Limitations of MSR (and ideas on how to solve them)

Outline

- Introduction to MSR
- The state of the art in MSR
- Limitations of MSR
  - Common threats to validity
  - Limitations of current software repositories
  - Some possible solutions
- Conclusions

Threats to the validity of MSR experiments

The categories of threats are the same as empirical studies in software engineering

- Construct validity
- Internal validity
- External validity
- Statistical conclusion
- Reliability

Construct validity

- Construct validity regard the relationship between theory and observation, i.e., the measured variables may not actually measure the conceptual variable.
- Threats in this category include:
  - measurement error
  - missing information
  - i.e, shortcomings of software repositories and data quality issues–more on this later

Internal validity

- Internal validity refers to the validity of inferences in an experimental setting
- Common threats in this category
  - Testing of subjects
  - Maturation (learning effect)
  - Experimental mortality (subjects dropping out)
- The analysis of historical data prevents these

Other threats to internal validity

- Representativeness of the chosen sample
- Quality of manual inspections relying on human judgement
  - Use several independent judges if possible, and compute their agreement

Ecological inferences and ecological fallacies

- Refers to the applicability of results made at a high level (e.g. modules) to entities at a lower level (e.g. files). Three factors can cause ecological fallacies:
  - Aggregation
  - Zonation (how the behavior under study is distributed at the lower level)
  - Sampling

Ecological Inference in
Empirical Software Engineering

Daryl Posnett Vladimir Filkov Premkumar Devanbu

Threats to external validity

- Threats to external validity are related to how generalizable the results are beyond the projects they were applied to.
- Empirical studies of systems are difficult to generalize to other systems; MSR studies are no exception.
- Development styles, practices and conditions vary between projects.

Threats to external validity

- Only open-source projects are used (are the results generalizable to the industry?)
- Only projects using a given SCM/Issue tracker are examined
- Only projects using a given programming language are used
- Other peculiarities of the development process
- Low sample sizes

Threats to statistical conclusion validity

- Concern the relationship between the treatment and the outcome
- Ensure that when statistical methods are employed, correct choices are made (significance level, correct statistical tests, multiple hypothesis testing, etc)

Reliability

- Reliability refers to the ability to consistently measure the effect of the approach
- Reliability makes reproducibility of experiments much easier
- MSR approaches feature a large degree of automation, easing reliability
- However, data confidentiality issues may arise

Limitations of software repositories

Software repositories are ad-hoc sources of data

> A **side effect** of the popularity and long-term use of SCM systems has been the recent discovery that they serve as an excellent source of data [...]. A new field, mining software repositories, has sprung up [...].

Impact of Software Engineering Research on the Practice of Software Configuration Management

JACKY ESTUBLIER
Grenoble University
DAVID LEBLANG
Massachusetts Institute of Technology
ANDRÉ VAN DER HOEK
University of California, Irvine
REIDAR CONRADI
NTNU
GEOFFREY CLEMM
Rational Software
WALTER TICHY
Universität Karlsruhe
and
DARCY WIBORG-WEBER
Telelogic

The tools were not designed with mining in mind, but to help developers solve other problems

- As such, the data stored may not be optimal for mining
- The process the developers follow may make mining more difficult (e.g., if there are only large commits, change prediction becomes really hard)

Potential issues in software repositories

- Accuracy of time window
- Amount of changes between versions
- Large commits
- Lost information
- Version sampling (compounding factor of the above)
- Bias in bug-fix datasets
- These are part of the threats to construct validity

Challenging the time window assumption

Table III: Precision estimated using postlists

Time interval	Precision (in %)		
(δ)	Max.	Min.	Avg.
200 seconds	100	50	93
1 hour	100	20	89
1 day	100	1	69
1 week	100	<1	31
1 month	95	<1	8

Table IV: Recall estimated using postlists

Time interval	Recall (in %)		
(δ)	Max.	Min.	Avg.
200 seconds	100	20	74
1 hour	100	20	84
1 day	100	22	92
1 week	100	24	94
1 month	100	25	94

Held since Zimmermann's paper in 2003

A one-hour time window gave the best results in a study

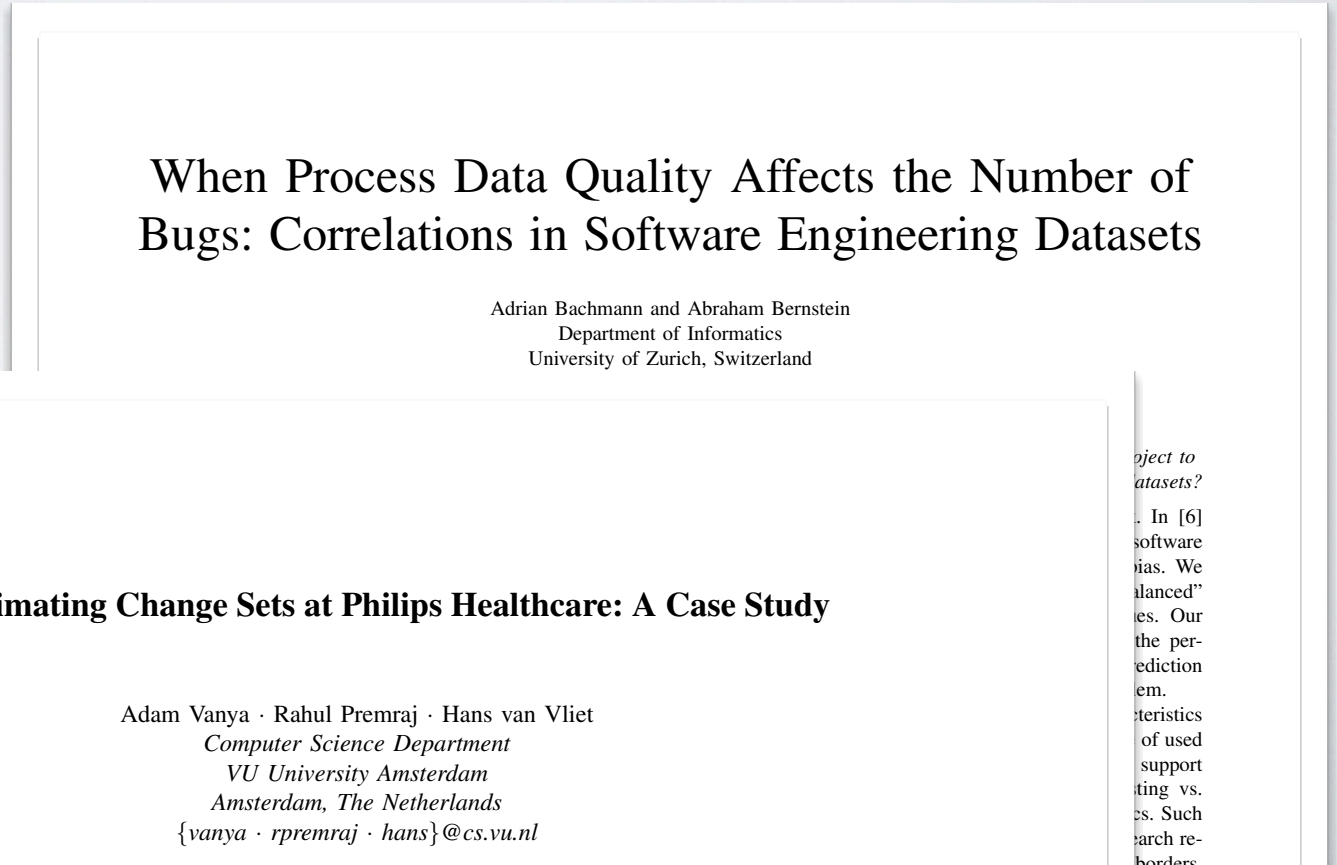
Approximating Change Sets at Philips Healthcare: A Case Study

Adam Vanya · Rahul Premraj · Hans van Vliet
Computer Science Department
VU University Amsterdam
Amsterdam, The Netherlands
{*vanya · rpremraj · hans*}*@cs.vu.nl*

Commit comments may be empty, or useless

> Note that in our study environment, such comments were extremely rare.

> The problem of empty commit messages arises mainly in the Eclipse dataset where about 20% of all commit messages are empty.



When Process Data Quality Affects the Number of Bugs: Correlations in Software Engineering Datasets

Adrian Bachmann and Abraham Bernstein
Department of Informatics
University of Zurich, Switzerland

Approximating Change Sets at Philips Healthcare: A Case Study

Adam Vanya · Rahul Premraj · Hans van Vliet
Computer Science Department
VU University Amsterdam
Amsterdam, The Netherlands
{*vanya · rpremraj · hans*}*@cs.vu.nl*

Developer duplication and identification

Type	Data Source	Primary Identities
(1)	Mailing lists	username@example.com
(1)	Mailing lists	Name Surname
(2)	Source Code	(c) Name Surname
(2)	Source Code	(c) username@example.com
(2)	Source Code	$id: username$
(3)	Versioning System	username
(4)	Bug Tracking	username@example.com

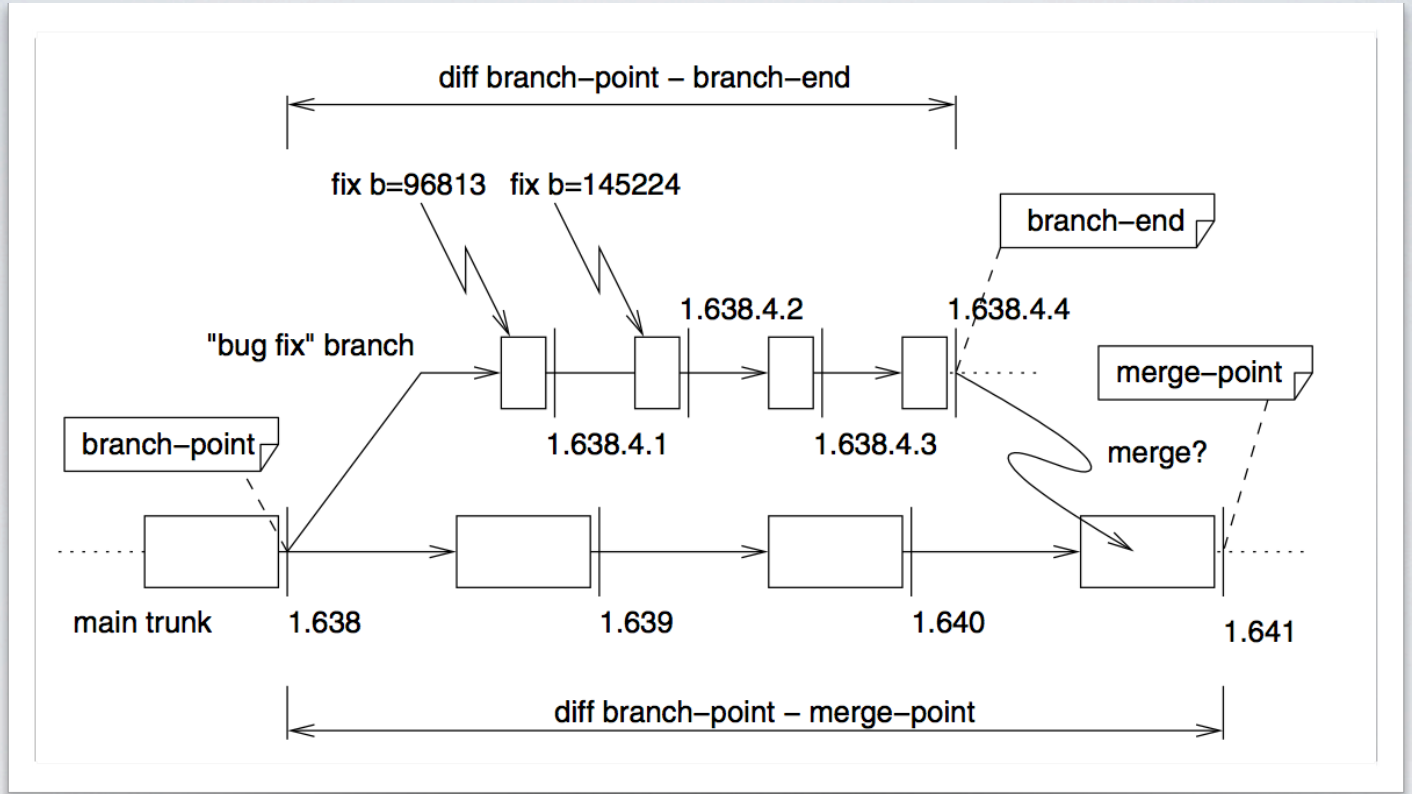
Table 1: Identities that can be found for each data source.

