Do Crosscutting Concerns Cause Defects?

M. Eaddy, T. Zimmermann, K. D. Sherwood, V. Garg,
G. C. Murphy, N. Nagappan, A. V. Aho.
IEEE Transactions on Software Engineering, 2008.

Presented by Danilo Ferreira e Silva

Motivation

Enormous effort goes into avoiding software defects

- Efforts might be better directed if we had a better understanding of their causes
- Empirical studies provide evidence that crosscutting concerns impact internal quality metrics

  (But, what about external quality?)

This study considers the possibility that **crosscutting concerns** impact one external quality: **defects**

Methodology

- Formal model to measure the extent to which concerns are crosscutting
- Three case studies to gather data on scattering and defect counts
- Correlation analysis between scattering and defects

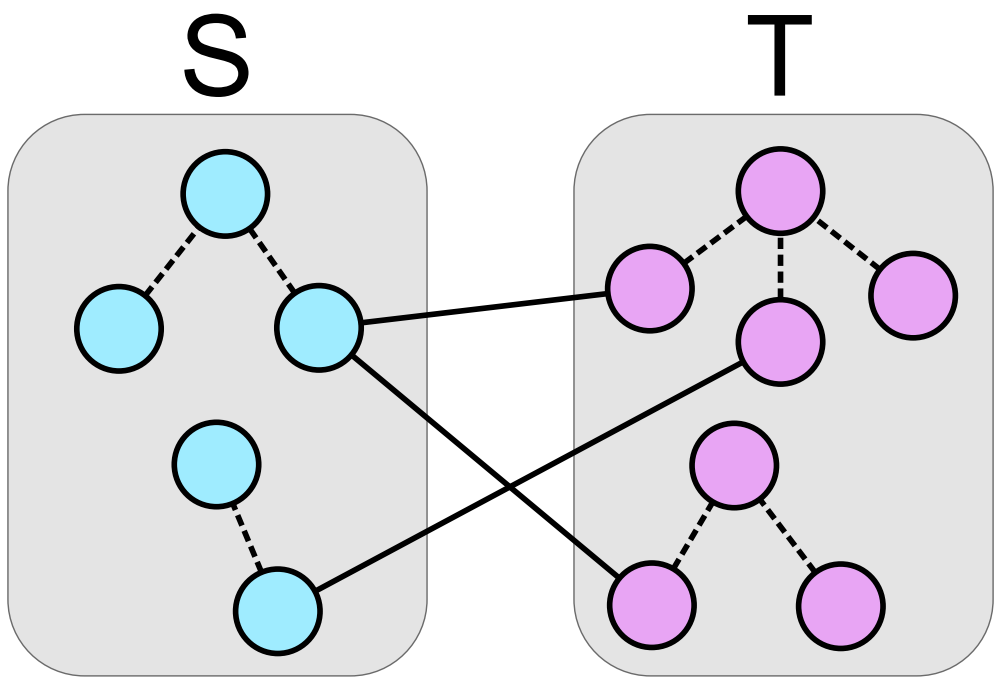
Methodology

- Formal model to measure the extent to which concerns are crosscutting
- Three case studies to gather data on scattering and defect counts
- Correlation analysis between scattering and defects

Moderate to strong correlation for all three case studies

A Model of Concerns

- S is a set of concerns (items from a program's specification)
- T is a set of program elements (AST nodes)
- $R = \{(s, t) \mid s \in S, t \in T\}$
  (relationship between concerns and program elements)



Metrics

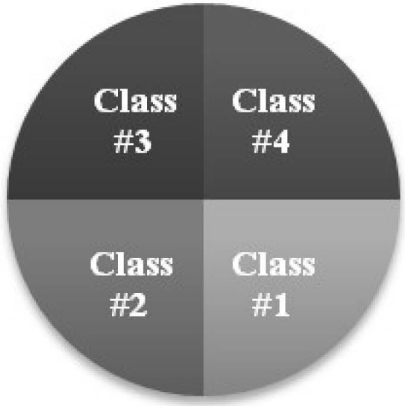
- Bug Count
- Lines of Concern Code (LOCC)
- Concern Diffusion over Components (CDC)
- Concern Diffusion over Operations (CDO)
- Degree of Scattering across Classes (DOSC)
- Degree of Scattering across Methods (DOSM)

