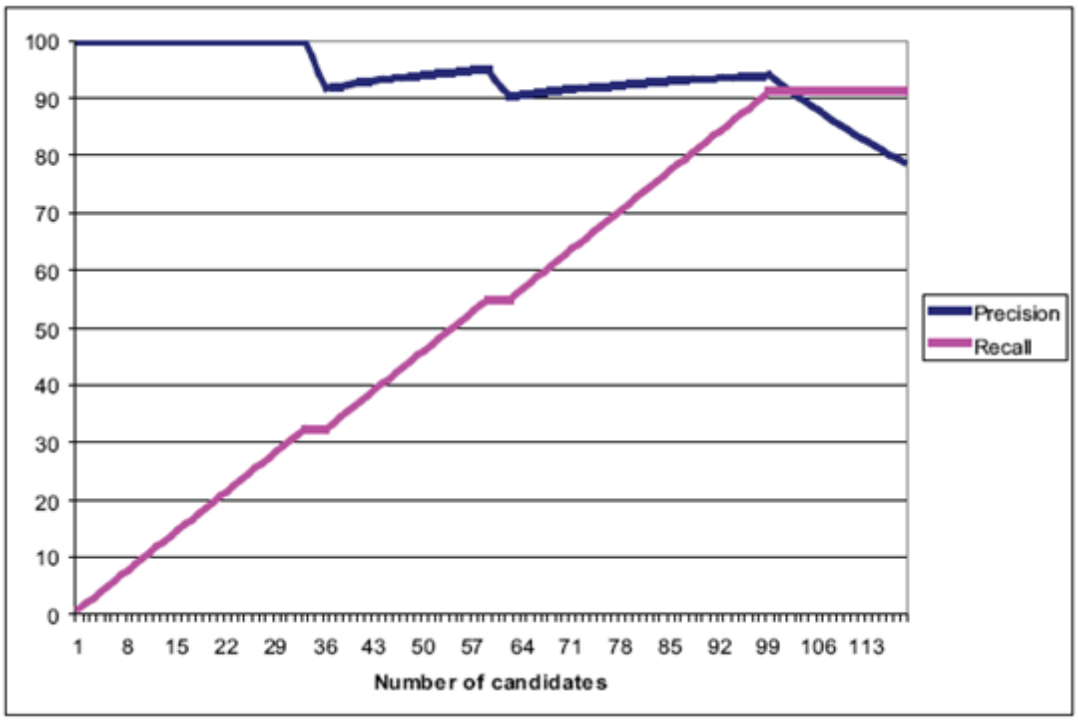
Evaluation (1/5)

- Research questions
  - RQ1: To what extent a BBN quality model built using our method is able to predict change/fault-prone classes in a system?
  - RQ2: Are the results of a BBN built using our method better than state-of-the-art prediction models with metrics?

Evaluation (2/5)

RQ1: precision/recall of BBNs (change-proneness)

