Design Smells in the Context of Agile Software Development

Abdou Maiga

DIRO, Université de Montréal, Québec, Canada
Ph.D. Defense

Nov. 2012

Context

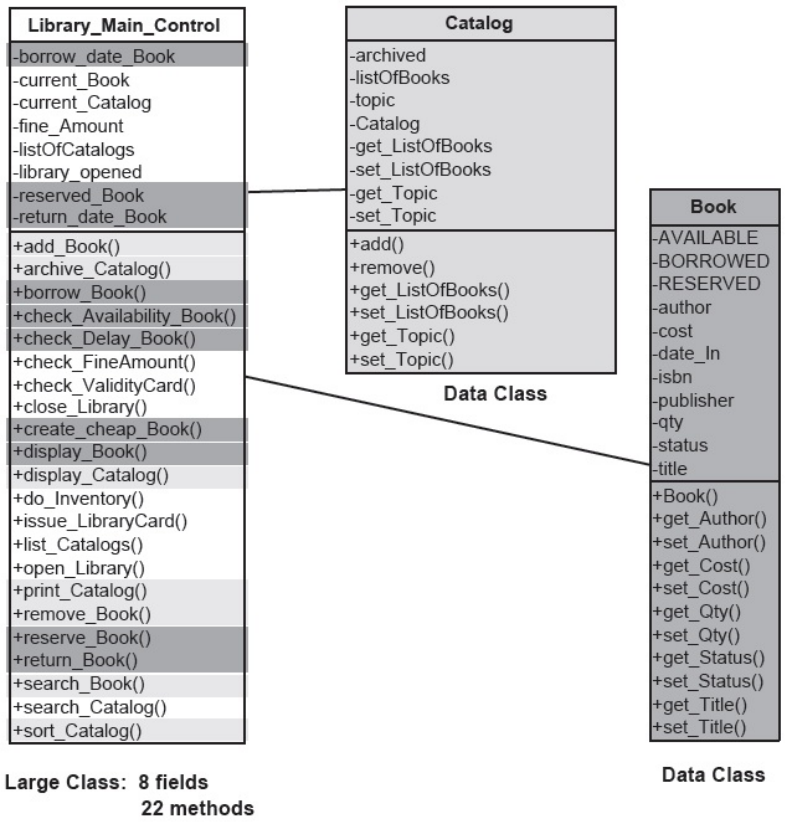
Agile Software Development (ASD) Context

- ASD is a set of collaborative, iterative, and incremental development methods [Beck, 2001]
- In general we have inadequation between documentation and source code

$\Rightarrow$

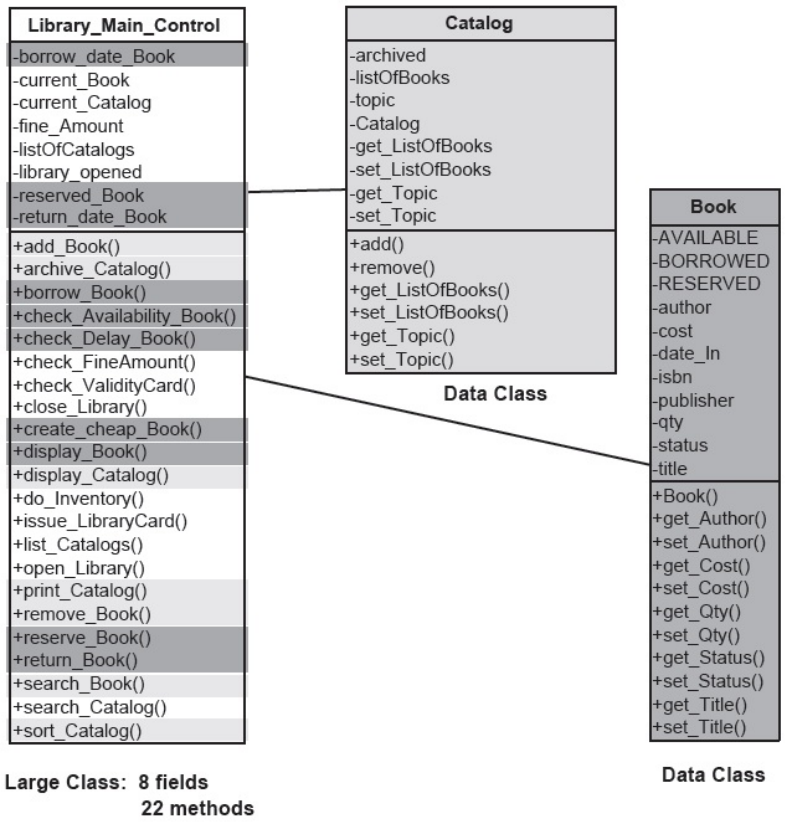
- The source code is the most important source of information to evolve, change and maintain the system.
- Maintenance activities have become very expensive. They reach more than 70% of the overall costs of software development [Pressman, 2001]