Metrics: Degree of Scattering

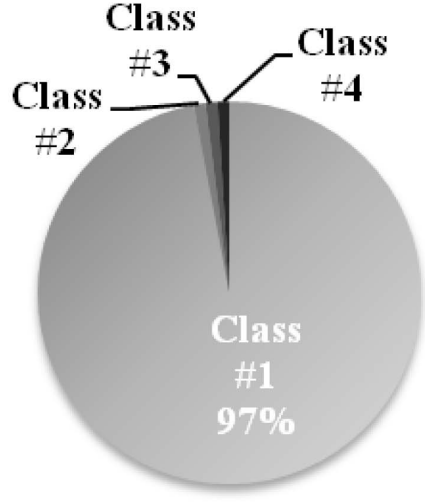
$$CONC(s,t) = \frac{\text{Source lines in element t related to concern s}}{\text{Source lines related to concern s}}$$

$$Variance(s) = \frac{\sum_{t \in T}(CONC(s,t) - CONC_{worst})}{|T|}$$

$$DOS(s) = 1 - \frac{Variance(s)}{Variance_{ideal}(s)}$$



DOSC = 1.00
CDC = 4



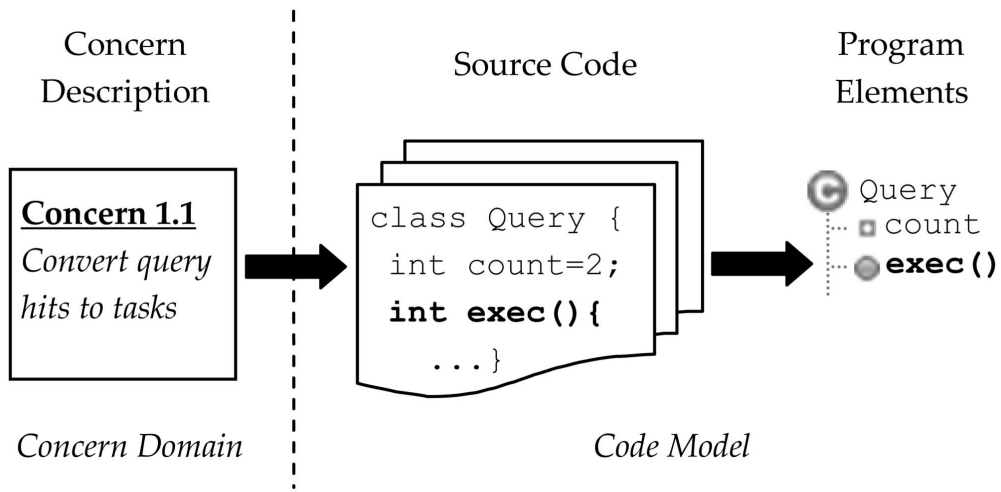
DOSC = 0.08
CDC = 4

Bug-concern mapping

1. Reverse engineer the concern-code mapping (manual)
   - **1.1** Concern selection
   - **1.2** Concern assignment (prune dependency rule)
2. Mine the bug-code mapping (partially automated)
   - **2.1** Associating bugs with bug fixes
   - **2.2** Associating bugs with program elements
3. Infer the bug-concern mapping

