



# Get Your Hands Dirty with Mining Software Repositories in 175 Minutes

Massimiliano Di Penta

University of Sannio, Italy

[dipenta@unisannio.it](mailto:dipenta@unisannio.it)

<http://www.ing.unisannio.it/mdipenta>

# Purpose of this Tutorial

Use data from software repositories to:

- Study the evolution of software systems
- Provide useful recommendations to developers

Practical hints plus examples of applications



# Not about

- Machine learning and statistical data analysis
- Mining unstructured data  
(covered by Andi Marcus)



# Outline

Introduction to MSR

Types of software repositories -  
How to mine them, promises and perils

Examples of applications

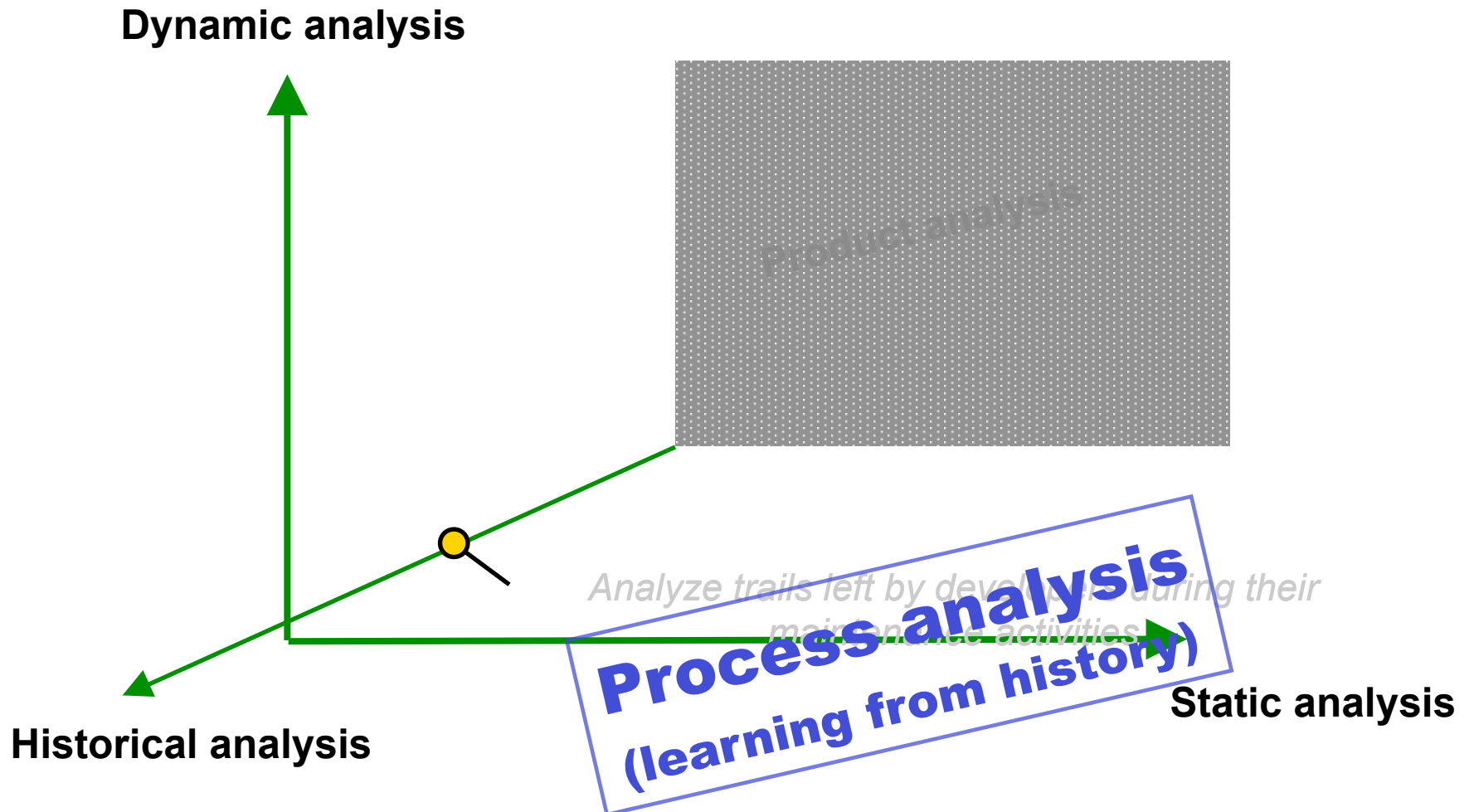
Key ingredients for conducting MSR research

# Introduction to Mining Software Repositories



# Key elements of Mining Software Repositories

# Historical analysis



# Historical analysis

Static and dynamic analysis do not capture information such as:

**How** does an artifact change during the time?

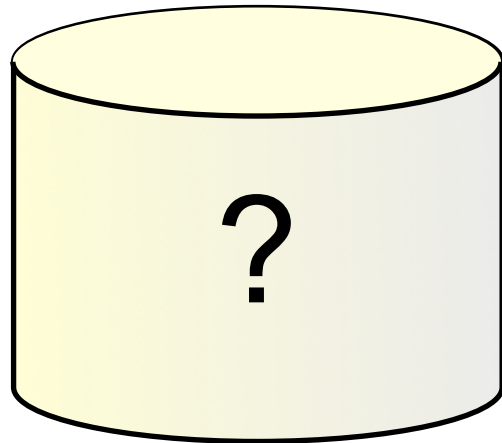
**When** was it changed?

**Who** changed it?

**Why** was it changed?

**What** artifacts changed together?

# What is a software repository?



Any data store containing artifacts regarding the life cycle of one (or more) software projects

**Versioning systems:** store changes to the code (e.g. CVS, SVN, git, Mercurial)

**Issue tracking systems:** follow the resolution of defects/requests for changes (e.g. Bugzilla, Jira)

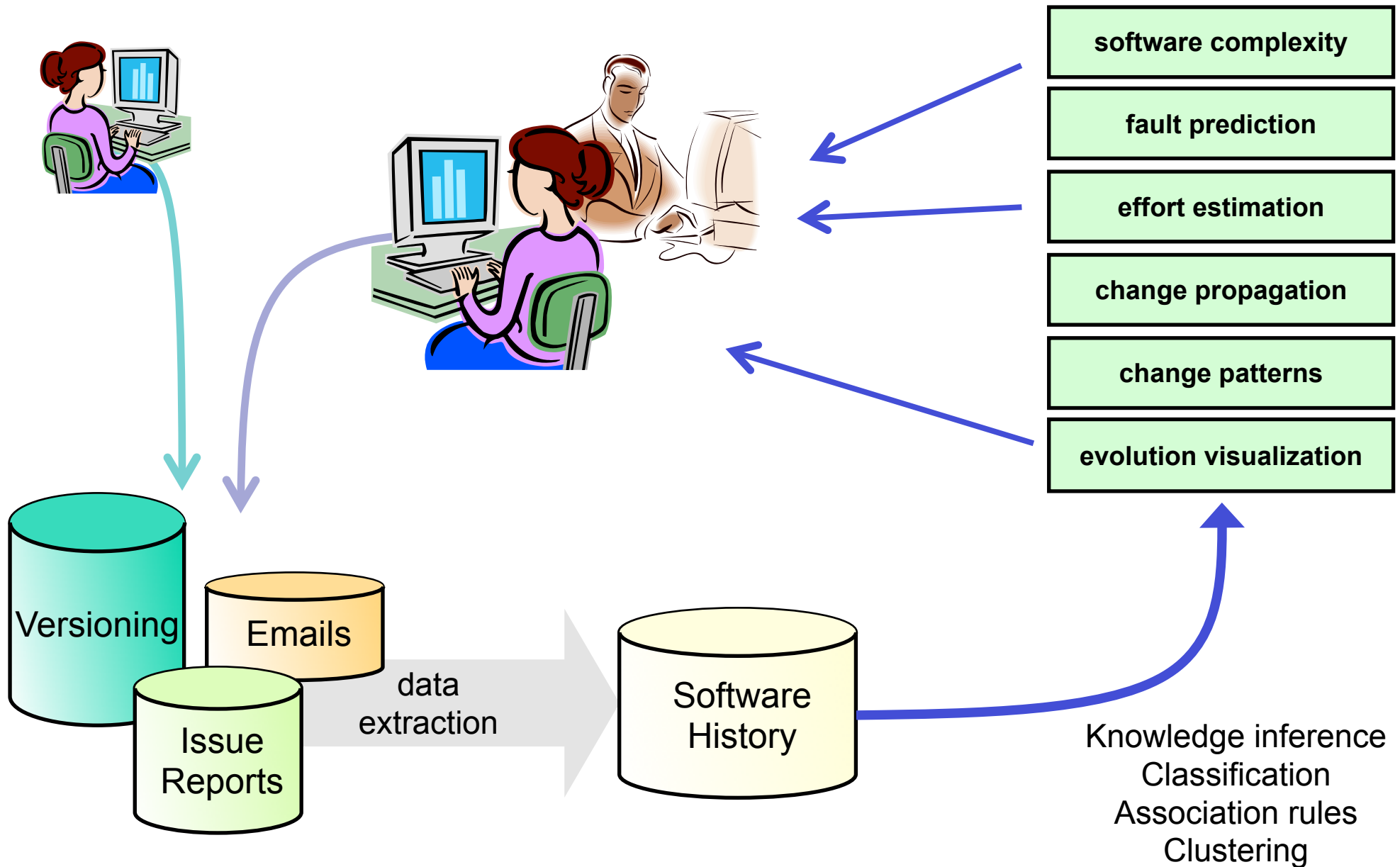
**Archived communication:** record rationale for decisions throughout the life of a project (e.g. Mailing Lists, Forums, Chat Logs)

**Runtime data:** system logs, execution traces, power monitor data

**Code repositories:** forges, code search engines

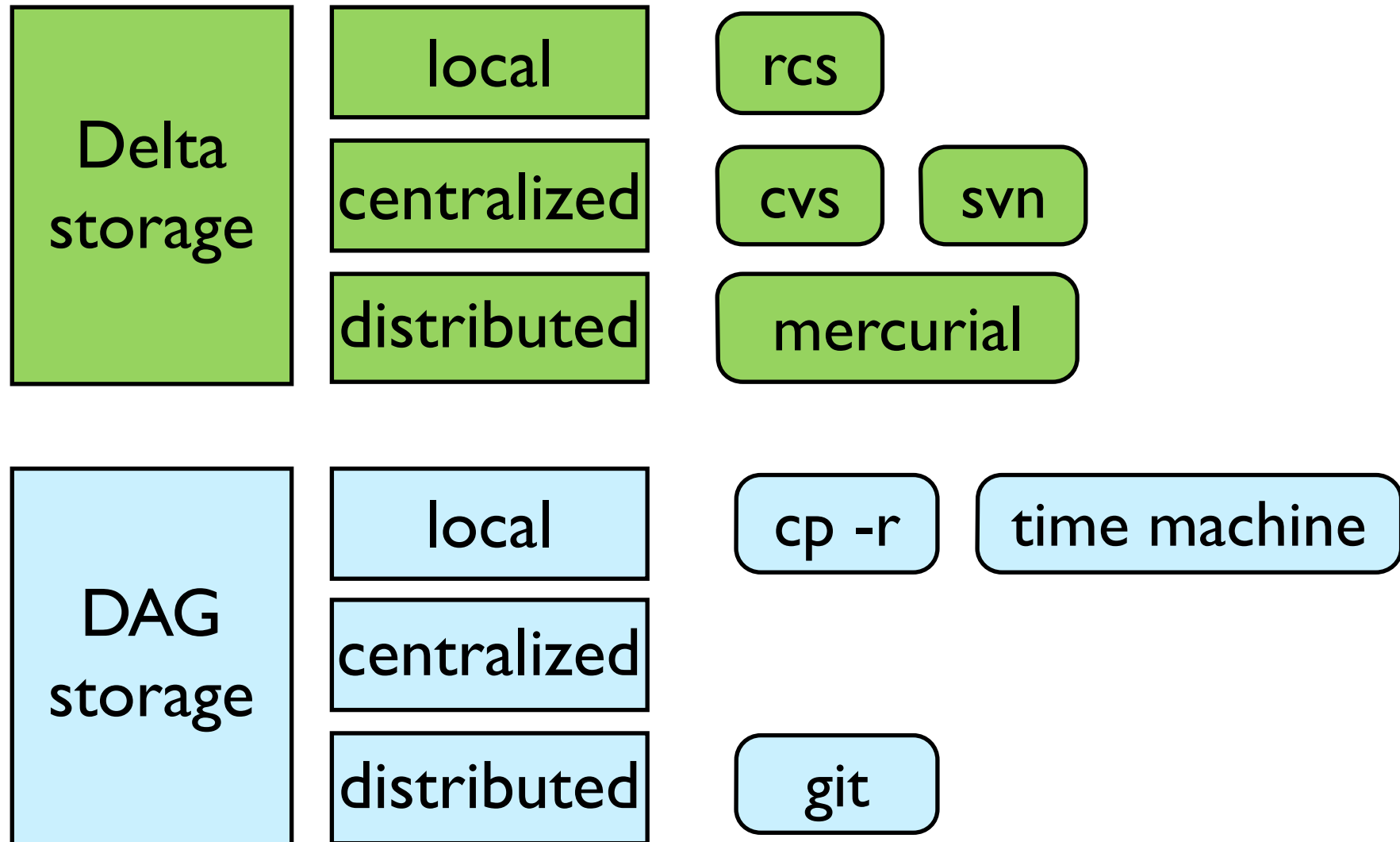


# Mining Software Repositories



# Versioning Systems

# Taxonomy

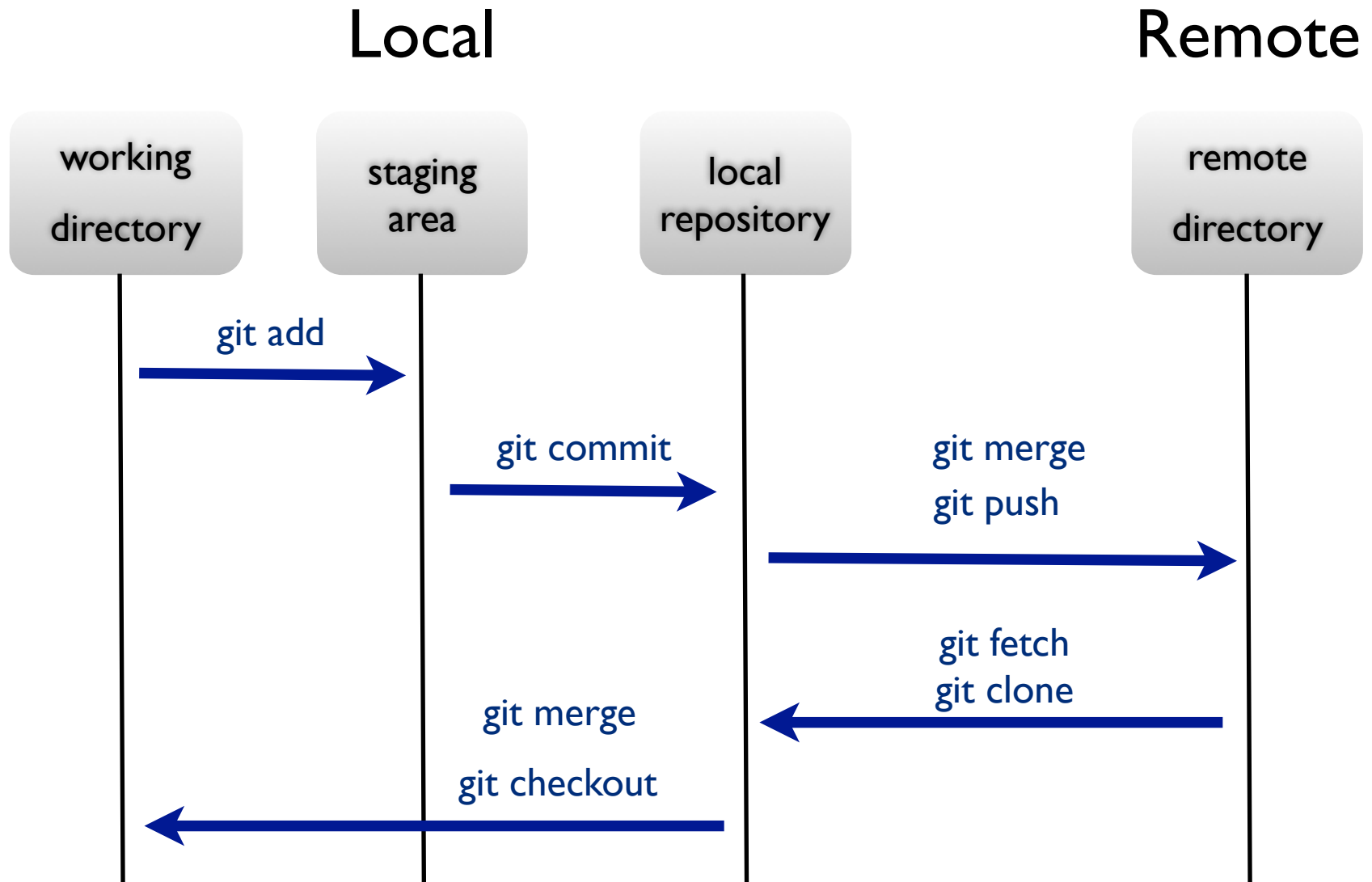


# git

<http://git-scm.com>

- Developed by Linus Torvalds as a tool for managing the Linux Kernel evolution
- Before Linux did not use a versioning system
- Entire kernel development based on patches

# Git usage



# Promises and Perils

## 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

### Abstract

We are now witnessing the rapid growth of decentralized source code management (DSCM) systems, in which every developer has her own repository. DSCMs facilitate a style of collaboration in which work output can flow sideways (and privately) between collaborators, rather than always up and down (and publicly) via a central repository. Decentralization comes with both the **promise** of new data and the **peril** of its misinterpretation. We focus on git, a very popular DSCM used in high-profile projects. Decentralization, and other features of git, such as automatically recorded contributor attribution, lead to richer content histories, giving rise to new questions such as “How do contributions flow between developers to the official project repository?” However, there are pitfalls. Commits may be reordered, deleted, or edited as they move between repositories. The semantics of terms common to SCMs and DSCMs sometimes differ markedly, potentially creating confusion. For example, a commit is immediately visible to all developers in centralized SCMs, but not in DSCMs. Our goal is to help researchers interested in DSCMs avoid these and other perils when mining and analyzing git data.

*Out of a stem that scored the hand  
I wrung it in a weary land.*

A. E. Housman, A Shropshire Lad

### 1. Introduction

Since the turn of the century, researchers have taken advantage of the data found in SCM repositories that has been made freely available for Open Source Software (OSS) projects. This data has been used to reconstruct the process by which the software was created [1], [2]. Researchers have also used this data to create recommender systems [3], [4], [5], study evolution patterns [6], [7], [8], predict bugs [9], [10], [11], and examine collaboration [12], [13], [14].

The number of software projects using DSCMs has increased, and looks set to continue to do so. Figure 1 shows

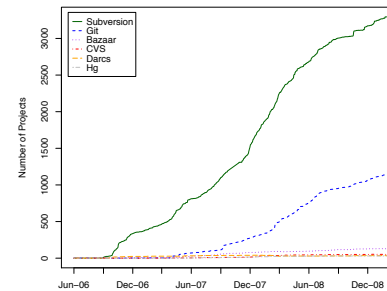


Figure 1. The Debian Project's Use of SCMs.

the number of projects with Debian packages that report using a given SCM over time<sup>1</sup>. As of February, 2009, 36% of the packages include SCM information. Although incomplete, this data gives a strong indication that git is second only to SVN in use and that its use is growing. Indeed, git has also been adopted by a number of high profile OSS projects such as X.org, Ruby on Rails, Wine, Samba, Perl, and the Glasgow Haskell Compiler.

The repositories of these and other projects are of interest to researchers, but their data differs in important ways from that which is found in their centralized counterparts.

Massey and Packard [15] have presented a method of converting CSCMs to git for mining data. However, to our knowledge, only one paper [16] has examined data mined from a git-based project. This paper presented results of analysis of data drawn from the Linux git repository. We have also found one article in the *Linux Weekly News* that uses data mined from git to track how patches find their way into the stable main line linux tree from subsystem git repositories [17]. Neither the paper nor the article addresses the core differences between git and centralized SCMs,

<sup>1</sup> According to data provided by projects using the `vcsc-` (SCM-) headers introduced to Debian package descriptions in 2006.

# Promises - I

Mining more history from distributed codebases



# Promises - II

A richer history can be mined, including implicit and explicit branches, merge points, pulls from other repositories

# Promises - III

Local storage of history

# Promises - IV

Faster than other SCM and less space used

# Promises - V

Easy migration from other SCM

# Promises - VI

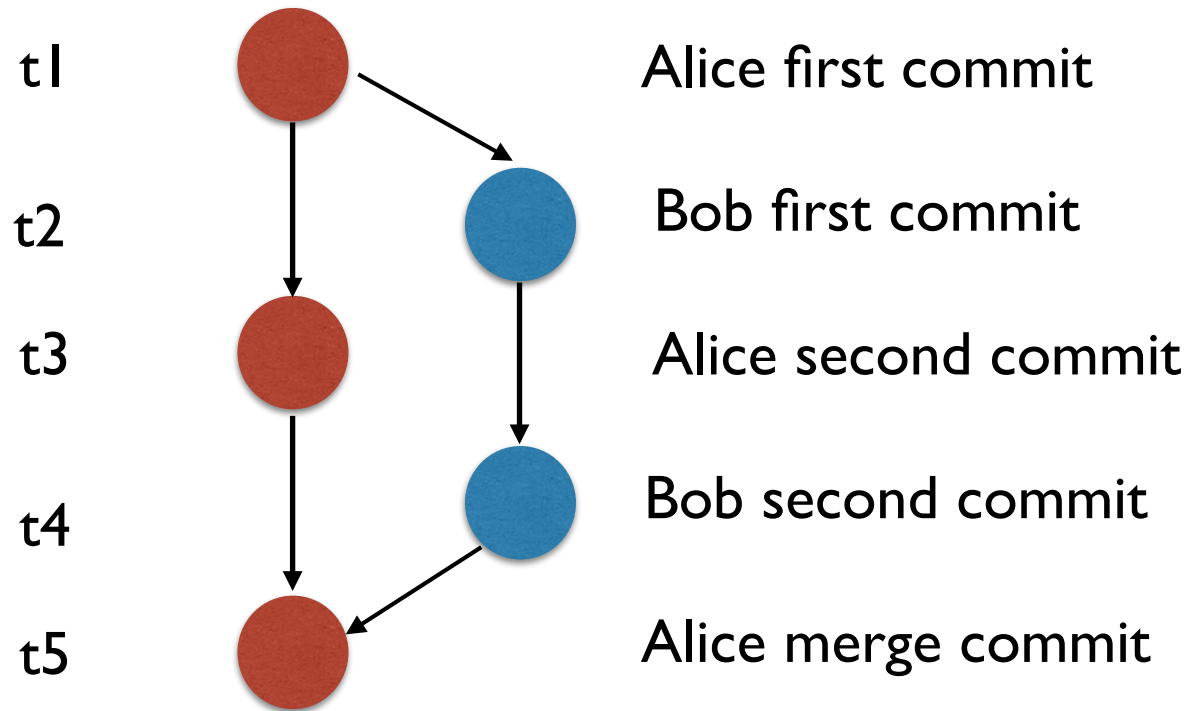
Track author information, not just committers



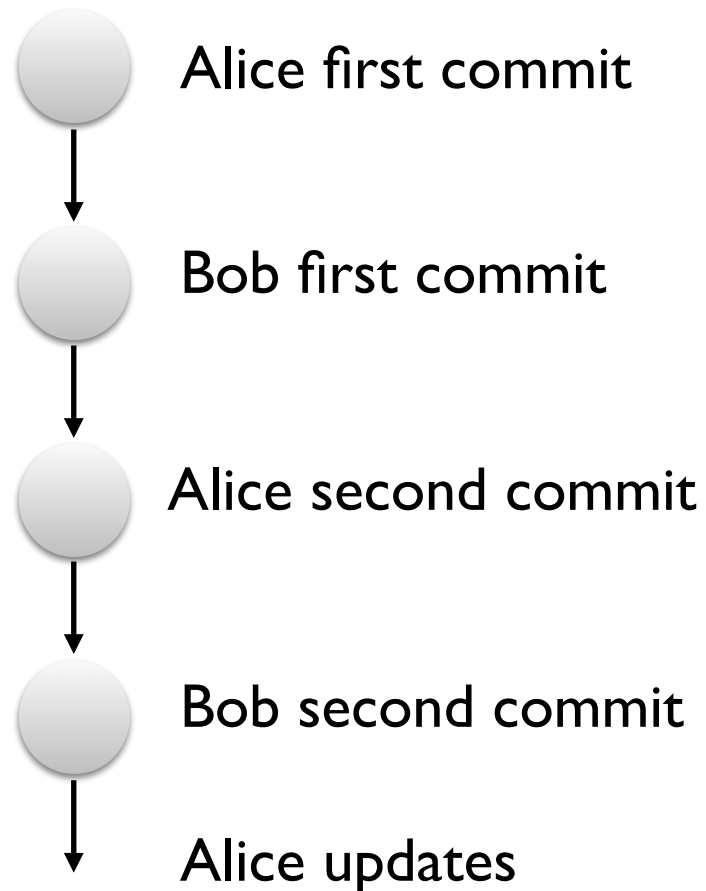
# Perils - I

Implicit branches

# Implicit Branch



# How it would be in SVN







# Perils - II

No linear history, but paths in the DAG



All branches



View

Branch

Create Branch

Search

Subject	Author	Date
○ ndr_compression: change debug levels	Stefan Metzmacher	2008-09-06 10:55:04
○ ndr_compression: use deflateReset() together with defalte...	Stefan Metzmacher	2008-09-06 16:16:00
○ ndr_compression: use inflateReset() and inflateSetDictiona...	Stefan Metzmacher	2008-09-05 20:18:07
○ Don't compare identity, it'll never be different. Jeremy.	Jeremy Allison	2008-09-06 06:47:06
○ Added tests that show that write time update is immediate...	Jeremy Allison	2008-09-05 23:24:36
○ Merge branch 'v4-0-test' of ssh://git.samba.org/data/git/s...	Andrew Tridgell	2008-09-04 04:49:29
○ Merge branch 'v4-0-test' of ssh://git.samba.org/data/git/...	Andrew Tridgell	2008-08-29 23:38:02
○ Add a setexpiry operation in samdb.py	Andrew Tridgell	2008-08-29 23:32:44
○ added a simple script for setting password expiry	Andrew Tridgell	2008-08-29 23:23:06
○ Simplify SetSecrets behaviour in line with RPC-LSA an...	Andrew Bartlett	2008-09-08 04:46:04
○ Try to implement the right logic for systemFlags	Andrew Bartlett	2008-09-08 03:10:24
○ Don't expose passwords, even to the administrator.	Andrew Bartlett	2008-09-08 03:09:02
○ More work towards trusted domains support in Samba...	Andrew Bartlett	2008-09-08 02:55:34
○ Remove <tab> in OpenLDAP MMR config	Oliver Liebel	2008-09-06 05:12:19
○ Make SMB signing work with Windows 2008 and kerberos.	Andrew Bartlett	2008-09-06 01:07:41
○ Add a new error code	Andrew Bartlett	2008-09-05 08:46:12
○ Update copyright	Andrew Bartlett	2008-09-05 08:45:58
○ Update copyright, I've been working here many long years...	Andrew Bartlett	2008-09-05 08:45:37
○ Move our DC to implement mandatory signing.	Andrew Bartlett	2008-09-05 08:45:10
○ With a windows 2008 client, even anonymous requires...	Andrew Bartlett	2008-09-05 08:24:44
○ More work to implement LSA CreateTrustedDomainEx2	Andrew Bartlett	2008-09-04 08:06:38
○ Merge commit 'origin/v4-0-test' into trusted-domains	Andrew Bartlett	2008-09-04 03:32:32
○ Regenerate SWIG file.	Jelmer Vernooij	2008-09-03 22:55:24
○ Avoid using version call for version string.	Jelmer Vernooij	2008-09-03 22:29:53
○ Allow overriding shared library policy using environmen...	Jelmer Vernooij	2008-09-03 22:26:02
○ Fix embedding of Samba 4.	Jelmer Vernooij	2008-09-03 14:10:35
○ Merge branch 'v4-0-test' of ssh://git.samba.org/data/gi...	Andrew Bartlett	2008-09-03 07:34:44
○ Implement NETLOGON PAC verification on the server-side	Andrew Bartlett	2008-09-03 07:30:17
○ Merge krb5_cksumtype_to_enctype from Heimdal svn...	Andrew Bartlett	2008-09-03 06:20:30
○ Test a few more error cases in RPC-PAC	Andrew Bartlett	2008-09-03 06:19:16