Context



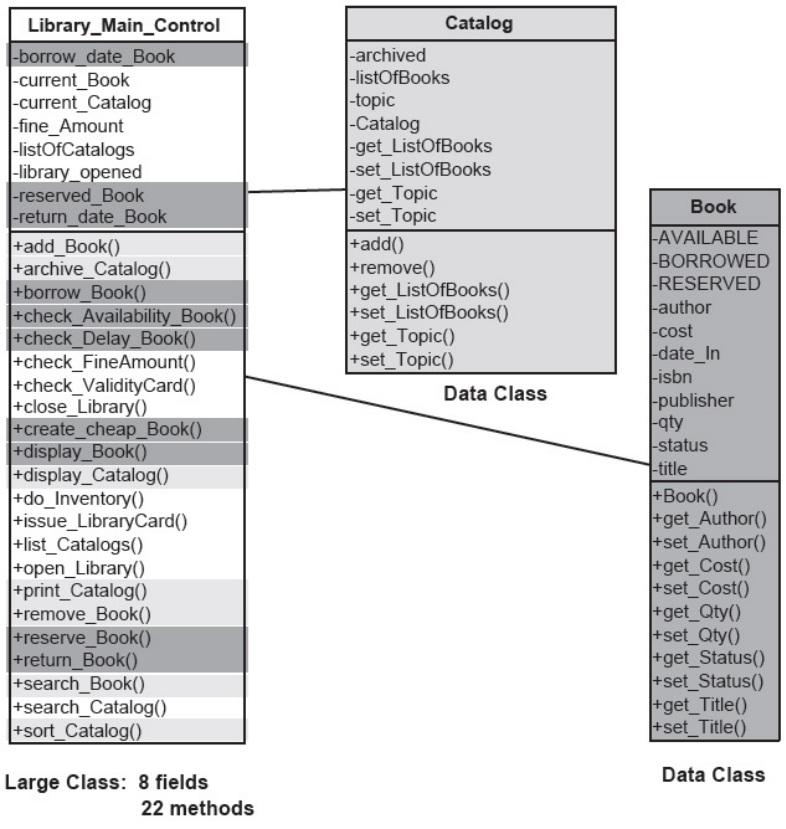
- Procedural thinking in object oriented language

Context



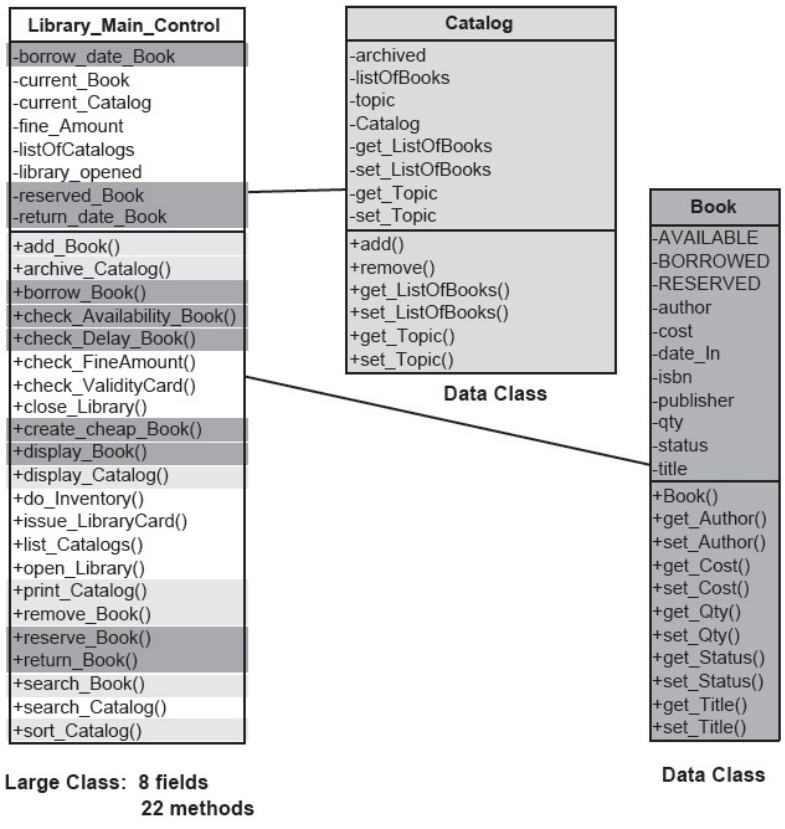
- Procedural thinking in object oriented language
- Impact of this design on comprehension?

Context



- Procedural thinking in object oriented language
- Impact of this design on comprehension?
- Impact of this design on fault-fixing activities?

Context



- Procedural thinking in object oriented language
- Impact of this design on comprehension?
- Impact of this design on fault-fixing activities?
- How to accurately detect this design in ASD context?

Outline

Outline

Design Smells in Software Systems

Design smells are poor solutions to recurring design or implementation problems [Webster, 1995]

- They are generally the result of misuse of the object-oriented paradigm and–or design patterns [Brown, 1998]

Design smells are present in sofware systems due to

- The time-to-market
- The lack of understanding
- The developers' experience

Developers cannot always follow standard designing and coding techniques

Motivation

Quantitative evidences on the relation between design smells, comprehension, and fault-fixing activities

- are important for practitioners to help developers reduced change costs
- help to reduce maintenance costs and improve software quality
- help practitioners to take rational decisions about how dealing with design smells
- help practitioners justify the removal of design smells

$\Rightarrow$

Empirical software engineering thesis

Thesis

There is an **impact** of **design smells** on source code **comprehension** and **fault-fixing activities** and we can provide a **tool** for **accurate incremental** design smells **detection** in ASD context.