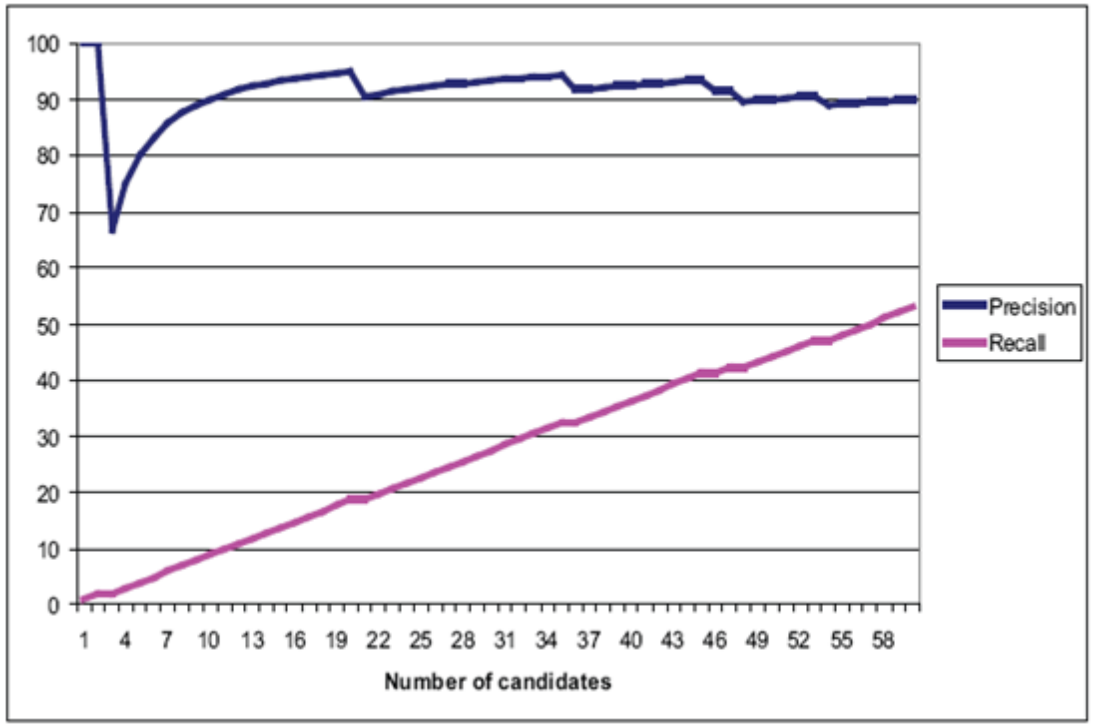
Intra-system (Rhino, Training: Rhino)



Evaluation (3/5)

RQ1: precision/recall of BBNs (change-proneness)

Inter-system (Rhino, Training: mylyn)



Evaluation (4/5)

- RQ2: Comparison with state-of the-art metrics models
  - Replication of Zimmermann's study
    - Logistic regression

Training	Testing	Metrics		Design		Metrics+Design	
		Precision	Recall	Precision	Recall	Precision	Recall
2.0	2.0	0.68	0.22	0.63	0.12	**0.71**	0.24
	2.1	0.42	0.25	**0.65**	0.13	**0.44**	0.26
2.1	2.1	0.61	0.16	**0.64**	0.14	**0.62**	0.17
	2.0	0.60	0.11	0.58	0.13	**0.62**	0.12

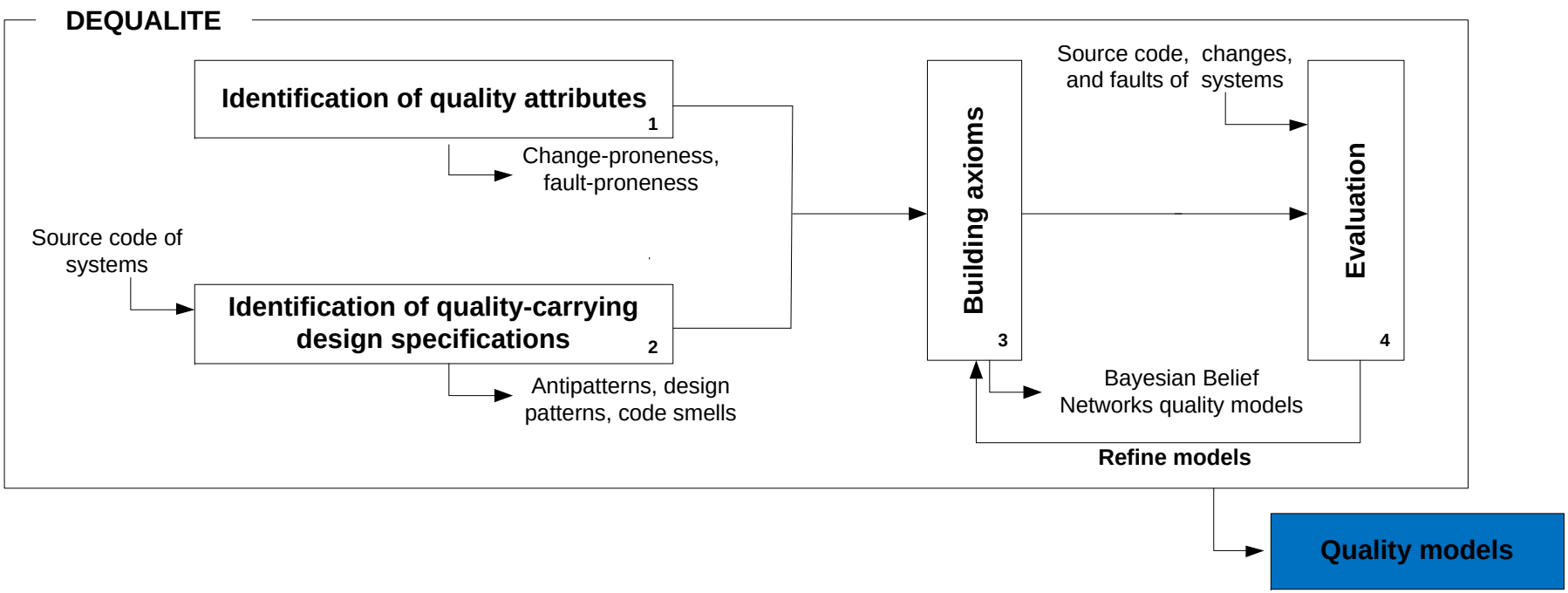
- A model taking into account the design of system have a better accuracy in predicting fault-prone classes than a model based on metrics solely