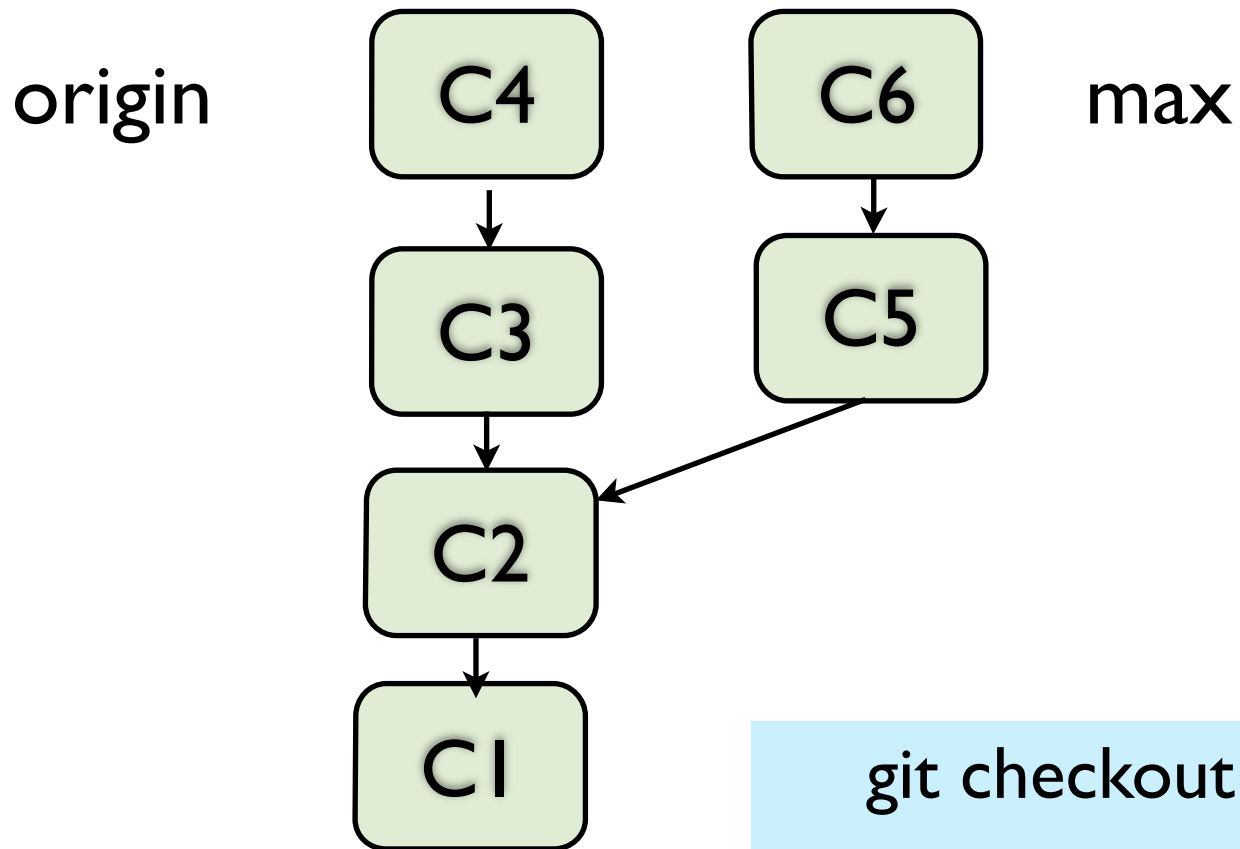




# Perils - III

Repository owner can rewrite the history through rebasing

# Rebasing



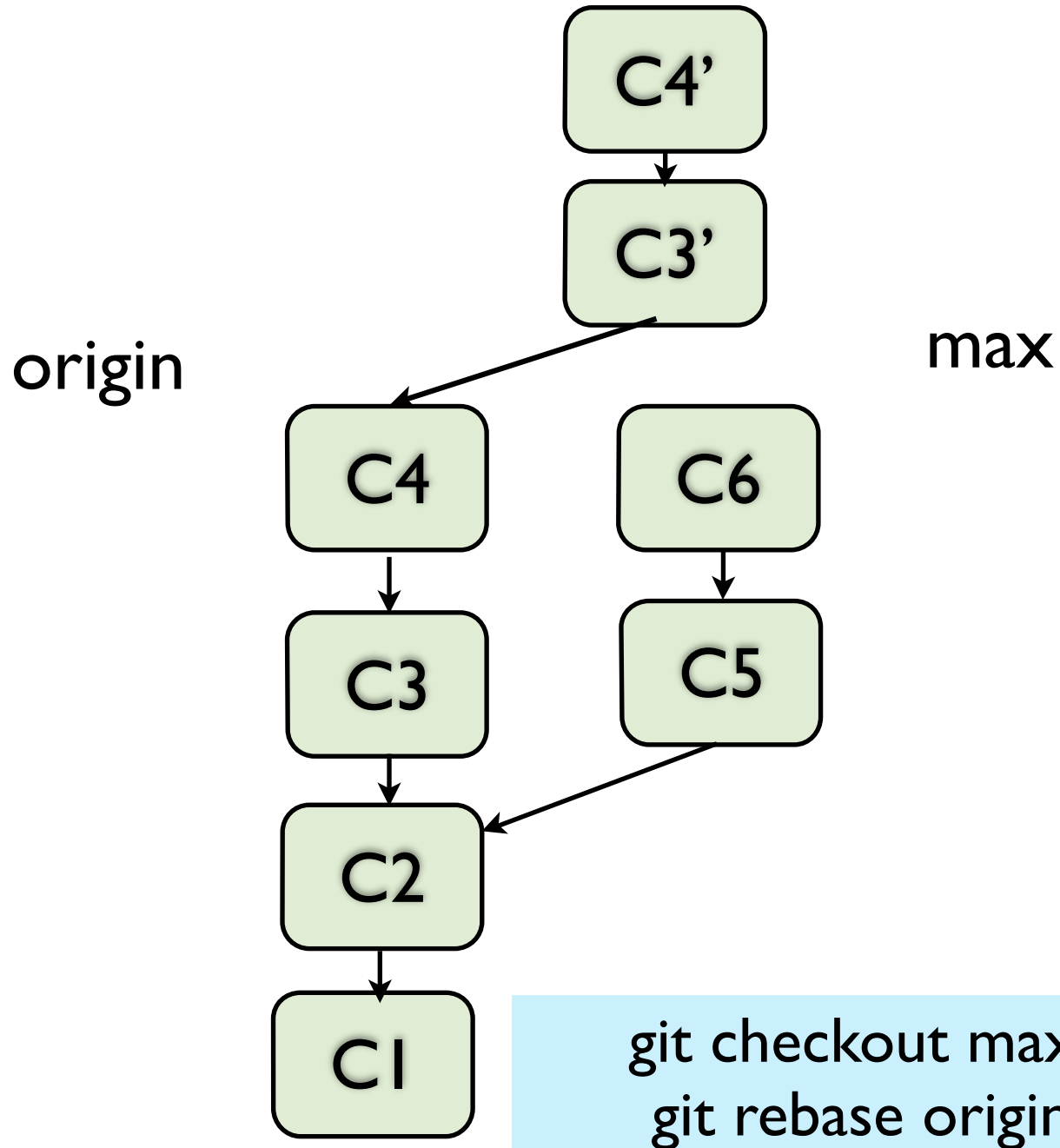
```
git checkout -b max origin
```

...

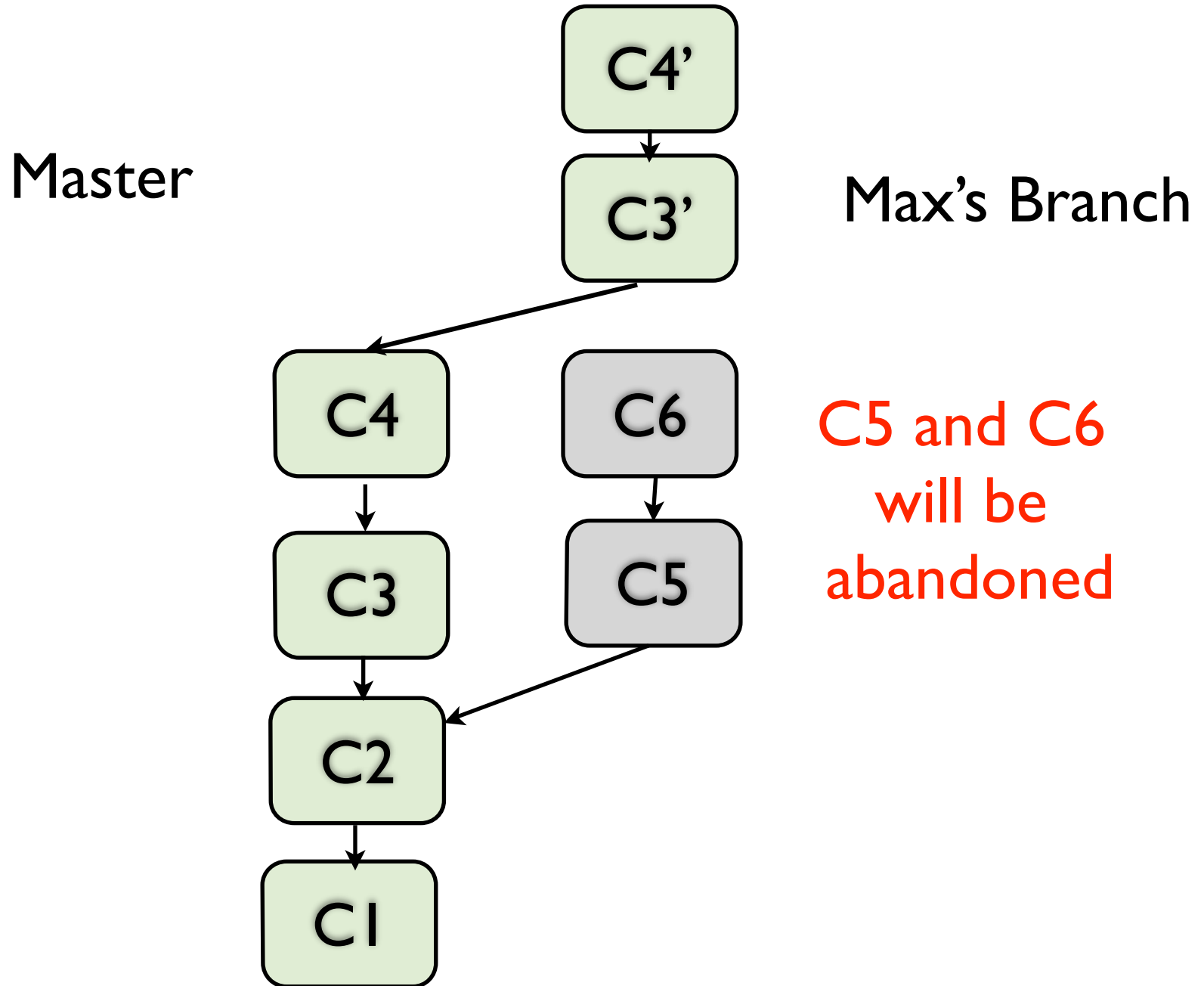
```
git commit
```

...

# Rebasing



# Rebasing



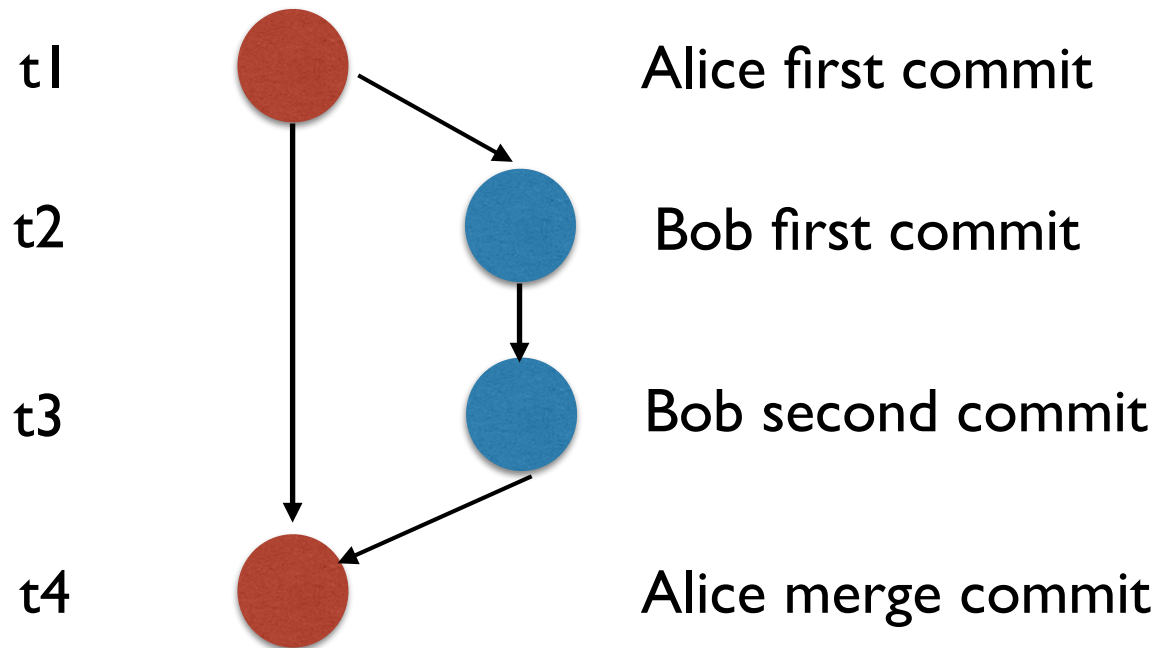


# Perils - IV

Cannot determine whether a merge occurred in a commit, and its source

Depends on the use or not of fast forward merges

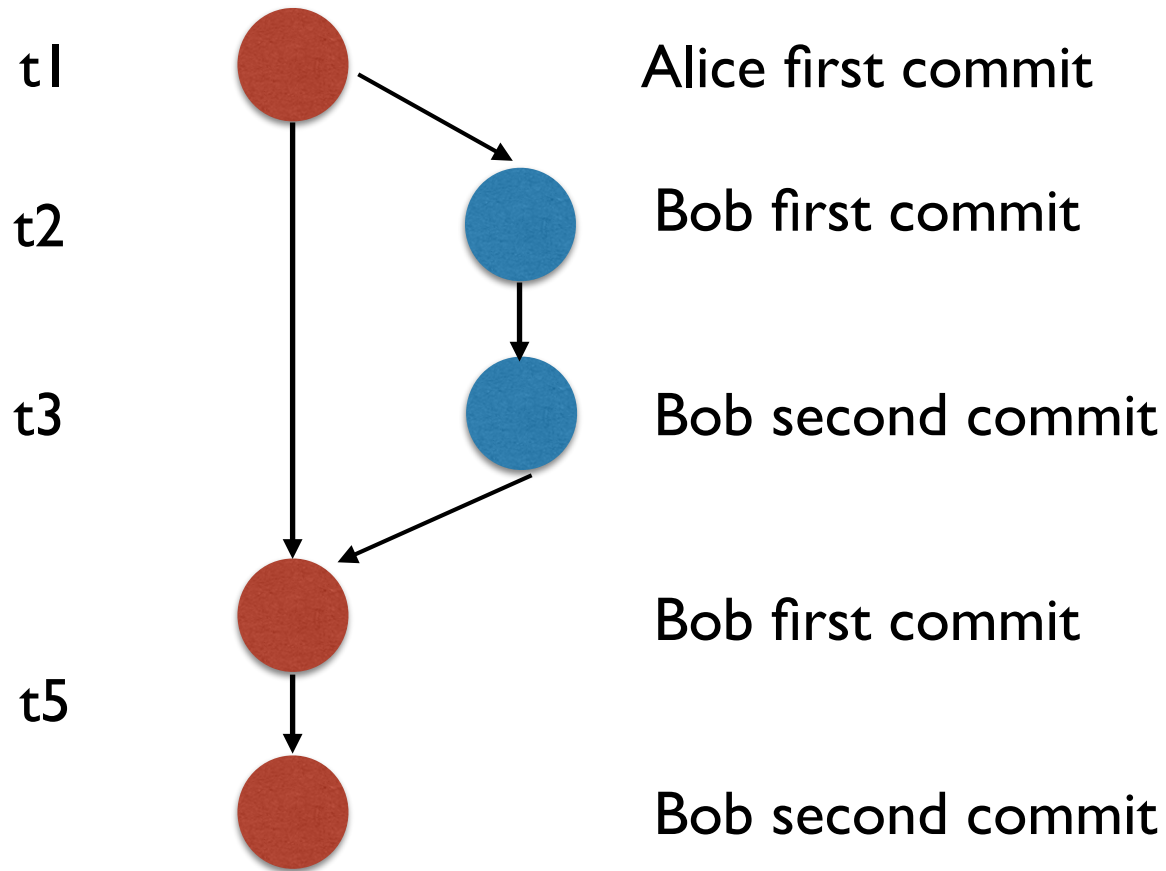
# Avoiding fast forward



Generated commit message:  
“Merge BobBranch into AliceBranch”



# Avoiding fast forward



# git - Cloning a Repository

```
git clone <URL>
```

e.g.

```
git clone https://git.samba.org/samba.git
```

Import from CVS/SVN also possible

# Import from CVS/SVN

**CVS Import:** `git cvsimport -v -d <CVSROOT> <module>`

e.g.

```
git cvsimport -v -d  
:pserver:anonymous@cvs.drupal.org:/cvs/drupal-contrib  
contrib
```

**SVN Import:** `git svn clone <URL>`

e.g.

```
git svn clone  
http://argouml.tigris.org/svn/argouml/trunk
```

# getting the log

```
git log --date=iso --stat HEAD
```

```
git log --name-status -date=iso --stat HEAD
```

commit e0e415e9877ec80db10b764ed969081c9ca91af5

Author: Eric Covener <[covener@apache.org](mailto:covener@apache.org)>

Date: 2013-03-04 21:54:24 +0000

PR54587: LDAP connections used for authn were not respecting  
LDAPConnectionPoolTimeout due to confusion over what "bound" means.

Added some LDAP trace at TRACE5 to track how LDAP connections are  
reused and rebound.

git-svn-id: <https://svn.apache.org/repos/asf/httpd/httpd/trunk@1452551>  
13f79535-47bb-0310-9956-ffa450edef68

include/ap_mmn.h		3	+-
include/util_ldap.h		2	++
modules/ldap/util_ldap.c		43	+++++++-----

3 files changed, 40 insertions(+), 8 deletions(-)

# git log --name-status

commit b911f3733c458dafacb90ee6b48de1f29c39a363

Author: Jim Jagielski <[jim@apache.org](mailto:jim@apache.org)>

Date: Tue Mar 5 17:29:28 2013 +0000

Rough start for simple, tunneling websocket proxy support.  
Compiles at this stage and that's all I know :)

git-svn-id: [https://svn.apache.org/repos/asf/httpd/httpd/  
trunk@1452911](https://svn.apache.org/repos/asf/httpd/httpd/trunk@1452911) 13f79535-47bb-0310-9956-ffa450edef68

M modules/proxy/config.m4

A modules/proxy/mod\_proxy\_websocket.c

# Some git log options

- Only the merges on the master:

```
git log --merges
```

- Exclude merges

```
git log --no-merges
```

- Only some branches:

```
git log master
```

```
git log exp
```

# git log - changed files

```
git log --oneline --name-status
```

bdd2551 modified file b in the exp

M b.txt

8c3fa37 modified file a in the master

M a.txt

2ca854b added file b

A b.txt

327262c added file a

A a.txt

427afe8 Initial commit



# git whatchanged

commit b911f3733c458dafacb90ee6b48de1f29c39a363

Author: Jim Jagielski <[jim@apache.org](mailto:jim@apache.org)>

Date: 2013-03-05 17:29:28 +0000

Rough start for simple, tunneling websocket proxy support.

Compiles at this stage and that's all I know :)

git-svn-id: <https://svn.apache.org/repos/asf/httpd/httpd/trunk@1452911>  
13f79535-47bb-0310-9956-ffa450edef68

:100644 100644 a5535a2... 0878363... M modules/proxy/config.m4

:000000 100644 0000000... 1a6b694... A modules/proxy/mod\_proxy\_websocket.c

# git whatchanged -p

commit 16d43c6001cd16a8117d57270cc7fefc443dbb14

Author: Jim Jagielski <[jim@apache.org](mailto:jim@apache.org)>

Date: Tue Mar 5 21:38:01 2013 +0000

Work around blocking issues...

git-svn-id: <https://svn.apache.org/repos/asf/httpd/httpd/trunk@1453022> 13f79535-47bb-0310-9956-ff

```
diff --git a/modules/proxy/mod_proxy_websocket.c b/modules/proxy/mod_proxy_websocket.c
```

```
index 10ea8ec..c233070 100644
```

```
--- a/modules/proxy/mod_proxy_websocket.c
```

```
+++ b/modules/proxy/mod_proxy_websocket.c
```

```
@@ -123,7 +123,7 @@ static int proxy_websocket_transfer(request_rec *r, conn_rec *c_i, conn_rec *c_o
```

```
        "error on %s - ap_pass_brigade",
```

```
        name);
```

```
    }
```

```
-    } else if (!APR_STATUS_IS_EAGAIN(rv)) {
```

```
+    } else if (!APR_STATUS_IS_EAGAIN(rv) && !APR_STATUS_IS_EOF(rv)) {
```

```
        ap_log_rerror(APLOG_MARK, APLOG_DEBUG, rv, r, APLOGNO())
```

```
            "error on %s - ap_get_brigade",
```

```
            name);
```

# Pretty printing

```
git log --pretty=format:"%h - %an, %ar : %s"
```

```
ca82a6d - Scott Chacon, 11 months ago : changed the version number
```

```
085bb3b - Scott Chacon, 11 months ago : removed unnecessary test code
```

```
a11bef0 - Scott Chacon, 11 months ago : first commit
```

<code>%H</code> Commit hash
<code>%h</code> Abbreviated commit hash
<code>%T</code> Tree hash
<code>%P</code> Parent hashes
<code>%p</code> Abbreviated parent hashes
<code>%an</code> Author name
<code>%ae</code> Author e-mail
<code>%ad</code> Author date
<code>%ar</code> Author date, relative
<code>%cn</code> Committer name
<code>%cd</code> Committer date
<code>%cr</code> Committer date, relative
<code>%s</code> Subject

# git ID?

- A commit is identified by a SHA hash
- When using such IDs in git commands you don't need the full IDs
- Often the first 4-5 characters suffice, provided they are unique
- To obtain a log with the shortest commit id:

```
git log --abbrev-commit
```

# Showing file revision

## Show a given revision

```
git show <commitID>:<fname>
```

```
git show 62f17cf15e1:modules/proxy/mod_proxy_http.c
```

## Previous revision

```
git show <commitID>^:<fname>
```

```
git show 62f17cf15e1^:modules/proxy/mod_proxy_http.c
```

# ..and downloading it

```
git checkout <commitID> <file>
```

e.g.

```
git checkout 62f17cf15e1modules/proxy/mod_proxy_http.c
```

```
git checkout 62f17cf15e1^ modules/proxy/mod_proxy_http.c
```

# Comparing them...

```
git diff <commitID>
```

```
git diff <commitID1>:<file> <commitID2>:<file>
```

```
git diff <commitID>^:<file> <commitID>:<file>
```

# blame

Given a file revision, for each line it reports when such a line was last modified, and by whom

```
git blame <filename>
```

or

```
git blame -w <filename>
```

(-w ignores whitespace changes)



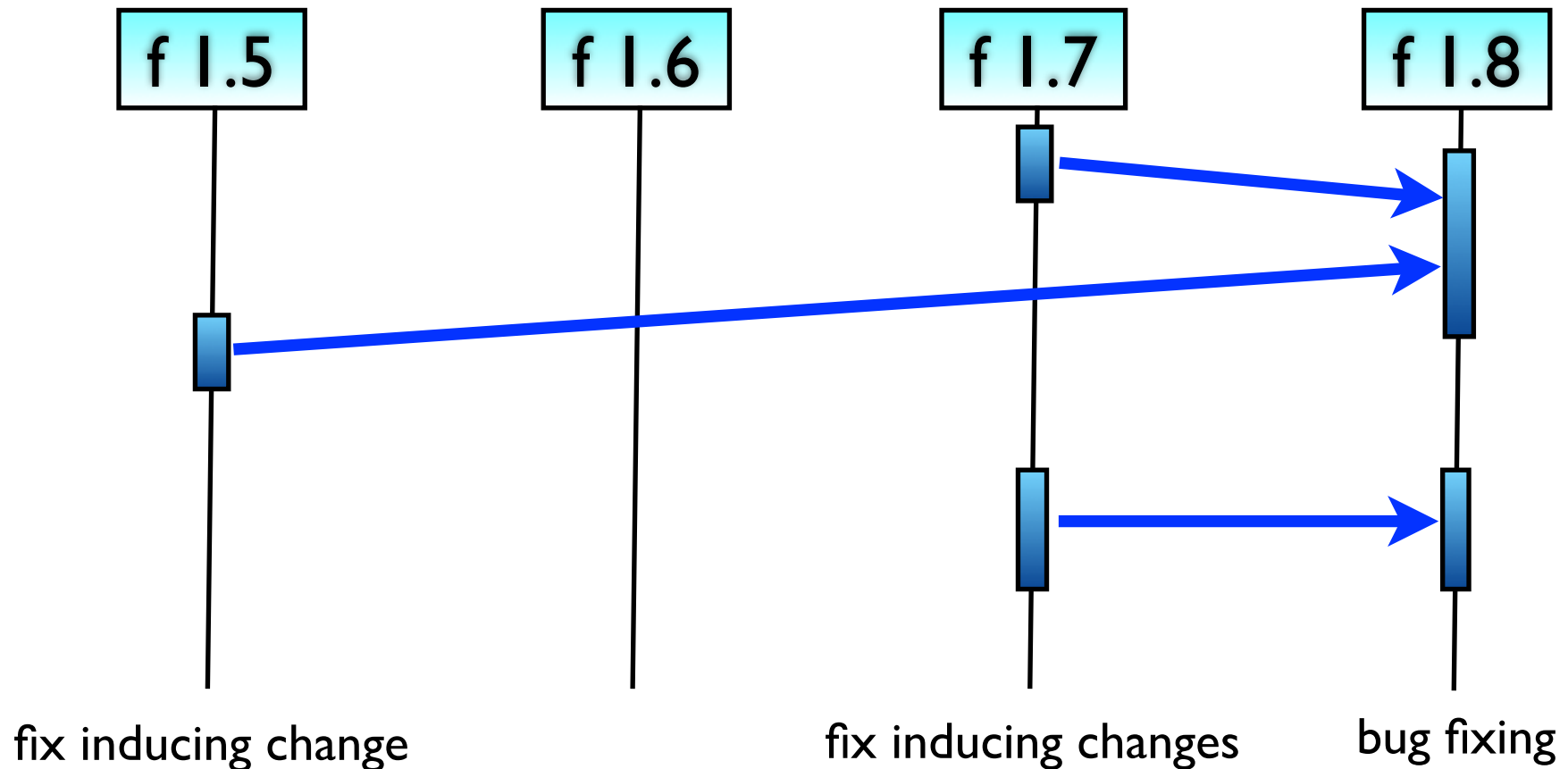
# git blame modules/proxy/mod\_proxy\_http.c

```
e17a716f (Roy T. Fielding 2006-07-11 20:33:53 +0000 1)/ * Licensed to the Apache Software Foundation (ASF) under one or more
e17a716f (Roy T. Fielding 2006-07-11 20:33:53 +0000 2) * contributor license agreements. See the NOTICE file distributed with
e17a716f (Roy T. Fielding 2006-07-11 20:33:53 +0000 3) * this work for additional information regarding copyright ownership.
e17a716f (Roy T. Fielding 2006-07-11 20:33:53 +0000 4) * The ASF licenses this file to You under the Apache License, Version 2.0
e17a716f (Roy T. Fielding 2006-07-11 20:33:53 +0000 5) * (the "License"); you may not use this file except in compliance with
e17a716f (Roy T. Fielding 2006-07-11 20:33:53 +0000 6) * the License. You may obtain a copy of the License at
5d855a48 (Roy T. Fielding 1999-08-24 06:55:44 +0000 7) *
c4ecf8b7 (Andre Malo 2004-02-06 22:58:42 +0000 8) * http://www.apache.org/licenses/LICENSE-2.0
5d855a48 (Roy T. Fielding 1999-08-24 06:55:44 +0000 9) *
c4ecf8b7 (Andre Malo 2004-02-06 22:58:42 +0000 10) * Unless required by applicable law or agreed to in writing, software
c4ecf8b7 (Andre Malo 2004-02-06 22:58:42 +0000 11) * distributed under the License is distributed on an "AS IS" BASIS,
c4ecf8b7 (Andre Malo 2004-02-06 22:58:42 +0000 12) * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
c4ecf8b7 (Andre Malo 2004-02-06 22:58:42 +0000 13) * See the License for the specific language governing permissions and
c4ecf8b7 (Andre Malo 2004-02-06 22:58:42 +0000 14) * limitations under the License.
5d855a48 (Roy T. Fielding 1999-08-24 06:55:44 +0000 15) */
5d855a48 (Roy T. Fielding 1999-08-24 06:55:44 +0000 16)
5d855a48 (Roy T. Fielding 1999-08-24 06:55:44 +0000 17)/* HTTP routines for Apache proxy */
5d855a48 (Roy T. Fielding 1999-08-24 06:55:44 +0000 18)
5d855a48 (Roy T. Fielding 1999-08-24 06:55:44 +0000 19)#include "mod_proxy.h"
6f67de98 ( Nick Kew 2007-10-09 15:59:32 +0000 20)#include "ap_regex.h"
5d855a48 (Roy T. Fielding 1999-08-24 06:55:44 +0000 21)
47716559 (Graham Leggett 2001-04-13 23:56:04 +0000 22)module AP_MODULE_DECLARE_DATA proxy_http_module;
47716559 (Graham Leggett 2001-04-13 23:56:04 +0000 23)
73ffcd0f (William A. Rowe Jr 2004-08-11 22:43:44 +0000 24)static apr_status_t ap_proxy_http_cleanup(const char *scheme,
73ffcd0f (William A. Rowe Jr 2004-08-11 22:43:44 +0000 25) request_rec *r,
56a7c0d8 (Graham Leggett 2002-03-09 07:15:33 +0000 26) proxy_conn_rec *backend);
```