Outline

Related Works – Design Smells Impact

Work	Limitations
[Deligiannis, 2003] Blob affect evolution of design structures and use of inheritance	No evidence of impact of DS on comprehension
[Du Bois, 2006] Decomposition of God classes into collaborating classes using refactorings can improve comprehension	No evidence of impact of DS on comprehension
[Olbrich, 2009] Blob and Shotgun Surgery are more change-prone than others	No evidence of impact of DS on Fault-fixing activities
[Abbes, 2011] Blob, Spaghetti Code design smells impedes developers' performance	Could not find any impact from one occurence of blob or spaguetti code
[Khomh 2012] Design smell classes are more change-prone and fault-prone than others	No link betwen fault-fixing activities and design smells

Related Work – Limitations

Design Smells Impact

- No evidence of the impact of DS on comprehension
  - $\Rightarrow$ Controlled experiments on impact of DS on comprehension

Related Work – Limitations

Design Smells Impact

- No evidence of the impact of DS on comprehension
  - ⇒ Controlled experiments on impact of DS on comprehension
- No impact found for single occurrence of DS on comprehension
  - ⇒ Experiments with combination of DS

Related Work – Limitations

Design Smells Impact

- No evidence of the impact of DS on comprehension
  - ⇒ Controlled experiments on impact of DS on comprehension
- No impact found for single occurrence of DS on comprehension
  - ⇒ Experiments with combination of DS
- No evidence of impact of DS on fault-fixing activities
  - ⇒ Empirical study on the impact of DS on fault-fixing activities

Related Work – Design Smells Detection

Approach	Limitations
[Travassos, 99] Manual inspection	Not applicable on large systems Time and money consuming
[Marinescu, 2004] Detection strategies based on metrics	Validated on one system Threshold problem
[Munro, 2005] Heuristics based on metrics	Specific to code smells
[Langelier, 2005], [Dhambri , 2008] Visualization approach	High human intervention
[AliKacem, 2006] Rules based on metrics detect violations quality principles	Not directly applicable to design smells detection Threshold problem No process of DS detection
[Moha, 2010] Based on set of metrics rules and relations between classes	Definition of rules cards are manual Extensive knowledge Threshold problem No feedback
[Khomh, 2011] Probabilistic approach based on Bayesian Belief Networks	Extensive knowledge Probabilistic approach Threshold problem

Related Work – Limitations

Design Smells Detection

The previous approaches are mostly based on the use of code/design quality metrics and thresholds to identify DS.

- They require extensive knowledge of DS
  $\Rightarrow$ Machine learning technique: SVM

Related Work – Limitations

Design Smells Detection

The previous approaches are mostly based on the use of code/design quality metrics and thresholds to identify DS.

- They require extensive knowledge of DS
  - $\Rightarrow$ Machine learning technique: SVM
- The choice of thresholds
  - $\Rightarrow$ Derived automatically from the application of SVM

Related Work – Limitations

Design Smells Detection

The previous approaches are mostly based on the use of code/design quality metrics and thresholds to identify DS.

- They require extensive knowledge of DS
  - ⇒ Machine learning technique: SVM
- The choice of thresholds
  - ⇒ Derived automatically from the application of SVM
- They did not integrate the subjectivity of the practitioners
  - ⇒ Integrating user's feedback

Contributions

1. Quantitative evidences of impact of DS on source code comprehension of systems
   - ⇒ Help developers to take rational decisions about their design quality
2. Quantitative evidences of impact of DS on fault-fixing activities
   - ⇒ Help developers to deal with DS
3. Incremental detection approach based on SVM and using users' feedback
   - ⇒ Make developers proactive and help to reduce maintenance costs

Outline

Context

- Comprehension: central element for the effectiveness of software maintenance and evolution
- Design smells are conjectured to negatively impact comprehension

$\Rightarrow$

Following the design in [Abbes, 2011], we conducted two experiments to assess the impact of co-occurrences of DS

- In [Abbes, 2011], single occurrence of DS did not impact comprehension but co-occurrences of Blob and Spaguetti Code impact comprehension
- We conducted a quasi quasi-replication to analyse whether co-occurrences of same DS, Blob or Spaghetti Code, in a single system, impact comprehension.

Study Design

Our design is like [Abbes, 2011]: 2∗3 factorial design

- Three different systems (per experiment), each with two possibilities (with or without co-occurrences of DS)
- For each combination, we prepare a set of comprehension questions, making up treatments.

Research Questions

RQ1	What is the impact of co-occurrences of Blob on code source comprehension?
RQ2	What is the impact of co-occurrences of Spaghetti Code on code source comprehension?