Evaluation (5/5)

- Bansiya's QMOOD model (Mylyn)
  - Among the top 20% of classes considered less reusable, less flexible, and less extensible by QMOOD:
    - 71% of them were change-prone classes ;
    - 98% of them were predicted as change-prone by the BBN with;
    - 69% of these classes being among the top 20% results of the BBN
  - Even though the BBN was not designed to measure the exact same attributes as QMOOD it can be almost as effective as QMOOD in detecting problematic classes in systems

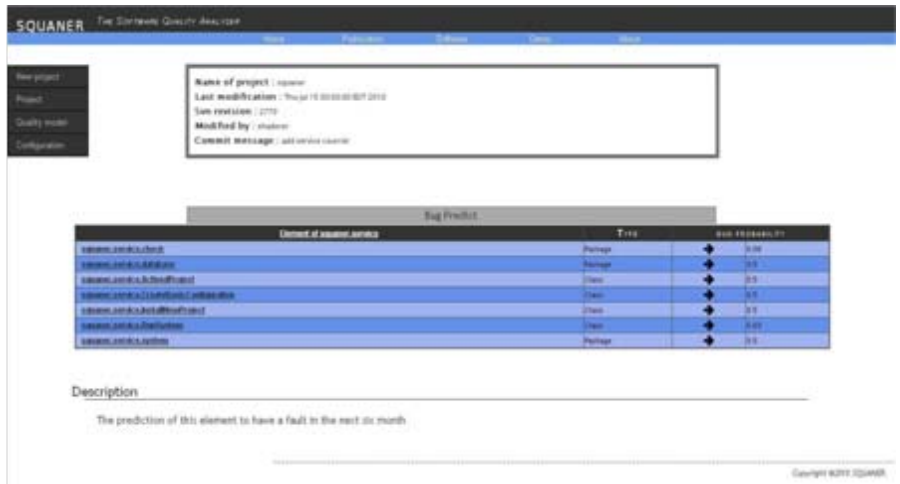
Summary on Quality Models

- BBNs built from DEQUALITE showed high precision and recall and a capability to assign high probabilities to candidate classes that are *indeed* change-prone
- BBNs obtained from DEQUALITE are in general equivalent or superior to these of a state-of-the-art model with metrics and that when BBNs are improved with metrics, their accuracy increase
- BBNs obtained from DEQUALITE could be as effective as QMOOD in detecting problematic classes in systems


DEQUALITE
Identification of quality attributes
1
Change-proneness, fault-proneness
Source code of systems
Identification of quality-carrying design specifications
2
Antipatterns, design patterns, code smells
Building axioms
3
Source code, changes, and faults of systems
Evaluation
4
Bayesian Belief Networks quality models
Refine models
Quality models

Implementation: SQUANER

- The quality models developed in this research are available online in our portal, SQUANER at: http://www.squaner.khomh.net/



Outline

- Introduction
- Related Work and Contributions
- Experimentations
- Quality Models and Implementation
- Threats to the Validity
- Conclusion and Future Work