# Fix-inducing changes

Jacek Sliwerski, Thomas Zimmermann, Andreas Zeller: When do changes induce fixes? MSR 2005

Sunghun Kim, Thomas Zimmermann, Kai Pan, E. James Whitehead Jr.: Automatic Identification of Bug-Introducing Changes. ASE 2006: 81-90





# Using changes to predict fault-proneness

Machine learning algorithm (SVM)

Independent variables: features of the change delta

Feature Group	Example
Added Delta	If, while, for, ==
Deleted Delta	True, 0, <, ++, int
Directory/File Name	Src, module, java
Change Log	Fix, added, new
New Revision Source Code	If,   , != do, while, string, false
Metadata	Author: hunkim, commit hour:12
Complexity Metrics	LOC: 34, Cyclomatic: 10



# Does a refactoring induce bugs?

## When does a Refactoring Induce Bugs? An Empirical Study

Gabriele Bavota<sup>1</sup>, Bernardino De Carluccio<sup>1</sup>, Andrea De Lucia<sup>1</sup>  
Massimiliano Di Penta<sup>2</sup>, Rocco Oliveto<sup>3</sup>, Orazio Strollo<sup>1</sup>

<sup>1</sup>University of Salerno, Fisciano (SA), Italy

<sup>2</sup>University of Sannio, Benevento, Italy

<sup>3</sup>University of Molise, Pesche (IS), Italy

gbavota@unisa.it, bernardino.decarluccio@gmail.com, adelucia@unisa.it

dipenta@unisannio.it, rocco.oliveto@unimol.it, oraziostrolle@hotmail.com

**Abstract**—Refactorings are—as defined by Fowler—behavior preserving source code transformations. Their main purpose is to improve maintainability or comprehensibility, or also reduce the code footprint if needed. In principle, refactorings are defined as simple operations so that are “unlikely to go wrong” and introduce faults. In practice, refactoring activities could have their risks, as other changes.

This paper reports an empirical study carried out on three Java software systems, namely Apache Ant, Xerces, and ArgoUML, aimed at investigating to what extent refactoring activities induce faults. Specifically, we automatically detect (and then manually validate) 15,008 refactoring operations (of 52 different kinds) using an existing tool (Ref-Finder). Then, we use the SZZ algorithm to determine whether it is likely that refactorings induced a fault.

Results indicate that, while some kinds of refactorings are unlikely to be harmful, others, such as refactorings involving hierarchies (e.g., pull up method), tend to induce faults very frequently. This suggests more accurate code inspection or testing activities when such specific refactorings are performed.

**Index Terms**—Refactoring, Fault-inducing changes, Mining software repositories, Empirical Studies.

### I. INTRODUCTION

Software systems are continuously subject to maintenance tasks to introduce new features or fix bugs [1]. Very often such activities are performed in an undisciplined manner due to strict time constraints, to lack of resources/skills, or to the limited knowledge some developers have of the system design [2]. As a result, the code underlying structure, and therefore the related design, tend to deteriorate.

This phenomenon was defined as “software aging” by Parnas [3], and was also described in the law of increasing complexity by Lehman [1]. Some researchers measured the phenomenon in terms of change entropy [4], [5], while others defined “antipatterns”, i.e., recurring cases of poor design choices occurring as a consequence of aging, or when the software is not properly designed from the beginning. Classes doing too much (*God classes* or *Blobs*), poorly structured code (*Spaghetti code*), or *Long Message Chains* used to develop a certain feature are only few examples of antipatterns that plague software systems [2].

In order to mitigate the above described issues, software systems are, time to time, subject to improvement activities,

aimed at enhancing the code and design structure. Such activities are often referred to as *refactoring*. Refactoring is defined by Fowler [2] as “a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior”. The aim of refactoring is to improve the structure of source code—and consequently of the system design—whenever its structure may possibly lead to maintainability or comprehensibility problems. Fowler’s catalogue [6] comprises a set of 93 refactorings, aimed at dealing with different antipatterns in source code, such as, extracting a class from a Blob, pulling up a method from a subclass onto a superclass, or modifying the navigability of an association between two classes.

In theory, a refactoring should not change the behavior of a software system, but only help in improving some of its non-functional attributes. In practice, a refactoring might be risky as any other change occurring in a system, causing possible bug introductions. Indeed, a recent study [10] showed that even automated refactoring as performed by Integrated Development Environments could be fault-prone as well.

While there are attempts to investigate the relation between some refactorings and fault-proneness [8], [9] or change entropy [7], to the best of our knowledge there is no study aimed at thoroughly investigating whether a wide set of (even undocumented) refactorings occurred in a software system during its evolution induced bugs, and what kind of refactorings might induce more bugs than others.

In this paper we report an empirical study aimed at investigating to what extent refactoring induces bug fixes in software systems. We use an existing tool, namely Ref-Finder [11], to automatically detect refactoring operations of 52 different types on 63 releases of three Java software systems, Apache Ant<sup>1</sup>, ArgoUML<sup>2</sup>, and Xerces-J<sup>3</sup>. Of the 15,008 refactoring operations detected by the tool, 12,922 operations have been manually validated as actually refactorings. Then, we use the SZZ algorithm [12], [13] to determine whether the 12,922

<sup>1</sup><http://ant.apache.org/>

<sup>2</sup><http://argouml.tigris.org/>

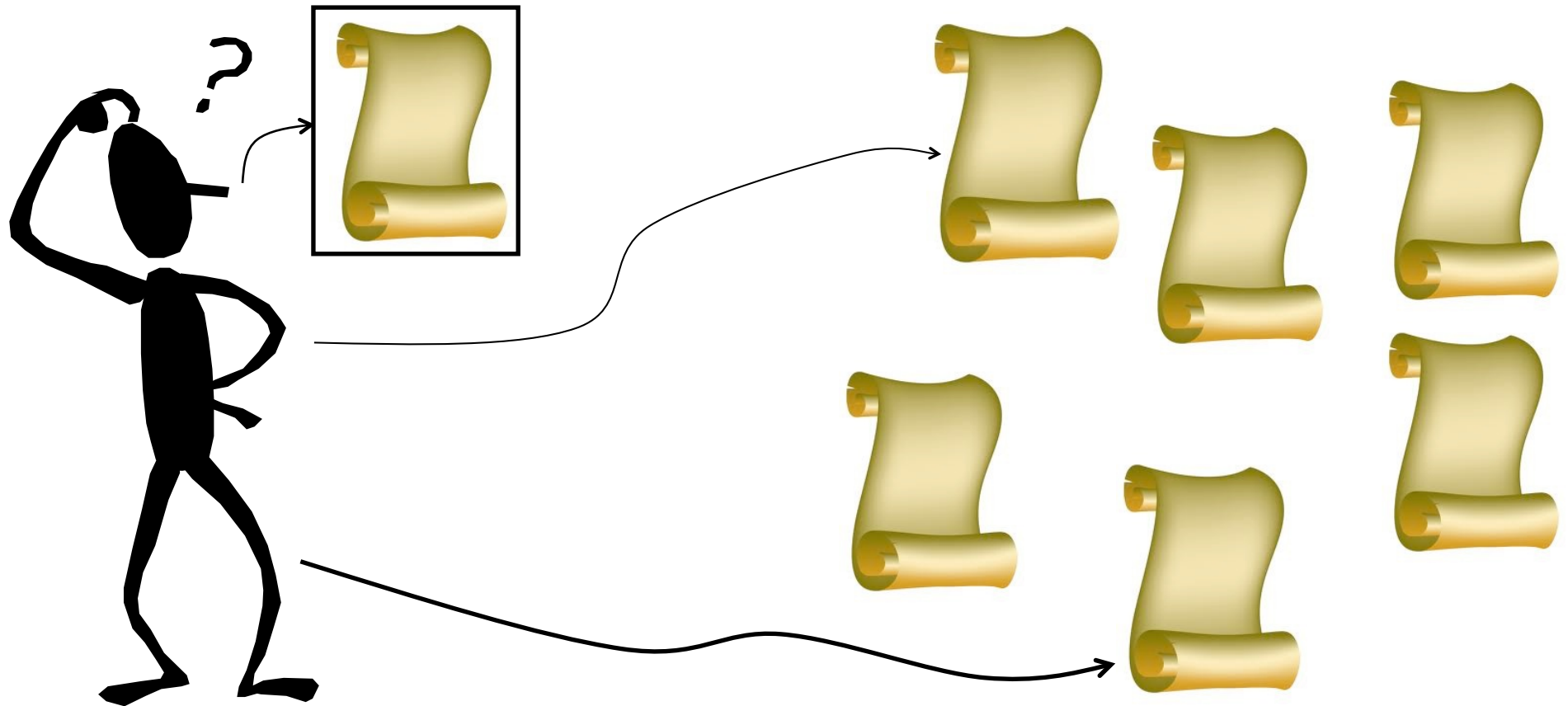
<sup>3</sup><http://xerces.apache.org/xerces-j/>

# Some results

- Changes to classes involved in a refactoring have 23 times more chances to induce fixes
- 158 times in the worst case
- Most dangerous refactoring: Pull up method and extract subclass

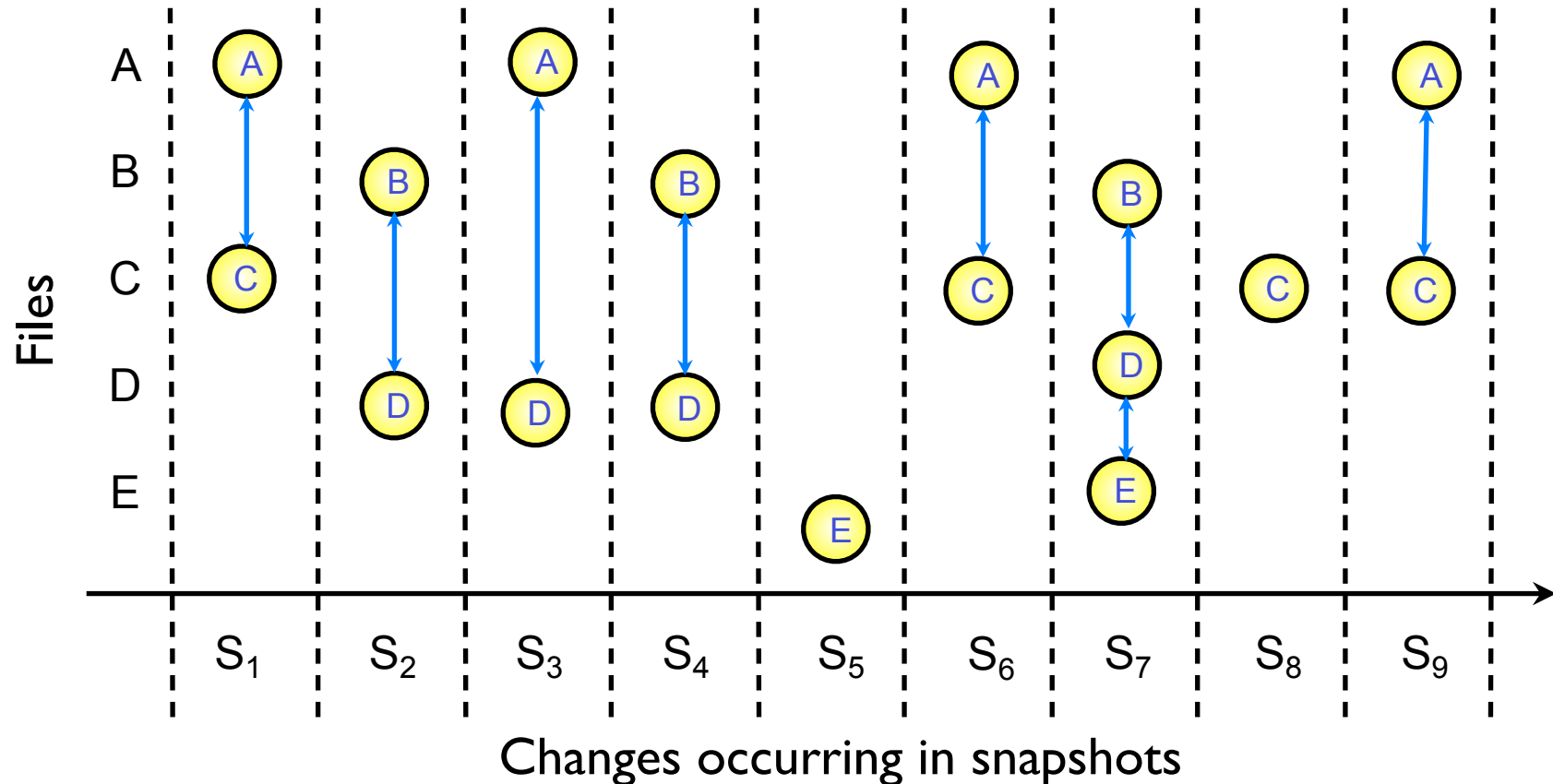


# Versioning system application: identifying logical coupling





# Association rule discovery



Thomas Zimmermann, Peter Weißgerber, Stephan Diehl, Andreas Zeller: Mining Version Histories to Guide Software Changes. ICSE 2004: 563-572

Annie T.T. Ying, Gail C. Murphy, Raymond T. Ng, Mark Chu-Carroll: Predicting Source Code Changes by Mining Change History. IEEE Trans. Software Eng. 30(9): 574-586 (2004)



# Association Rule Discovery in R

itemsets.csv

A,C
B,D
A,D
B,D
E
A,C
B,D,E
C
A,C

```
library(arules)
```

```
t<-read.transactions("itemsets.csv",sep=" ",  
rm.duplicates=TRUE)
```

```
m<-apriori(t,parameter=list(supp=0.2,conf=0.8))
```

```
inspect(m)
```

	lhs	rhs	support	confidence	lift
1	{"C"}	= {"A"}	0.375	1	2
2	{"B"}	= {"D"}	0.375	1	2





# Support, Confidence, Lift

$Supp(rule) =$  % of instances where the rule occur

$$Conf(X \Rightarrow Y) = \frac{Supp(X \cup Y)}{Supp(X)}$$

$$lift(X \Rightarrow Y) = \frac{Supp(X \cup Y)}{Supp(X) \cdot Supp(Y)}$$



# Historical smell detection

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2014.2372760, IEEE Transactions on Software Engineering

## Mining Version Histories for Detecting Code Smells

Fabio Palomba<sup>1</sup>, Gabriele Bavota<sup>2</sup>, Massimiliano Di Penta<sup>2</sup>, Rocco Oliveto<sup>3</sup>, Denys Poshyvanyk<sup>4</sup>, Andrea De Lucia<sup>1</sup>

<sup>1</sup>University of Salerno, Fisciano (SA), Italy

<sup>2</sup>University of Sannio, Benevento, Italy

<sup>3</sup>University of Molise, Pesche (IS), Italy

<sup>4</sup>The College of William and Mary, Williamsburg, VA, USA

fpalomba@unisa.it, gbavota@unisannio.it, dipenta@unisannio.it, rocco.oliveto@unimol.it, denys@cs.wm.edu, adelucia@unisa.it

**Abstract**—Code smells are symptoms of poor design and implementation choices that may hinder code comprehension, and possibly increase change- and fault-proneness. While most of the detection techniques just rely on structural information, many code smells are intrinsically characterized by how code elements change over time. In this paper, we propose HIST (Historical Information for Smell deTectio)n, an approach exploiting change history information to detect instances of five different code smells, namely Divergent Change, Shotgun Surgery, Parallel Inheritance, Blob, and Feature Envy. We evaluate HIST in two empirical studies. The first, conducted on twenty open source projects, aimed at assessing the accuracy of HIST in detecting instances of the code smells mentioned above. The results indicate that the precision of HIST ranges between 72% and 86%, and its recall ranges between 58% and 100%. Also, results of the first study indicate that HIST is able to identify code smells that cannot be identified by competitive approaches solely based on code analysis of a single system's snapshot. Then, we conducted a second study aimed at investigating to what extent the code smells detected by HIST (and by competitive code analysis techniques) reflect developers' perception of poor design and implementation choices. We involved twelve developers of four open source projects that recognized more than 75% of the code smell instances identified by HIST as actual design/implementation problems.

**Index Terms**—Code Smells, Mining Software Repositories, Empirical Studies.

### 1 INTRODUCTION

Code smells have been defined by Fowler [15] as symptoms of poor design and implementation choices. In some cases, such symptoms may originate from activities performed by developers while in a hurry, e.g., implementing urgent patches or simply making suboptimal choices. In other cases, smells come from some recurring, poor design solutions, also known as anti-patterns [9]. For example a *Blob* is a large and complex class that centralizes the behavior of a portion of a system and only uses other classes as data holders. *Blob* classes can rapidly grow out of control, making it harder and harder for developers to understand them, to fix bugs, and to add new features.

Previous studies have found that smells hinder comprehension [1], and possibly increase change- and fault-proneness [24], [25]. In summary, smells need to be carefully detected and monitored and, whenever necessary, refactoring actions should be planned and performed to deal with them.

This paper is an extension of "Detecting Bad Smells in Source Code Using Change History Information" that appeared in the Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE 2013), Palo Alto, California, pp. 268-278, 2013 [40].

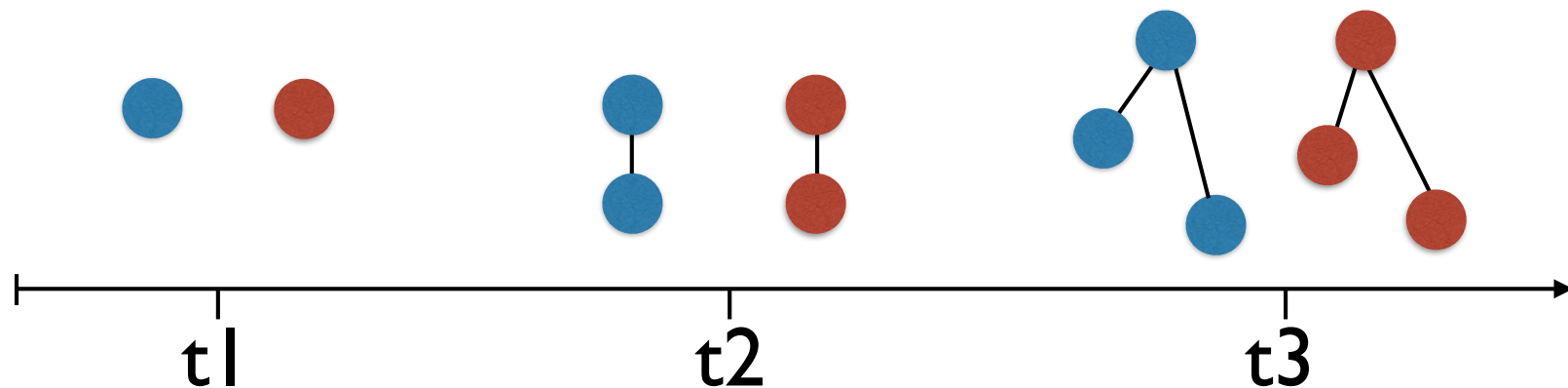
There exist a number of approaches for detecting smells in source code to alert developers of their presence [31], [34], [49]. These approaches rely on structural information extracted from source code, for example, by means of constraints defined on some source code metrics. For instance, according to some existing approaches, such as *DECOR* [34], *LongMethod* or *LargeClass* smells are based on the size of the source code component in terms of LOC, whereas other smells like *ComplexClass* are based on the McCabe cyclomatic complexity [33]. Other smells, such as *Blob*, might use more complex rules.

Although existing approaches exhibit good detection accuracy, they still might not be adequate for detecting many of the smells described by Fowler [15]. In particular, there are some smells that, rather than being characterized by source code metrics or other information extracted from source code snapshots, are *intrinsically characterized by how source code changes over time*. For example, a *Parallel Inheritance* means that two or more class hierarchies evolve by adding code to both classes at the same time. Also, there are smells that are traditionally detected using structural information, where historical information can aid in capturing complementary, additionally useful properties. For example, a *Feature Envy*



# Example: Parallel Inheritance Detection

Pairs of classes for which the addition of a subclass for one class implies the addition of a subclass for the other class





# Threat: Irrelevant changes

- We count commits as proxy of amount of changes
- Many commits are related to formatting, change of copyright year, commenting, refactoring

David Kawrykow, Martin P. Robillard:

Non-essential changes in version histories. ICSE 2011: 351-360

# Pruning out them...

- [Kawrykow et al. \(2011\)](#) developed an approach to identify non-essential changes (3%-15% of total in their study)
- They pruned out them to build better change impact prediction (-20% of erroneous and -4% of true recommendations)

However... hard to tell  
what is irrelevant and  
what not...

# Mining text from SCM



SCM contain a lot of unstructured data

Besides code identifiers and comments,  
above all commit notes

# Example

- Add Checkclipse preferences to all projects so Checkstyle is preconfigured
- Fix for issue 4503.
- Fix for issue 4517: "changeability" on association ends not displayed properly. This was never adapted since UML 1.3 & NSUML were replaced.

What kind of problem do you see here?





# Be aware!

- Commit notes are often insufficient to know everything about a change
- Need to merge with issue tracker data

# Issue Trackers

# Bugzilla (Mozilla)

The screenshot shows a Mozilla Bugzilla bug report page for Bug 33467. The browser's address bar shows the URL [https://bugzilla.mozilla.org/show\\_bug.cgi?id=33467](https://bugzilla.mozilla.org/show_bug.cgi?id=33467). The page title is "Bugzilla@Mozilla - Bug 33467 Cookie Manager should allow direct entry of entries to reject/allow - add site button". The page is in English and was last modified on 2004-06-03 02:32:11 PDT. The bug is categorized as "Verified Fixed".

**Bug 33467 - Cookie Manager should allow direct entry of entries to reject/allow - add site button** [Last Comment](#)

**Status:** VERIFIED FIXED **Reported:** 2000-03-27 12:38 PST by Richard Ekle  
**Whiteboard:** **Modified:** 2004-06-03 02:32 PDT ([History](#))  
**Keywords:** helpwanted **CC List:** 30 users ([show](#))

**Product:** Core ([show info](#)) **See Also:**  
**Component:** Networking: Cookies ([show info](#)) **Crash Signature:**  
**Version:** Trunk  
**Platform:** All All

**Importance:** P3 enhancement with 6 votes ([vote](#))  
**Target Milestone:** mozilla1.6beta  
**Assigned To:** Mike Connor [:mconnor]  
**QA Contact:** benc

**URL:**

**Duplicates:** [26742](#) [57416](#) [67431](#) [69148](#) [89221](#) [183830](#) [201060](#) ([view as bug list](#))

**Depends on:** [222553](#)  
**Blocks:** [100573](#)  
[Show dependency tree / graph](#)

---

**Attachments**

<a href="#">first try</a> (6.21 KB, patch) 2003-08-13 13:55 PDT, Michiel van Leeuwen (email: mvl+moz@)	<a href="#">dwtite: review+</a> <a href="#">jag-mozilla: superreview+</a>	<a href="#">Details</a>   <a href="#">Diff</a>   <a href="#">Review</a>
<a href="#">Target UI for this enhancement</a> (128.62 KB, image/bmp) 2003-08-25 00:33 PDT, Christopher Meeke	<a href="#">no flags</a>	<a href="#">Details</a>

# JIRA

[#XNIO-15] ConnectionAddress produces a warning when used in vararg lists

https://issues.jboss.org/browse/XNIO-15?page=com.atlassian.jira.plugin.system.issuetabpanels:all-tabpanel

183 of 149308  
Return to search

## XNIO / XNIO-15

### ConnectionAddress produces a warning when used in vararg lists

Agile Board More Actions Views

#### Details

Type:	Bug	Status:	Closed (View Workflow)
Priority:	Major	Resolution:	Done
Affects Version/s:	None	Fix Version/s:	1.0.0.CR1
Component/s:	None		
Labels:	None		
Estimated Difficulty:	Low		
Similar Issues:	<a href="#">Show 10 results &gt;</a>		

#### People

Assignee: David Lloyd  
Reporter: David Lloyd  
Vote (0) Watch (0)

#### Dates

Created: 10/Jun/08 10:21 PM  
Updated: 11/Jun/08 4:53 AM  
Resolved: 11/Jun/08 4:53 AM

#### Agile

[View on Board](#)

#### Description

Need to come up with a cleaner solution here.