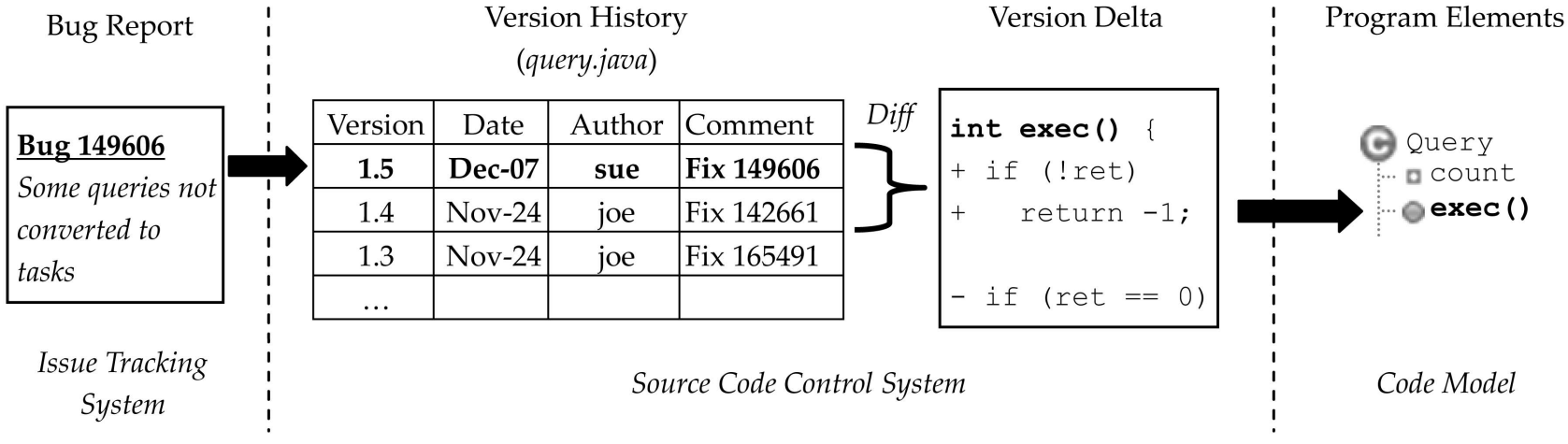
Bug-concern mapping

1. Reverse engineer the concern-code mapping (manual)
   - **1.1** Concern selection
   - **1.2** Concern assignment (prune dependency rule)
2. Mine the bug-code mapping (partially automated)
   - **2.1** Associating bugs with bug fixes
   - **2.2** Associating bugs with program elements
3. Infer the bug-concern mapping



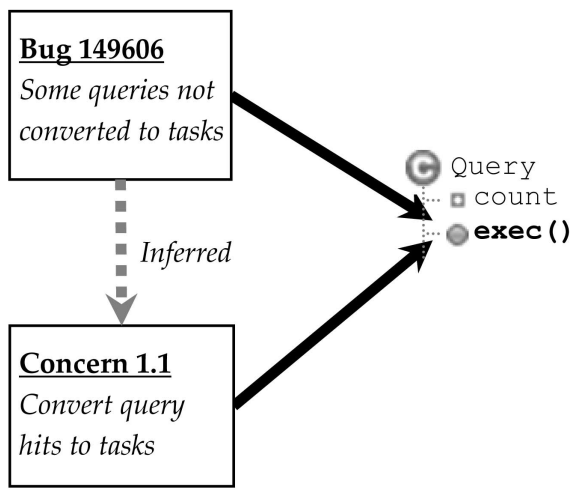
Bug-concern mapping

1. Reverse engineer the concern-code mapping (manual)
   - **1.1** Concern selection
   - **1.2** Concern assignment (prune dependency rule)
2. Mine the bug-code mapping (partially automated)
   - **2.1** Associating bugs with bug fixes
   - **2.2** Associating bugs with program elements
3. Infer the bug-concern mapping



Bug-concern mapping

1. Reverse engineer the concern-code mapping (manual)
   - **1.1** Concern selection
   - **1.2** Concern assignment (prune dependency rule)
2. Mine the bug-code mapping (partially automated)
   - **2.1** Associating bugs with bug fixes
   - **2.2** Associating bugs with program elements
3. Infer the bug-concern mapping



Case Studies

Project characteristics:

- Open-source
- Java (tooling limitation)
- Production quality
- Identifiable concerns (at least 30)
- Accessible Issue Tracking System (ITS)
- Consistently referenced bugs (commit messages)

Case Study 1

Mylyn-Bugzilla

- Plug-in for the Eclipse IDE that enables task-focused methodology
- 28 concerns identified
  - *Convert query hits to tasks*
  - *Support search for duplicates*
- Concerns and bugs assigned manually

Case Study 2

Rhino

- Javascript interpreter
- Hierarchy of 480 concerns (357 leaves) extracted from the ECMAScript Standard
  - *Regular Expression Literals*
  - *Scope Chain and Identifier Resolution*
- Concerns assigned manually

Case Study 3

iBATIS

- Object-relational mapping tool
- Hierarchy of 183 concerns (132 leaves) extracted from the Developer's Guide
  - *Caching*
  - *Request Caching*
  - *Class Caching*
- Concerns assigned manually

Coverage Statistics

	Mylyn–Bugzilla			*Rhino*			*iBATIS*		
	All	Mapped[a]	%	All	Mapped	%	All	Mapped	%
Classes	56	44	79	138	80	57	212	207	97
Methods	427	253	59	1870	1415	75	1844	1807	97
Fields	457	230	50	1339	962	71	536	529	98
Lines[b]	13649	5914	43	32134	28308	88	13314	13144	98
Concerns[c]	28	28	100	480	417	86	183	173	94
Bugs[d]	110	101	92	241	160	66	87	47	53