Threats to the Validity

- Construct validity: relation between theory and observation
  - Manually validated instances of motifs
- Internal validity: causal inferences
  - No claim of causation, only relation
- Conclusion validity: relation between the treatment and the outcome
  - Statistic tests properly used
- Reliability validity: possibility of replicating this study
  - Details for replication available at: http://khomh.net/experiments/thesis/
- External validity: possibility to generalise our results
  - Generalisation requires further studies

Outline

- Introduction
- Related Work and Contributions
- Experimentations
- Quality Models and Implementation
- Threats to the Validity
- Conclusion and Future Work

Conclusion (1/3)

- Quality models built with DEQUALITE achieve high precision and recall in predicting change-prone classes
- Results are in general equivalent or superior to these of state-of-the-art models with metrics when predicting fault-prone classes
- The accuracy of fault-proneness models built with DEQUALITE increases when they are improved with new information on systems, like class sizes

Conclusion (2/3)

- Contrary to quality models, DEQUALITE BBNs-based model, provides in addition to the probability that a class is of bad quality,
  - The list of design patterns on the class
  - The list of antipatterns on the class

Conclusion (3/3)

"By considering system design; in particular the presence of design patterns and antipatterns, it is possible to build better quality models than simply by considering the internal attributes of classes"

We have provided:

- Quantitative evidence that design patterns and antipatterns have an impact on the quality of systems
- And that taking them into account improve prediction

Thus proving our thesis

Lessons Learned (1/2)

- Tangled implementations of design patterns exist and significantly affect the structure of classes
  - A particular attention should be paid to classes playing roles in design motifs; in particular classes playing two roles

Lessons Learned (2/2)

- Classes participating in antipatterns are significantly more likely to be subject to changes and to be involved in fault-fixing changes than other classes
  - MessageChains, a violation of the Law of Demeter, are consistently related to more changes and faults
- A not negligible percentage of classes participate in co-occurrences of antipatterns and design patterns in systems
  - Design patterns have a positive effect in mitigating antipatterns

Future Work (1/2)

- Extend DEQUALITE to include new sources of information on systems, like source code identifiers
- Extend DEQUALITE to assess more subjective quality attributes like understandability
  - We are currently performing a series of controlled experiments to analyse the effect of various antipatterns on the understandability of systems

Future Work (2/2)

- Study the usability of a quality model in a software development environment
- Replicate our study to build quality models for multi-language systems
- Replicate our study to control for faults when studying changes, and for changes when studying faults

Publications (1/4)

- Articles in journals
  1. Foutse Khomh, Massimiliano Di Penta, Yann-Gaël Guéhéneuc, and Giuliano Antoniol, (2010) An Exploratory Study of the Impact of Antipatterns on Class Change- and Fault-Proneness, Journal of Empirical Software Engineering (EMSE) (under revision).
  2. Foutse Khomh, Stéphane Vaucher, Yann-Gaël Guéhéneuc, and Houari Sahraoui, (2010) BDTEX: A GQM-based Bayesian Approach for the Detection of Antipatterns, Journal of Systems and Software (JSS) (under revision).

- Book chapter
  1. Foutse Khomh and Yann-Gaël Guéhéneuc, (2010) Construction de modèles de qualité prenant en compte la conception des systèmes et présentation d'un tel modèle de qualité, Evolution et Rénovation des Systèmes Logiciels , Hermes, (To appear)

- Conference articles
  1. Nicolas Haderer, Foutse Khomh, and Giuliano Antoniol, SQUANER: A Framework for Monitoring the Quality of Software Systems, Proceedings of the 26th IEEE International Conference on Software Maintenance (ICSM'10), Tool Demonstrations track, September 12-18, 2010, Timișoara, Romania. IEEE Computer Society Press.
  2. Salima Hassaine, Foutse Khomh, Yann-Gaël Guéhéneuc, and Sylvie Hamel (2010) IDS: An Immunology-inspired Approach for the Detection of Software Design Smells, In Proceedings of the Quality in Reengineering and Refactoring track at the 7th International Conference on the Quality of Information and Communications Technology (QUATIC).