Hypothesis

For RQs, we formulate following null hypothesis

- $H_{01_{2Blobs}}$: No statistically significant difference between the subjects' average correct answers
- $H_{02_{2Blobs}}$: No statistically significant difference between the subjects' average time spent
- $H_{03_{2Blobs}}$: No statistically significant difference between the subjects' average effort

Objects of the Study

Experiments	Systems	Classes	SLOCs	Release
1	Azureus v2.3.0.6	1,449	191,963	2005
	iTrust v11.0	565	21,901	2010
	SIP v1.0	1,771	486,966	2010
2	ArgoUml v0.20	1,230	113,017	2006
	JHotDraw v5.4b2	484	72,312	2004
	Rhino v1.6R5	108	48,824	2009

Table : Object Systems

Subjects of the Study

- 59 anonymous subjects
- M.Sc. and Ph.D. students at ÉPM and UdeM
- M.Sc. and Ph.D. students at Carleton University
- Subjects were volunteers and could withdraw at any time, for any reason

Independent Variables

Experiment	Variable
1	*Has2b* : Boolean (presence or not) of co-occurences of Blob
2	*Has2SC* : Boolean (presence or not) of co-occurences of Spaghetti Code

Mitigating Variables

Three mitigating variables could possibly impact the measures of the dependent variables

- Subject's knowledge level in Java.
- Subject's knowledge level of Eclipse.
- Subject's knowledge level in software engineering.

Dependent Variables

Dependent variables measure subjects' performance

- Subjects' effort using the NASA Task Load Index (TLX)
- Time spent to answer the comprehension questions
- Percentage of correct answers

Analysis Method

- Non-parametric Mann-Whitney test for RQ1 and RQ2
- Non-parametric effect size measure Cliff's d
- Kruskal-Wallis one-way analysis of variance to analyse the impact of the mitigating variables

Results

Table : Mann-Whitney tests and Cliff's *d* results for 2 Blob

	M.-W. *p*	Cliff's *d*
Time	$<$**0.01**	**0.87** (Large)
Correct Answers	$<$**0.01**	**0.53** (Large)
Effort	$<$**0.01**	**0.73** (Large)

Table : Mann-Whitney tests and Cliff's *d* results for 2 SC

	M.-W. *p*	Cliff's *d*
Time	$<$**0.01**	**0.93** (Large)
Correct Answers	$<$**0.01**	**0.93** (Large)
Effort	$<$**0.01**	**0.55** (Large)

Impact of Mitigating Variables

Kruskal-Wallis test shows no effect on the results of the mitigating variables

1. Java Knowledge
2. Eclipse Knowledge
3. Software Engineering Knowledge

Results

Subjects working with systems with co-occurrences of DS

- Put more effort in doing comprehension tasks
- Have few percentages of correct answers
- Spend more time

Co-occurrences of DS are sufficient to impact developer's comprehension of source code

Outline

Context

We know that

- Design smells are prevalent in systems
- Several studies have shown that design smells negatively impact maintenance

However, none of them empirically investigated the impact of design smells on fault-fixing activities

$\Rightarrow$

Gather quantitative evidence on the relation between design smells, faults, and the developers' effort to fix the faults

Research Questions

RQ1	Relation between number of DS in a class and its fault-proneness?
RQ2	Relation between duration of fixing period and the presence of DS?
RQ3	Relation between number of elements (fields and methods) impacted by fault-fixes and the presence of DS?
RQ4	Relation between entropy of fault-fixes and the presence of DS?
RQ5	Relation between fault-fixes and the number of occurrences of DS before and after fault-fixes?

Independent Variables

RQ1	*HasF*: (Boolean) presence or not of fault in a class
(RQ2,RQ3,RQ4)	$HasDS_t$: (Boolean) presence or not of DS in a fault t
RQ5	$DS_{t,b}$: Number of occurrences of DS in a fault t before the fault-fix

Dependent Variables

RQ1	DS_i: Number of occurrences of DS in class i
RQ2	D_t: Duration of fixing period (number of days between date of report of the fault day of the fix)
RQ3	Nbr_{Elts}: Number of elements (methods and fields) impacted by a fault-fix
RQ4	E_t: Entropy of fault-fix
RQ5	$DS_{t,a}$: Number of occurrences of DS in a fault t after the fault-fix

Analysis Methods

We use the following analysis methods:

- Non-parametric Mann-Whitney test
- Non-parametric effect size measure Cliff's d
- Multivariate regression analyses

Results

RQ	ArgoUML	Eclipse	Mylyn	Rhino
RQ1 (Fault-prone)	✓	✓	✓	✓
RQ2 (Duration)	x	✓	✓	x
RQ3 (Elements)	x	✓	✓	✓
RQ4 (Entropy)	✓	✓	✓	✓
RQ5 (Nbr DS)	✓	✓	✓	✓

Results

Quantitative evidence of impact of DS on fault-fixing activities

- Faulty classes have higher number of DS
- Faults involving DS take more time to be fixed and their fixes impact more fields and methods with higher entropy
- Fixing faults reduces the number of DS

$\Rightarrow$ Help developers to deal with DS

Outline

Context

We know from the study above that

- DS impact code source comprehension
- DS impact fault-fixing activities
- Automatically remove DS can improve the quality of source code and reduce maintenance cost