Results

Spearman correlation coefficients

	DOSM	*CDC*	*CDO*	*LOCC*	***Bugs***
DOSC	.64	.84	.57	.38	**.39**
DOSM	—	.77	.91	.63	**.50**
CDC	—	—	.78	.65	**.57**
CDO	—	—	—	.71	**.61**
LOCC	—	—	—	—	**.77**

(a) Mylyn-Bugzilla

	DOSM	*CDC*	*CDO*	*LOCC*	***Bugs***
DOSC	.62	.96	.74	.60	**.67**
DOSM	—	.63	.88	.68	**.66**
CDC	—	—	.80	.67	**.73**
CDO	—	—	—	.80	**.77**
LOCC	—	—	—	—	**.90**

(b) Rhino

	DOSM	*CDC*	*CDO*	*LOCC*	***Bugs***
DOSC	.67	.90	.73	.43	**.46**
DOSM	—	.67	.90	.64	**.29**
CDC	—	—	.78	.55	**.58**
CDO	—	—	—	.77	**.44**
LOCC	—	—	—	—	**.53**

(c) iBATIS

Results

Spearman correlation coefficients

	DOSM	*CDC*	*CDO*	*LOCC*	***Bugs***
DOSC	.64	.84	.57	.38	**.39**
DOSM	—	.77	.91	.63	**.50**
CDC	—	—	.78	.65	**.57**
CDO	—	—	—	.71	**.61**
LOCC	—	—	—	—	**.77**

(a) Mylyn-Bugzilla

	DOSM	*CDC*	*CDO*	*LOCC*	***Bugs***
DOSC	.62	.96	.74	.60	**.67**
DOSM	—	.63	.88	.68	**.66**
CDC	—	—	.80	.67	**.73**
CDO	—	—	—	.80	**.77**
LOCC	—	—	—	—	**.90**

(b) Rhino

	DOSM	*CDC*	*CDO*	*LOCC*	***Bugs***
DOSC	.67	.90	.73	.43	**.46**
DOSM	—	.67	.90	.64	**.29**
CDC	—	—	.78	.55	**.58**
CDO	—	—	—	.77	**.44**
LOCC	—	—	—	—	**.53**

(c) iBATIS

- Moderate to strong correlation for all three case studies

Results

Spearman correlation coefficients

	DOSM	*CDC*	*CDO*	*LOCC*	***Bugs***
DOSC	.64	.84	.57	.38	**.39**
DOSM	—	.77	.91	.63	**.50**
CDC	—	—	.78	.65	**.57**
CDO	—	—	—	.71	**.61**
LOCC	—	—	—	—	**.77**

(a) Mylyn-Bugzilla

	DOSM	*CDC*	*CDO*	*LOCC*	***Bugs***
DOSC	.62	.96	.74	.60	**.67**
DOSM	—	.63	.88	.68	**.66**
CDC	—	—	.80	.67	**.73**
CDO	—	—	—	.80	**.77**
LOCC	—	—	—	—	**.90**

(b) Rhino

	DOSM	*CDC*	*CDO*	*LOCC*	***Bugs***
DOSC	.67	.90	.73	.43	**.46**
DOSM	—	.67	.90	.64	**.29**
CDC	—	—	.78	.55	**.58**
CDO	—	—	—	.77	**.44**
LOCC	—	—	—	—	**.53**

(c) iBATIS

- Moderate to strong correlation for all three case studies
- CDC and CDO were more strongly correlated

Testing for the Confounding Effect of Size

- The size of the concern has a strong correlation with defects
- There is also a strong correlation between scattering metrics and size
- Test for a confounding effect
  - Stepwise Regression Analysis
  - Principal Component Analysis (PCA)

Conclusion

Size is not the single dominating factor, the scattering metrics contribute toward explaining the variance in bug count

Threats to Validity

Internal Validity

- Concern assignment unreliability
- Bug assignment errors
- Concern and bug assignments at the member level (not at statement level)

External Validity

- Results may not generalize
  - Programming language
  - Complexity of the problem domain
  - Tool support
  - etc

Conclusion

Main Contribution

Empirical evidence suggesting that crosscutting concerns cause defects

- Further studies are needed to draw general conclusions

Remaining Questions

- Can we reduce the likelihood of defects by reducing crosscutting?
- Are crosscutting concerns a byproduct of programming technology, developer aptitude, or the inherent complexity of the concern?
- What is the relationship between code churn and scattering?

Thanks!