The same person can go by different identifiers.
Changing email address is a very common example.

Developer identification methods for integrated data from various sources

Gregorio Robles, Jesus M. Gonzalez-Barahona
{grex, jgb}@gsyc.escet.urjc.es
Grupo de Sistemas y Comunicaciones
Universidad Rey Juan Carlos
Madrid, Spain

Detecting branches and merging



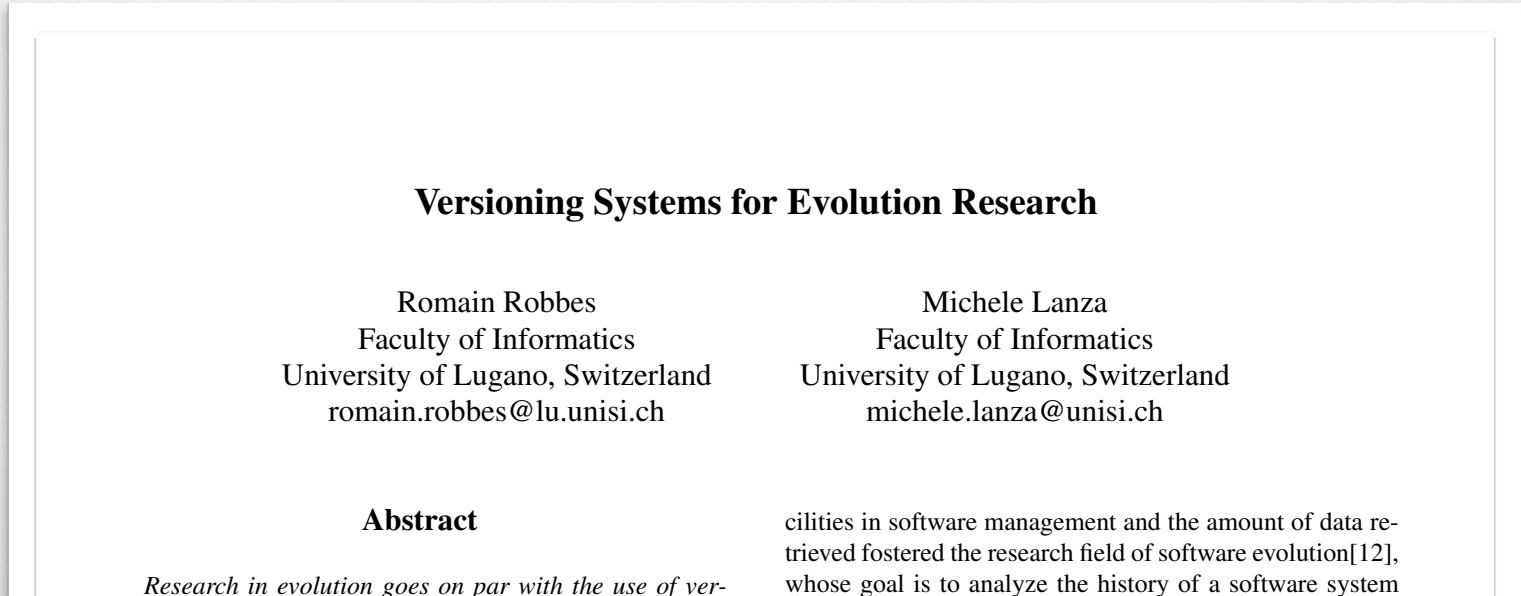
Merging branches can produce large transactions.
In CVS and early SVN (< 1.5), merges need to be detected.

Populating a Release History Database from Version Control and Bug Tracking Systems*

Michael Fischer, Martin Pinzger, and Harald Gall
Distributed Systems Group, Vienna University of Technology

Common SCM systems make it hard to recover precise changes

- Text-based versioning implies heavy preprocessing
  - (but makes an SCM more versatile)
- Taking snapshots at commit time loses information
  - (but simplifies the SCM)



Versioning Systems for Evolution Research

Romain Robbes
Faculty of Informatics
University of Lugano, Switzerland
romain.robbes@lu.unisi.ch

Michele Lanza
Faculty of Informatics
University of Lugano, Switzerland
michele.lanza@unisi.ch

Abstract

Research in evolution goes on par with the use of ver-

cilities in software management and the amount of data retrieved fostered the research field of software evolution[12], whose goal is to analyze the history of a software system

Text-based versioning inflates the importance of some kind of changes

- Indentation changes
- License changes
- Moving code (files renamed, split, moving code inside a file)
- Refactorings (methods renamed ...)
- Changes to cloned code
- Committing large patches by external contributors

There are several kinds of large commits

Categories of Large Commits	Description
Implementation	New requirements
Maintenance	Maintenance activities.
Module Management	Changes related to the way the files are named and organized into modules.
Legal	Any change related to the license or authorship of the system.
Non-functional source-code changes	Changes to the source code that did not affect the functionality of the software, such as reformatting the code, removal of white-space, token renaming, classic refactoring, code cleanup (such as removing or adding block delimiters without affecting the functionality)
SCS Management	Changes required as a result of the manner the Source Control System is used by the software project, such as branching, importing, tagging, etc.
Meta-Program	Changes performed to files required by the software, but which are not source code, such as data files, documentation, and Makefiles.

Table 5: Categories of Large Commits. They reflect better the types of large commits we observed than those by Swanson.

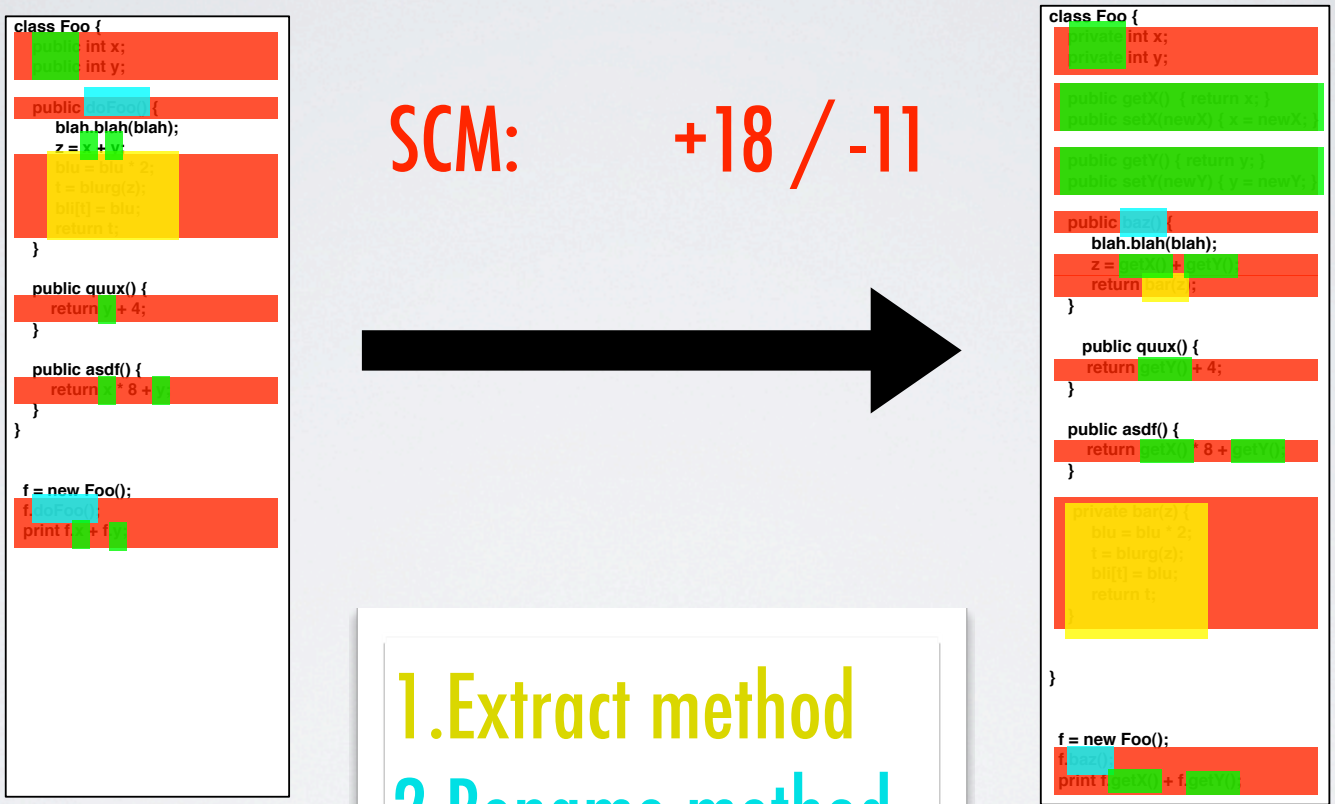
What Do Large Commits Tell Us? A taxonomical study of large commits

Abram Hindle
David Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario

Daniel M. German
Department of Computer Science University of Victoria
Victoria, British Columbia
Canada

Ric Holt
David Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario

Automated refactorings can have a large impact, in a short amount of time



1. Extract method
2. Rename method
3. Create accessors

How common are these changes?

Between 2.8% and 22.9% of modified code lines were updated only via non-essential differences.

In the individual systems analyzed, between 2.6% and 15.5% of all method updates were non-essential.

(numbers excluding indentation, comment changes)

Non-Essential Changes in Version Histories

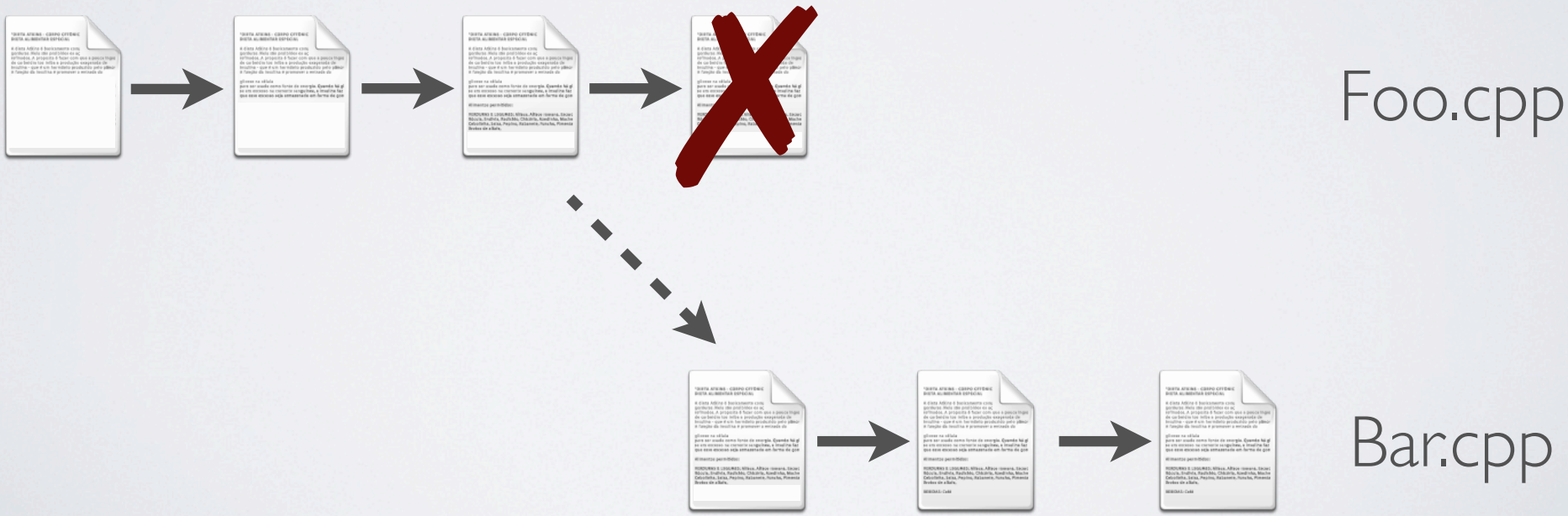
David Kawrykow and Martin P. Robillard
McGill University
Montréal, Canada
{dkawry,martin}@cs.mcgill.ca

ABSTRACT

Numerous techniques involve mining change data captured in software archives to assist engineering efforts, for example to identify non-obvious relationships between code elements [8, 23, 25]. We refer to approaches operating on change data as *change-based approaches*.