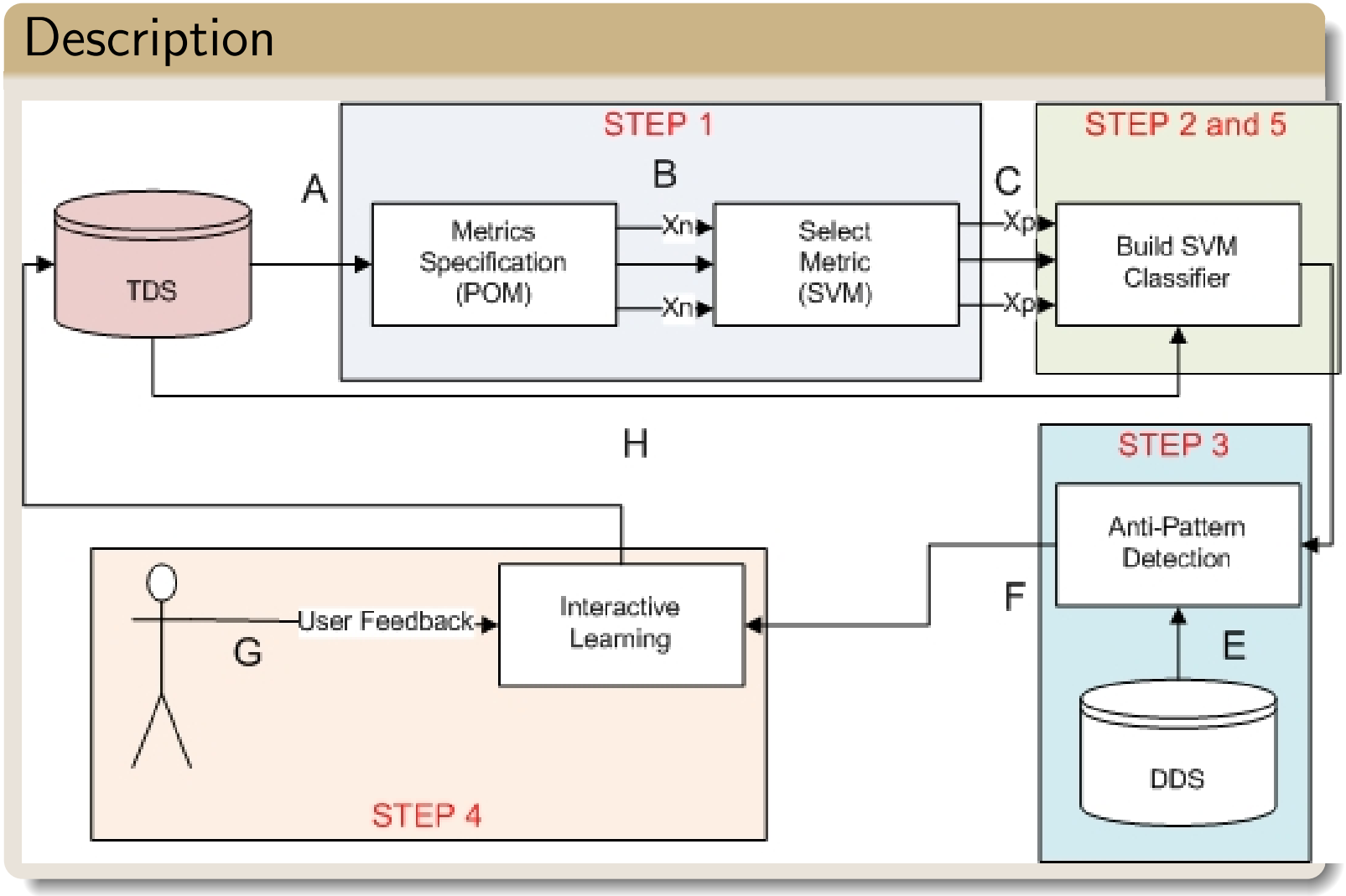
However previous approaches have some limitations

- They have **limited** precision and recall
- They require extensive knowledge of DS
- They cannot be applied on **subsets of systems**
- They did not integrate the subjectivity of the practitioners
- They are not **incremental**.

Contributions: SMURF

- SMURF approach to detect DS using SVM and practitioners' feedback
  - Classical learning systems with same precision as neural networks or higher
  - Can integrate users' feedback
  - SVM is robust to noises [Taylor and Cristianini, 2004;Rychetsky, 2001]

Description



Research Questions

RQ1,RQ2	How does the accuracy of **SMURF** compare with that of **DETEX** and **BDTEX**?
RQ3	How does the accuracy of SMURF change when used in **inter-system** configuration?
RQ4	How does the accuracy of SMURF change when using users' **feedback**?

Objects

Table : Description of the objects of the study

Names	Versions	# Lines of Code	# Classes	# Interfaces
ArgoUML	0.19.8	113,017	1,230	67
A design tool UML				
Azureus	2.3.0.6	191,963	1,449	546
A peer-to-peer client that implements the protocol BitTorrent				
Xerces	2.7.0	71,217	513	162
A syntaxic analyser				

Subjects

The subjects of our study are the following four DS

- Blob
- Functional Decomposition (FD)
- Spaghetti Code (SC)
- Swiss Army Knife (SAK)

We use these four DS because they well **known** and **previously studied** [Moha, 2010; Kohmh 2011]

Subsets of System: $RQ1_1$

Table : Precision of SMURF in subsets

DS	ArgoUML	Azureus	Xerces	Average
Blob	97.09	97.32	95.51	**96.64**
FD	70.68	72.01	66.93	**69.87**
SC	85.00	88.00	86.00	**86.33**
SAK	75.46	84.54	80.76	**80.25**