#### Activity

All Comments Work Log History Activity Commits Source Reviews

David Lloyd made changes - 10/Jun/08 10:21 PM

Field	Original Value	New Value
Fix Version/s		1.0.0.CR1 [ 12312410 ]

david.lloyd@jboss.com submitted changeset 34 to trunk in XNIO (2 files) - 11/Jun/08 4:53 AM

XNIO-15 - fix lame warning by changing API

- xnio-base/trunk/api/src/main/java/org/jboss/xnio/io/Utils.java (+10 -7)
- xnio-base/trunk/api/src/test/java/org/jboss/xnio/test/io/UtilsTest.java (+3 -2)

David Lloyd made changes - 11/Jun/08 4:53 AM

Status	Open [ 1 ]	Closed [ 6 ]
Resolution		Done [ 1 ]

# Downloading issue reports

Some trackers allow export but... often you need to download them as web pages

What you need:

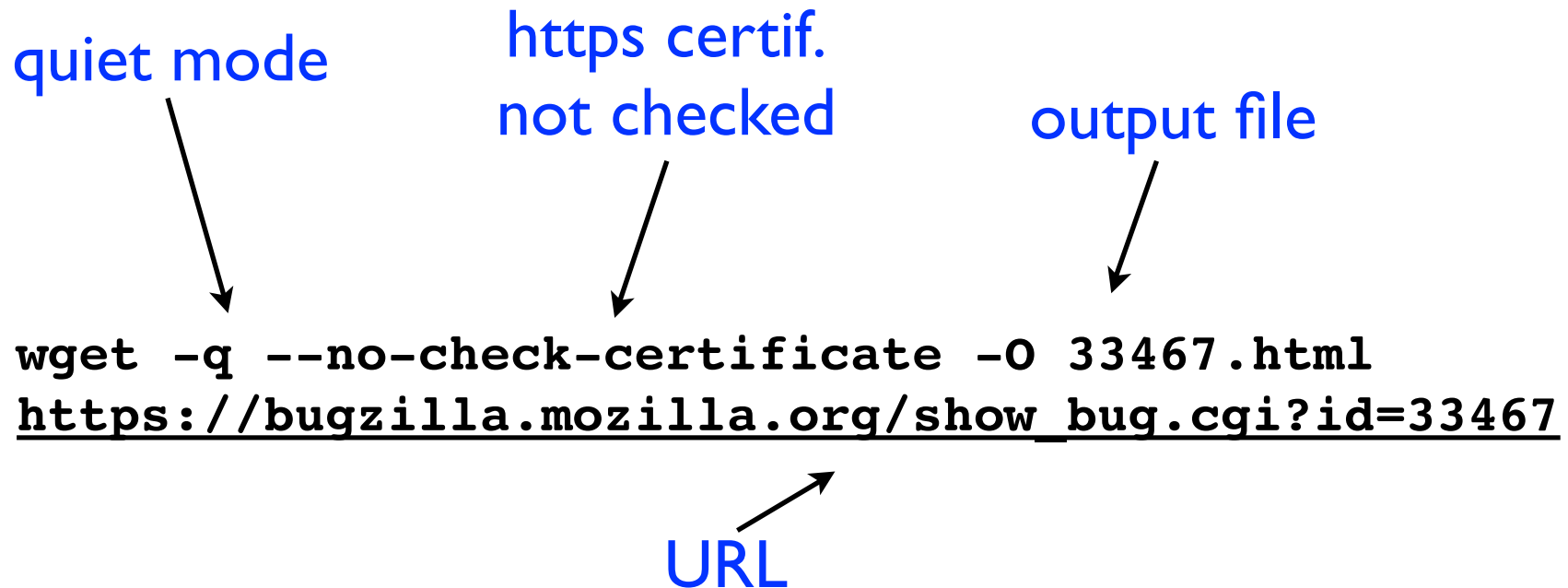
- Generic URL of the issues:
- [https://bugzilla.mozilla.org/show\\_bug.cgi?id=33467](https://bugzilla.mozilla.org/show_bug.cgi?id=33467)
- List of bug ids (e.g. results of a query, or from commit notes)
- `wget`
- `lynx`

# Let's download it...

quiet mode      https certif.  
not checked      output file

```
wget -q --no-check-certificate -O 33467.html  
https://bugzilla.mozilla.org/show_bug.cgi?id=33467
```

URL



once you have the IDs, you can do a simple script to automate this

# Parse the HTML

Why not if you really want to do it...

Alternative: render the bug report into a textual file

```
lynx -dump 33467.html 33467.txt
```

Output...



#[1]Top [2]Bugzilla@Mozilla

Bugzilla@Mozilla - Bug 33467

Cookie Manager should allow direct entry of entries to reject/allow -

add site button

Last modified: 2004-06-03 02:32:11 PDT

\* [3]Home

\* | [4]New

\* | [5]Browse

\* | [6]Search

\* | \_\_\_\_\_ Search

[[7]help]

\* | [8]Reports

\* | [9]Requests

\* | [10]Product Dashboard

\* | [11]Help

\* | [12]New Account

\* | [13]Log In Sign in with Persona or

---

[\_] Remember Log in

[14][x]

\* | [15]Forgot Password Login:  
Reset Password

---

[16][x]

[17]Last Comment [18]Bug 33467 - Cookie Manager should allow direct

entry of entries to reject/allow - add site button

Summary: Cookie Manager should allow direct entry of entries to

reject/allow - add sit...

[19]Status: VERIFIED FIXED

Whiteboard:

[20]Keywords: helpwanted

[21]Product: Core

[22]Classification: Components

[23]Component: Networking: Cookies

Version: Trunk

Platform: All All

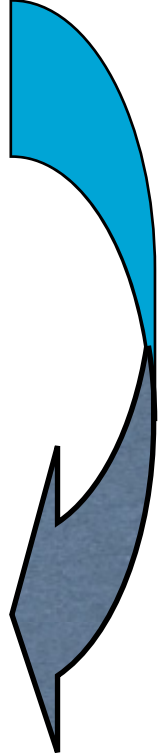
[24]Importance: P3 enhancement with [25]6 votes ([26]vote)

# Linking issues to commmits

“*fix 367920 setting pop3 messages as junk/not junk ignored when message quarantining turned on sr=mscott*”

Solution: Regular expression matching e.g.

```
$1=~ /BR (\d+) / || $1=~ /fix\s+(\d+) /i ||  
$1=~ /PR\s+(\d+) / || $1=~ /Bugzilla\s+(\d  
+) /i || $1=~ /Bug\s+(\d+) /i || $1=~ /^\# (\d  
+) /i
```



Bugzilla@Mozilla - Bug 367920

[Home](#) | [New](#) | [Search](#) |  [Find](#) | [Reports](#) | [Requests](#) | [New Account](#) | [Log In](#)

[First](#) [Last](#) [Prev](#) [Next](#) No search results available

### Bug 367920 - Filter action Set Junk Status to Junk does not work correctly

<b>Status:</b>	VERIFIED FIXED	<b>Reported:</b>	2007-01-23 11:42 PDT by <a href="#">Nick Howitt</a>
<b>Severity:</b>	normal	<b>Modified:</b>	2007-03-26 14:13:35 PDT ( <a href="#">View Bug Activity</a> )
<b>Keywords:</b>	fixed1.8.1.2, regression	<b>Votes:</b>	0
<b>Whiteboard:</b>		<b>CC:</b>	<div style="border: 1px solid black; padding: 2px;">bienvenu@nventure.com mscott@mozilla.org patrick.brunschwig@gmx.net vseerror@lehigh.edu</div>
<b>URL:</b>		<b>Flags:</b>	mscott: blocking-thunderbird2+
<b>Product:</b>	Thunderbird		
<b>Component:</b>	General		
<b>Version:</b>	2.0		
<b>Hardware:</b>	All		
<b>OS:</b>	All		
<b>Assigned To:</b>	<a href="#">David Bienvenu</a>		
<b>QA Contact:</b>	<a href="#">general@thunderbird.bugs</a>		
<b>Priority:</b>	--		
<b>Target Milestone:</b>	---		



# Threat: Missing Links

## Explicit link

Quieten level 0 debug when probing for modules. We shouldn't display so loud an error when a `smb_probe_module()` fails. Also tidy up debugs a bit. [Bug 375](#).

## Missing Link

`nmbd_incomingdgrams.c`: Fix bug with Syntax 5.1 servers reported by SGI where they do host announcements to `LOCAL_MASTER_BROWSER_NAME<00` rather than `WORKGROUP<Id`

# Beyond Explicit links

Rongxin Wu, Hongyu  
Zhang, Sunghun Kim,  
Shing-Chi Cheung:  
ReLink: recovering  
links between bugs  
and changes.  
SIGSOFT FSE 2011:  
15-25

**Change Log** (Revision: 148; Author: srowen; Date: Jan 22, 2008)  
Name of midlet is "ZXingMIDlet", not "ZXingMidlet"!

---

**Bug Report** (Issue 18; Status: Fixed):

Reported by herf...@yahoo.com, Jan 11, 2008

Issue 18: does not work on nokia 5300

Comment 1 by project member srowen@gmail.com, Jan 11, 2008

Which version, regular or basic? ...

...

Comment 3 by herf...@yahoo.com, Jan 12, 2008

I tested both of them (regular and basic) but in both case it just make an exception...

...

Comment 5 by project member srowen@gmail.com, Jan 22, 2008

...The class is called "ZXingMIDlet" but the exception mentions "ZXingMidlet" (note different capitalization). It looks like the manifest file I wrote gets this wrong. ...

Comment 6 by project member srowen@gmail.com, Jan 22, 2008

I think I have fixed this particular problem by correcting...

Can you try the most recent version from...



# Application: defect prediction

# Purpose of defect prediction



Can we really build models predicting future faults in our system?

No!

However, we can predict which artifacts will **likely** exhibit faults

- These artifacts must be better tested/validated

# Predicting variables you can use

- Code metrics
- Past bugs
- Process metrics (e.g. data)
- Features from fix-inducing changes
- ....

# Want to learn more?

Recent tutorial at ESEC-FSE by Leandro Minku

[http://www.cs.bham.ac.uk/~minkull/  
publications/fse15-tutorial.pdf](http://www.cs.bham.ac.uk/~minkull/publications/fse15-tutorial.pdf)





# Threat: Incorrect Classification

- Issue tracking systems contain various kinds of changes
- Classified using inadequate fields, or just poorly and subjectively classified

## **Status:**

UNCONFIRMED  
NEW  
ASSIGNED  
REOPENED  
RESOLVED  
VERIFIED  
CLOSED

## **Resolution:**

---  
FIXED  
INVALID  
WONTFIX  
DUPLICATE  
WORKSFORME  
MOVED

## **Severity:**

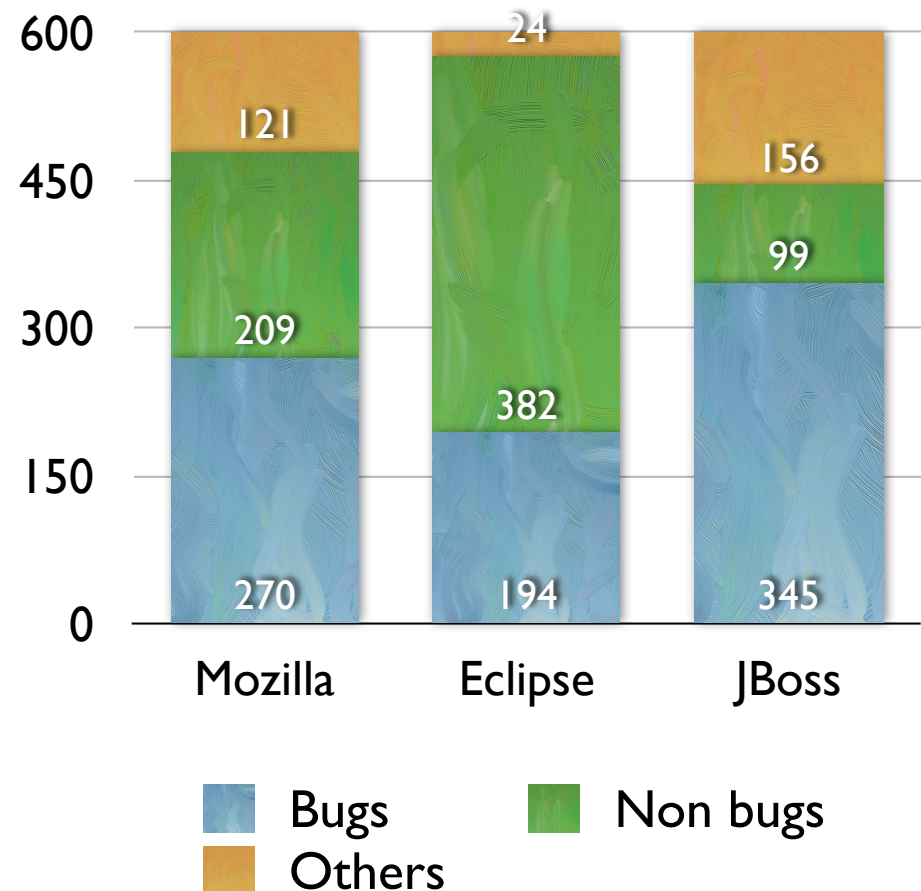
blocker  
critical  
major  
normal  
minor  
trivial  
enhancement

## **Priority:**

P1  
P2  
P3  
P4  
P5

# Results of a manual classification

We manually classified  
1,800 randomly  
selected bugs from  
Mozilla, Eclipse, JBoss



# It's not a Bug, it's a Feature: How Misclassification Impacts Bug Prediction

Kim Herzig  
Saarland University  
Saarbrücken, Germany  
herzig@cs.uni-saarland.de

Sascha Just  
Saarland University  
Saarbrücken, Germany  
just@st.cs.uni-saarland.de

Andreas Zeller  
Saarland University  
Saarbrücken, Germany  
zeller@cs.uni-saarland.de

**Abstract**—In a manual examination of more than 7,000 issue reports from the bug databases of five open-source projects, we found 33.8% of all bug reports to be *misclassified*—that is, rather than referring to a code fix, they resulted in a new feature, an update to documentation, or an internal refactoring. This misclassification introduces *bias* in bug prediction models, confusing bugs and features: On average, 39% of files marked as defective actually never had a bug. We estimate the impact of this misclassification on earlier studies and recommend manual data validation for future studies.

**Index Terms**—mining software repositories; bug reports; data quality; noise; bias

## I. INTRODUCTION

In empirical software engineering, it has become commonplace to mine data from change and bug databases to detect where bugs have occurred in the past, or to predict where they will occur in the future. The accuracy of such measurements and predictions depends on the *quality of the data*. Therefore, mining software archives must take appropriate steps to assure data quality.

A general challenge in mining is to separate *bugs* from *non-bugs*. In a bug database, the majority of issue reports are classified as *bugs*—that is, requests for corrective code maintenance. However, an issue report may refer to “perfective and adaptive maintenance, refactoring, discussions, requests for help, and so on” [1]—that is, activities that are unrelated to errors in the code, and would therefore be classified in a non-bug category. If one wants to mine code history to locate or predict error prone code regions, one would therefore only consider issue reports classified as bugs. Such filtering needs nothing more than a simple database query.

However, all this assumes that the category of the issue report is accurate. In 2008, Antoniol et al. [1] raised the problem of *misclassified* issue reports—that is, reports classified as *bugs*, but actually referring to *non-bug issues*. If such mix-ups (which mostly stem from issue reporters and developers interpreting “bug” differently) occurred frequently and systematically they would introduce *bias* in data mining models threatening the external validity of any study that builds on such data: Predicting the most error-prone files, for instance, may actually yield files most prone to new features. But how often does such misclassification occur? And does it actually bias analysis and prediction?

TABLE I  
PROJECT DETAILS.

	Maintainer	Tracker type	# reports
HTTPclient	APACHE	Jira	746
Jackrabbit	APACHE	Jira	2,402
Lucene-Java	APACHE	Jira	2,443
Rhino	MOZILLA	Bugzilla	1,226
Tomcat5	APACHE	Bugzilla	584

These are the questions we address in this paper. From five open source projects (Section II), we manually classified more than 7,000 issue reports into a fixed set of issue report categories clearly distinguishing the kind of maintenance work required to resolve the task (Section III). Our findings indicate substantial data quality issues:

**Issue report classifications are unreliable.** In the five bug databases investigated, more than 40% of issue reports are inaccurately classified (Section IV)

**Every third bug is not a bug.** 33.8% of all bug reports do not refer to corrective code maintenance (Section V).

After discussing the possible sources of these misclassifications (Section VI), we turn to the consequences. We find that the validity of studies regarding the distribution and prediction of bugs in code is threatened:

**Files are wrongly marked to be error-prone.** Due to misclassifications, 39% of files marked as defective actually have never had a bug (Section VII).

**Files are wrongly predicted to be error-prone.** Between 16% and 40% of the top 10% most defect-prone files do not belong in this category after reclassification (Section VIII).

Section IX details studies affected and unaffected by these issues. After discussing related work (Section X) and threats to validity (Section XI), we close with conclusion and consequences (Section XII).

## II. STUDY SUBJECTS

We conducted our study on five open-source JAVA projects described in Table I. We aimed to select projects that were under active development and were developed by teams that follow strict commit and bug fixing procedures similar to industry. We also aimed to have a more or less homogenous data

# It's not a bug...

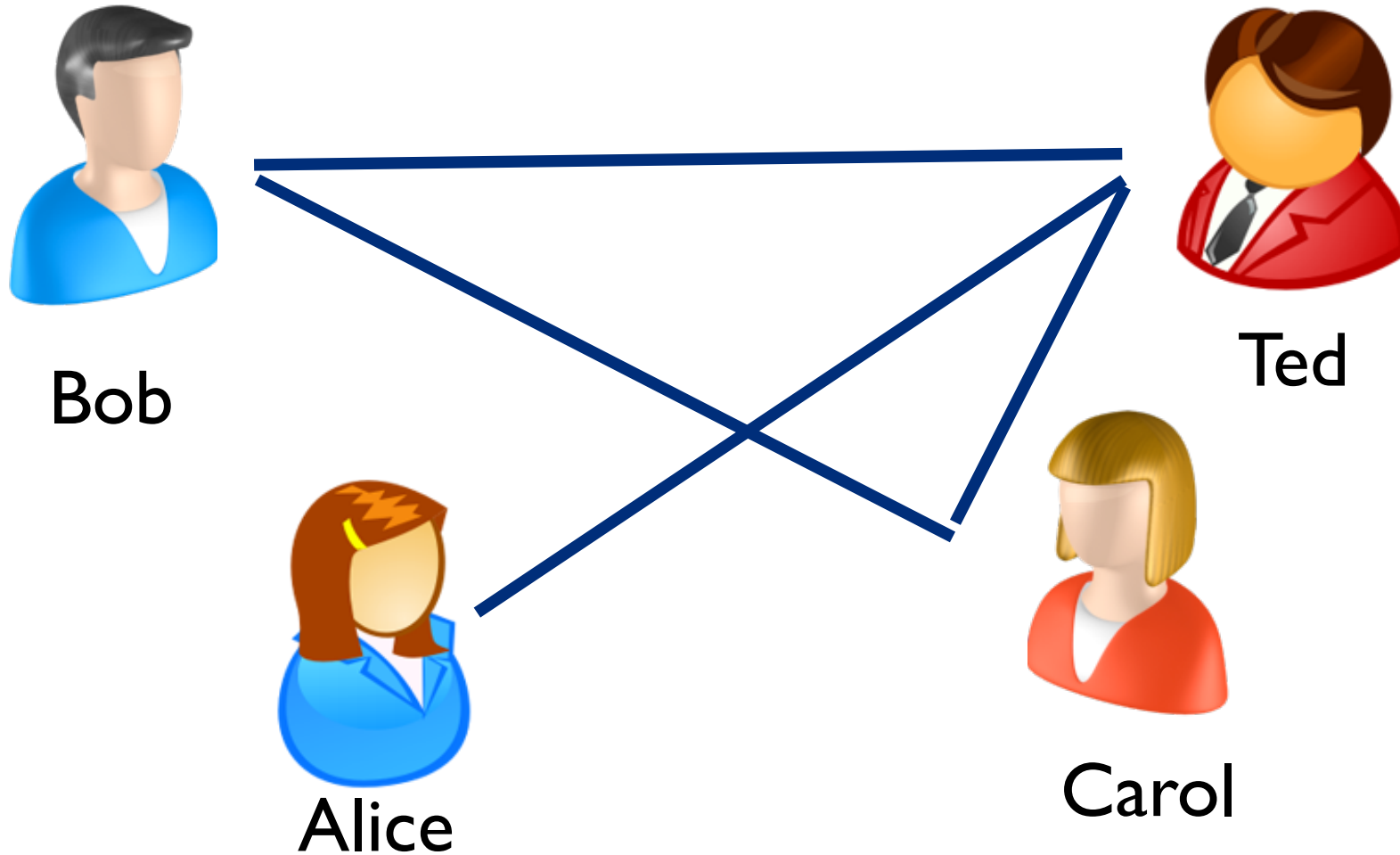
- Confirmed our results: 1/3 of bug reports are not about bugs
- When predicting the top 10% defect-prone files, 16% to 40% do not belong to that category

# Mailing Lists

# Availability

- Mailing list archives in Web-based form (more difficult to grab), e.g. SourceForge
- ...or in archives
- Emails are often in MBOX format
- Use specific Parsers
  - `Mail::Box::Parser` in Perl
  - Tika in Java <http://tika.apache.org>

# Communication Network



# Identifying a communication network

- Use **message-ID** and “**In-Reply-To**” to link emails
- Then, for each email, identify the **Sender**
- Finally, add a link between senders of linked emails



# Example

From lisa at usna.navy.MIL Thu Jul 1 15:42:27 1999

From: lisa at usna.navy.MIL (Lisa Beckettold {CADIG STAFF})

Date: Tue Dec 2 03:03:10 2003

Subject: nmbd/nmbd\_processlogon.c - CODE 12???

Message-ID: <99Jul1.114236-0400edt.4995-357+39@jupiter.usna.navy.mil

Hi:

I have Samba 2.1.0-prealpha running on a Sun Ultra 10. My NT PC joined the domain without a problem, but I can't logon. Every time I attempt to log into the Samba domain, the NT screen blanks as it usually does during login, but then the "Begin Login" screen reappears.

I see this message in my samba/var/log.nmb file whenever I try to login:

# Example

From: allen at driversoft.com (Allen Reese)

Date: Tue Dec 2 03:03:10 2003

Subject: nmbd/nmbd\_processlogon.c - CODE 12???

In-Reply-To: <[99Jul1.114236-0400edt.4995-357+39@jupiter.usna.navy.mil](mailto:99Jul1.114236-0400edt.4995-357+39@jupiter.usna.navy.mil)>

Message-ID: <[Pine.LNX.4.04.9907011046420.7499-100000@rat.driversoft.com](mailto:Pine.LNX.4.04.9907011046420.7499-100000@rat.driversoft.com)>

about 6 months ago I had a similar problem where login's wouldn't work du to an NT machine fighting with my server over who got to be the lmb. I set the registry key on all of my NT boxes and that fixed it for me.

Allen Reese

Senior Software Engineer

Driversoft, Inc.

[allen@driversoft.com](mailto:allen@driversoft.com)

On Fri, 2 Jul 1999, Lisa Beckettold {CADIG STAFF} wrote:

Hi:

I have Samba 2.1.0-prealpha running on a Sun Ultra 10. My NT PC joined

# Result



Allen



Lisa

# Resolving ambiguities

- People use multiple names:  
M. Di Penta, Max Di Penta, Di Penta
- And multiple emails:  
[dipenta@unisannio.it](mailto:dipenta@unisannio.it), [dipenta@gmail.com](mailto:dipenta@gmail.com)
- Sometimes emails do not contain names, only email addresses
- When mining issue trackers, names may be the only resource

Christian Bird, Alex Gourley, Premkumar T. Devanbu, Michael Gertz, Anand Swaminathan: Mining email social networks. MSR 2006: 137-143

Gerardo Canfora, Luigi Cerulo, Marta Cimitile, Massimiliano Di Penta: Social interactions around cross-system bug fixings: the case of FreeBSD and OpenBSD. MSR 2011: 143-152

# Heuristics

- **Use of Initials**, first name may be full or abbr.  
M. Di Penta = Max Di Penta
- **Middle Name Missing**  
John Fitzgerald Kennedy = John Kennedy
- **First name missing**  
Di Penta = Max Di Penta
- **Mapping names-emails/IDs** (if needed):  
M. Di Penta → mdipenta@foo.bar  
Di Penta → dipenta@foo.bar

# Analyze projects social network

- Not going to provide you with much details on that
- Treat the data as a list of edges and a list of nodes
- [igraph](#) and [sna](#) packages in R
- Further reading: [J. P. Scott. Social Network Analysis: A Handbook \(2nd edition\). Sage Publications Ltd, Englewood Cliffs, NJ, 2000.](#)

# Example

edges.csv

```
FROM,TO  
  
sebastian h,szymon brandys  
  
bill higgins,michael valenta  
  
anders peterson,jeff mcaffer  
  
jean michel lemieux,rafael chaves  
  
....
```

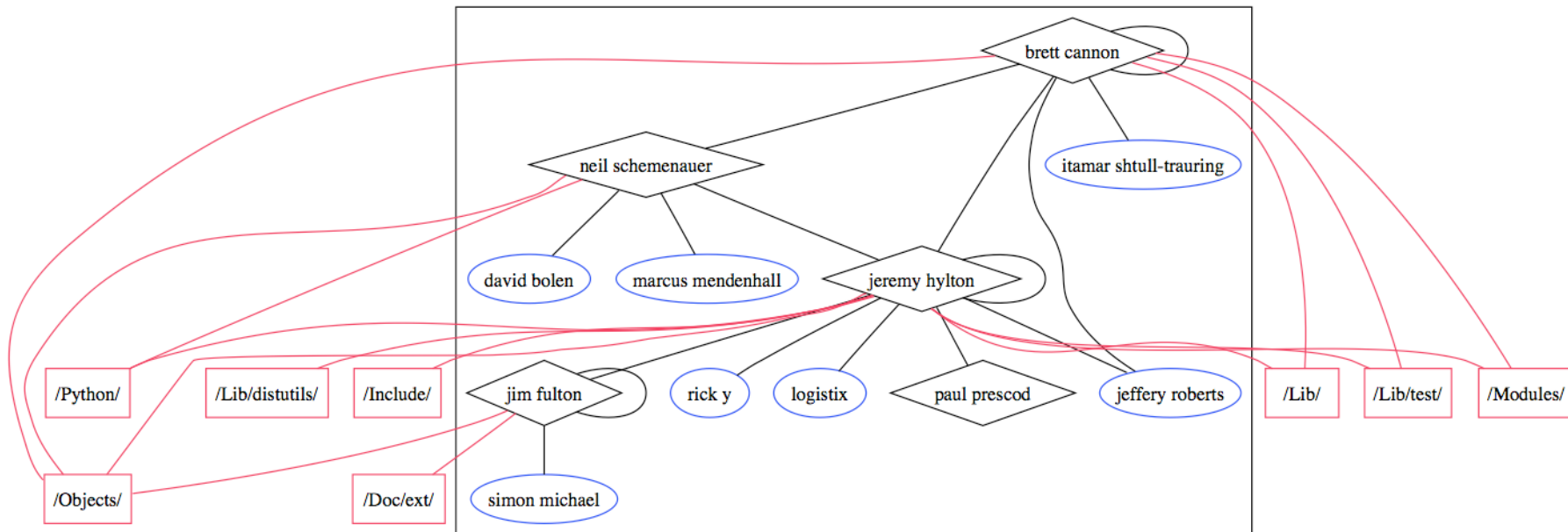
nodes.csv

```
NAME,COMMITTER,HASFIXED,ISGOOD,ISBAD  
  
sebastian h,NO,NO,NO,NO  
  
szymon brandys,YES,NO,YES,NO  
  
bill higgins,NO,NO,NO,NO  
  
michael valenta,YES,NO,NO,YES  
  
....
```

```
tedges<-read.csv("edges.csv"); #loads edges  
tnodes<-read.csv("nodes.csv"); #loads nodes  
# builds graph  
g<-graph.data.frame(tedges, directed=TRUE, vertices=tnodes);  
a<-get.adjacency(g); #adjacency matrix for sna  
deg<-degree(a); #sna degree  
bw<-betweenness(a); #sna betweenness
```



# Application: latent structure of software projects



The social structure of the project reflects developers' collaboration





# Identifying mentors in software projects

Enough **expertise** about the topic of interest for the newcomer...



Demonstrated **ability to help** other people...