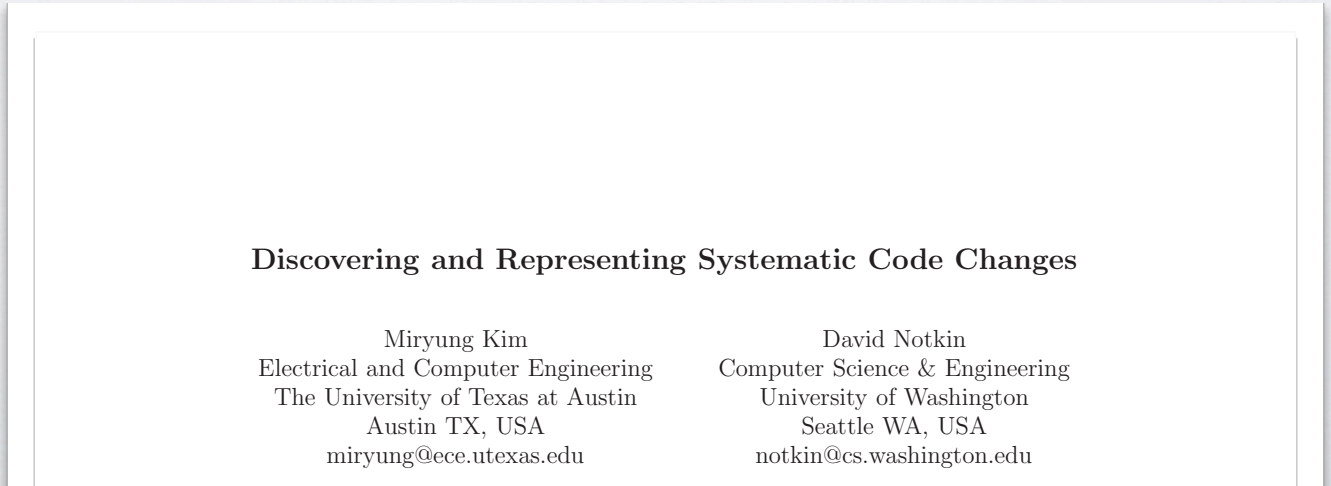
Other refactorings can have an even larger impact

- What if foo.cpp is renamed to bar.cpp?
- In the SCM, foo.cpp will be removed, and bar.cpp will be added



Several techniques deal with these kind of changes

- Origin analysis detect renames, and merging and splitting of files.
- Refactoring detection detects renames to methods, etc.
- Kim's approach detects systematic changes.



Discovering and Representing Systematic Code Changes

Miryung Kim
Electrical and Computer Engineering
The University of Texas at Austin
Austin TX, USA
miryung@ece.utexas.edu

David Notkin
Computer Science & Engineering
University of Washington
Seattle WA, USA
notkin@cs.washington.edu

Filtering noise

- For change-log analysis techniques, one can drop large transactions
- Defect or change prediction with large transaction are problematic

These techniques have a cost

- Additional computation makes version sampling more likely
- version sampling in turn makes changes larger, further degrading quality
- Bi-weekly snapshots are already costly to process

Is the situation better when mining a DVCS such as Git?

9 "Promises" including:
access to more history
better origin analysis
better performance

7 "Perils", including:
complex branches
rewritable history
hard to detect merges

The Promises and Perils of Mining Git

Christian Bird*, Peter C. Rigby†, Earl T. Barr*, David J. Hamilton*, Daniel M. German†, Prem Devanbu*

*University of California, Davis, USA
†University of Victoria, Canada

{bird,barr,hamiltod,devanbu}@cs.ucdavis.edu {pcr,dmg}@cs.uvic.ca

Git commits are a bit smaller, but not that much

> We therefore performed an analysis on the size of commits, measured in LOC added and removed, prior to and after migration to git for a number of projects. Although statistically significant, the magnitude of the difference was uninteresting (only 2 lines less per commit).

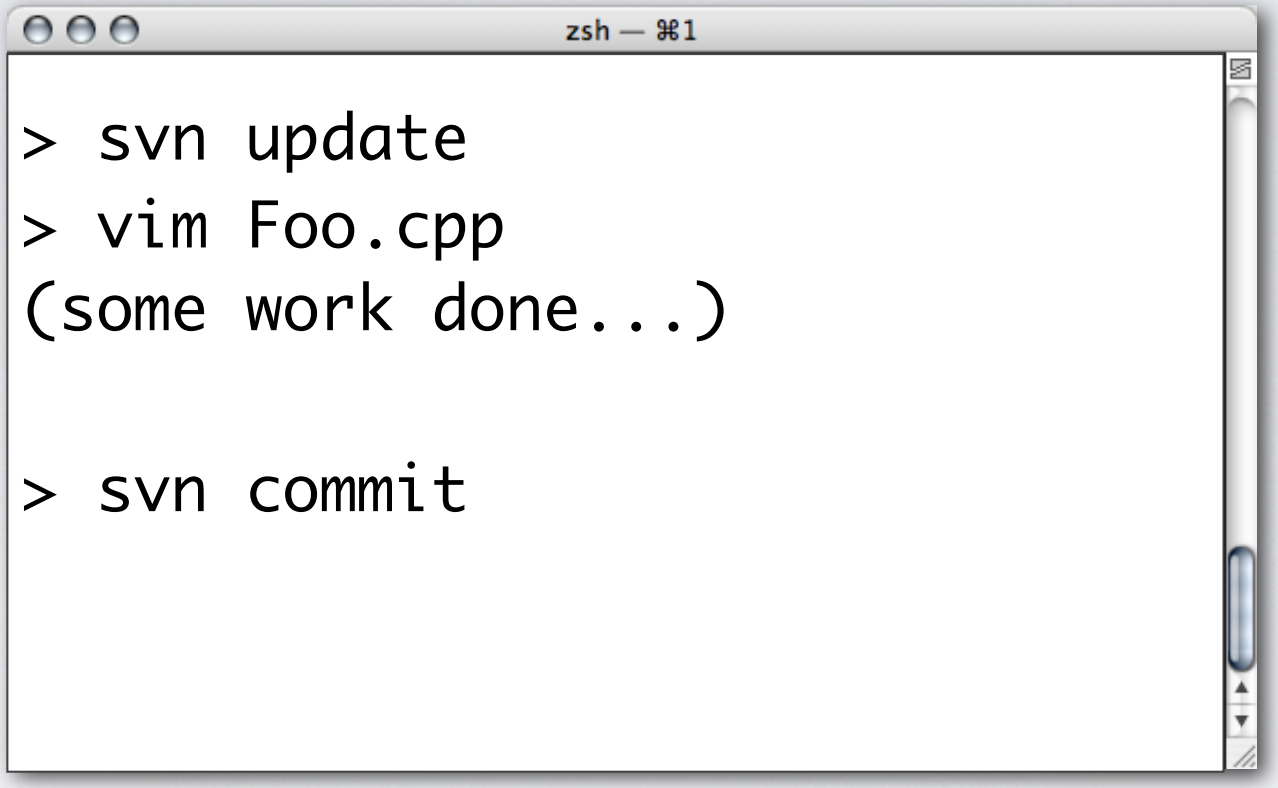
The Promises and Perils of Mining Git

Christian Bird*, Peter C. Rigby†, Earl T. Barr*, David J. Hamilton*, Daniel M. German†, Prem Devanbu*

*University of California, Davis, USA
†University of Victoria, Canada

{bird,barr,hamiltod,devanbu}@cs.ucdavis.edu {pcr,dmg}@cs.uvic.ca

In all cases, SCM only record snapshots ...



... so they lose information

Limitations of defect repositories

- Some bugs are actually not bugs, but enhancements
- Automated linking does not find all the links
- Not all bugs are reported in the bug database

Some bugs are not really bugs, but rather requests for enhancements

Systems	Bug	Non bugs	Others
Mozilla	270	209	121
Eclipse	194	382	24
JBoss	345	99	156

Table 1: Final classification over the 1,800 BTS issues for Mozilla, Eclipse and JBoss.

A large proportion of the bugs are "not bugs"

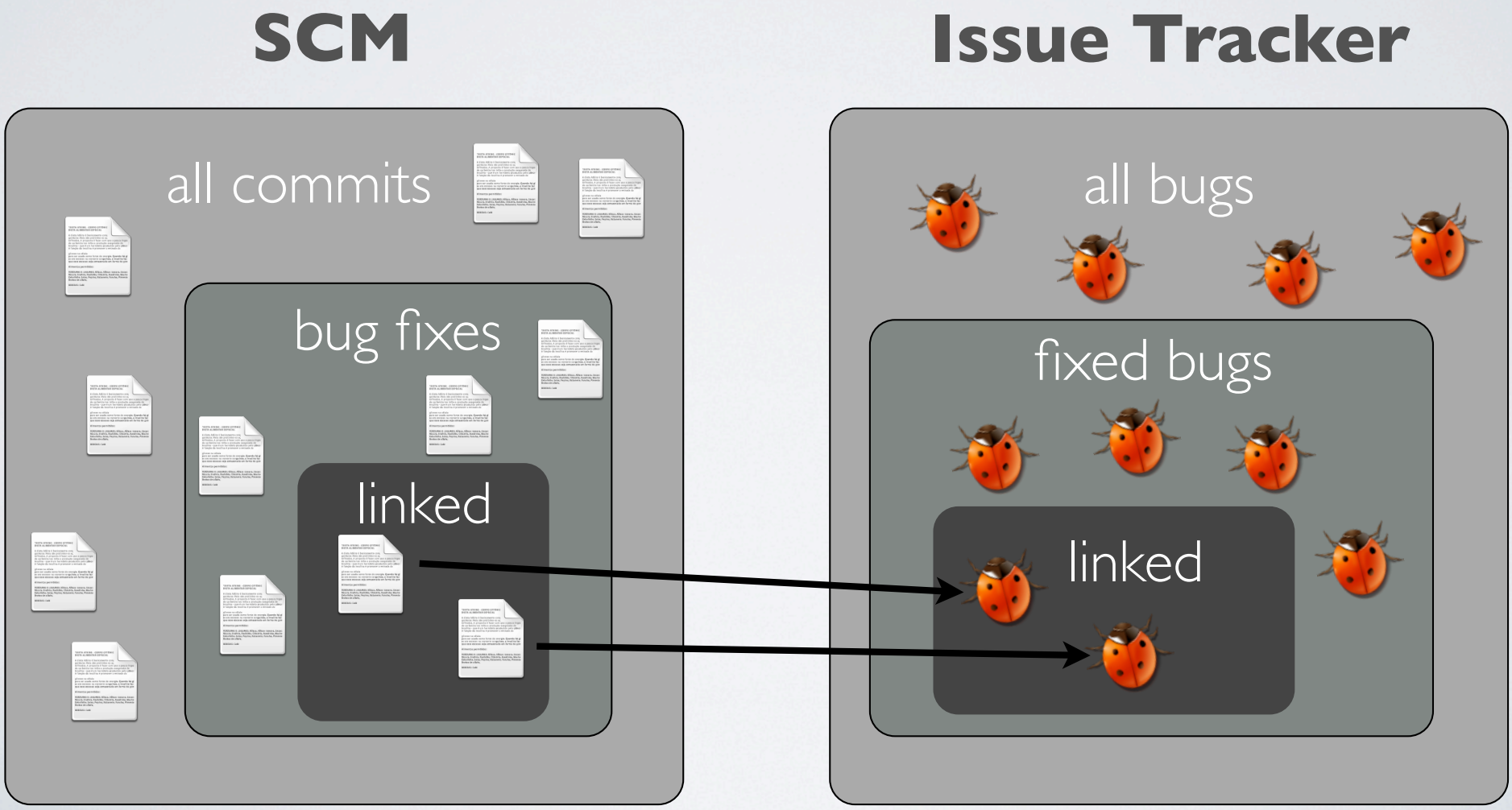
Is it a Bug or an Enhancement? A Text-based Approach to Classify Change Requests