Table : Recall of SMURF in subsets

DS	ArgoUML	Azureus	Xerces	Average
Blob	84.09	91.33	95.29	**90.24**
FD	57.50	84.28	70.00	**70.59**
SC	71.00	89.00	86.00	**82.00**
SAK	77.14	85.71	75.50	**79.45**

Subsets of System: $RQ1_1$

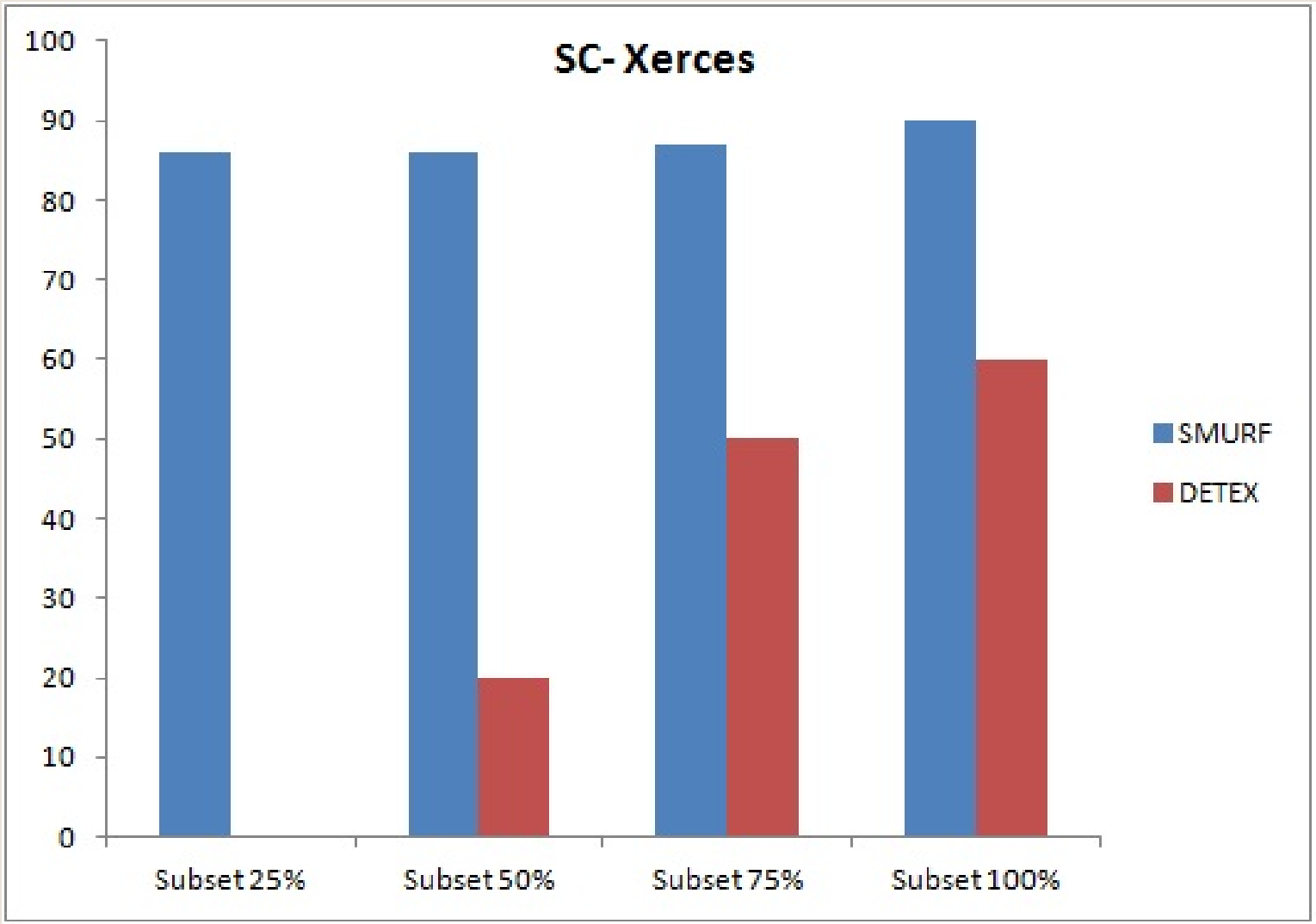


Figure : Trends of precision and recall when increasing the size of subset for SC in Xerces

Study Results

Complete System: $RQ1_2$

Table : Total recovered true occurrences of Blob by DETEX and SMURF

	DETEX	**SMURF**
ArgoUML	25	40
Azureus	38	48
Xerces	39	55
Total	102	**143**

- DETEX is not applicable on subsets of systems while SMURF has good precison and recall
- On entire systems, SMURF **detects more true occurrences** of Blob than DETEX