Publications (2/4)

3. Rocco Oliveto, Foutse Khomh, Giuliano Antoniol, and Yann-Gaël Guéhéneuc (2010) Numerical Signatures of Antipatterns: An Approach based on B-Splines, In Proceedings of the 14th European Conference on Software Maintenance and Reengineering (CSMR).

4. Foutse Khomh, Massimiliano Di Penta and Yann-Gaël Guéhéneuc, (2009) An Exploratory Study of the Impact of Code Smells on Software Change-proneness, In Proceedings of the 16th Working Conference on Reverse Engineering (WCRE), October 13-16, Lille, France. IEEE Computer Society Press.

5. Stéphane Vaucher, Foutse Khomh, Naouel Moha and Yann-Gaël Guéhéneuc, (2009) Tracking Design Smells: Lessons from a Study of God Classes, In Proceedings of the 16th Working Conference on Reverse Engineering (WCRE), October 13-16, Lille, France. IEEE Computer Society Press.

6. Foutse Khomh, Yann-Gaël Guéhéneuc, and Giuliano Antoniol, (2009) Playing Roles in Design Patterns: An Empirical Descriptive and Analytic Study, In Proceedings of the 25th IEEE International Conference on Software Maintenance (ICSM), September 20-26, Edmonton, Alberta, Canada. IEEE Computer Society Press.

7. Foutse Khomh, Stéphane Vaucher, Yann-Gaël Guéhéneuc, and Houari Sahraoui, (2009) A Bayesian Approach for the Detection of Code and Design Smells, In Proceedings of the 9th International Conference on Quality Software (QSIC), August 24-25, Jeju, Korea. IEEE Computer Society Press.

8. Foutse Khomh, Yann-Gael Gueheneuc, (2008) Do Design Patterns Impact Software Quality Positively?, In Proceedings of the 12th European Conference on Software Maintenance and Reengineering (CSMR), du 1-4 avril, Athènes, Grèce. IEEE Computer Society Press.

Publications (3/4)

9. Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, Yann-Gaël Guéhéneuc, (2008) Is it a Bug or an Enhancement? A Text-based Approach to Classify Change Requests, In Proceedings of the 18th IBM Centers for Advanced Studies Conference (CASCON), Toronto, CA, October 27 - 30. ACM Press.

10. Naouel Moha, Foutse Khomh, Yann-Gaël Guéhéneuc, (2008) Génération automatique d'algorithmes de détection des défauts de conception, In Proceedings of the 14ème Colloque International sur les Langages et Modèles à Objet (LMO), du 2 -7 mars, Montréal, Quebec, Canada. Éditions Cépaduès.

11. Foutse Khomh, (2009) SQUAD: Software Quality Understanding through the Analysis of Design, Doctoral Symposium, 16th Working Conference on Reverse Engineering (WCRE), October 13-16, Lille, France. IEEE Computer Society Press.

12. Foutse Khomh, Yann-Gaël Guéhéneuc, (2008) DEQUALITE: Building Design-based Software Quality Models, In Proceedings of the 2nd PLoP Workshop on Software Patterns and Quality (SPAQu), October 18-20, Nashville, Tennessee, USA. ACM Press.

13. Foutse Khomh, Yann-Gael Gueheneuc, (2007) Perception and Reality: What are Design Patterns Good For? In Proceedings of the 11th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE), July 31st, Berlin, Germany. Springer-Verlag.

Publications (4/4)

- Posters and tools
  1. Foutse Khomh, (2009) SQUAD: Software Quality Understanding through the Analysis of Design, Consortium for Software Engineering Research (CSER), April 26-27, Montréal, Canada.
  2. Yann-Gaël Guéhéneuc, Janice Ka-Yee Ng, Duc-Loc Huynh, Foutse Khomh, (2006) Ptidej: A Tool Suite, IBM CASCON, Oct, 2006, Toronto, Canada.