Giuliano Antoniol and Kamel Ayari[1] Massimiliano Di Penta[2]
Foutse Khomh and Yann-Gaël Guéhéneuc[3]

[1] SOCCER Lab. – DGIGL, École Polytechnique de Montréal, Québec, Canada
[2] RCOST, University of Sannio, I-82100 Benevento, Italy
[3] Ptidej Team – GEODES, DIRO, Université de Montréal, Québec, Canada

Evidence of bias in bug-fix datasets

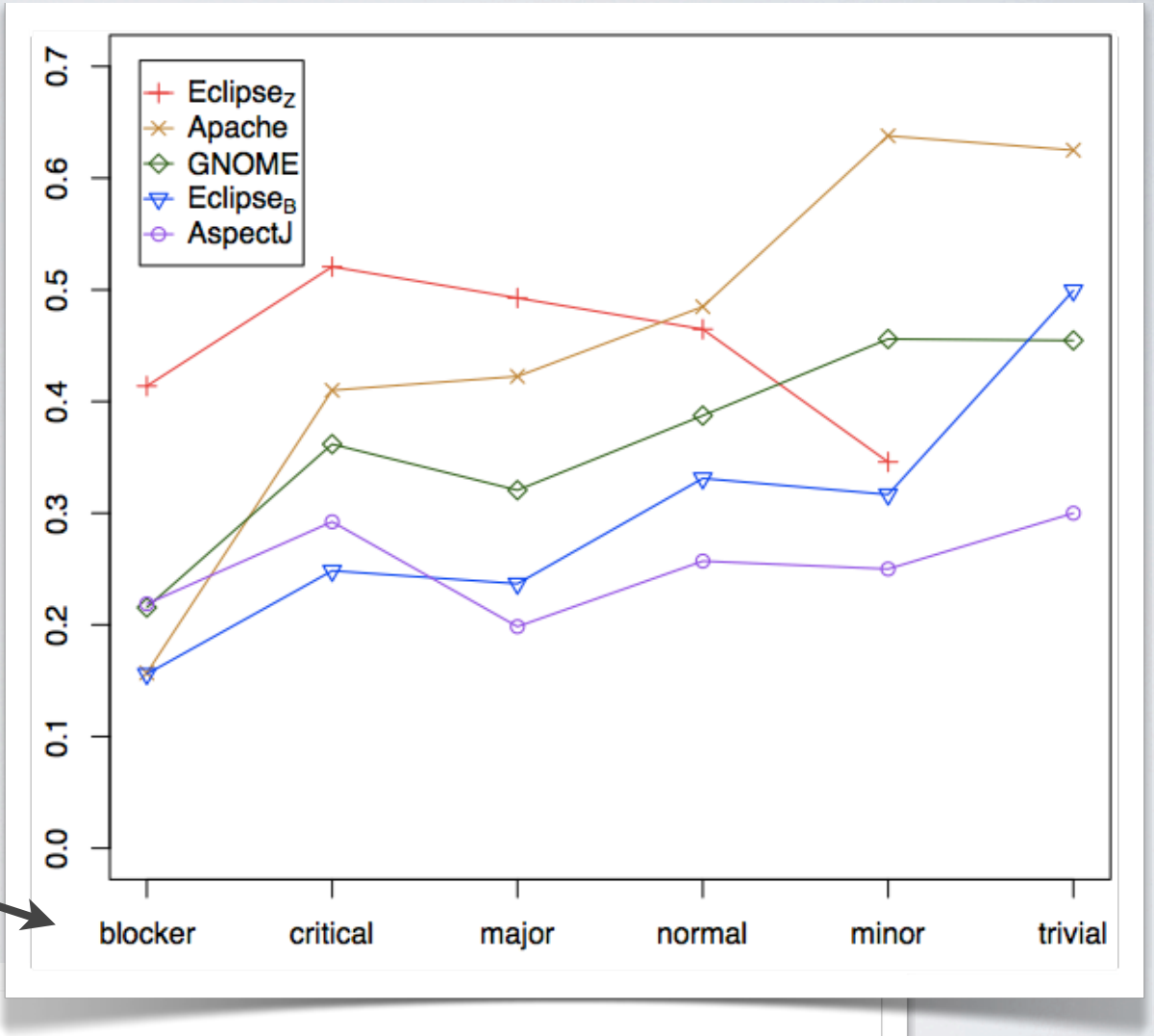
Not all the bugs that are fixed are linked.
Not all types of bugs are linked equally.



Less severe bugs are better linked!

linkage proportion

bug severity



Fair and Balanced? Bias in Bug-Fix Datasets

Christian Bird[1], Adrian Bachmann[2], Eirik Aune[1], John Duffy[1]
Abraham Bernstein[2], Vladimir Filkov[1], Premkumar Devanbu[1]

[1]University of California, Davis, USA
[2]University of Zurich, Switzerland

Manual linking gives different results

- A sample of bug reports manually linked by an expert Apache developer gave different results.
- He examined all the commits in a six week period of the history of Apache (493 commits), finding 69 bug fixes, and 7 security fixes.

The Missing Links: Bugs and Bug-fix Commits

Adrian Bachmann[1], Christian Bird[2], Foyzur Rahman[2],
Premkumar Devanbu[2] and Abraham Bernstein[1]

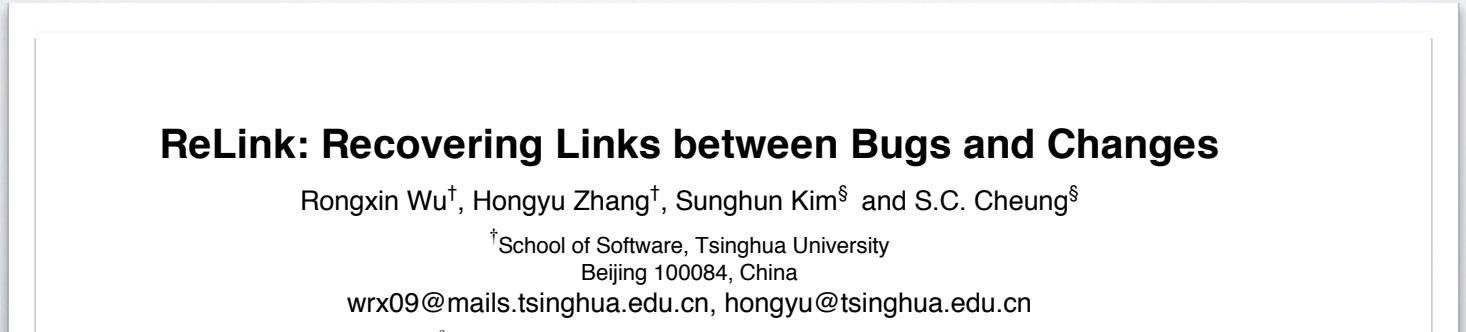
[1]Department of Informatics, University of Zurich, Switzerland
[2]Computer Science Department, University of California, Davis, USA

Only 47.6% of the bug fixing commits were documented in the bug fix database

- Others were only discussed in the mailing list.
- Other findings:
  - Several commits may be involved in the same bug fix
  - Bugs may be fixed outside of the project (in librairies)
  - The reason of a commit can be hard to find (empty commits) or unrelated to changes (documentation)

Better linking techniques have been proposed

- ReLink uses three similarities to improve bug linking
  - Similarity in time between bug fix and commits
  - Similarity in committer and bug fixer names
  - Similarity in vocabulary between bug fix and report
- Precision/Recall go from 91%/64% to 89%/78%



ReLink: Recovering Links between Bugs and Changes

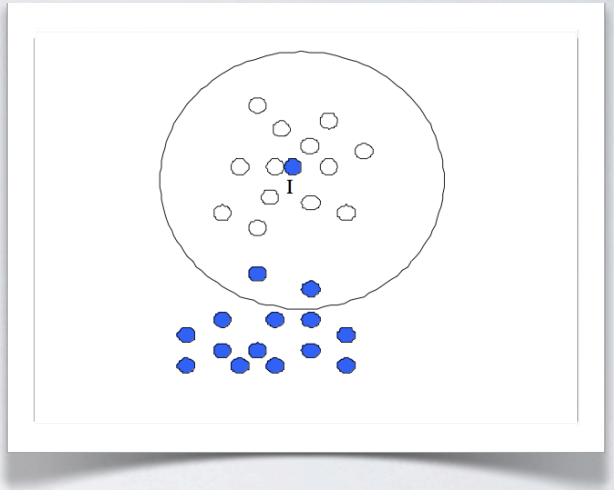
Rongxin Wu[†], Hongyu Zhang[†], Sunghun Kim[§] and S.C. Cheung[§]

[†]School of Software, Tsinghua University
Beijing 100084, China
wrx09@mails.tsinghua.edu.cn, hongyu@tsinghua.edu.cn

The CLNI algorithm detects noise in bug prediction datasets

Table 4.The defect prediction performance (F-measure) after identifying and removing noisy instances (SWT)

Remove Noises ?	Noise Rate	Bayes Net	Naïve Bayes	SVM	Bagging
No	15%	0.781	0.305	0.594	0.841
	30%	0.777	0.308	0.339	0.781
	45%	0.249	0.374	0.353	0.350
Yes	15%	0.793	0.429	0.797	0.838
	30%	0.802	0.364	0.706	0.803
	45%	0.762	0.418	0.235	0.505



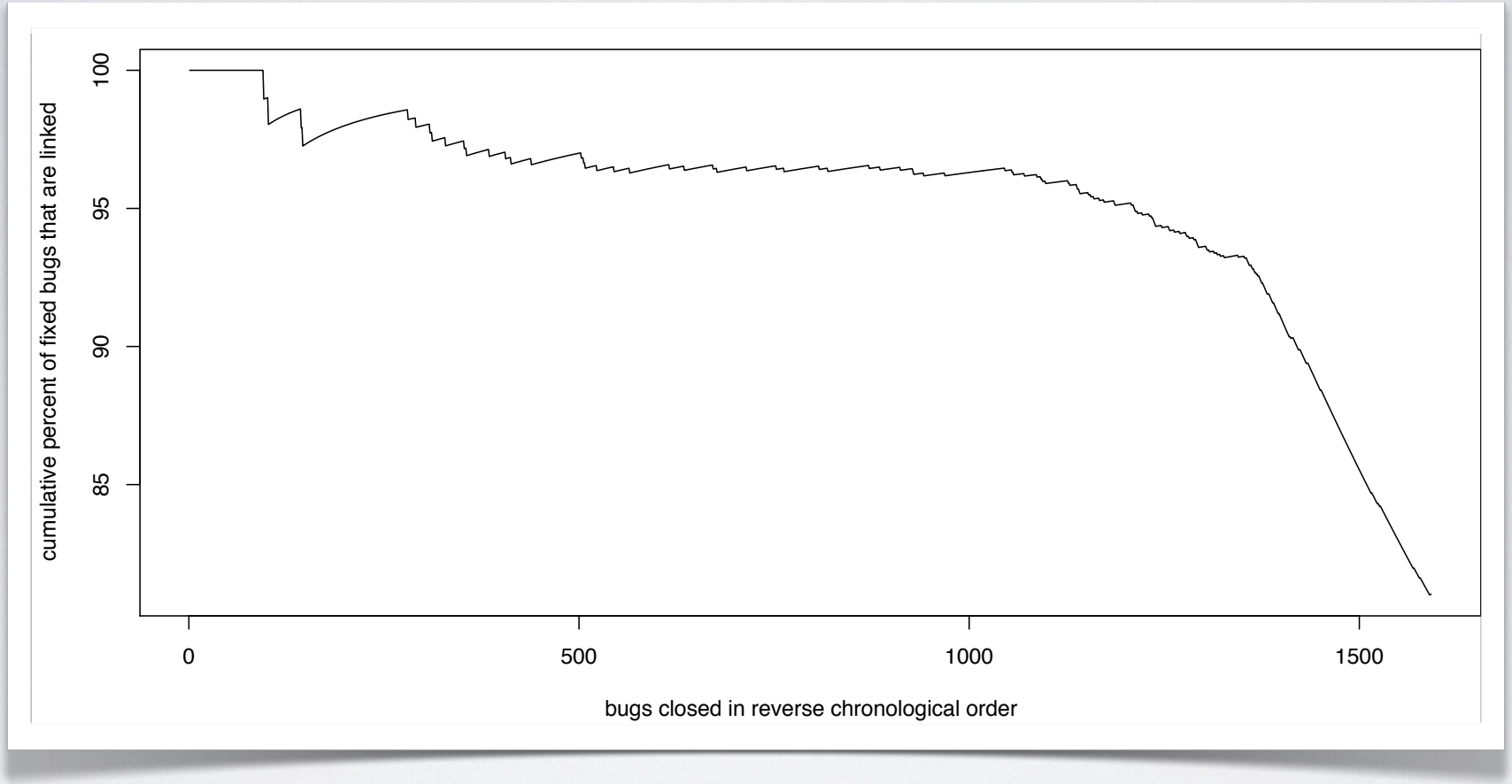
CLNI in a nutshell

Dealing with Noise in Defect Prediction

Sunghun Kim[1], Hongyu Zhang[2], Rongxin Wu[2] and Liang Gong[2]