The accuracy of SMURF compared with that of BDTEX

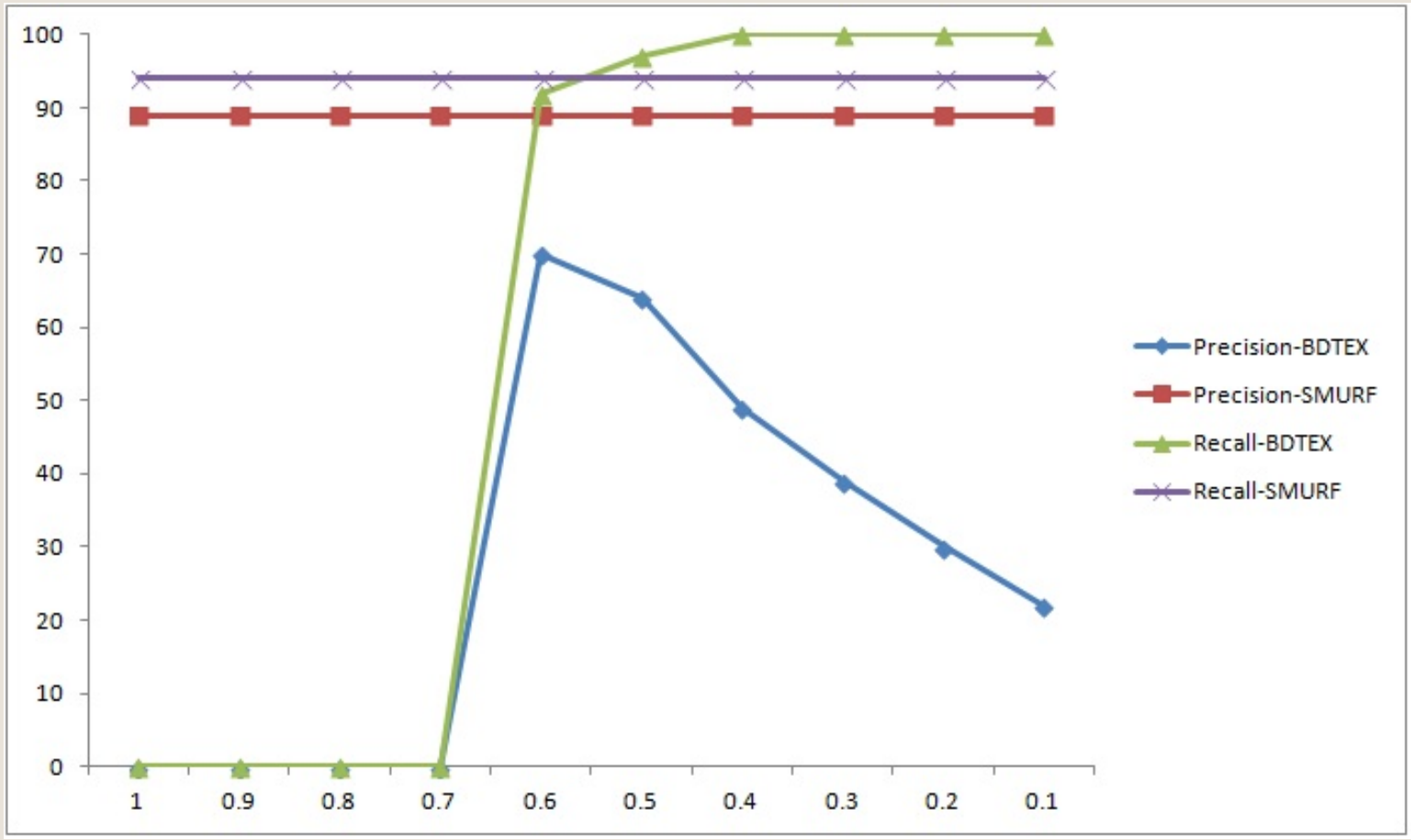


Figure : Trends in the increase of precision and recall when decreasing the probabilty of being a design smell for Blob in Xerces

The accuracy of SMURF in inter-systems configuration

Table : Precision of SMURF in inter-systems

	ArgoUML (%)	Azureus (%)	Xerces (%)
Blob	92.00	96.00	89.00
FD	57.00	62.00	36.00
SC	77.00	74.00	91.00
SAK	56.00	73.00	90.00

Table : Recall of SMURF in inter-systems

	ArgoUML (%)	Azureus (%)	Xerces (%)
Blob	62.00	48.00	94.00
FD	40.00	100.00	20.00
SC	96.00	88.00	91.00
SAK	68.00	84.00	56.00