# Mentor Identification in Academia

<http://arnetminer.org>

M-Di-Penta's Ego Network:

Rebuild



Advisor Co-author Advisee



# Mentor Identification Criteria

f1 Mentor/mentee exchanged many emails

f2 Mentor more active than mentee

f3 Mentor more senior (in the project) than mentee

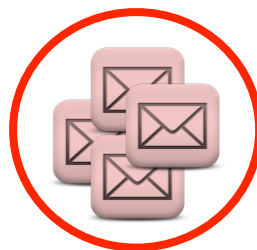
f4 Mentee exchanged her first emails mainly with the mentor

f5 Mentor did a high number of commits



# Recommending Mentors

Past mentors



Time



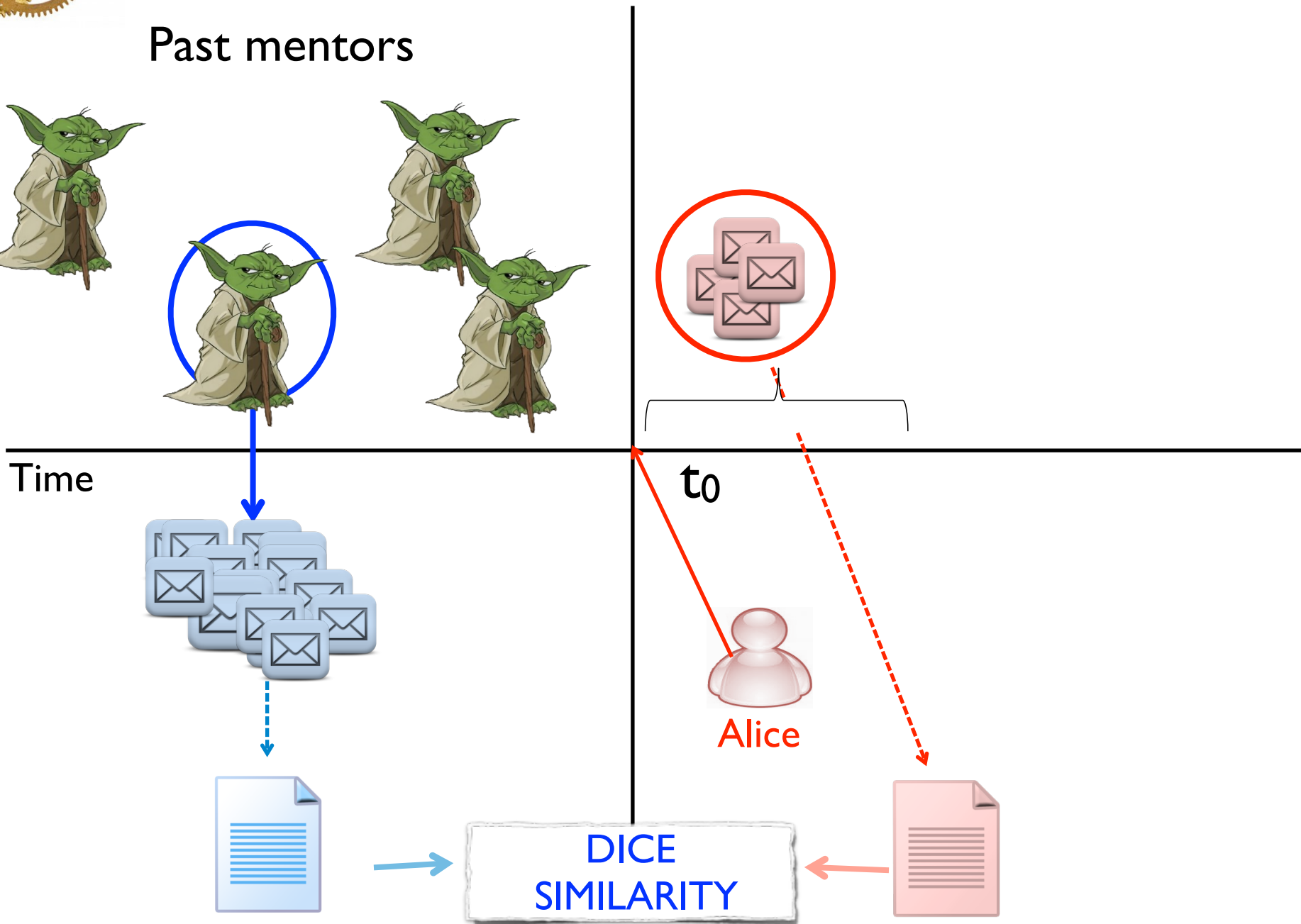
$t_0$



Alice

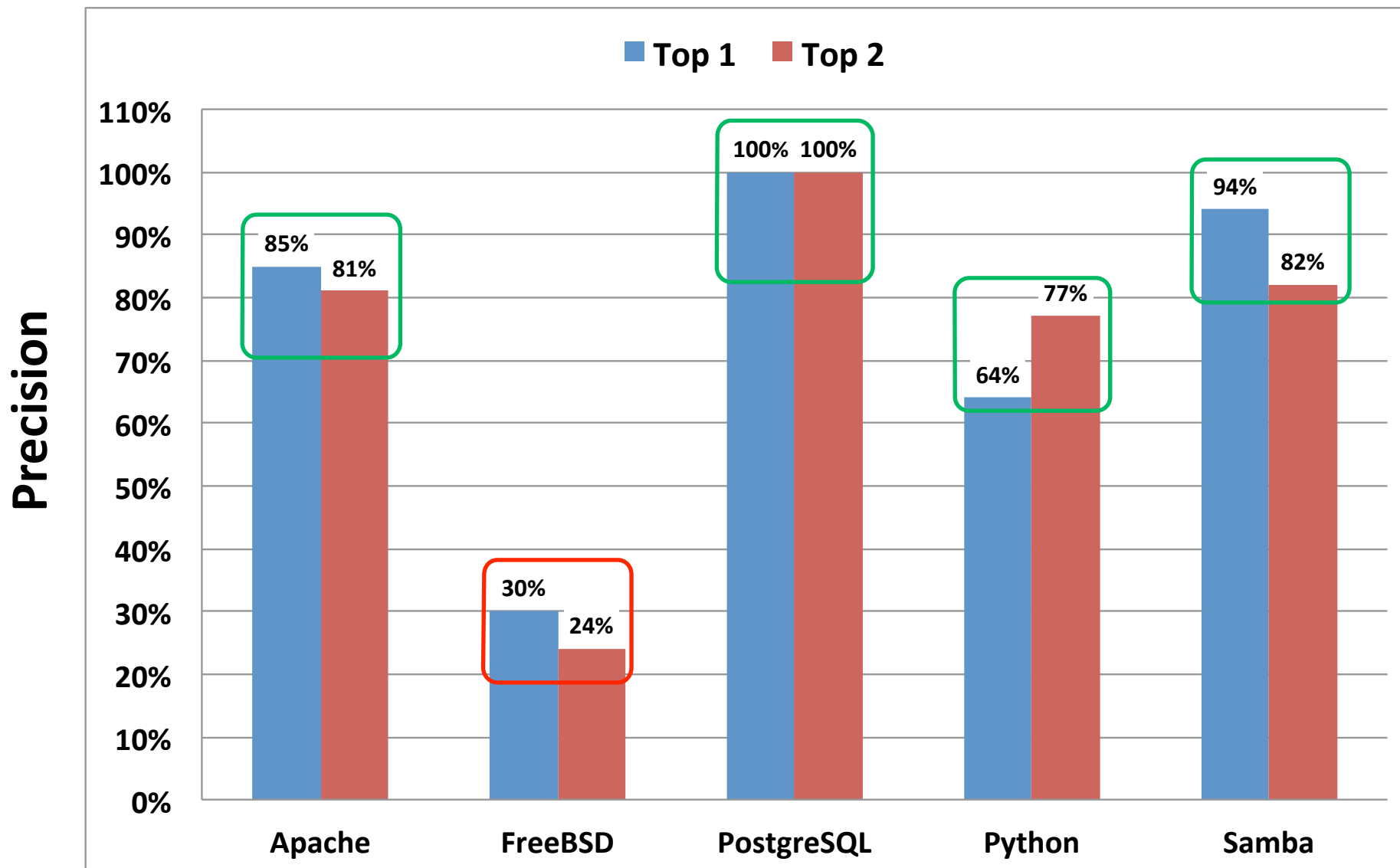


DICE  
SIMILARITY



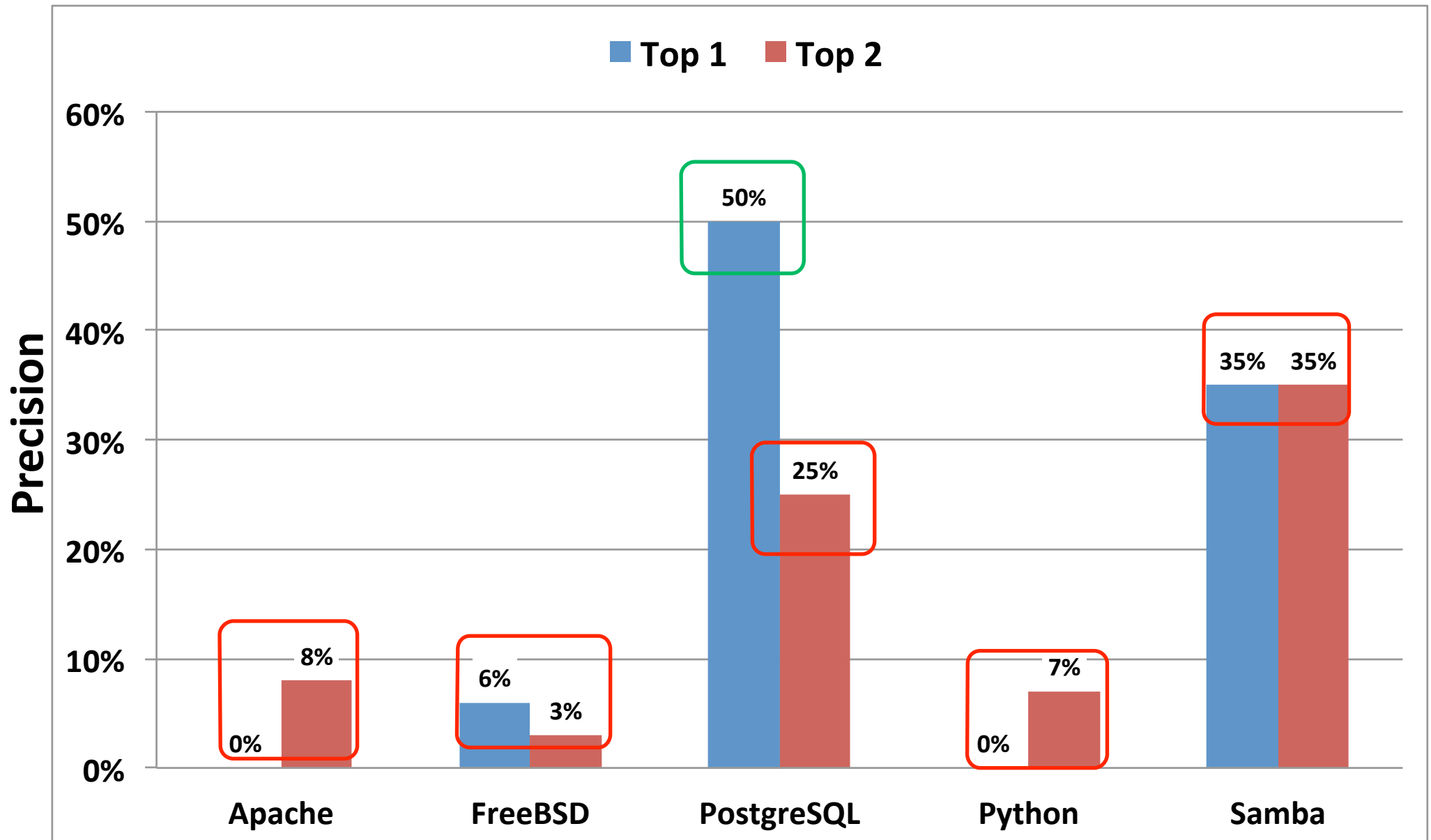


# Recommendation Accuracy





# Why don't just using Top Committers?





# Threat: secret life



- Software repositories do not capture everything of a software project
- Not all discussions, not all decisions, and after all also not all changes
- This could be especially true in industrial projects
- Should be less common in FLOSS

# Different ways of identifying emerging teams



# Sources

- Mailing lists
- Issue trackers
- Chat log
- Code changes (commits of multiple authors in the same time window)

# Social networks from issue trackers

- Identifying communication trickier than for mailing lists
- **Assumption:** whoever posts a comment communicates with those who previously posted a comment

# Example (Firefox #1432)

**Angus Davis 1998-11-21 17:22:59 PST**

This also occurs when logging into Netcenter.....

**Gagan 1998-11-23 14:09:59 PST**

Looks like this might be related to Steve Morse's recent work....

**Stephen P. Morse 1998-11-23 14:41:59 PST**

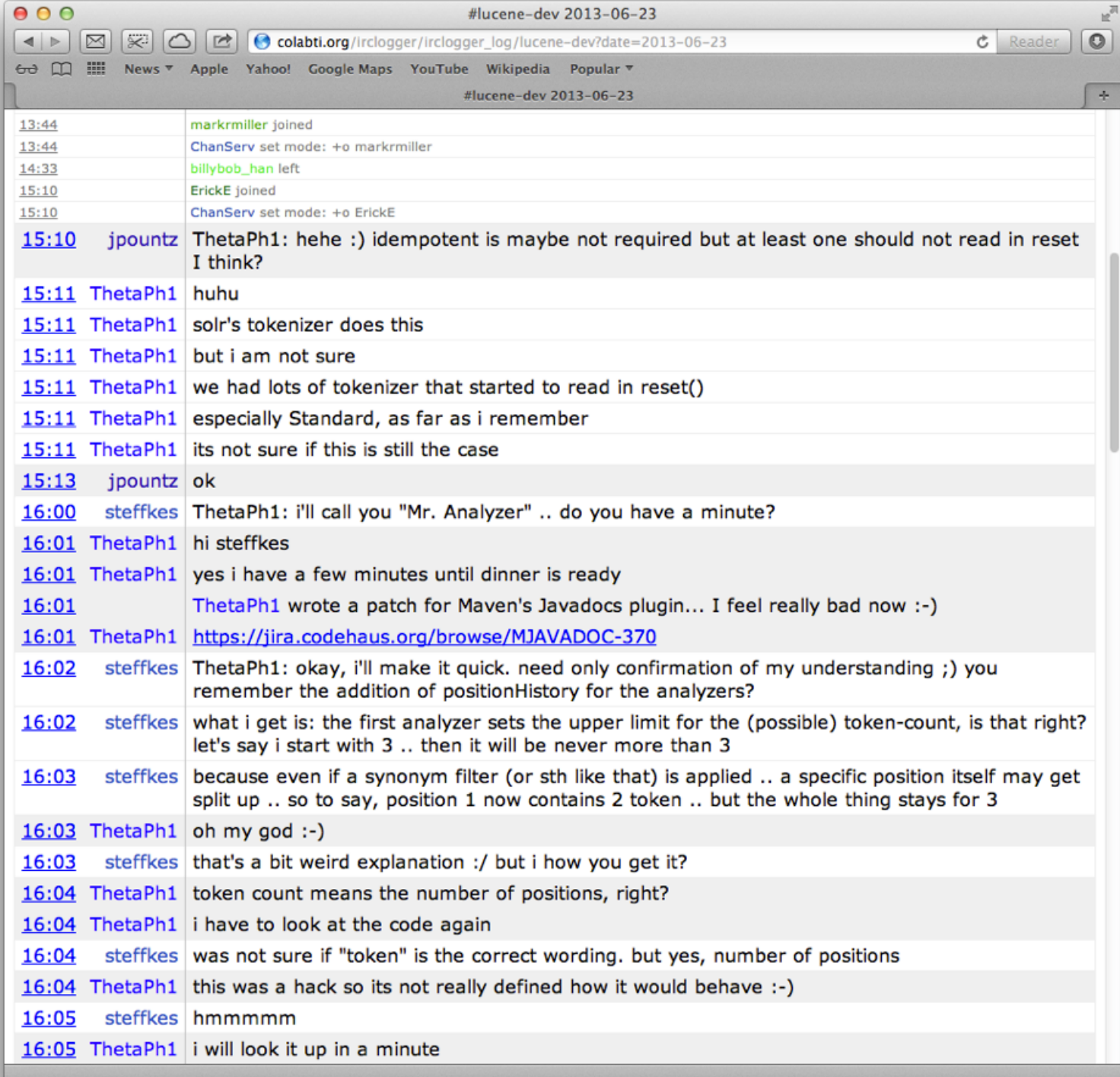
Yes, this is from single signon. ....

**Gagan 1998-11-23 15:48:59 PST**

As much as I agree with morse, I am getting inclined towards suggesting that we should leave SingleSignon for Dec.

Also in this case Mining  
Unstructured Data  
Techniques can Help!

# Chat Logs



```
#lucene-dev 2013-06-23
colabti.org/irclogger/irclogger_log/lucene-dev?date=2013-06-23
#lucene-dev 2013-06-23

13:44 markrmiller joined
13:44 ChanServ set mode: +o markrmiller
14:33 billybob_han left
15:10 ErickE joined
15:10 ChanServ set mode: +o ErickE
15:10 jpountz ThetaPh1: hehe :) idempotent is maybe not required but at least one should not read in reset I think?
15:11 ThetaPh1 huhu
15:11 ThetaPh1 solr's tokenizer does this
15:11 ThetaPh1 but i am not sure
15:11 ThetaPh1 we had lots of tokenizer that started to read in reset()
15:11 ThetaPh1 especially Standard, as far as i remember
15:11 ThetaPh1 its not sure if this is still the case
15:13 jpountz ok
16:00 steffkes ThetaPh1: i'll call you "Mr. Analyzer" .. do you have a minute?
16:01 ThetaPh1 hi steffkes
16:01 ThetaPh1 yes i have a few minutes until dinner is ready
16:01 ThetaPh1 wrote a patch for Maven's Javadocs plugin... I feel really bad now :-)
16:01 ThetaPh1 https://jira.codehaus.org/browse/MJAVADOC-370
16:02 steffkes ThetaPh1: okay, i'll make it quick. need only confirmation of my understanding ;) you remember the addition of positionHistory for the analyzers?
16:02 steffkes what i get is: the first analyzer sets the upper limit for the (possible) token-count, is that right? let's say i start with 3 .. then it will be never more than 3
16:03 steffkes because even if a synonym filter (or sth like that) is applied .. a specific position itself may get split up .. so to say, position 1 now contains 2 token .. but the whole thing stays for 3
16:03 ThetaPh1 oh my god :-)
16:03 steffkes that's a bit weird explanation :/ but i how you get it?
16:04 ThetaPh1 token count means the number of positions, right?
16:04 ThetaPh1 i have to look at the code again
16:04 steffkes was not sure if "token" is the correct wording. but yes, number of positions
16:04 ThetaPh1 this was a hack so its not really defined how it would behave :-)
16:05 steffkes hmmm
16:05 ThetaPh1 i will look it up in a minute
```

Identifying  
collaborations through  
code changes

# Example

F1



F2



F3



Bob



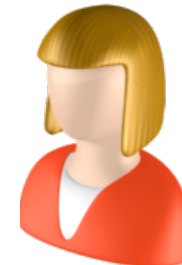
Jim



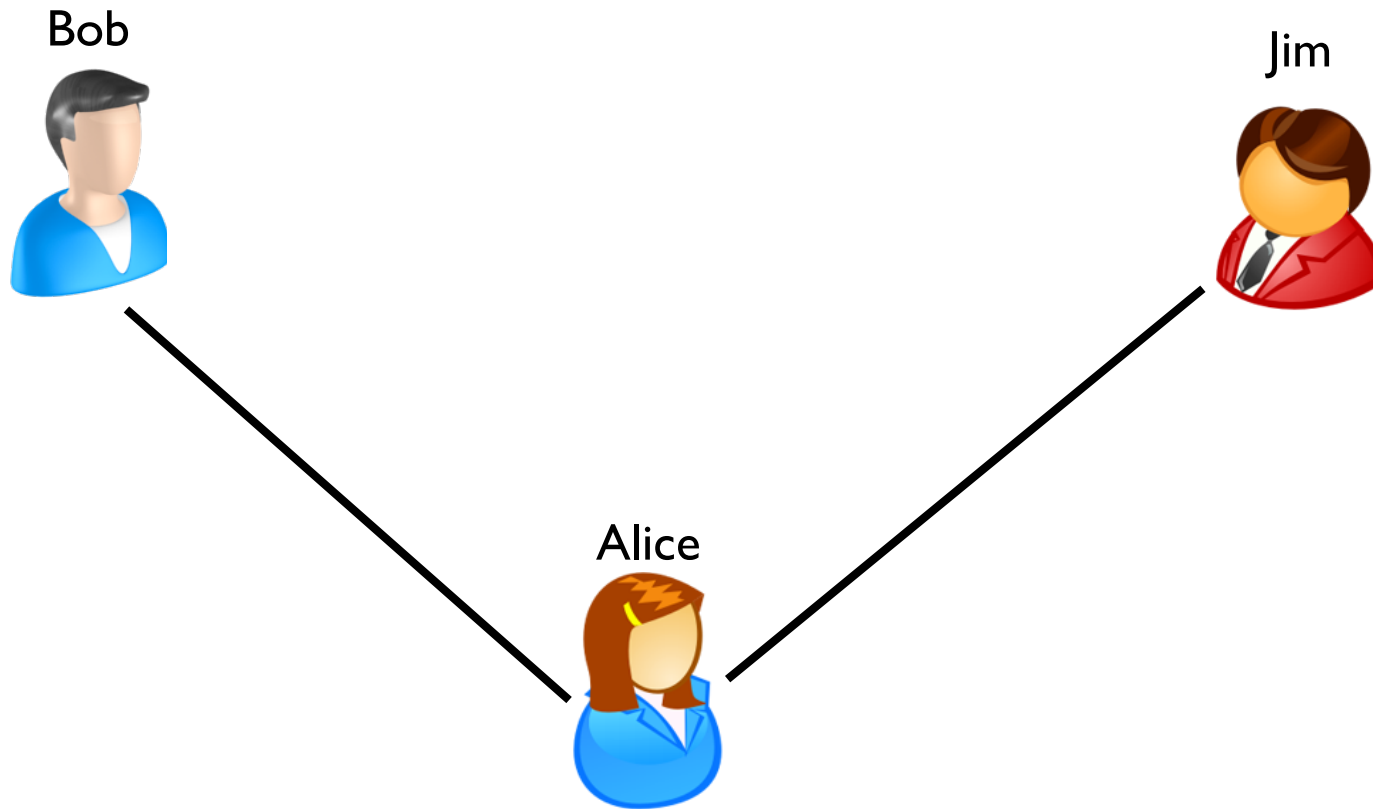
Alice



Jane

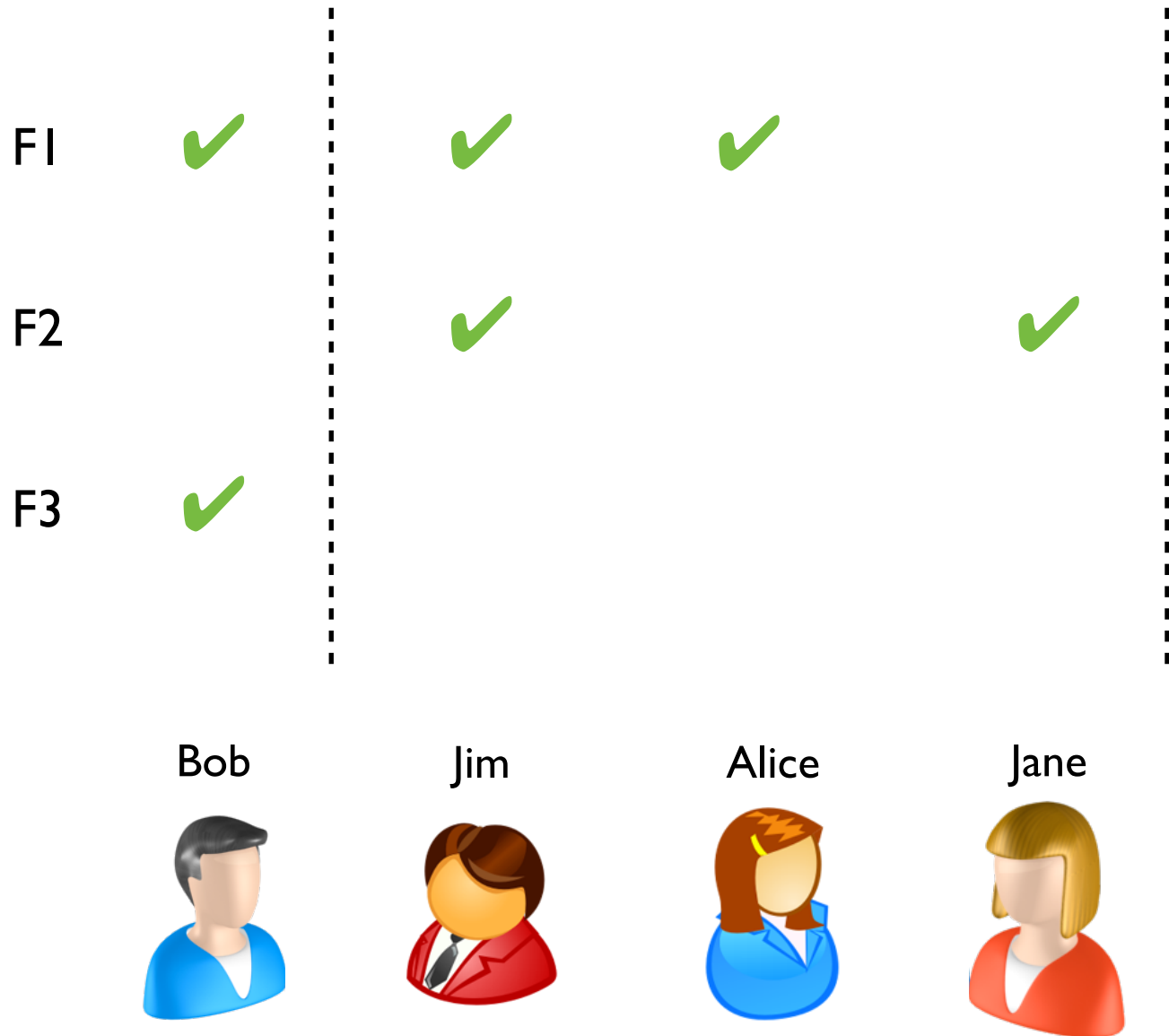


# Example





# Example



# Example

Bob



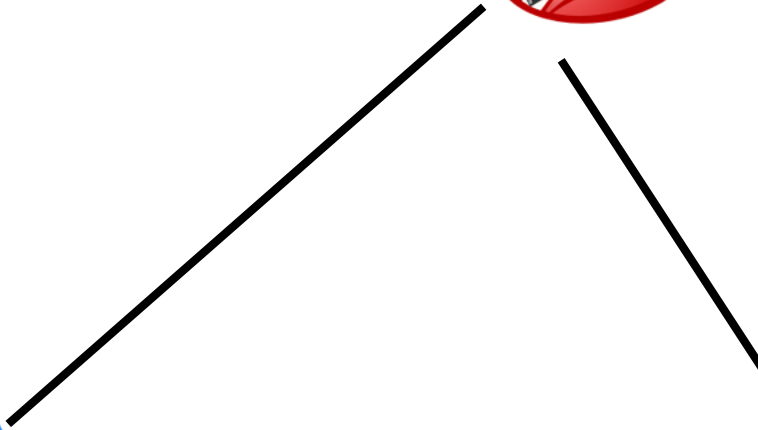
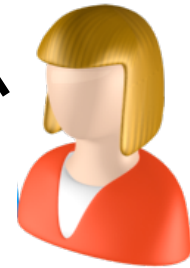
Jim



Alice



Jane





# Be careful

- People may modify the same artefacts but never get in touch
- The finer the granularity of the analysis (e.g. method level) the better

# Overlap between sources

# How Developers' Collaborations Identified from Different Sources Tell us About Code Changes

Sebastiano Panichella<sup>1</sup>, Gabriele Bavota<sup>1</sup>, Massimiliano Di Penta<sup>1</sup>, Gerardo Canfora<sup>1</sup>, Giuliano Antoniol<sup>2</sup>  
<sup>1</sup>Dept. of Engineering, University of Sannio, Italy, <sup>2</sup> École Polytechnique de Montréal, Canada

**Abstract**—Written communications recorded through channels such as mailing lists or issue trackers, but also code changes, have been used to identify emerging collaborations in software projects. Also, such data has been used to identify the relation between developers' roles in communication networks and source code changes, or to identify mentors aiding newcomers to evolve the software project. However, results of such analyses may be different depending on the communication channel being mined. This paper investigates how collaboration links vary and complement each other when they are identified through data from three different kinds of communication channels, i.e., mailing lists, issue trackers, and IRC chat logs. Also, the study investigates how such links overlap with links mined from code changes, and how the use of different sources would influence (i) the identification of project mentors, and (ii) the presence of a correlation between the social role of a developer and her changes. Results of a study conducted on seven open source projects indicate that the overlap of communication links between the various sources is relatively low, and that the application of networks obtained from different sources may lead to different results.

**Keywords**—Developers, Developer Social Network, Empirical Study

## I. INTRODUCTION

The communication among projects' members plays a paramount role in any successful software project. Indeed, team coordination and communication has always been the crux of people involved in software project management [1]. Notwithstanding the nature of a project (i.e., open source versus industrial/closed source), its domain, or size, the involved people need to exchange information effectively, minimizing the communication overhead and making sure they are up to date with the project status.