[1] Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong
[2] School of Software, Tsinghua University, Beijing, China
hunkim@cse.ust.hk, hongyu@tsinghua.edu.cn, {se.wu.rongxin, jacksongl1988}@gmail.com

Other solution: use projects with better bug linking practices and better issue tracking



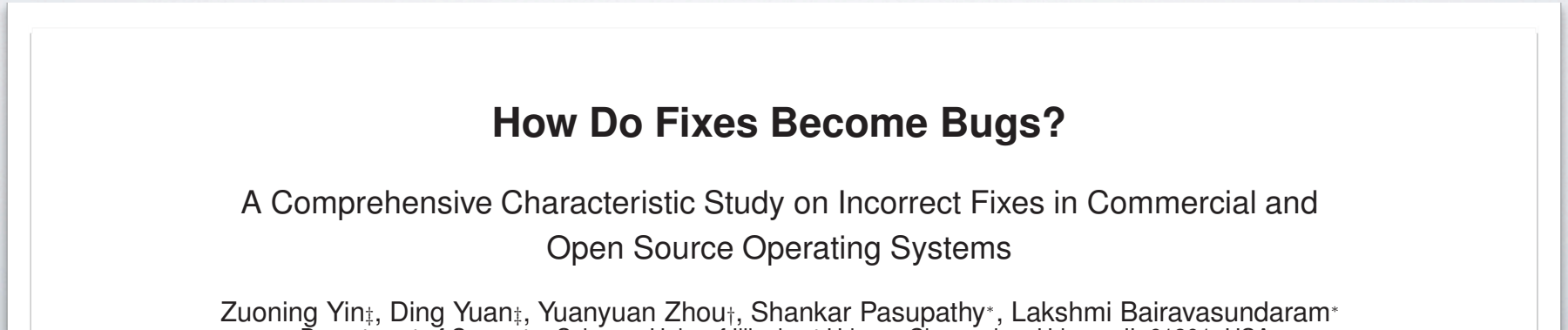
Lucene has a link rate around 95%, using JIRA
(from personal communication with Christian Bird)

A general solution: increase the amount of manual verification and qualitative analyses

To make sure the results make sense and are computed correctly

To characterize finer phenomena not easily observable through the lens of automation

Example: understanding why bug fixes can become bugs, based on an inspection of 1,000 actual instances



How Do Fixes Become Bugs?

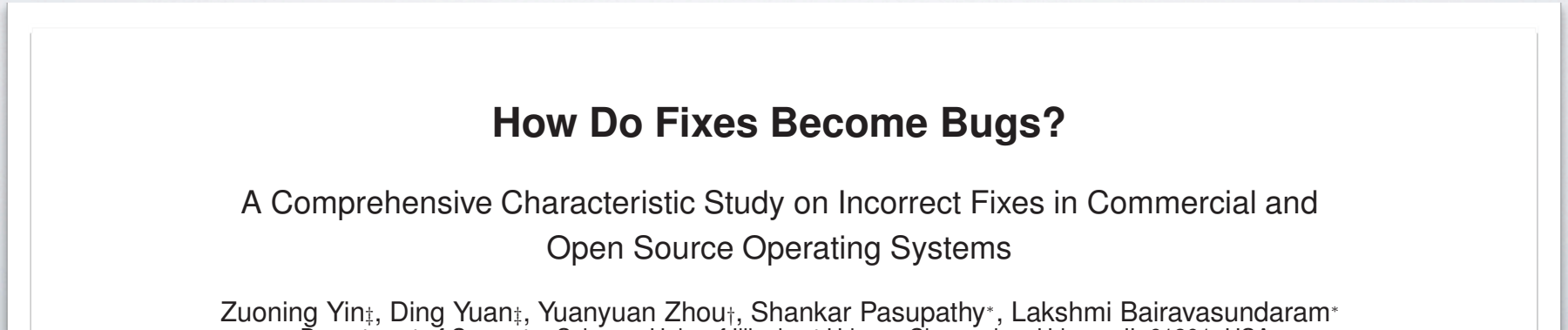
A Comprehensive Characteristic Study on Incorrect Fixes in Commercial and Open Source Operating Systems

Zuoning Yin‡, Ding Yuan‡, Yuanyuan Zhou†, Shankar Pasupathy*, Lakshmi Bairavasundaram*

Two findings from the nine in this paper

> At least 14.8%~24.4% of examined fixes for post-release bugs are incorrect. 43% of the examined incorrect fixes can cause crashes, hangs, data corruptions or security problems.

> Interestingly, in most of the cases, the developers who are most familiar (5~6 times of the actual fixers) with the relevant code of these incorrect fixes are still working on the project, but unfortunately were not selected to do the fixes.



How Do Fixes Become Bugs?

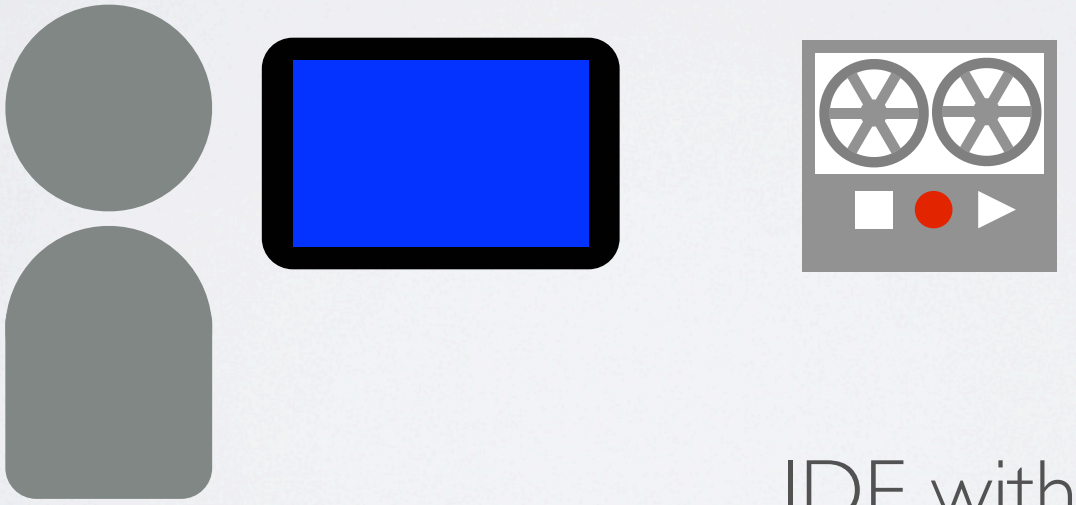
A Comprehensive Characteristic Study on Incorrect Fixes in Commercial and Open Source Operating Systems

Zuoning Yin‡, Ding Yuan‡, Yuanyuan Zhou†, Shankar Pasupathy*, Lakshmi Bairavasundaram*

Alternative repositories to overcome the limitations

Usage of alternative repositories can address some of the issues

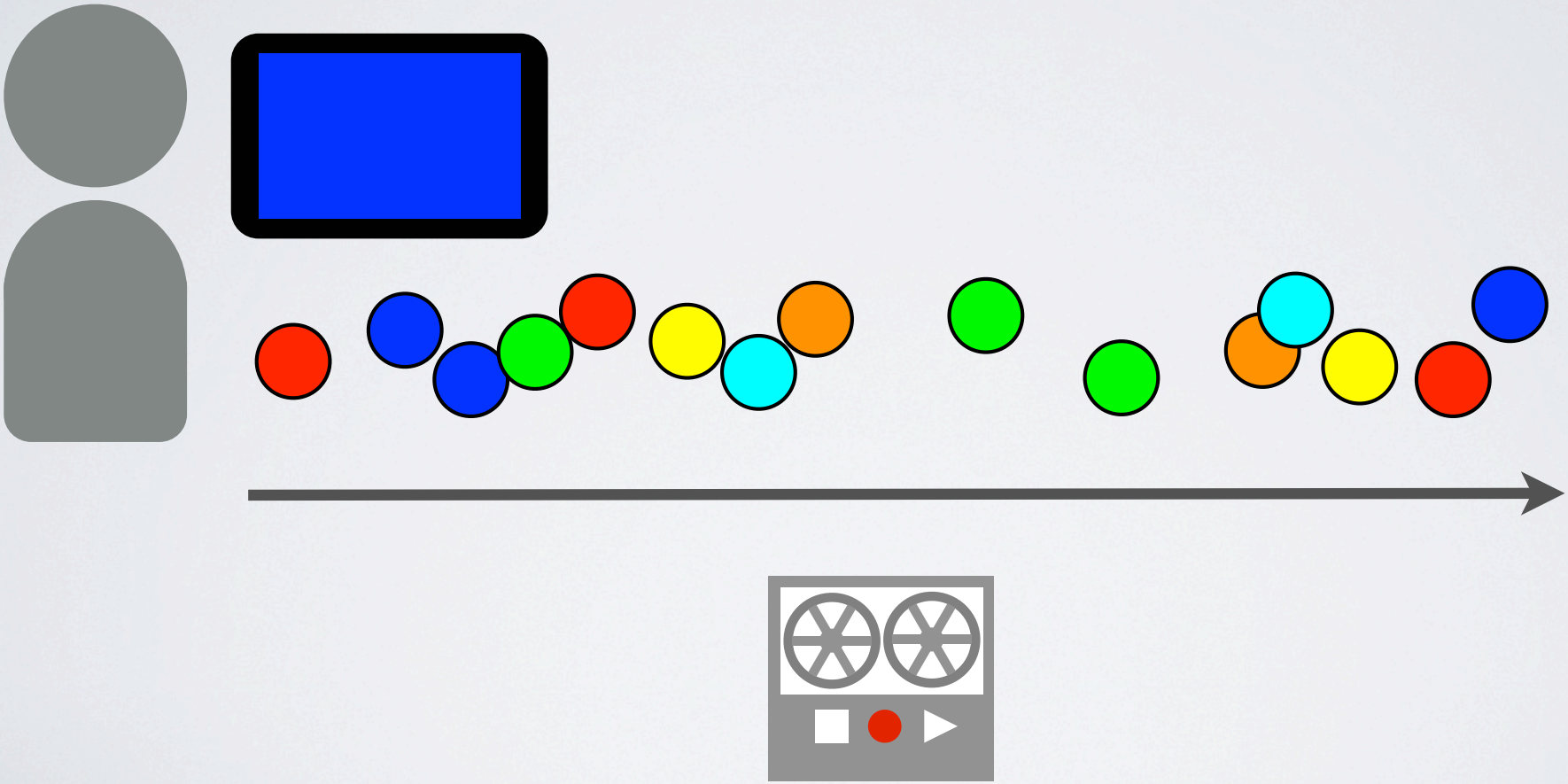
- Recorded IDE interactions offer a finer view of developer behavior than the coarse-grained view offered by versioning systems.