The accuracy of SMURF with feedback

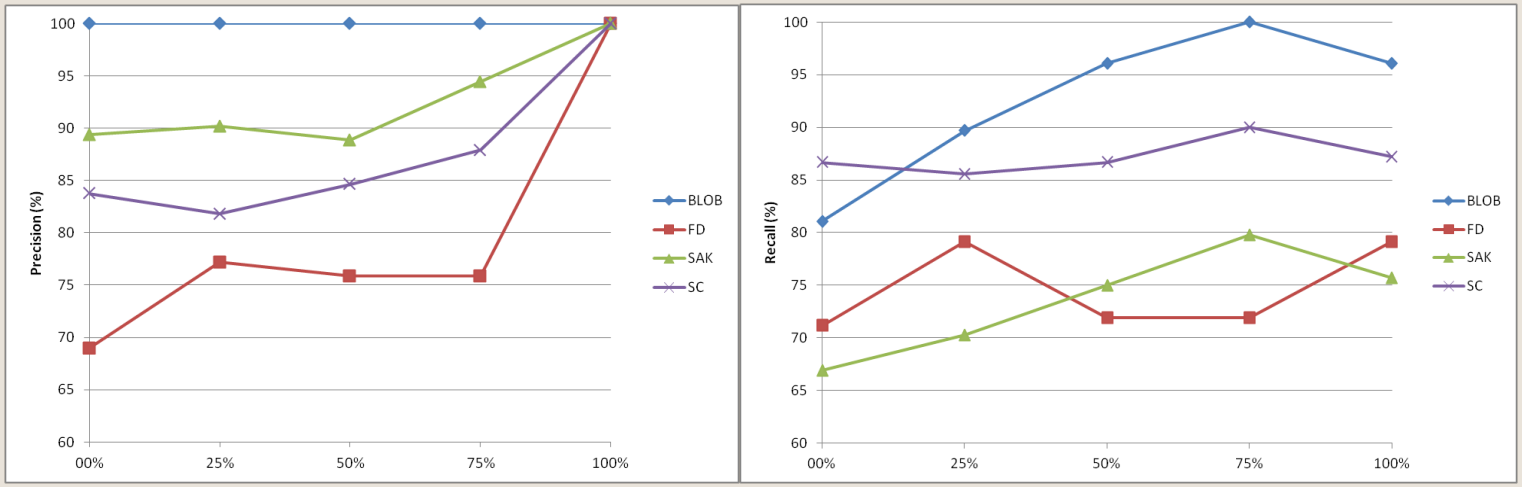
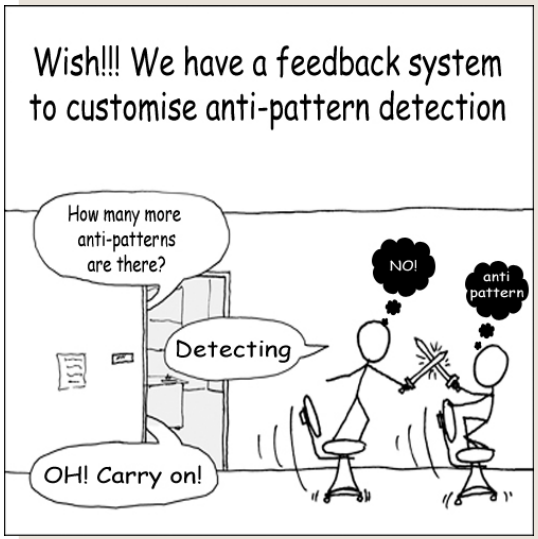


Figure : Trends in the increase of precision and recall when integrating incremental feedback

Results



Incremental detection approach based on SVM and using users' feedback

- We test SMURF on 3 systems and 4 DS
- The accuracy of SMURF is greater than that of DETEX
- SMURF is more stable than the probabilistic approach BDTEX
- SMURF is an incremental detection approach: could be applied in continuous integration context

$\Rightarrow$ Make developers proactive and help to reduce maintenance costs

Outline

Conclusion 1/3

1. Quantitative evidences of impact of DS on source code comprehension of systems
   - $\Rightarrow$ More time to understand system with co-occurrences of Blob or Spaguetti Code with large effect size
   - $\Rightarrow$ More effort to understand system with co-occurrences of Blob or Spaguetti Code with large effect size
   - $\Rightarrow$ Lower percentage of good answers when performing task on system with co-occurrences of Blob or Spaguetti Code with large effect size

Developers should take rational decisions about their design quality to avoid the negative impact of DS on code source comprehension

Conclusion 2/3

1. Quantitative evidences of impact of DS on fault-fixing activities
   - ⇒ The duration of the fixing period is longer for faults involving classes with DS
   - ⇒ Fixing faults in classes with DS impacts more files, more fields/methods with higher entropy
   - ⇒ After a fault is fixed, the number of occurrences of design smells in the classes involved in the fault decreases

Help developers to deal with DS when fixing a fault

Conclusion 3/3

1. Incremental detection approach based on SVM and using users' feedback
   - ⇒ The accuracy of SMURF is greater than that of DETEX and BDTEX
   - ⇒ SMURF can be applied in both intra-system and inter-system configurations
   - ⇒ SMURF accuracy improves when using practitioners' feedback

Make developers proactive and help to reduce maintenance costs

Short Term Perspectives

- Replicate the studies in other contexts, with other subjects, questions, kind of design smells, and other systems
- Investigate the relation between the number of occurrences of design smells and the level of comprehension
- Study the impact of design pattern on source code comprehension and fault-fixing activities
- Use our incremental detection tool, SMURF, in real-world environments

Long Term Perspectives

- How long does a design smell Survive?
- What are the factors of introduction and propagation of design smells in software system?
- What are the factors of extinction of design smells?
- How to automatically remove and propose refactoring for design smells?

Thesis

There is an **impact** of **design smells** on source code **comprehension** and **fault-fixing activities** and we can provide a **tool** for **accurate incremental** design smells **detection** in ASD context.

Contributions

1. Quantitative evidences of impact of DS on source code comprehension of systems
   - $\Rightarrow$ Help developers to take rational decisions about their design quality
2. Quantitative evidences of impact of DS on fault-fixing activities
   - $\Rightarrow$ Help developers to deal with DS
3. Incremental detection approach based on SVM and using users' feedback
   - $\Rightarrow$ Make developers proactive and help to reduce maintenance costs