In everybody's experience, different communication channels play different, sometimes complementary sometimes alternative, roles: news can be gathered from the radio, by reading a newspaper, watching a TV broadcast or surfing blogs. Each channel has its pros and cons: TV/radio tend to be timely; Internet in addition has less control; newspapers could provide a deeper and focused treatment of some topics. Besides that, which communication channel is preferred is a mere personal choice influenced by various factors, such as the information need, the age, the culture or the life style. Much in the same way, people contributing to a project may prefer a particular communication channel. For example, general discussions about a project's perspective, software design, or future development strategies may happen in mailing lists, whereas discussions related to specific features or to the resolution of bugs occur on issue trackers. Another factor is the size, structure and general organization of the project. For

example, some projects tend to have in the past most of the discussion over mailing lists, and only in recent years they tend to use issue trackers much more. Finally, in industrial projects part of the discussion occurs through face-to-face or phone meetings [2].

In recent and past years, (written) communication has been analyzed by several authors for different purposes and exploited to support software evolution tasks. For example, Bird *et al.* [3] and Hong *et al.* [4] studied to what extent emerging teams identified from email and issue tracker communication reflect the latent structure of software projects. Bird *et al.* [5] found a correlation between social network metrics and change activities. Finally, Bettenburg *et al.* [6] and Kumar *et al.* [7] studied how social network metrics could be used for bug prediction purposes. Canfora *et al.* [8] used data from mailing lists and issue trackers to recommend mentors.

The studies mentioned above have analyzed projects' communication by observing one or two sources of communication. The conjecture we want to investigate is that, *different communication channels would provide different views of developers' interaction. As a consequence, the use of such information in recommender systems could produce different results.*

To this aim, we analyze written communication between developers (i.e., people changing the code) recorded through mailing lists, issue trackers, IRC chat logs, and code changes. The overarching goal is to provide evidence that by analyzing a single communication channel one may obtain a misleading portrait of people interaction, and that in general different combinations of the sources may provide different views of the project's interaction.

By analyzing the communication occurring in seven open source projects we show that (i) not all developers use all communication sources; (ii) people interacting using a given channel may or may not communicate through other channels; (iii) the identification of key project roles—such as developers with a high communication degree or mentors [8]—leads to different results if done over different communication channels; (iv) a study performed in the literature [5] would have achieved different findings when looking into different communication channels.

**Paper structure.** Section II presents the details of the empirical study design, selected system, approach adopted to collect and analyze data. Section III reports empirical findings and is followed by Section IV where we discuss the threats to validity. After a discussion of related work in Section V, Section VI concludes the paper and outlines directions for future work.

# Study overview

Data from 7 projects (Apache HTTPD, CXF, Hibernate, Infinispan, Lucene, Samba, Weld)

Research questions:

- 1.Overlap in terms of nodes (contributors)
- 2.Overlap in terms of edges
- 3.Topic variation
- 4.Impact on social network metrics and mentor identification

# Main findings

## **Not all contributors use all channels**

- Overlap between 40% (issue tracker and chat) and 86% (issue tracker and mailing lists)

## **The overlap between networks built upon channels is fairly limited**

- Between 20% (mail and chat) and 38% (issue tracker and mailing lists)

# Also...

- Topic model analysis revealed that contributors discuss significantly different topics across different channels
- Social network analysis metrics (degree, betweenness, etc.) significantly change
- Complementing different sources may be useful to better identify mentors



Much more beyond  
versioning systems,  
issue trackers, emails...

# Security Advisories

The screenshot shows a web browser window displaying the Mozilla Foundation Security Advisory page for MFSa 2013-62. The browser's address bar shows the URL [www.mozilla.org/security/announce/2013/mfsa2013-62.html](http://www.mozilla.org/security/announce/2013/mfsa2013-62.html). The page features the Mozilla logo and navigation links. The main heading is "Mozilla Foundation Security Advisory 2013-62". The advisory details are as follows:

<b>Title:</b>	Inaccessible updater can lead to local privilege escalation
<b>Impact:</b>	High
<b>Announced:</b>	June 25, 2013
<b>Reporter:</b>	Seb Patane
<b>Products:</b>	Firefox

**Fixed in:** Firefox 22.0

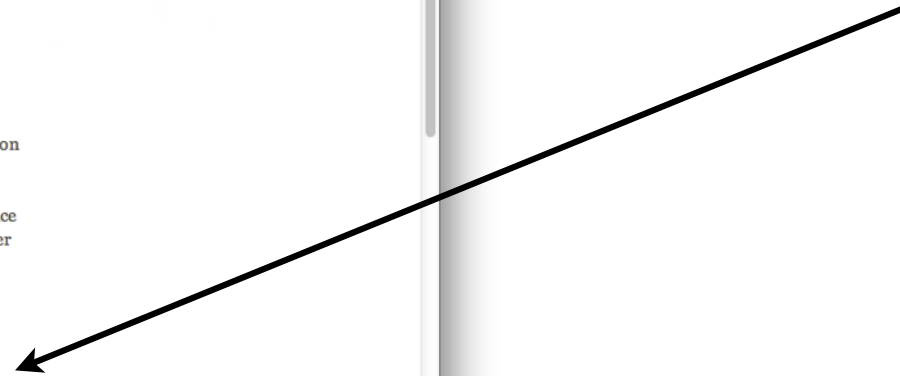
**Description**

Security researcher **Seb Patane** reported an issue with the Mozilla Maintenance Service on Windows. He discovered that when the Mozilla Updater executable was inaccessible, the Maintenance Service will behave incorrectly and can be made to use an updater at an arbitrary location. This updater will run with the system privileges used by the Maintenance Service, allowing for local privilege escalation. Local file system access is necessary in order for this issue to be exploitable and it cannot be triggered through web content.

**References**

- [Arbitrary code execution using a temporarily inaccessible file \(CVE-2013-1700\)](#)

May  
(or may  
not) be  
linked to  
issue reports



# Forums

- Projects have specialized discussion forums
- Plus, there are some general-purpose forums
- People ask questions, provide answers
  - Questions/Answers get votes
- Discussions organized according to tags
- Contributors profiled with statistics, ratings, and “badges”

# Stack Overflow

sql - selecting parent child from same table in mysql - Stack Overflow

stackoverflow.com/questions/17447762/selecting-parent-child-from-same-table-in-mysql

StackExchange v sign up log in careers 2.0 search

stackoverflow Questions Tags Tour Users Ask Question

Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required. Tell me more x

## selecting parent child from same table in mysql

It was the #Feed action

In the index.php

With the ...Empty delimiter

Stop guessing, we'll show you [Get Started Now](#)

After searching a lot here, I could not find solution to my problem. So, I am posting this question.

1 I have a Database Table which has the structure like this:

folder_id	folder_name	parent_id
1	Private	0
2	Public	0
3	Photos	0
4	December	3
5	Bday	4
6	Celeb	5

In hierarchical form, it will be like a folder structure :

```
-- Private
-- Public
-- Photos
---- December
----- Bday
----- Celeb
```

Now, I would like to select a Path upto a particular folder, like to **Bday** or **Celeb** folder. Thus, I want a MySQL query which will return me only the rows containing the folders between the path to a specific folder.

For Example, If I want a path to **Celeb** folder, then the Query should return these rows only :

tagged

- mysql x 172870
- sql x 140470
- database x 57644

asked today  
viewed 39 times  
active today

**Community Bulletin**  
blog [Say Hi to Nine of Our Newest Newbies](#)

**WOOF FROM HOME**  
WORK @ StackExchange

CAREERS 2.0



## Tagged Questions

info

newest

3 featured

frequent

votes

active

unanswered

A database is an organized collection of data typically used to model certain situations. Use this tag if you have questions about designing a database. If it is about a particular database management system, like MySQL for instance, please use that tag instead.

[learn more...](#) | [top users](#) | [synonyms \(3\)](#)

0

votes

**Suggestions for an independent BI project**

I'm attending a summer Uni course within applied programming and would appreciate a few suggestions for a simple BI project. The solution shouldn't take more than six weeks to implement, and since I ...

0

answers

[sql-server](#) [database](#) [projects-and-solutions](#) [data-warehouse](#) [business-intelligence](#)

3 views

asked 10 mins ago



Annika Nielsen

1

0

votes

1

answer

**Connecting to database at a servlet or at a managed bean**

I am new to javaEE and trying to make database connection. I can do it but i think my way of doing it is inefficient. Here is what i do: static String dbUrl="jdbc:mysql://localhost:3306/Bank"; ...

[java](#) [mysql](#) [database](#) [java-ee](#)

6 views

asked 11 mins ago



bigO

1,140 • 1 • 7

-1

votes

1

answer

**delete data on SQL table based on datatable input**

This method below goes into a table and deletes all of the table. But in the datatable "table" below in the first column there are dates which i would like to be deleted from the database. how would ...

[c#](#) [database](#) [datatable](#)

13 views

asked 17 mins ago



user2545743

4

0

votes

**Deleting Selected Part of Access Database**

I am trying to select a certain part of my access 2010 database to delete. Anything in my database older than 7 days. I have to code to compact the database i just dont know where to start for ...

[.net](#) [database](#) [vb.net](#) [delete](#) [ms-access-2010](#)

5 views

asked 23 mins ago



Chase Ernst

84 • 5

-5

**user profile front end of your site**

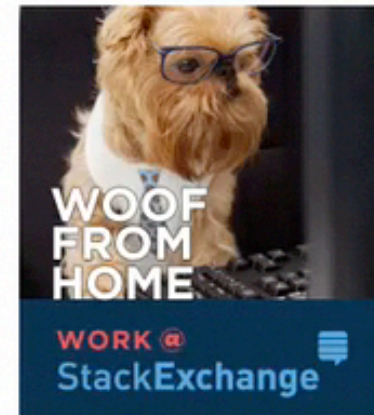
58,090

questions tagged

[database](#) [about »](#)

**Community Bulletin**

blog [Say Hi to Nine of Our Newest Newbies](#)

**CAREERS 2.0**

Senior front-end web developer  
Redbet Gaming  
Sliema, Malta / relocation

Software Developer - Network  
Applications (m/f)  
OMICRON electronics GmbH  
Feldkirch, Austria / relocation

GAME DEVELOPER AS3 H/F  
RoyalCactus  
Aix-en-Provence, France /...

**Related Tags**



# Prompter

<http://prompter.inf.usi.ch/>

The screenshot displays the Prompter IDE interface. The top window shows a code editor with the following Java code:

```
/**
 * Unzip it
 * @param zipFile input zip file
 * @param output zip file output folder
 */
public void unzipIt(String zipFile, String outputFolder){

    byte[] buffer = new byte[1024];

    try{

        //create output directory is not exists
        final File folder = new File(OUTPUT_FOLDER);
        if(!folder.exists()){
            folder.mkdir();
        }

        //get the zip file content
        ZipInputStream zis = new ZipInputStream(new FileInputStream(zipFile));
        //get the zipped file list entry
```

The right side of the interface features a Notification Center with two notifications:

- Notification 1: "Java ZIP - how to unzip folder?" with a sensitivity of 66% and a timestamp of 6/9/2013 16:04.
- Notification 2: "How to add a progress bar?" with a sensitivity of 61% and a timestamp of 6/9/2013 16:04.

The bottom window shows a Stack Overflow document titled "Java ZIP - how to unzip folder?". The document content includes the question: "Is there any sample code, how to partially unzip folder from ZIP into my desired directory? I have read all files from folder "FOLDER" into byte array, how do I recreate from its file structure?" and tags for "java" and "zip".

Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Michele Lanza: Mining StackOverflow to turn the IDE into a self-confident programming prompter. MSR 2014: 102-111





# CODES: mining method descriptions from StackOverflow

```
Document.java Query.java *IndexReader.java IndexWriter.java

public abstract void document(int docID, StoredFieldVisitor visitor) throws IOException

/**
 * To get the value of a stored field in Lucene by the internal Lucene ID,
 * use IndexReader.document(int n). If you have your own UID's indexed, you'll
 * need to search by that term, get the Lucene ID, and then call
 * IndexReader.document(int n). Are you trying to calculate PageRank on the
 * fly? If you are, that seems crazy to me. Usually PageRank is a batch
 * process that runs, and the static PageRank score that is assigned for each
 * document is added as a boost during indexing time.
 */

public final Document document(int docID) throws IOException {
    final DocumentStore documentStore = documentStore;
    documentStore.document(docID, visitor);
    return visitor.getFields();
}

public final Document document(int docID) throws IOException {
    final DocumentStore documentStore = documentStore;
    documentStore.document(docID, visitor);
    return visitor.getFields();
}
```

1

**Document org.apache.lucene.index.IndexReader.document(int docID) throws IOException**

To get the value of a stored field in Lucene by the internal Lucene ID, use IndexReader.document(int n). If you have your own UID's indexed, you'll need to search by that term, get the Lucene ID, and then call IndexReader.document(int n). Are you trying to calculate PageRank on the fly? If you are, that seems crazy to me. Usually PageRank is a batch process that runs, and the static PageRank score that is assigned for each document is added as a boost during indexing time.

Parameters:  
docID

Throws:  
IOException

Press 'F2' for focus

Sebastiano Panichella, Jairo Aponte, Massimiliano Di Penta, Andrian Marcus, Gerardo Canfora:  
Mining source code descriptions from developer communications. ICPC 2012: 63-72

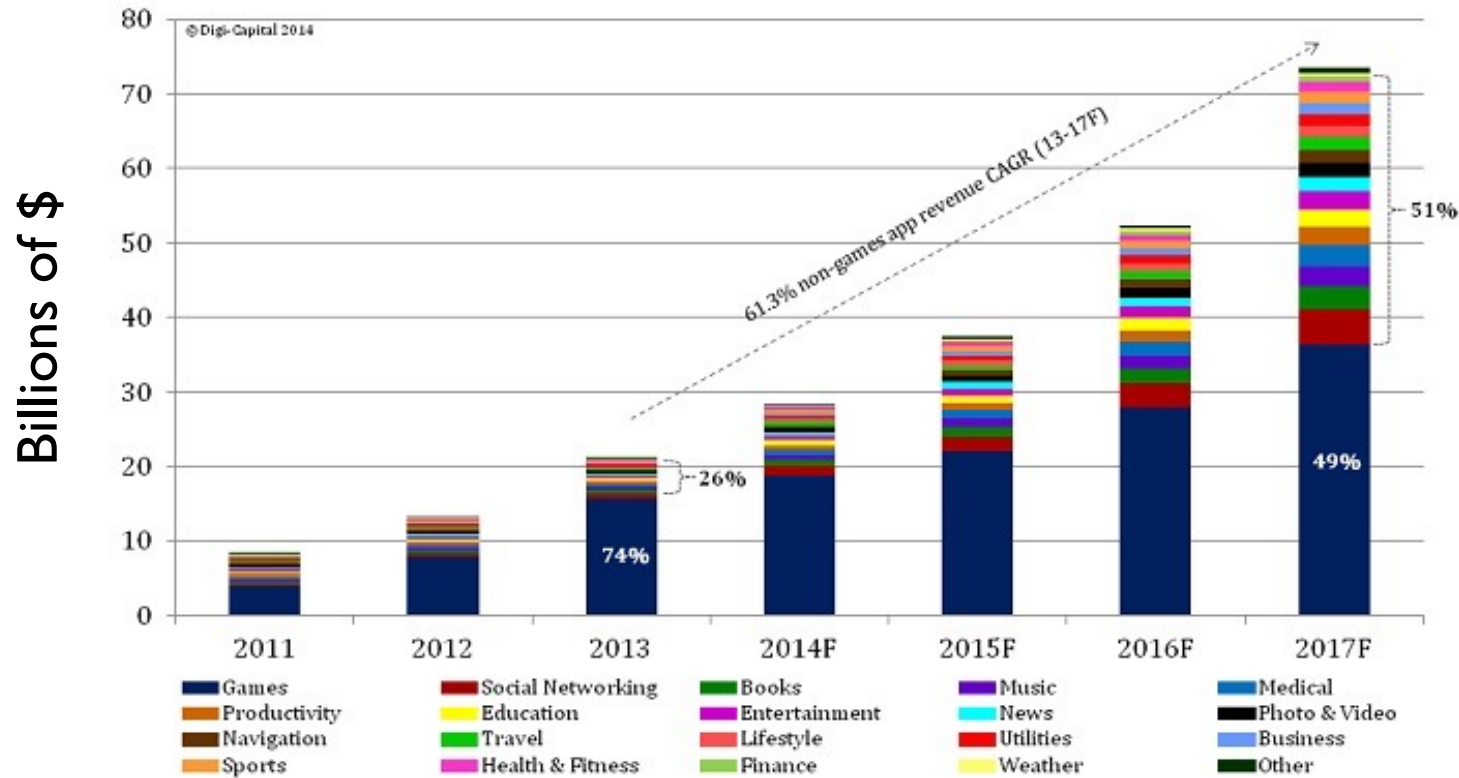
# Mining in the mobile era





# Mobile App Market Revenue

Global Mobile Apps Sector Revenue (\$B)

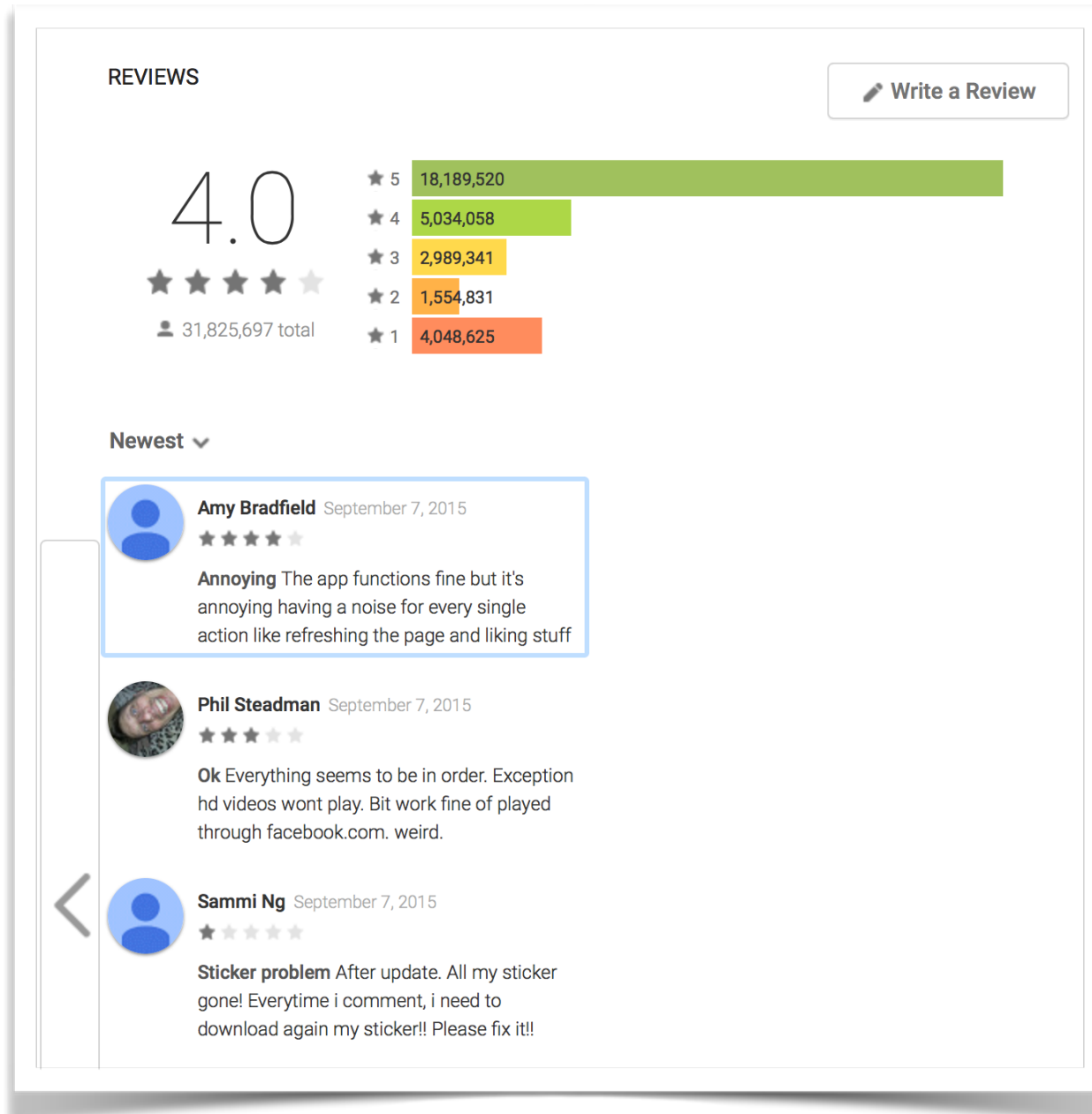


Digi-Capital™ Knowledge Relationships Ideas

Sources: Digi-Capital, PWC, App Annie, Comscore, Gartner

# Opportunities for the miner

# Reviews and user ratings



# Seminal work (one of..)

## App Store Mining and Analysis: MSR for App Stores

Mark Harman, Yue Jia, and Yuanyuan Zhang  
University College London, Malet Place, London, WC1E 6BT, UK.

**Abstract**—This paper introduces app store mining and analysis as a form of software repository mining. Unlike other software repositories traditionally used in MSR work, app stores usually do not provide source code. However, they do provide a wealth of other information in the form of pricing and customer reviews. Therefore, we use data mining to extract feature information, which we then combine with more readily available information to analyse apps' technical, customer and business aspects. We applied our approach to the 32,108 non-zero priced apps available in the Blackberry app store in September 2011. Our results show that there is a strong correlation between customer rating and the rank of app downloads, though perhaps surprisingly, there is no correlation between price and downloads, nor between price and rating. More importantly, we show that these correlation findings carry over to (and are even occasionally enhanced within) the space of data mined app features, providing evidence that our 'App store MSR' approach can be valuable to app developers.

### I. INTRODUCTION

App stores provide a rich source of information about apps concerning their customer-, business- and technically- focussed attributes. Customer information is available concerning the ratings accorded to apps by the users who have downloaded them. This provides both qualitative and quantitative data about the customer perception of the apps. Business information is available concerning the downloads and price of apps. Technical information is also available in the descriptions of apps, but it is in free text format, so data mining is required to extract the technical details required for analysis.

This is perhaps a unique situation in the history of software engineering: never before has there been a nexus of readily available information that combines the users' view, the developers' claims and the sales information pertinent to a large corpus of software products from many different providers. The combination of these three types of information provides a rich and inter-related set of data from which we can analyse and understand this new software engineering paradigm of app development. We argue that app store data mining and analysis will support the nascent app development industry, providing insights into the added value of features under consideration for new products and next releases.

To support these claims, we mine and analyse relationships between the technical, business and user perspectives for the Blackberry app store, showing how the findings can be used to inform and guide developers and managers. We study the relationships between three areas of interest: technical (through features offered), customer perceptions (through ratings and download rankings) and business (through price). In order to focus on the relationship between all three of these concerns, we consider only those apps for which there is a non-zero price.

This is the first time that such an empirical analysis of app relationships has been attempted in the literature. With this

paper we seek to introduce the study of what might be termed 'App Store Repository Mining', which is closely related to more traditional approaches to Mining Software Repositories, as we explained in the Related Work Section (Section V).

### II. APP ANALYSIS FRAMEWORK

Our approach to app store analysis consists of the four phases shown in Figure 1. The first phase extracts raw data from the app store (in this case BLACKBERRY APP WORLD<sup>1</sup>, though our approach can be applied to other app stores with suitable changes to the extraction front end). In the second phase we parse the raw data extracted in the first phase to retrieve all of the available attributes of each app relating to price, ratings and textual descriptions of the app itself. The third phase uses data mining to extract feature information from the textual descriptions and the final phase computes metrics concerning the technical, business and customer information extracted. The rest of this section explains the first three steps of our extraction and analysis approach in more detail.



Figure 1. Overall App Analysis Architecture: A four phase approach extracts, refines and stores app information for subsequent analysis.

**Phase 1 (Data Extraction):** We implemented a web crawling system to collect the raw webpage data from the app store. The crawler first collects all category information of the app store and then scans each category page to find the list of addresses of all the apps in each category, using this to locate and extract raw data on each app within each category.

**Phase 2 (Parsing):** The raw data is parsed according to a set of pattern templates, the attributes of which specify a unique searchable signature for each attribute of interest. Some attribute fields are populated by humans, so we created templates that account for the various ways in which the human might provide the equivalent information. However, once this manual step is complete the entire process is fully automated (until such time that the app store changes structure). We developed patterns to capture information about Category, Description, Price, Customers' Rating, and the Rank of Downloads of each app. To apply our approach to a different app store we need modify only the data extractor and the parsing phase to accommodate the different app store structure and data representations respectively.

**Phase 3 (Data Mining Features):** There are many ways to define a 'feature'. For our purposes, feature information

<sup>1</sup><http://appworld.blackberry.com/webstore/>

Mark Harman, Yue Jia, Yuanyuan Zhang: App store mining and analysis: MSR for app stores. MSR 2012: 108-111

# What they found...

Price is not correlated with rating, nor with the number of downloads



# The Impact of API-Change and Fault-Proneness on the User Ratings of Android Apps



# API Change and Fault Proneness: A Threat to the Success of Android Apps

Mario Linares-Vásquez<sup>1</sup>, Gabriele Bavota<sup>2</sup>, Carlos Bernal-Cárdenas<sup>3</sup>  
Massimiliano Di Penta<sup>2</sup>, Rocco Oliveto<sup>4</sup>, Denys Poshyvanyk<sup>1</sup>

<sup>1</sup>The College of William and Mary, Williamsburg, VA, USA

<sup>2</sup>University of Sannio, Benevento, Italy

<sup>3</sup>Universidad Nacional de Colombia, Bogotá, Colombia

<sup>4</sup>University of Molise, Pesche (IS), Italy

mlinarev@cs.wm.edu, gbavota@unisannio.it, cebernalc@unal.edu.co,  
dipenta@unisannio.it, rocco.oliveto@unimol.it, denys@cs.wm.edu

## ABSTRACT

During the recent years, the market of mobile software applications (apps) has maintained an impressive upward trajectory. Many small and large software development companies invest considerable resources to target available opportunities. As of today, the markets for such devices feature over 850K+ apps for Android and 900K+ for iOS. Availability, cost, functionality, and usability are just some factors that determine the success or lack of success for a given app. Among the other factors, reliability is an important criteria: users easily get frustrated by repeated failures, crashes, and other bugs; hence, abandoning some apps in favor of others.

This paper reports a study analyzing how the fault- and change-proneness of APIs used by 7,097 (free) Android apps relates to applications' lack of success, estimated from user ratings. Results of this study provide important insights into a crucial issue: making heavy use of fault- and change-prone APIs can negatively impact the success of these apps.

## Categories and Subject Descriptors

D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement

## General Terms

Measurement

## Keywords

Mining Software Repositories, Empirical Studies, Android, API changes

## 1. INTRODUCTION

According to a recent study by VisionMobile [27], the mobile handset industry has been growing at 23% Compound

Annual Growth Rate (CAGR) in revenues since 2009. The "App" economy is a tremendous success: iOS, BlackBerry, and Android were the most lucrative software platforms in 2012, with average monthly revenue of over \$4,800, \$3,700, and \$3,300 per app, respectively [26]. Additionally, the developers' mindshare index during the last three years (2010-2012) shows that iOS and Android are the top two software platforms being used by developers worldwide [26, 27].

What are the hidden forces that contribute to the app economy's success? Typical answers are: ubiquitous computing, low cost of handsets (especially, the Android devices), monetization models, customers' loyalty to brands such as iPhone or BlackBerry, etc. However, beyond explaining the "hidden forces" that drive consumer/developer decisions and define the reasons for the success of the apps, that success can be influenced by the software infrastructure that developers use to build apps (i.e., Application Programming Interfaces - APIs).