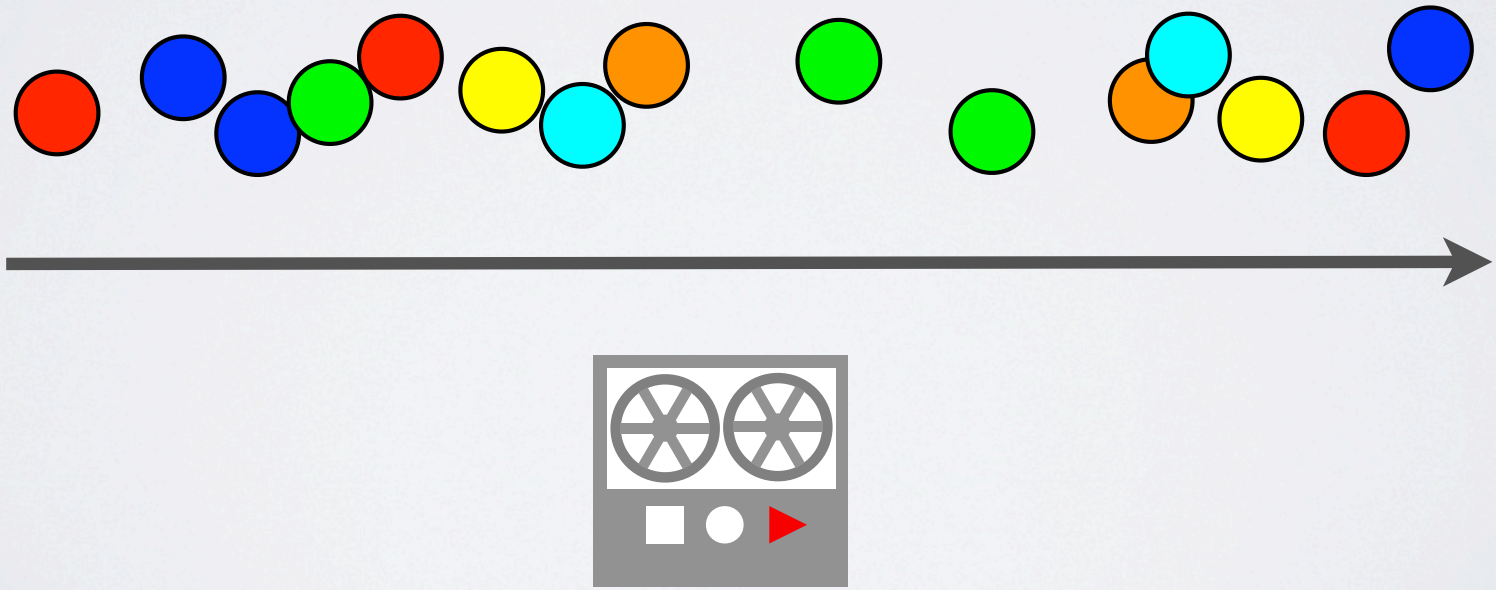
Developer

IDE with instrumentation

All the information previously lost is recorded ...



... and can be replayed



There are two levels of recording:

- Recording interactions: navigation and edition events
- Recoding changes: recording the content of editions

Interaction recording

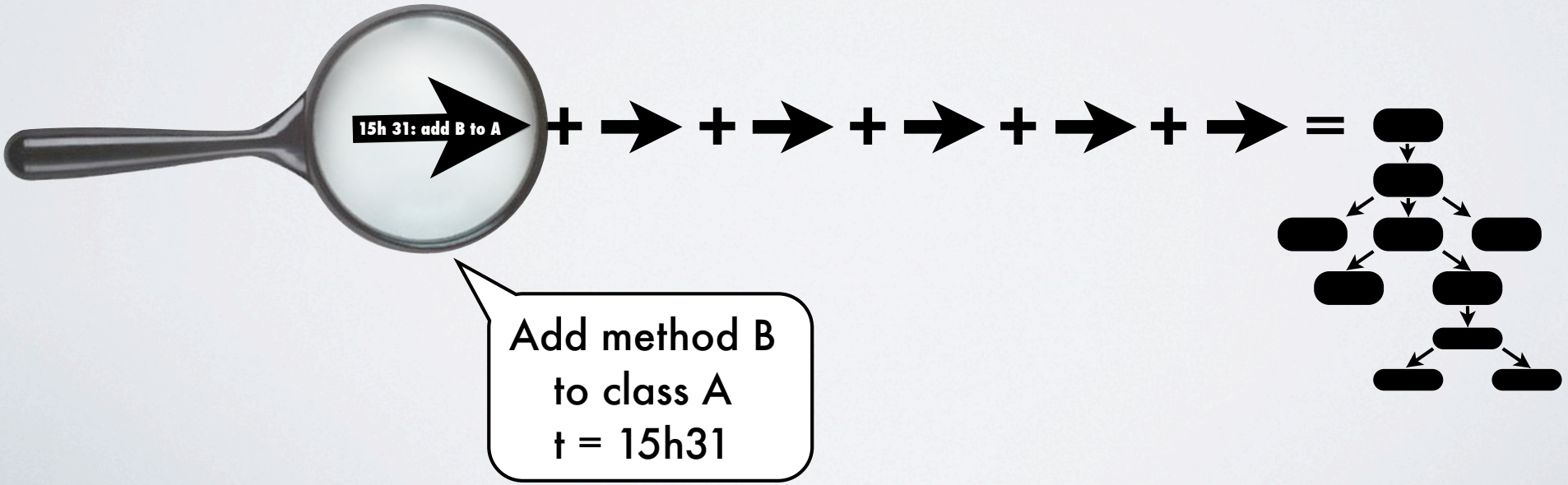
- The Mylyn dataset has been used with success
  - Mylyn records selections and editions in Eclipse
  - selection: "looking" at an entity (file, method)
  - edition: the entity has been changed (but not how)

Type	Description
Selection	Select a file in the explorer
Edit	Edit a file in the editor
Command	Invoke command by developer
Propagation	Propagated interaction
Prediction	Predict future interaction
Manipulation	Manipulate DOI value.

Change recording approaches

- Record changes as they are performed in the IDE
  - SpyWare/CBSE (Smalltalk)
  - Syde/synchronous changes (Eclipse–collaborative)
- Changes are:
  - addition/removals of packages classes, methods
  - insertion/deletion of program statements

How does change recording work?

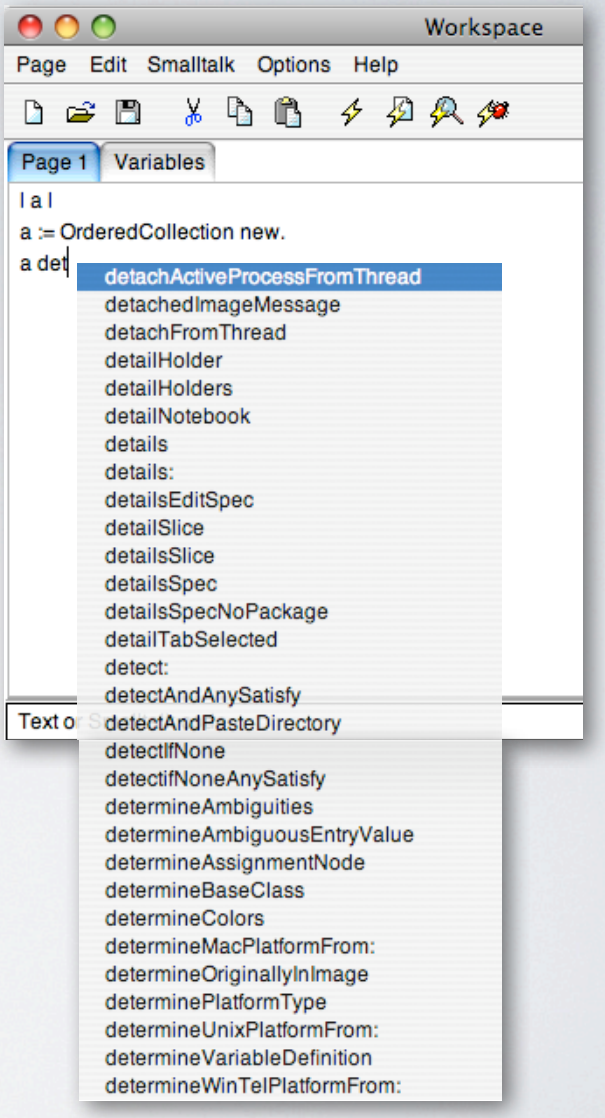


Recorded changes and/or interactions have been used in a variety of cases

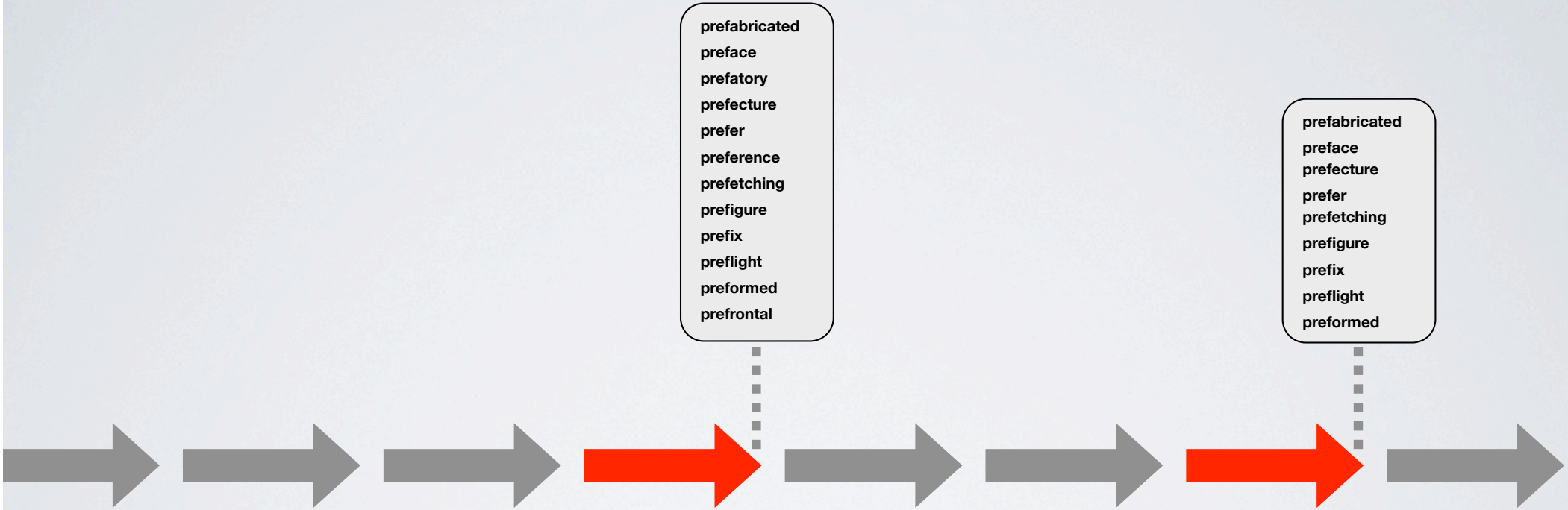
- Better code completion
- Change prediction
- Ownership/expertise
- Defect prediction
- Others: task identification, task resumption, empirical studies of programmer behavior

Code completion is used by every developer

Command	% of all commands
Delete	14.3
Save	11.3
Next Word	7.3
Paste	6.8
Code Completion	6.7
Previous Word	5.9
Copy	4.6
Select Previous Word	3.4
Step	3.2