- Technical reports
  1. Foutse Khomh, Massimiliano Di Penta and Yann-Gaël Guéhéneuc, (2009) An Exploratory Study of the Impact of Code Smells on Software Change-proneness,Technical report, Ecole Polytechnique de Montréal.
  2. Foutse Khomh, Massimiliano Di Penta, Yann-Gaël Guéhéneuc and Guiliano Antoniol, (2009) An Exploratory Study of the Impact of Antipatterns on Software Changeability, Technical report EPM-RT-2009-02, Ecole Polytechnique de Montréal.
  3. Foutse Khomh, Yann-Gaäel Guéhéneuc and Giuliano Antoniol, (2009) Playing Roles in Design Patterns: An Empirical Descriptive and Analytic Study, Technical report EPM-RT-2009-03, Ecole Polytechnique de Montréal.
  4. Foutse Khomh, Naouel Moha and Yann-Gaël Guéhéneuc, (2009) DEQUALITE : méthode de construction de modèles de qualité prenant en compte la conception des systèmes, Technical report EPM-RT-2009-04, Ecole Polytechnique de Montréal.
  5. Simon Denier, Foutse Khomh, and Yann-Gael Guéhéneuc, (2008) Reverse-Engineering the Literature on Design Patterns and Reverse-Engineering, Technical report EPM-RT-2008-09, Ecole Polytechnique de Montréal.
  6. Foutse Khomh and Yann-Gael Guéhéneuc, (2008) An Empirical Study of Design Patterns and Software Quality, Technical report 1315, University of Montréal.

Questions

Thank you for listening

References

- Boehm [1976] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2nd international conference on Software engineering*, pages 592-605. IEEE Computer Society Press, 1976.McCall et al. [1977] J. A. McCall, P. K. Richards, and G. F Walters. Factors in software quality. In Nat'l Tech. Information Service, editor, *Nat'l Tech. Information Service*, 1, 2 and 3, 1977.
- ISO 9126 [1991] ISO 9126. *Information Technology-Software Product Evaluation-Quality Characteristics and Guidelines for their Use*. ISO/IEC, December 1991. ISO/IEC 9126:1991(E)
- Dromey [1995] R. Geoff Dromey. A model for software product quality. *IEEE Transactions on Software Engineering*, 21(2):146-162. IEEE, february 1995.
- Bansiya and Davis [2002] Jagdish Bansiya and Carl G. Davis. A hierarchical model for object-oriented design quality assessment. In IEEE CS Press, editor, *IEEE Trans. on Software Engineering*, 28:4-17, Jan. 2002.
- [Moha et al., 2009] Naouel Moha, Yann-Gaeel Gueheneuc, Laurence Duchien, and Anne-Francoise Le Meur. DECOR: A method for the specification and detection of code and design smells. In Mark Harman, editor, *Transactions on Software Engineering (TSE)*. IEEE Computer Society Press, 2009.
- [Guéhéneuc and Antoniol, 2008] Yann-Gael Guéhéneuc and Giuliano Antoniol. DeMIMA: A multi-layered framework for design pattern identification. In Sebastian Elbaum and David S. Rosenblum, editors, *Transactions on Software Engineering (TSE)*, 34(5):667-684. IEEE Computer Society Press, September 2008. 18 pages.
- [Gamma *et al.*, 1994] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns-Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1*st* edition, 1994. isbn: 0-201-63361-2.
- [Briand and WÄust, 2002] Lionel C. Briand and JÄurgen WÄust. Empirical studies of quality models in object-oriented systems. In Marvin Zelkowitz, editor, *Advances in Computers*. Academic Press, 2002.
- [Brown *et al.*, 1998] William J. Brown, Raphael C. Malveau, William H. Brown, Hays W. McCormick III, and Thomas J. Mowbray. *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley and Sons, 1*st* edition, March 1998. isbn: 0-471-19713-0.
- [Ignatios *et al.*, 2003] Deligiannis Ignatios, Stamelos Ioannis, Angelis Lefteris, Roumeliotis Manos, and Shepperd Martin. A controlled experiment investigation of an object oriented design heuristic for maintainability. In Elsevier, editor, *Journal of Systems and Software*, 65(2). Elsevier, February 2003.
- [Bois *et al.*, 2006] Bart Du Bois, Serge Demeyer, Jan Verelst, Tom Mens, and Marijn Temmerman. Does god class decomposition affect comprehensibility? In *Proceedings of the IASTED Inter national Conference on Software Engineering*, pages 346{355. IASTED/ACTA Press,2006.