APIs encapsulate the complexity of low-level programming details, and provide developers with a high-level model for using the underlying hardware. However, the ease-of-use of these APIs is impacted by factors related to API design and quality. For instance, top categories of API learning obstacles are related to learning resources (e.g., documentation, or code examples) and API structure (e.g., design or name of API elements) [19]. Also, APIs not ensuring backward compatibility support are typically hard to use because their instability [28], and API breaking-changes could introduce bugs in the client code. Moreover, since developers often assume correctness behind underlying APIs, faults in APIs can drastically impact the client code quality as perceived by the end-users. For example, Zibran *et al.* [29] found that among 1,513 bug reports related to various components of Eclipse, GNOME, MySQL, Python 3.1, and Android projects, 562 bug-reports were related to API usability issues; and about 175 (31.1%) of those issues were related to API correctness. *Although one can possibly assume that API instability (change-proneness) and fault-proneness may impact the success of software applications, to the best of our knowledge such relations have not been empirically investigated yet.*

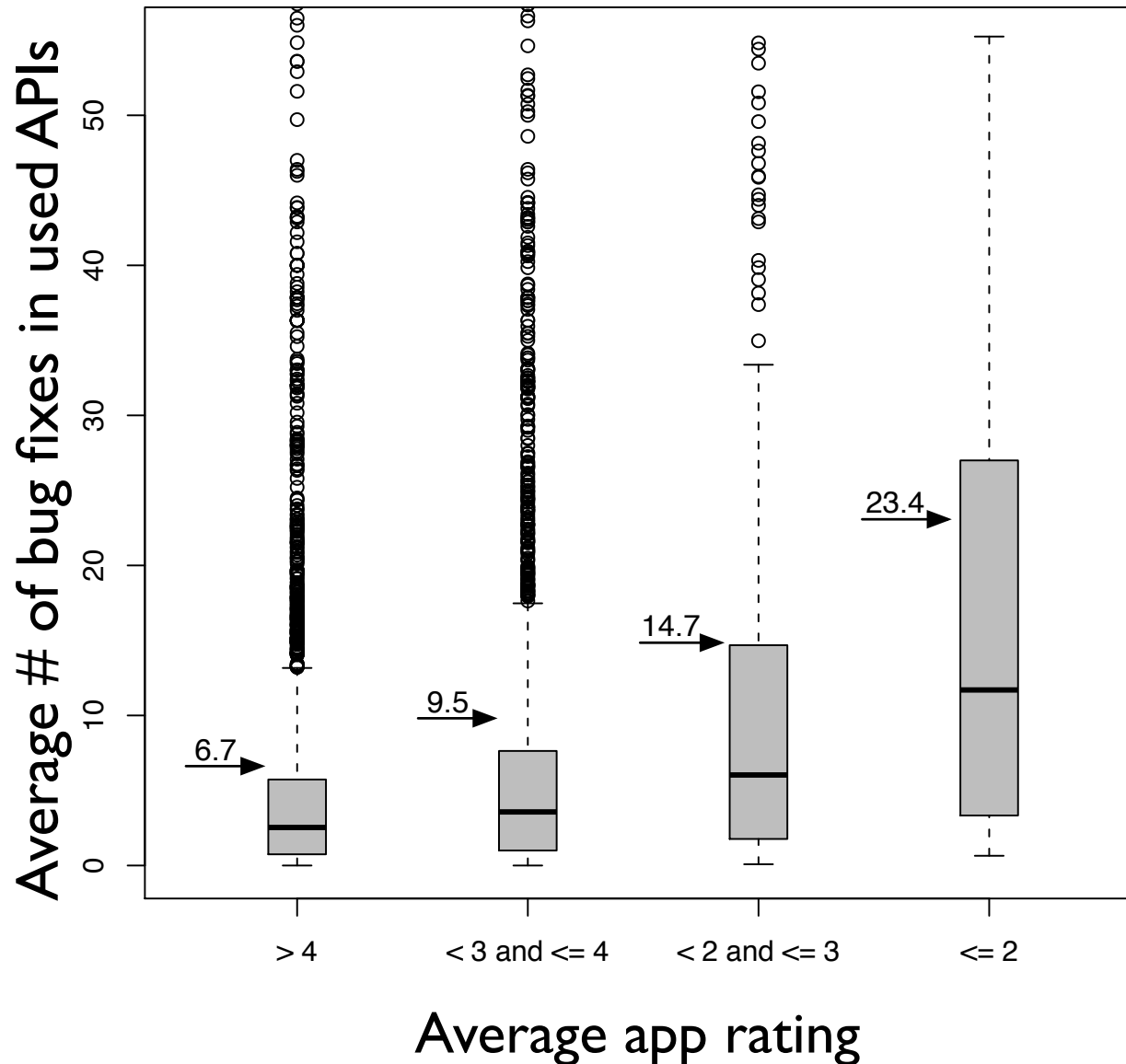
The goal of this paper is to provide solid empirical evidence about the relation between the success of apps (in terms of user ratings), and the change- and fault-proneness of the underlying APIs. The study has been conducted on

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

ESEC/FSE'13, August 18–26, 2013, Saint Petersburg, Russia  
Copyright 2013 ACM 978-1-4503-2237-9/13/08...\$15.00  
<http://dx.doi.org/10.1145/2491411.2491428>

Mario Linares Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Massimiliano Di Penta, Rocco Oliveto, Denys Poshyvanyk: API change and fault proneness: a threat to the success of Android apps. ESEC/SIGSOFT FSE 2013: 477-487

# Rating and Fault Prone APIs



Wilcoxon U test yield significant differences, medium effect size

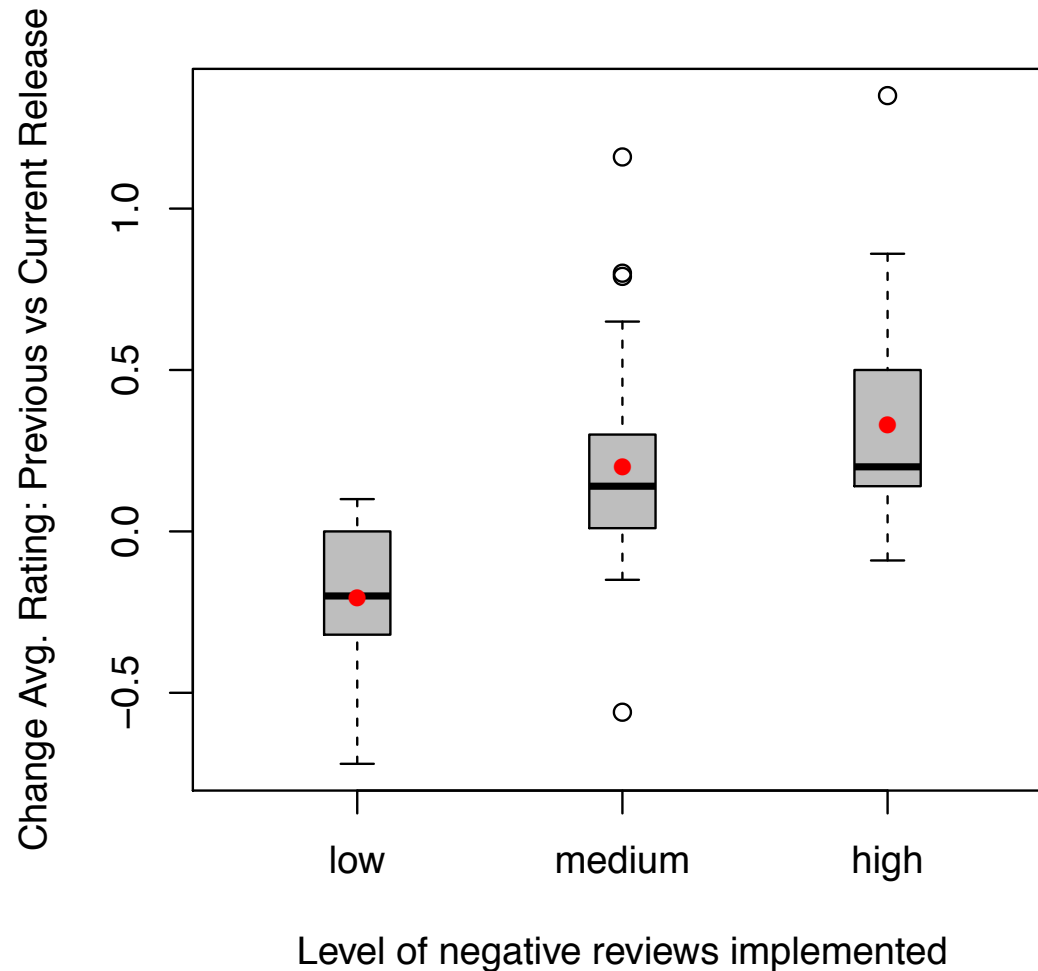




# CRISTAL: Crowdsourcing Reviews to Support Apps evolution

Fabio Palomba, Mario Linares-Vasquez, Gabriele Bavota,  
Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk and Andrea De Lucia

# What happens if negative reviews are implemented?



Low:  $\leq Q1$  Medium:  $Q1-Q3$  High:  $>Q3$

# Mining Mobile Apps: Challenges and Perils

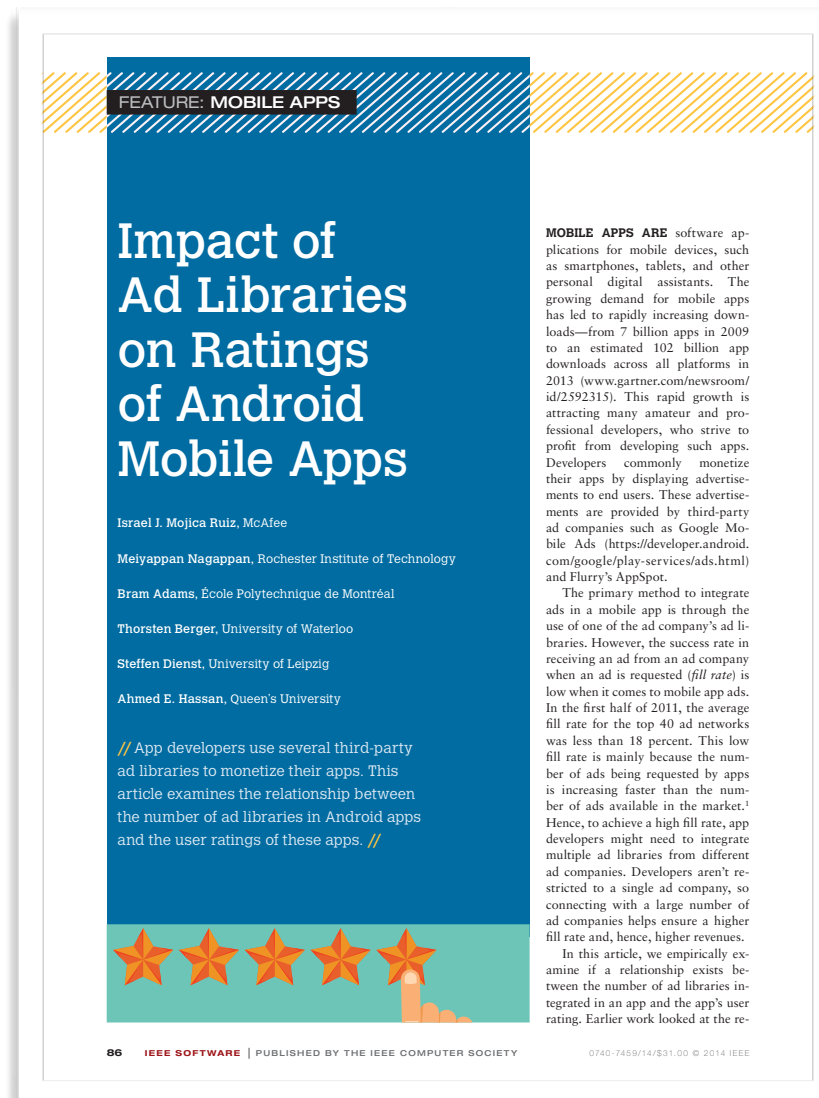


# Ad (side) effects

Many app dependencies are not towards libraries/services implementing features

Negative role on.....

# Effects on rating



Israel J. Mojica Ruiz, Meiyappan Nagappan, Bram Adams, Thorsten Berger, Steffen Dienst, Ahmed E. Hassan: Impact of Ad Libraries on Ratings of Android Mobile Apps. IEEE Software 31(6): 86-92 (2014)

# Effects on performance and consumption

## Truth in Advertising: The Hidden Cost of Mobile Ads for Software Developers

Jiaping Gui\*, Stuart Mcilroy<sup>†</sup>, Meiyappan Nagappan<sup>‡</sup> and William G. J. Halfond\*

\*University of Southern California, Los Angeles, CA, USA

Email: {jgui, halfond}@usc.edu

<sup>†</sup>Queen's University, Kingston, Canada

Email: mcilroy@cs.queensu.ca

<sup>‡</sup>Rochester Institute of Technology, Rochester, NY, USA

Email: mei@se.rit.edu

**Abstract**—The “free app” distribution model has been extremely popular with end users and developers. Developers use mobile ads to generate revenue and cover the cost of developing these free apps. Although the apps are ostensibly free, they in fact do come with hidden costs. Our study of 21 real world Android apps shows that the use of ads leads to mobile apps that consume significantly more network data, have increased energy consumption, and require repeated changes to ad related code. We also found that complaints about these hidden costs are significant and can impact the ratings given to an app. Our results provide actionable information and guidance to software developers in weighing the tradeoffs of incorporating ads into their mobile apps.

**Index Terms**—Mobile advertisements, mobile devices

### I. INTRODUCTION

Mobile advertising has become an important part of many software developers’ marketing and advertising strategy [1]. This development has come about in just a matter of a few years. In 2010, the mobile advertising industry’s revenue was just over half a billion dollars [2], but by 2013 it reached over 17 billion dollars [3], and in the first quarter of 2014, had already reached over 11 billion dollars [4]. By 2017, analysts predict that revenue from mobile advertising will exceed that of TV advertisements [5] and account for one in three dollars spent on advertising [6].

The presence of mobile ads has become pervasive in the app ecosystem with, on average, over half of all apps containing ads [7]. This has been driven by the development of large scale advertising networks, such as Google Mobile Ads and Apple iAd, that facilitate the interaction between developers and advertisers. To earn ad revenue, developers display ads in their apps by making calls to APIs provided by an advertising network. When the ads are displayed on an end user’s device, the developer receives a small payment. A typical business model for a developer is to place ads in their apps and then release the app for free with the hope that the ad revenue will offset the cost of the app’s development. In general, this model is perceived as a win-win situation for both developers and end users: developers receive a steady, and sometimes large, ad-driven revenue stream, and end users receive a “free” app.

A key problem in this model is that it depends on the perception that, aside from app development, there are no

additional costs to either the end user or software developer. While this is true for direct costs, this fails to account for the indirect hidden costs of embedding mobile ads in an app. On the end users’ side, indirect hidden costs come in several forms: loading ads from a remote server requires network usage, for which many users are billed by the amount of bytes; loading and rendering ads requires CPU time and memory, which can slow down the performance of an app; and finally, all of these activities require battery power, which is a limited resource on a mobile device. Developers have hidden costs as well. It is necessary to maintain the code that interacts with the advertisements, which requires developer effort. The ratings and reviews a developer receives can also be affected. Studies have shown that over 70% of users find in-app ads “annoying” [8] and such users may give an app a lower rating or write negative reviews. This negative response may then affect the number of downloads of an app, which in turn can affect the developer’s future ad revenue.

In this paper we present the results of our investigation into the hidden costs of mobile advertising for software developers. To carry out this investigation, we performed an extensive empirical analysis of 21 real world apps from the Google Play app store that make use of mobile advertising. Our analysis considered five types of hidden costs: app performance, energy consumption, network usage, maintenance effort for ad-related code, and app reviews. The results of our investigation show that there is, in fact, a high hidden cost of ads in mobile apps. Our results show that apps with ads consume, on average: 48% more CPU time, 16% more energy, and 79% more network data. We also found that developers, on average, make ad related changes in 23% of their releases. The presence of mobile ads also has a rating and review cost, as we found that complaints related to ads and these hidden costs were relatively frequent and had a measurable impact on an app’s rating. Overall, we believe that these findings are significant and will help to inform software developers so they can better weigh the tradeoffs of incorporating ads into their mobile apps, understand the impact ads have on their end users, and improve end users’ app experience.

# Mining challenge: limited availability of repositories



Even for open source apps, you might not be able to find all kinds of software repositories

Often the observable history is fairly limited



# Don't in mining mobile apps

- In most cases apps are small projects
- They have been developed by small (or singleton) teams
- Some MSR research questions might be less relevant in this context
  - Bug triaging / identifying experts
  - (maybe) Identifying fault prone components





# Not all reviews are equal

## AR-Miner: Mining Informative Reviews for Developers from Mobile App Marketplace

Ning Chen, Jialiu Lin<sup>1</sup>, Steven C. H. Hoi, Xiaokui Xiao, Boshen Zhang  
Nanyang Technological University, Singapore, <sup>1</sup>Carnegie Mellon University, USA  
{nchen1, chhoi, xkxiao, bszhang}@ntu.edu.sg, <sup>1</sup>jialiul@cs.cmu.edu

### ABSTRACT

With the popularity of smartphones and mobile devices, mobile application (a.k.a. “app”) markets have been growing exponentially in terms of number of users and downloads. App developers spend considerable effort on collecting and exploiting user feedback to improve user satisfaction, but suffer from the absence of effective user review analytics tools. To facilitate mobile app developers discover the most “informative” user reviews from a large and rapidly increasing pool of user reviews, we present “AR-Miner” — a novel computational framework for App Review Mining, which performs comprehensive analytics from raw user reviews by (i) first extracting informative user reviews by filtering noisy and irrelevant ones, (ii) then grouping the informative reviews automatically using topic modeling, (iii) further prioritizing the informative reviews by an effective review ranking scheme, (iv) and finally presenting the groups of most “informative” reviews via an intuitive visualization approach. We conduct extensive experiments and case studies on four popular Android apps to evaluate AR-Miner, from which the encouraging results indicate that AR-Miner is effective, efficient and promising for app developers.

### Categories and Subject Descriptors

D.2 [Software]: Software Engineering; H.4 [Information Systems Applications]: Miscellaneous

### General Terms

Algorithm and Experimentation

### Keywords

User feedback, mobile application, user reviews, data mining

## 1. INTRODUCTION

The proliferation of smartphones attracts more and more software developers to devote to building mobile applications (“apps”). As the market competition is becoming more

intense, in order to seize the initiative, developers tend to employ an iterative process to develop, test, and improve apps [23]. Therefore, timely and constructive feedback from users becomes extremely crucial for developers to fix bugs, implement new features, and improve user experience agilely. One key challenge to many app developers is how to obtain and digest user feedback in an effective and efficient manner, i.e., the “user feedback extraction” task. One way to extract user feedback is to adopt typical channels used in traditional software development, such as (i) bug/change repositories (e.g., Bugzilla [3]), (ii) crash reporting systems [19], (iii) online forums (e.g., SwiftKey feedback forum [6]), and (iv) emails [10].

Unlike the traditional channels, modern app marketplaces, such as Apple App Store and Google Play, offer a much easier way (i.e., the web-based market portal and the market app) for users to rate and post app reviews. These reviews present user feedback on various aspects of apps (such as functionality, quality, performance, etc), and provide app developers a new and critical channel to extract user feedback. However, comparing with traditional channels, there are two outstanding obstacles for app developers to obtain valuable information from this new channel. First of all, the proportion of “informative” user reviews is relatively low. In our study (see Section 5.1), we found that only 35.1% of app reviews contain information that can directly help developers improve their apps. Second, for some popular apps, the volume of user reviews is simply too large to do manual checking on all of them. For example, Facebook app on Google Play receives more than 2000 user reviews per day, making it extremely time consuming to do manual checking.

To our best knowledge, very few studies were focused on extracting valuable information for developers from *user reviews in app marketplace* [28, 21, 22]. This paper formally formulates this as a new research problem. Specifically, to address this challenging problem and tackle the aforementioned two obstacles, we propose a novel computational framework, named “AR-Miner” (App Review Miner), for extracting valuable information from raw user review data with minimal human efforts by exploring effective data mining and ranking techniques. Generally speaking, given a bunch of user reviews of an app collected during a certain time interval, AR-Miner first filters out those “non-informative” ones by applying a pre-trained classifier. The remaining “informative” reviews are then put into several groups, and prioritized by our proposed novel ranking model. Finally, we visualize the ranking results in a concise and

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [Permissions@acm.org](mailto:Permissions@acm.org).

ICSE '14, May 31 – June 7, 2014, Hyderabad, India  
Copyright 2014 ACM 978-1-4503-2756-5/14/05...\$15.00  
<http://dx.doi.org/10.1145/2568225.2568263>

Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, Boshen Zhang: AR-miner: mining informative reviews for developers from mobile app marketplace. ICSE 2014: 767-778

# Informative reviews...

Functional

None of the pictures will load in my news feed.

Performance

It lags and doesn't respond to my touch which almost always causes me to run into stuff.

Feature (change)  
request

Amazing app, although I wish there were more themes to choose from

Remove ads

So many ads its unplayable!

Fix permissions

This game is adding for too much unexplained permissions.

# Non-Informative

Purely emotional

Great fun can't put it down!

This is a crap app.

Description  
of app/actions

I have changed my review from 2 star to 1 star.

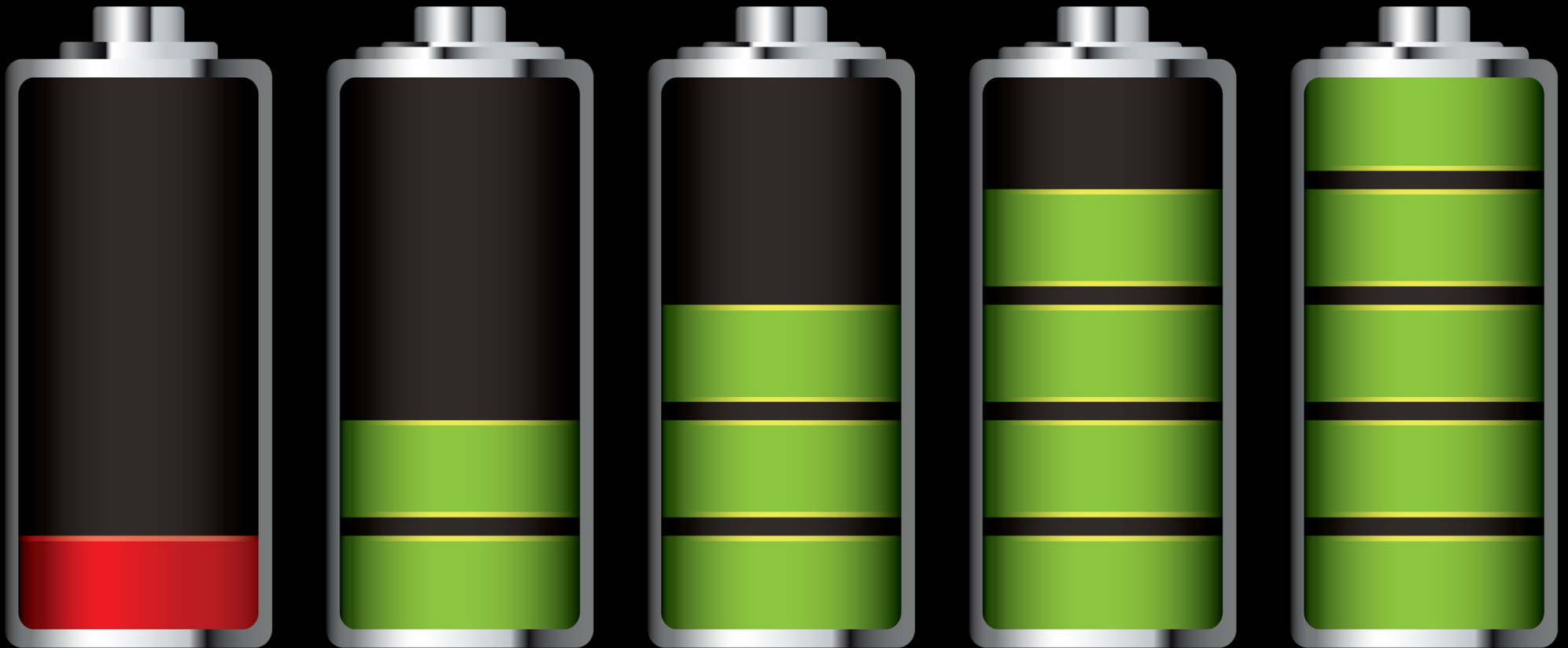
Unclear issue  
description

Bad game this is not working on my phone.

Questions

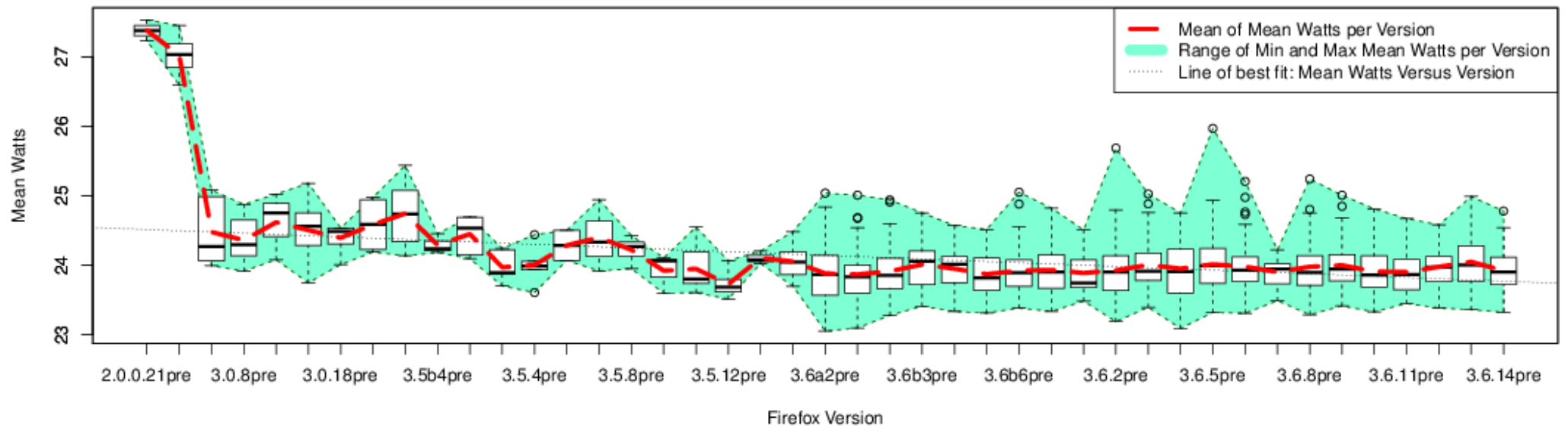
How can I get more points?

# Green (or Energy) Mining



# Green mining

Wattage of Browsing Tests per version of Firefox 3.6 (and a sampling of earlier versions)



<http://softwareprocess.es/static/GreenMining.html>

Abram Hindle: Green mining: A methodology of relating software change to power consumption. MSR 2012: 78-87



# Identifying Energy Greedy APIs

## Mining Energy-Greedy API Usage Patterns in Android Apps: An Empirical Study

Mario Linares-Vásquez<sup>1</sup>, Gabriele Bavota<sup>2</sup>, Carlos Bernal-Cárdenas<sup>3</sup>  
Rocco Oliveto<sup>4</sup>, Massimiliano Di Penta<sup>2</sup>, Denys Poshyvanyk<sup>1</sup>

<sup>1</sup>The College of William and Mary, Williamsburg, VA, USA    <sup>2</sup>University of Sannio, Benevento, Italy  
<sup>3</sup>Universidad Nacional de Colombia, Bogotá, Colombia    <sup>4</sup>University of Molise, Pesche (IS), Italy  
mlinarev@cs.wm.edu, gbavota@unisannio.it, cebernalc@unal.edu.co,  
rocco.oliveto@unimol.it, dipenta@unisannio.it, denys@cs.wm.edu

### ABSTRACT

Energy consumption of mobile applications is nowadays a hot topic, given the widespread use of mobile devices. The high demand for features and improved user experience, given the available powerful hardware, tend to increase the apps' energy consumption. However, excessive energy consumption in mobile apps could also be a consequence of energy greedy hardware, bad programming practices, or particular API usage patterns. We present the largest to date quantitative and qualitative empirical investigation into the categories of API calls and usage patterns that—in the context of the Android development framework—exhibit particularly high energy consumption profiles. By using a hardware power monitor, we measure energy consumption of method calls when executing typical usage scenarios in 55 mobile apps from different domains. Based on the collected data, we mine and analyze energy-greedy APIs and usage patterns. We zoom in and discuss the cases where either the anomalous energy consumption is unavoidable or where it is due to suboptimal usage or choice of APIs. Finally, we synthesize our findings into actionable knowledge and recipes for developers on how to reduce energy consumption while using certain categories of Android APIs and patterns.

### Categories and Subject Descriptors

D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement

### General Terms

Measurement

### Keywords

Energy consumption, Mobile applications, Empirical Study

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [Permissions@acm.org](mailto:Permissions@acm.org).