Using recorded changes to evaluate code completion



Each time a method call is inserted,
we can test the completion algorithm

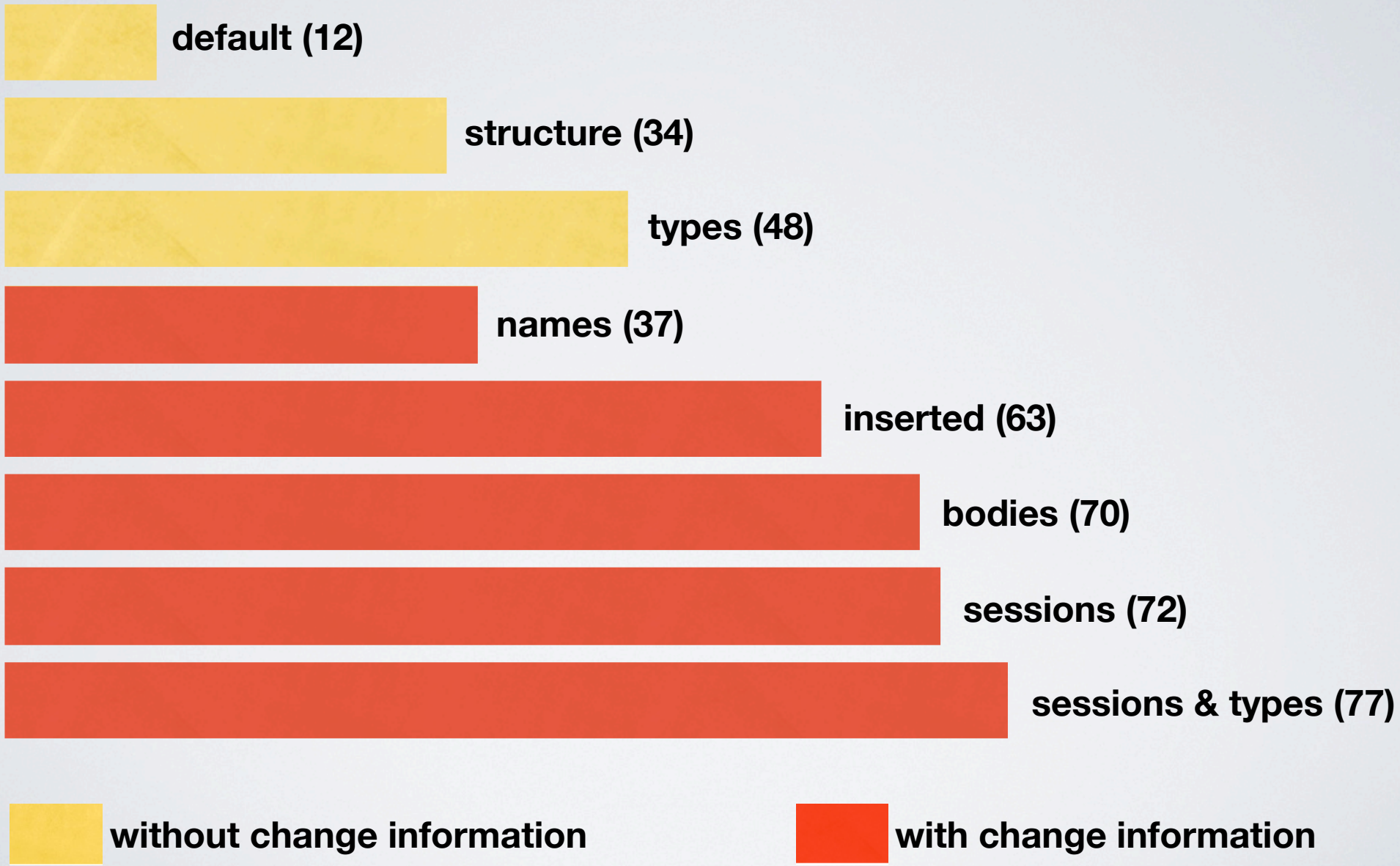
Testing the insertion of “prescient”

prefix	l=2	l=3	l=4	l=5	l=6	l=7	l=8
result	fail	fail	6	2	n/a	n/a	n/a

“pr”	“pre”	“pres”	“presc”
practical	preach	presage	preschool
practice	preadapt	presager	prescient
praesidium	preamble	presbyopia	
praetor	preamp	presbyter	
pragmatic	prebiotic	preschool	
prairie	precalculus	prescient	
praline	precarious	preselect	
prank	precast	presence	
pray	precaution	preserve	
preach	precede	president	
preadapt	predator	press	
▪▪▪	▪▪▪	▪▪▪	

cutoff

Performance of various completion algorithms



Using CBSE to evaluate change prediction

Our simplifying assumptions imply that the ordering of selections and queries to a heuristic are immaterial.

Replaying development history to assess the effectiveness of change propagation tools
Hassan and Holt, Empir. Software Eng. 11 (3), 2006

What if the ordering matters?

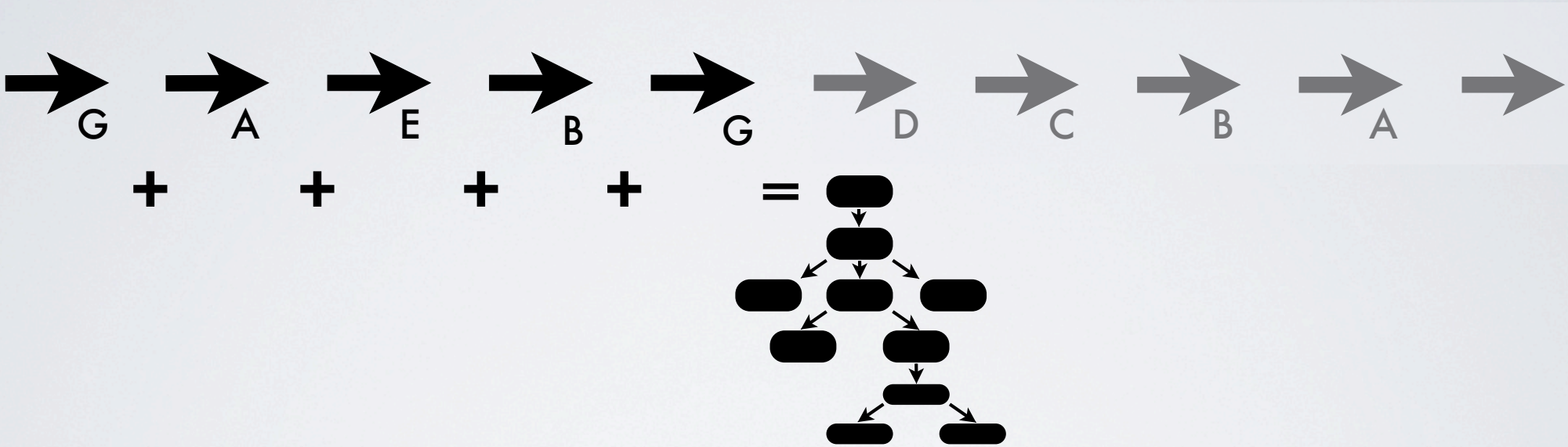
Replaying IDE Interactions to Evaluate and Improve Change Prediction Approaches

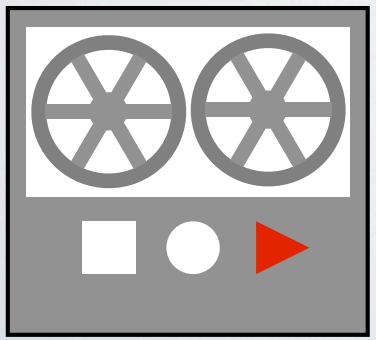
Romain Robbes
University of Chile

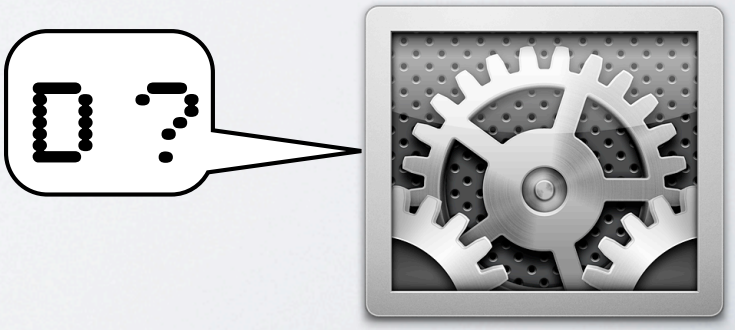
Damien Pollet
Inria – Université de Lille 1, France

Michele Lanza
University of Lugano, Switzerland

Taking the order of changes in account changes the evaluation process







Using CBSE to evaluate change prediction

- Recorded change sequences better fit the way developers actually work
- Change prediction based on SCM archives does not take into account the sequence of changes; doing so improves performance

Project	SW	SA	X	A–F	Avg
Classes	10.17	12.88	6.46	21.24	11.82
Methods	1.66	3.35	1.15	3.90	2.41

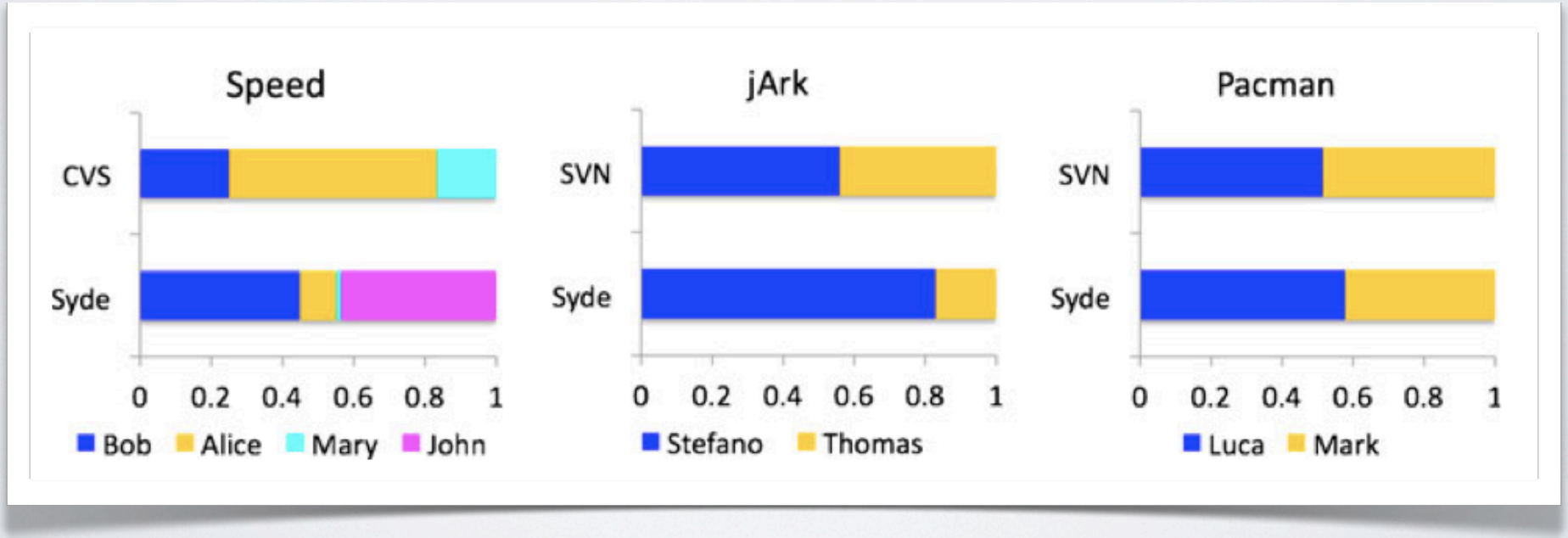
Table III
RESULTS OF ASSOCIATION RULES MINING

Project	SW	SA	X	A–F	Avg
Classes	21.04	21.37	25.80	30.21	22.99
Methods	9.70	11.31	10.27	11.45	10.48

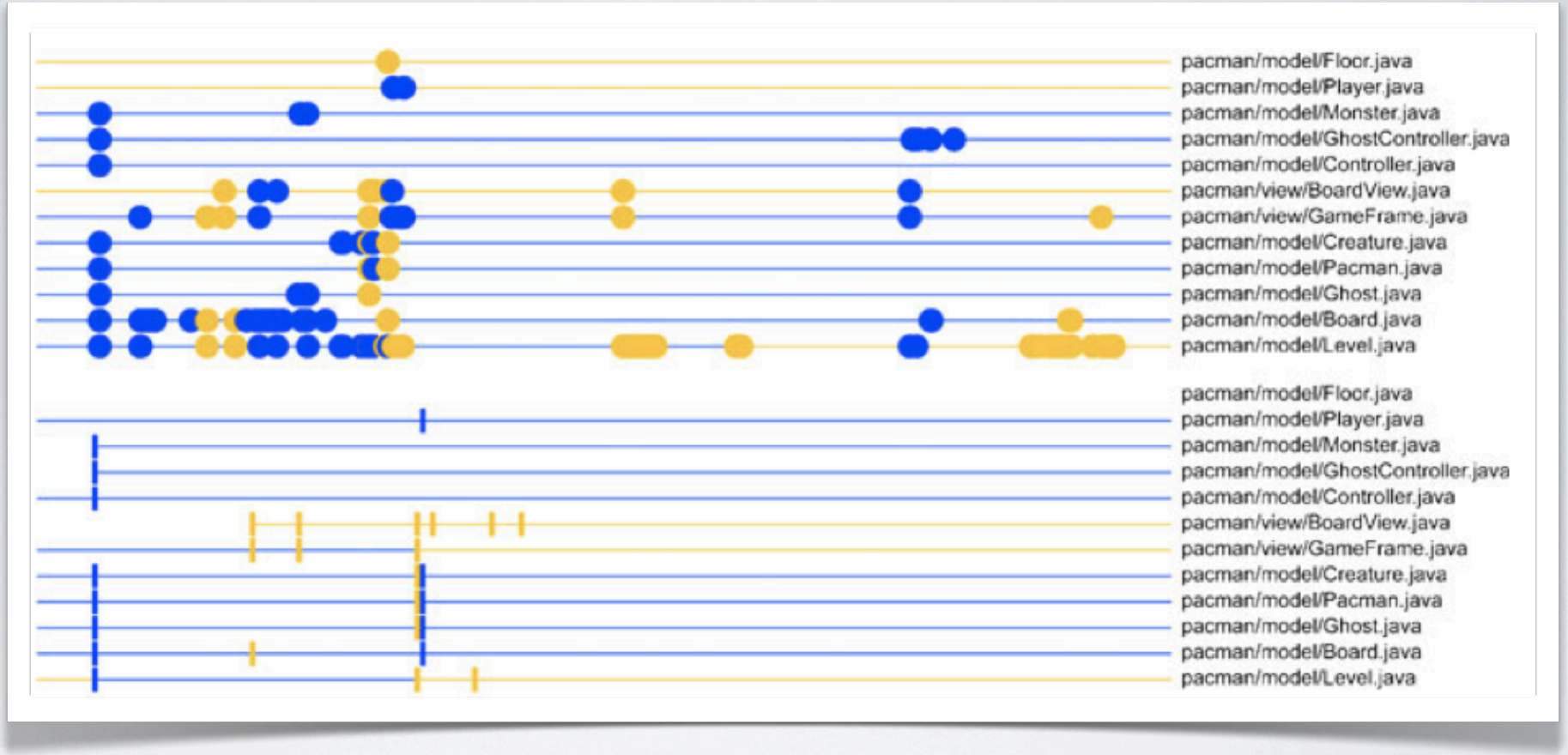
Table IV
RESULTS OF IMMEDIATE RULES MINING

Using CBSE to refine ownership metrics

- Ownership based on recorded change interactions is more precise than ownership based on SCM
  - Programmers sometimes do not commit for long time intervals (e.g. two weeks)



Difference between Syde and SVN logs



Refining code ownership with synchronous changes

Lile Palma Hattori · Michele Lanza · Romain Robbes

Combining navigation and change information to define a model of knowledge

$$\mathbf{DOK} = 3.293 + 1.098 * \mathbf{FA} + 0.164 * \mathbf{DL} - 0.321 * ln(1 + \mathbf{AC}) + 0.19 * ln(1 + \mathbf{DOI})$$

FA = First authorship
DL = Deliveries of changes
AC = Acceptance of changes
DOI = Degree of interest

A Degree-of-Knowledge Model to Capture Source Code Familiarity

Thomas Fritz, Jingwen Ou, Gail C. Murphy and Emerson Murphy-Hill
Department of Computer Science
University of British Columbia
Vancouver, BC, Canada

Three usages of the model were explored

- This model perform better than expertise recommenders, when compared with who developers think is the expert (based on two teams of developers)
- It also shows promising results when highlighting activity on defects relevant to a developer
- However this model does not correctly identify API elements, highlighting implementation elements instead

Using recorded interactions to improve bug prediction

- Developer submit Mylyn traces to the issue tracker of Eclipse; these were harvested
- 56 interaction metrics were computed based on Mylyn data, including:
  - number of selection, edition event
  - length of edition intervals, degree of interest

Micro Interaction Metrics for Defect Prediction

Taek Lee[†*], Jaechang Nam[§], DongGyun Han[§], Sunghun Kim[§], Hoh Peter In[†]

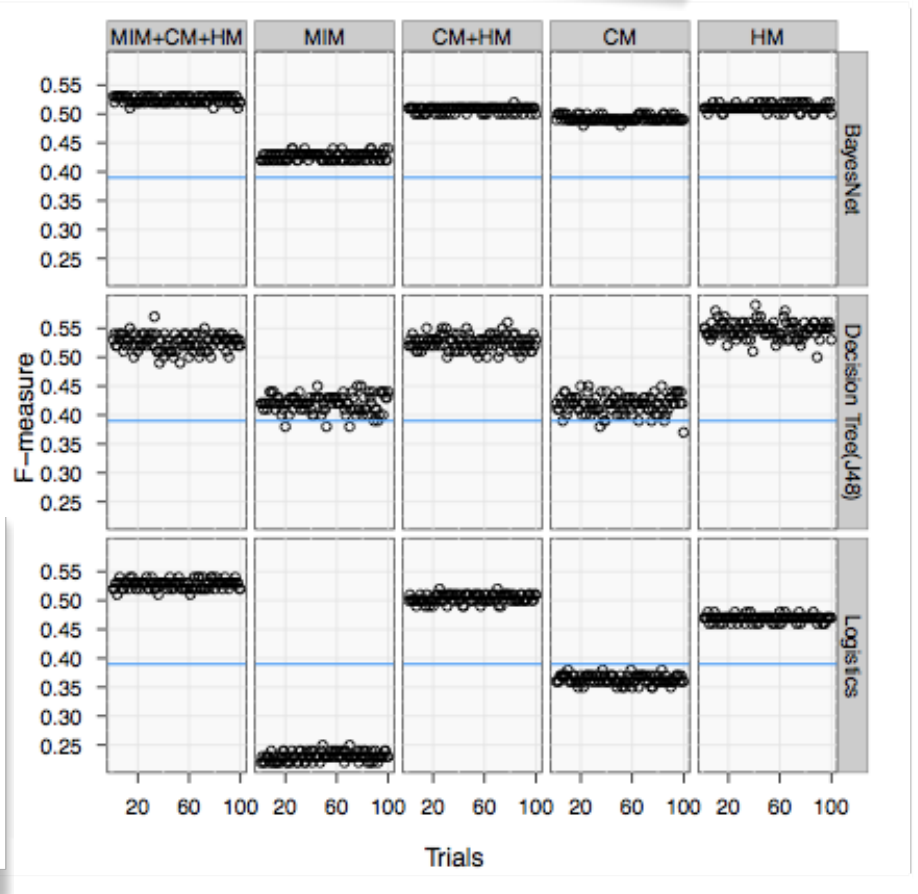
[†]Department of Computer Science and Engineering
Korea University, Seoul, Korea

Micro-interaction metrics improve bug prediction

Measures	MIM+CM+HM	MIM	CM+HM	CM	HM	DUMMY
BayesNet	**0.525**	0.43	0.50	0.49	0.51	0.39
Decision Tree	0.525	0.42	0.525	0.42	0.55	0.39
Logistics	**0.528**	0.23	0.50	0.36	0.47	0.39

“Among the top 56 important metrics for classification models, 44 (79%) are from MIMs.”

Subjects	# of instances (files)	% of defects
Mylyn	1061	14.3%
Team	239	35.5%
Etc.	1041	5.4%
All	2341	12.5%



Conclusions

What's in the future for MSR?

What do leaders in the field think?

roundtable

Editor: Dale Strok ■ dstrok@computer.org

Future of Mining Software Archives: A Roundtable

When mining software archives, we want to learn from the past to shape the future. But what does the research so far tell us about the future of the field itself? For this special issue, we invited and collected statements from nine research leaders in the field. These statements show opportunities for data collection and exploitation (Michael Godfrey, Ahmad Hassan, and James Herbsleb), enhancing programmer productivity (Gail Murphy and Martin Robillard), examining the role of social networking (Prem Devanbu), leveraging data for industry (Audris Mockus), and answering open research questions (Dewayne Perry). David Notkin, though, warns against too much enthusiasm: "Let us not mine for fool's gold." Enjoy!

—Nachiappan Nagappan, Andreas Zeller, and Thomas Zimmermann, Guest Editors

Answer Commonly Asked Project Questions

Michael W. Godfrey, *University of Waterloo*

Sensemaking is a term from cognitive psychology that concerns trying to understand a situation in the presence of ambiguous, contradictory, and complex data. The term is

it cost to develop feature X?" and "What modules are proving unusually problematic?"

Second, we need to do some persuading. We must convince tool designers to insert hooks into their tools that can record useful information as painlessly as possible. We must also persuade developers to better instrument their own practices. To do so, we must be able to argue that the return on investment of their time and effort will be rewarded with improved knowledge about their system's evolution. That's the goal of today's MSR research:

Selected quotes

We must convince tool designers to insert hooks into their tools that can record useful information as painlessly as possible. We must also persuade developers to better instrument their own practices. – Prof. Mike Godfrey

Other development tools, such as build scripts, debuggers, and code editors, will start logging their operations in repositories so that MSR researchers can explore how developers are using them. – Prof. Ahmed Hassan

While the MSR community must now generally settle for analyzing data generated by software tools created for other purposes, future tool design should consider data generation as an explicit design criterion. – Prof. Jim Herbsleb

As integrated-development environment tools become more standard, we can capture more details of how the development proceeds, although privacy concerns and the complexity of underlying data pose formidable challenges for anyone trying to use this information. – Dr. Audris Mockus

Future software repositories will likely contain much broader information about a software development project. – Prof. Gail Murphy

Recap

- There are several common threats to the validity of MSR studies that one should consider, address, or document.
- Construct validity threats stem from the imperfect quality of data coming from ad-hoc repositories
- Internal & construct validity can be affected by aggregation phenomena
- External validity threats rise from the limited amount of data under study (open-source projects, projects using a given type of repository)

Recap

- Since software repositories were not designed to be explicitly mined, data quality issues are prevalent. Some examples are:
  - Missing links between bugs and changes; bias in the type of linked bugs
  - Missing information, specifically finer-grained information between checkouts and commits
- Approaches have been developed to recover more data, but they can't recover lost information

Recap

- The usefulness of lost information has been shown, in the rare cases where it is available
  - Manually annotated bug fixes. This stresses the importance of manual inspections when confronted with imperfect data
  - Recorded IDE and change traces have applications in a lot of cases
- Researchers in the field hope that the next crop of repositories will contain more information

Recap of the recap

- There are several common threats to the validity of MSR studies that one should consider, address, or document.
- The main source of issues is the one of data quality; some work resolves the issues of current repositories, while other work examines alternatives, more precise software repositories.
- Manual, qualitative investigation is a worthwhile strategy to bypass some data quality issues, and uncover novel findings.
- We hope that these repositories will become the norm in the future.

Acknowledgements

- Alberto Bacchelli
- ICSM organizers
- Michele Lanza
- Mario Linarez Vasquez
- Richard Wettel

References

http://dl.dropbox.com/u/7172490/tutorial-references.txt