MSR '14, May 31 – June 1, 2014, Hyderabad, India  
Copyright 2014 ACM 978-1-4503-2863-0/14/05...\$15.00  
<http://dx.doi.org/10.1145/2597073.2597085>

### 1. INTRODUCTION

In recent years, we are observing rapid evolution of mobile devices in terms of available hardware, operating systems, and, as a consequence of that, the growing lists of features that mobile applications' (apps) users demand. These modern apps have virtually the same features as their equivalent desktop applications. For instance, many top video games for mobile devices provide similar levels of user experience as compared to those console analogs. Such evident step-ahead has, however, a price to be paid. Nowadays, multi-core processors, high-performance Graphical Processing Units (GPUs), and large screens on mobile devices are becoming more energy demanding as ever. Also, apps fully exploiting available hardware can easily drain devices' batteries in no time.

From a user's perspective, this produces tangible and pertinent problems. The use of energy-draining apps could quickly leave a user with empty battery, preventing her from using the smartphone even for phone calls. In addition, having and running such apps might require frequent battery re-charges. This represents a problem because modern battery's life is quite limited, often to a finite amount of charging cycles (for Lithium-ion batteries), ranging between 300 and 500 cycles (with only 100-200 cycles after a mid-life point) and gradually decreasing with time [4, 5].

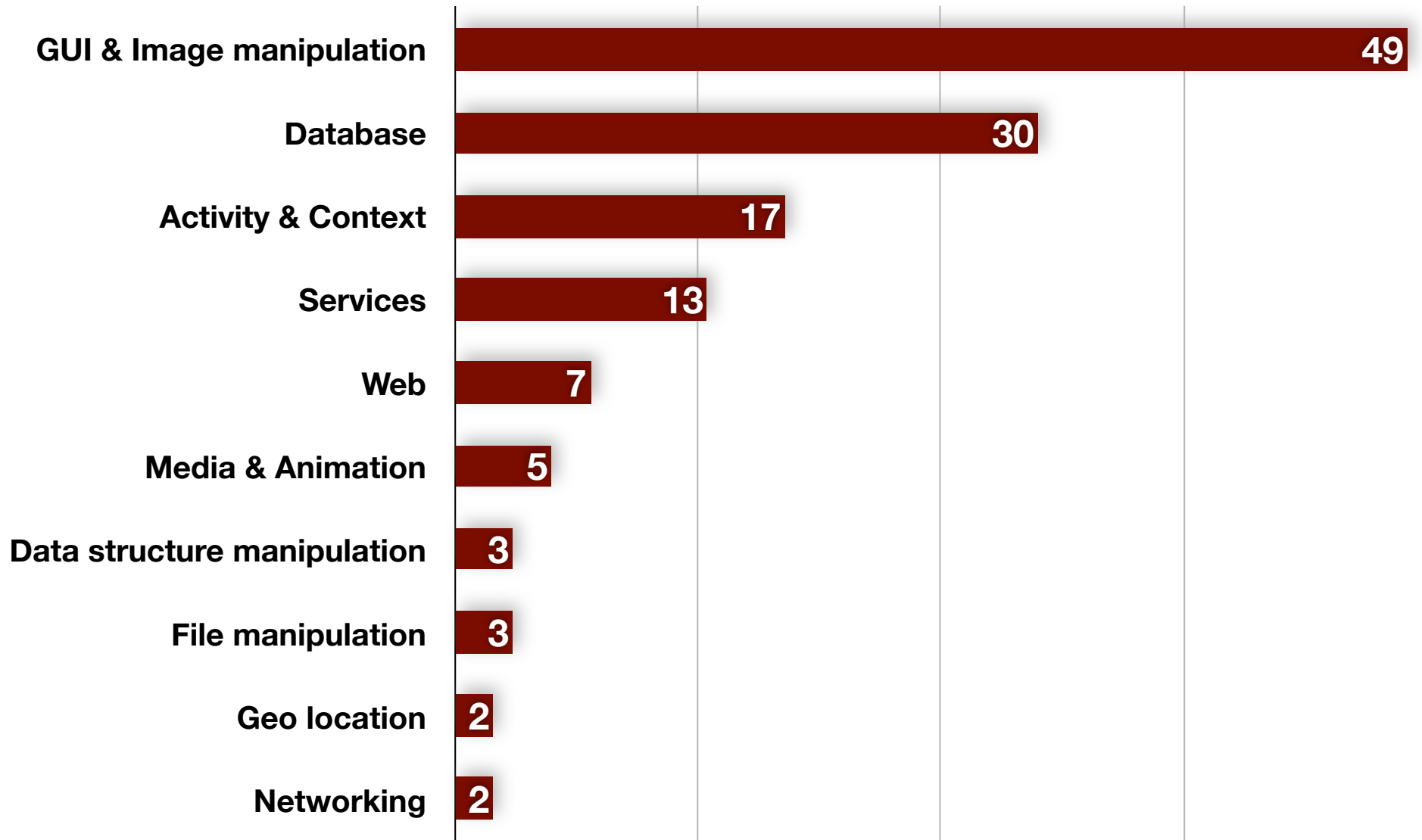
A practical, although naïve advice for preventing rapid discharges and for improving batteries' life, is to use mobile devices only for low energy consuming scenarios. However, while it might be obvious that some apps are likely to be power demanding—e.g., video games or those apps using devices such as Global Positioning Systems (GPS)—it can often happen that some apps might quickly drain the battery without any apparent reason [32, 33]. For instance, several studies identified misuses of *wakelocks* that keep hardware components unnecessarily awake as causes of high energy consumption in mobile devices [22, 32, 33, 35].

Also, programming errors, hardware interactions, and API misuses can cause high levels of energy consumption (also known as *energy bugs*) in mobile apps [32]. To identify such problems, effective strategies for measuring energy consumption in mobile devices are needed. In the literature, several different strategies have been proposed, based on real measurements [6, 11, 21, 23, 25, 40] and power modeling [19, 20, 33, 34, 43, 46]. While previous work attempted at characterizing energy bugs in mobile devices [6, 7, 19, 33, 40], most of these classifications have been done either by mining software repositories (e.g., bug reports, forums, commit logs)

Ok these are the obvious things...



# Android Energy-Greedy APIs







# Mining biometric info

# Stuck and Frustrated or In Flow and Happy: Sensing Developers' Emotions and Progress

Sebastian C. Müller, Thomas Fritz  
Department of Informatics, University of Zurich, Switzerland  
{smueller, fritz}@ifi.uzh.ch

**Abstract**—Software developers working on change tasks commonly experience a broad range of emotions, ranging from happiness all the way to frustration and anger. Research, primarily in psychology, has shown that for certain kinds of tasks, emotions correlate with progress and that biometric measures, such as electro-dermal activity and electroencephalography data, might be used to distinguish between emotions. In our research, we are building on this work and investigate developers' emotions, progress and the use of biometric measures to classify them in the context of software change tasks. We conducted a lab study with 17 participants working on two change tasks each. Participants were wearing three biometric sensors and had to periodically assess their emotions and progress. The results show that the wide range of emotions experienced by developers is correlated with their perceived progress on the change tasks. Our analysis also shows that we can build a classifier to distinguish between positive and negative emotions in 71.36% and between low and high progress in 67.70% of all cases. These results open up opportunities for improving a developer's productivity. For instance, one could use such a classifier for providing recommendations at opportune moments when a developer is stuck and making no progress.

## I. INTRODUCTION

Frustration, anger, happiness and enthusiasm are emotions that software developers frequently experience during their work [1]. These emotions are commonly intertwined with the progress one makes, such as experiencing positive emotions leading to more progress [2] or the state of being stuck and making no progress leading to frustration [3]. Research in psychology has already shown that there is a correlation between these two dimensions, the emotions and the progress people experience for certain kinds of tasks (e.g. [4]). To help ensure a developer's time is spent as productive as possible, an indicator for a developer's emotions could thus be used to prevent interruptions when a developer is "in flow", making a lot of progress and should not be disturbed, or to provide recommendations at opportune moments when the developer is getting frustrated and close to being stuck.

With the recent advances in biometric (*aka* psychophysiological) sensor technology, an increasing amount of research in psychology has shown that a person's biometric features, such as skin temperature, facial expression or respiration rate, can be used to detect and distinguish between emotions (e.g. [5], [6]). Psychology research has also shown that biometric measures can be used to determine a flow or stuck state (e.g. [3], [7]). However, these studies are focused on small analytical tasks or physics exercises and do not provide

any evidence on its applicability to software development tasks, in particular, given the complexity and emotions as well as cognitive skills these kinds of tasks stress in humans.

In software engineering, only little research has focused on developers' emotions and the use of biometric measures. For emotions, researchers have looked at the emotions that developers experience [1], [8], how they might affect productivity [9], [10], and whether one could use interaction logs to predict them [11], [12]. Using biometric sensors, in particular eye-tracking and fMRI, researchers have mainly studied how software developers comprehend code or use tools [13]–[15]. In a previous study, we looked at the use of biometric sensors to assess the difficulty of small code comprehension tasks [16].

In the research presented in this paper, we built upon existing work in software engineering and psychology and further investigate emotions and progress developers experience, as well as the use of biometric sensors to predict them in the context of change tasks. In particular, we are interested in the following three research questions:

- RQ1:** What is the range of developers' emotions during change tasks and are developers' emotions correlated with their perceived progress?
- RQ2:** What are aspects and practices that affect developers' emotions and progress during change tasks?
- RQ3:** Can we use biometric sensors to determine developers' emotions and progress during change tasks?

To address our research questions, we performed a study with 17 participants. In this study, participants worked on two change tasks for 30 minutes each while we recorded various biometric measures and periodically probed the participants for their emotions and progress. The results of our study show that developers experience a broad range of positive and negative emotions during change tasks that are similar to the ones experienced in other situations and that these emotions are highly correlated with progress, further supporting Graziotin *et al.*'s finding [9]. The results also show that the localization and understanding of relevant code are the most common aspects for emotions and progress to change. Using the biometric data gathered throughout the study, we trained a machine learning classifier that is able to distinguish between positive and negative emotions with an accuracy of 71.36% and between low and high progress with an accuracy of 67.70%.

This paper makes the following contributions:

Sebastian C. Müller, Thomas Fritz: Stuck and Frustrated or in Flow and Happy: Sensing Developers' Emotions and Progress. ICSE (I) 2015: 688-699

## Understanding Understanding Source Code with Functional Magnetic Resonance Imaging

Janet Siegmund<sup>\*</sup>, Christian Kästner<sup>\*,</sup> Sven Apel<sup>\*,</sup> Chris Parnin<sup>†,</sup> Anja Bethmann<sup>‡,</sup>  
 Thomas Leich<sup>§,</sup> Gunter Saake<sup>¶,</sup> and André Brechmann<sup>¶</sup>  
<sup>\*</sup>University of Passau, Germany <sup>†</sup>Carnegie Mellon University, USA  
<sup>‡</sup>Georgia Institute of Technology, USA <sup>§</sup>Leibniz Inst. for Neurobiology Magdeburg, Germany  
<sup>¶</sup>Metop Research Institute, Magdeburg, Germany <sup>¶</sup>University of Magdeburg, Germany

### ABSTRACT

Program comprehension is an important cognitive process that inherently eludes direct measurement. Thus, researchers are struggling with providing suitable programming languages, tools, or coding conventions to support developers in their everyday work. In this paper, we explore whether *functional magnetic resonance imaging (fMRI)*, which is well established in cognitive neuroscience, is feasible to more directly measure program comprehension. In a controlled experiment, we observed 17 participants inside an fMRI scanner while they were comprehending short source-code snippets, which we contrasted with locating syntax errors. We found a clear, distinct activation pattern of five brain regions, which are related to working memory, attention, and language processing—all processes that fit well to our understanding of program comprehension. Our results encourage us and, hopefully, other researchers to use fMRI in future studies to measure program comprehension and, in the long run, answer questions, such as: Can we predict whether someone will be an excellent programmer? How effective are new languages and tools for program understanding? How should we train developers?

### 1. INTRODUCTION

As the world becomes increasingly dependent on the billions lines of code written by software developers, little comfort can be taken in the fact that we still have no fundamental understanding of how developers understand source code.

Understanding program comprehension is not limited to theory building, but can have real downstream effects in improving education, training, and the design and evaluation of tools and languages for programmers. If direct measures of cognitive effort and difficulty could be obtained and correlated with programming activity, then researchers could identify and quantify which types of activities, segments of code, or kinds of problem solving are troublesome or improved with the introduction of a new language or tool.

In studying programmers, decades of psychological and observational experiments have relied on indirect techniques, such as com-

<sup>\*</sup>This author published previous work as Janet Feigenspan.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.  
 Copyright 20XX ACM X-XXXXX-XX-XX/XX/XX ...\$10.00.

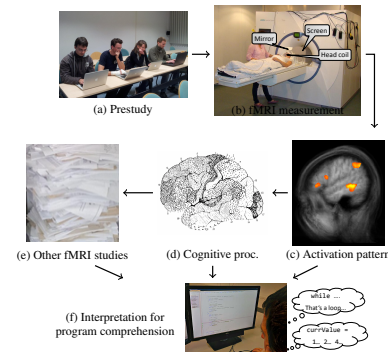


Figure 1: Workflow of our fMRI study.

paring task performance or having programmers articulate their thoughts in think-aloud protocols. Each method, when skillfully applied, can yield important insights. However, these common techniques are not without problems. In human studies of programming, individual [13] and task variance [18] in performance often mask any significant effects hoping to be found when evaluating, say, a new tool. Think-aloud protocols and surveys rely on self-reporting and require considerable manual transcription and analysis that garner valuable but indefinite and inconsistent insight.

In the past few decades, psychologists and cognitive neuroscientists have collectively embraced methods that measure physiological correlates of cognition as a standard practice. One such method is *functional magnetic resonance imaging (fMRI)*, a non-invasive means of measuring blood-oxygenation levels that change as a result of localized brain activity.

In this paper, we report on results and experience from applying fMRI in a program-comprehension experiment. While our experiment is a first step toward measuring program comprehension with fMRI, and as such inherently limited, we believe this study can illuminate a path toward future studies that systematically explore hypotheses and that can be used to build stronger theories of program comprehension.

# Mining Forges and Code Search Engines

# Some Examples

The Maven Repository <http://search.maven.org>

SourceForge <http://sourceforge.net>

GitHub <https://github.com>

OpenHub <https://www.openhub.net>

# Mining GitHub: Promises and Perils

# The Promises and Perils of Mining GitHub

Eirini Kalliamvakou  
University of Victoria  
ikalliam@uvic.ca

Leif Singer  
University of Victoria  
lsinger@uvic.ca

Georgios Gousios  
Delft University of Technology  
G.Gousios@tudelft.nl

Daniel M. Germán\*  
University of Victoria  
dmg@uvic.ca

Kelly Blincoe  
University of Victoria  
kblincoe@acm.org

Daniela Damian  
University of Victoria  
danielad@cs.uvic.ca

## ABSTRACT

With over 10 million `git` repositories, GitHub is becoming one of the most important source of software artifacts on the Internet. Researchers are starting to mine the information stored in GitHub's event logs, trying to understand how its users employ the site to collaborate on software. However, so far there have been no studies describing the quality and properties of the data available from GitHub. We document the results of an empirical study aimed at understanding the characteristics of the repositories in GitHub and how users take advantage of GitHub's main features—namely commits, pull requests, and issues. Our results indicate that, while GitHub is a rich source of data on software development, mining GitHub for research purposes should take various potential perils into consideration. We show, for example, that the majority of the projects are personal and inactive; that GitHub is also being used for free storage and as a Web hosting service; and that almost 40% of all pull requests do not appear as merged, even though they were. We provide a set of recommendations for software engineering researchers on how to approach the data in GitHub.

## Categories and Subject Descriptors

D.2.8 [Software Engineering]: Management—*Software configuration management*

## General Terms

Software Engineering

## Keywords

Mining software repositories, `git`, GitHub, code reviews.

## 1. INTRODUCTION

GitHub is a collaborative code hosting site built on top of the `git` version control system. GitHub introduced a

\*Corresponding Author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.  
MSR14 Hyderabad, India  
Copyright 2007 ACM X-XXXXX-XX-XX/XX ...\$15.00.

“fork & pull” model in which developers create their own copy of a repository and submit a pull request when they want the project maintainer to pull their changes into the main branch. In addition to code hosting, collaborative code review, and integrated issue tracking, GitHub has integrated social features. Users are able to subscribe to information by “watching” projects and “following” users, resulting in a feed of information on those projects and users of interest. Users also have profiles that can be populated with identifying information and contain their recent activity within the site.

With over 10.6 million repositories<sup>1</sup> hosted as of January 2014, GitHub is currently the largest code hosting site in the world. Its popularity, the integrated social features, and the availability of metadata through an accessible API have made GitHub very attractive for software engineering researchers. Existing research has been both qualitative [4, 7, 16, 17, 19] and quantitative [10, 24, 25, 26]. Qualitative studies have focused on how developers use GitHub's social features to form impressions and draw conclusions on other developers' and projects' activity to assess success, performance, and possible collaboration opportunities. Quantitative studies have aimed to systematically archive GitHub's publicly available data and use that to investigate development practices and network structure in the GitHub environment.

As part of our research on collaboration on GitHub [15], we conducted an exploratory online survey in 2013 to assess the reasons for developers using GitHub and how it supports them in working with others. Through analyzing the survey data we noticed that GitHub repositories were also used for purposes other than strictly software development. Many respondents were using repositories to host personal projects, without any plans to collaborate on their work. This signalled that there might be unseen, significant perils in using GitHub data “as-is” for software engineering research. The variety of repository contents and activity, as well as developers' intentions, can alter research conclusions if care is not taken to first establish that the data fits the research purpose.

The potential risks of misinterpretation in publicly mined data has also been noted with regard to SourceForge mined data [14]. Furthermore, Bird et al. [6] described the promises associated with exploiting the information stored in a decentralized version control system. We, therefore formulated the following research question to address with this study:

**RQ:** What are the promises and perils of mining GitHub for software engineering research?

<sup>1</sup><https://github.com/features>

# Promises - I

Many projects to mine (>20 Millions, 9 Millions of users)



# Promises - II

Full integration of SCM and Issue tracker

# Promises - III

Pull request data, including code reviews

# Promises - IV

Gist data

# Gist

<https://gist.github.com/discover>

Code snippets pasted by developers

Available and searchable on GitHub

# Gist



**mairbek** / logging.java

Created 4 years ago

Logging

1 file

0 forks

0 comments

0 stars

```
1 log.error("SOMETHING REALLY BAD HAPPENED")
2 log.warn("Something bad happened")
3 log.info("Startup")
4 log.debug("Processing request" + request)
```



**phaustin** / write\_history.py

Last active 10 days ago

logging

1 file

0 forks

0 comments

0 stars

```
1 import logging
2 logging.basicConfig()
3
4 my_log=logging.getLogger('my_logger')
5 my_log.setLevel(logging.WARN)
6 my_log.propagate=True
7 my_log.propagate=False
8
9 try:
10     int('A')
```



# Perils - I

Presence of forked or personal repositories



# Perils - II

Projects with very few commits or inactive



# Perils - III

Presence of non-software projects





# Perils - IV

Projects conduct development elsewhere



# Perils - V

Pull requests available for few projects

# Selecting your dataset

# Selection of objects for your studies

- **Huge amount of data available**
- **Intensity (specialized) vs. variation (diversified) sampling**

# Diversity in Software Engineering Research

Meiyappan Nagappan  
Software Analysis and Intelligence Lab  
Queen's University, Kingston, Canada  
mei@cs.queensu.ca

Thomas Zimmermann  
Microsoft Research  
Redmond, WA, USA  
tzimmer@microsoft.com

Christian Bird  
Microsoft Research  
Redmond, WA, USA  
Christian.Bird@microsoft.com

## ABSTRACT

One of the goals of software engineering research is to achieve generality: Are the phenomena found in a few projects reflective of others? Will a technique perform as well on projects other than the projects it is evaluated on? While it is common sense to select a sample that is representative of a population, the importance of diversity is often overlooked, yet as important. In this paper, we combine ideas from representativeness and diversity and introduce a measure called *sample coverage*, defined as the percentage of projects in a population that are similar to the given sample. We introduce algorithms to compute the sample coverage for a given set of projects and to select the projects that increase the coverage the most. We demonstrate our technique on research presented over the span of two years at ICSE and FSE with respect to a population of 20,000 active open source projects monitored by Ohloh.net. Knowing the coverage of a sample enhances our ability to reason about the findings of a study. Furthermore, we propose reporting guidelines for research: in addition to coverage scores, papers should discuss the target population of the research (*universe*) and dimensions that potentially can influence the outcomes of a research (*space*).

## Categories and Subject Descriptors

D.2.6 [Software Engineering]: Metrics

## General Terms

Measurement, Performance, Experimentation

## Keywords

Diversity, Representativeness, Sampling, Coverage

## 1. INTRODUCTION

Over the past twenty years, the discipline of software engineering research has grown in maturity and rigor. Researchers have worked towards maximizing the impact that software engineering research has on practice, for example, by providing techniques and results that are as general (and thus as useful) as possible. However, achieving generality is not easy: Basili et al. [1] remarked that “*general conclusions from empirical studies in software engineering are difficult because any process depends on a potentially large number of relevant context variables*”.

With the availability of OSS projects, the software engineering research community has moved to more extensive validation. As an extreme example, the study of Smalltalk feature usage by Robbes

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ESEC/FSE'13, August 18–26, 2013, Saint Petersburg, Russia  
Copyright 2013 ACM 978-1-4503-2237-9/13/08... \$15.00.

et al. [2] examined 1,000 projects. Another example is the study by Gabel and Su that examined 6,000 projects [3]. But if care isn't taken when selecting which projects to analyze, then increasing the sample size does not actually contribute to the goal of increased generality. *More is not necessarily better.*

As an example, consider a researcher who wants to investigate a hypothesis about say distributed development on a large number of projects in an effort to demonstrate generality. The researcher goes to the json.org website and randomly selects twenty projects, all of them JSON parsers. Because of the narrow range of functionality of the projects in the sample, any findings will not be very representative; we would learn about JSON parsers, but little about other types of software. While this is an extreme and contrived example, it shows the importance of systematically selecting projects for empirical research rather than selecting projects that are convenient. With this paper we provide techniques to (1) assess the quality of a sample, and to (2) identify projects that could be added to further improve the quality of the sample.

Other fields such as medicine and sociology have published and accepted methodological guidelines for subject selection [2] [4]. While it is common sense to select a sample that is *representative* of a population, the importance of *diversity* is often overlooked yet as important [5]. As stated by the Research Governance Framework for Health and Social Care by the Department of Health in the UK:

*“It is particularly important that the body of research evidence available to policy makers reflects the diversity of the population.”* [6]

Similarly the National Institutes of Health in the United States developed guidelines to improve diversity by requiring that certain subpopulations are included in trials [4]. The aim of such guidelines is to ensure that studies are relevant for the entire population and not just the majority group in a population.

Intuitively, the concepts of diversity and representativeness can be defined as follows:

- **Diversity.** A diverse sample contains members of every subgroup in the population and within the sample the subgroups have *roughly equal size*. Let's assume a population of 400 subjects of type X and 100 subjects of type Y. In this case, a perfectly diverse sample would be  $1 \times X$  and  $1 \times Y$ .
- **Representativeness.** In a representative sample the size of each subgroup in the sample is *proportional* to the size of that subgroup in the population. In the example above, a perfectly representative sample would be  $4 \times X$  and  $1 \times Y$ .

Note that based on our definitions diversity (“roughly equal size”) and representativeness (“proportional”) are orthogonal concepts. A highly diverse sample does not guarantee high representativeness and vice versa.

**Much more...**

# Build your own list!

- 100 Things
1. have 6 weeks in Jan D.C.
  2. go to grad school / law school
  3. become a freelance photographer
  4. get married
  5. have children
  6. take my children to Disney world
  7. visit India
  8. visit any country
  9. get on the subway
  10. become a senator
  11. work in the white house
  12. write a cook book
  13. meet Derek zeter
  14. vacation in Hawaii
  15. learn Italian
  16. visit Egypt
  17. go skydiving
  18. learn to skateboard
  19. catch a squirrel
  20. have a pet bird
  21. live in New York
  - 22.
  - 23.
  - 24.
  - 25.

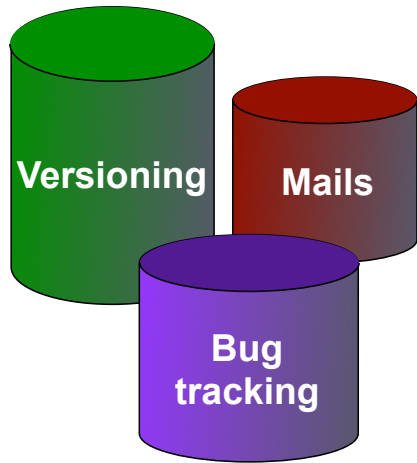




# Key Ingredients of a Mining Software Repository Study



# Some key ingredients...



Quantitative+  
**qualitative** analysis



Interviewing/  
surveying developers



Appropriate statistics

# Qualitative vs Quantitative Analyses

**Quantitative:** to get numerical relations among variables

**Qualitative:** to interpret a phenomenon just observing it in its context

# Qualitative Analysis of repository data

- Dig into data, code, emails, bug reports
- Not necessarily statistical sampling, but rather **purposeful sampling**
- Use of techniques inspired to grounded theory, e.g. open coding

# Examples - Commit

- Understand the purpose of a commit
- Look at the message and/or actual change

“...I had previously fixed the identical bug in `oper_select_candidate`, but didn't realize that the same error was repeated over here...”

# Examples - Discussion

- Understand what developers discuss over mailing lists
- We found that certain API cause bugs, when and how are developers facing the problem?

# Characteristics of a good categorisation

- Categories should be internally consistent
- Reasonably inclusive
- Multiple people should be involved
- Categories should be credible for developers

# Contacting project members

- Don't be surprised if you get about 10% of the responses
- Don't expect them to perform long and tiresome tasks



# More about qualitative analysis

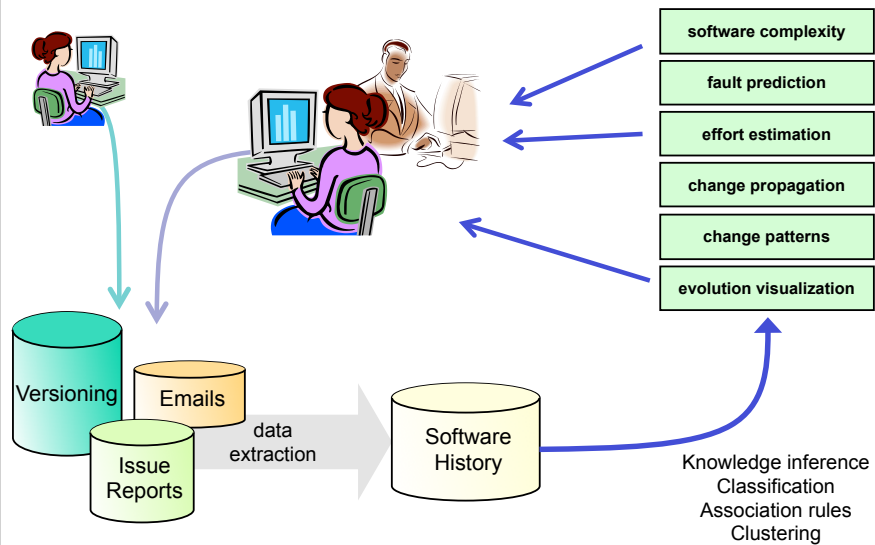
Massimiliano Di Penta, Damian Tamburri: **Combining Quantitative and Qualitative Methods in Empirical Software Engineering Tutorial @ESEC-FSE 2015**

<http://tinyurl.com/eseqfseqq>



# Summary

# Mining Software Repositories



# Identifying Mentors in software projects

Enough **expertise** about the topic of interest for the newcomer...



Demonstrated **ability to help** other people...



Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, Sebastiano Panichella: Who is going to mentor newcomers in open source projects? SIGSOFT FSE 2012: 44

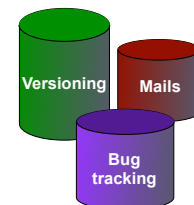


## Threat: Incorrect Classification

- Bug tracking systems contain various kinds of changes
- Classified using inadequate fields, or just poorly and subjectively classified

Status:	Resolution:	Severity:	Priority:
UNCONFIRMED	---	blocker	P1
NEW	FIXED	critical	P2
ASSIGNED	INVALID	major	P3
REOPENED	WONTFIX	normal	P4
RESOLVED	DUPLICATE	minor	P5
VERIFIED	WORKSFORME	trivial	
CLOSED	MOVED	<u>enhancement</u>	

## Some key ingredients...



Quantitative+ qualitative analysis



Interviewing/ surveying developers



Appropriate statistics