Mining Software Repositories for Intelligent Software Maintenance

Thomas Weibel <weibelt@ethz.ch>

SETLabs, Infosys Tech. Ltd., Bangalore

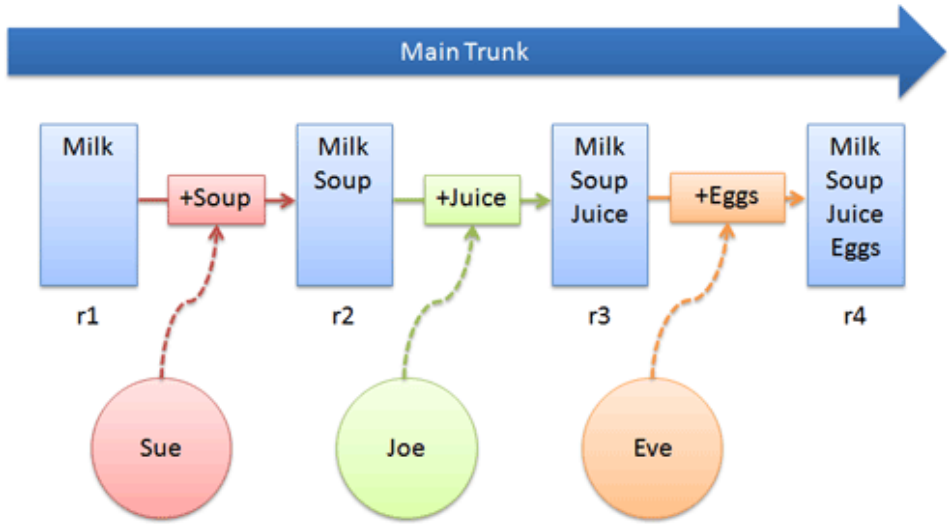
December 1, 2009

Executive Summary

- Change management with version control systems
- Improving software maintenance through software repository mining
- Framework for preventive maintenance
- Novelty: Metrics for localization
- Study of Open Source projects

Outline

Version Control



- Management of changes to computer files in a repository
- Changes identified by a number or letter code (“revision”)
- Each revision associated with timestamp and person making the change
- Version control systems: CVS, Subversion, Git, ...

Working Copy, Commits and Change Sets

- Working copy: Local copy of files from a repository
- Commit: Writing changes to the working copy into the repository
- Change set: Set of changes made in a single commit

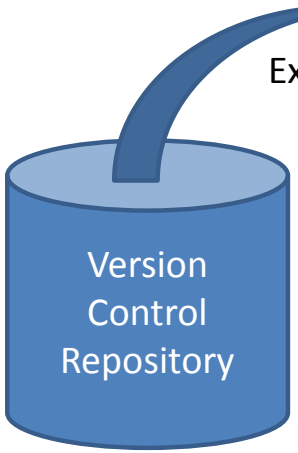
```
commit 3d2d827f5ca5e32816194119d5c980c7e04474a6
Author: Michael S. Tsirkin <mst@redhat.com>
Date:   Mon Sep 21 17:03:51 2009 -0700

    mm: move use_mm/unuse_mm from aio.c to mm/

M       fs/aio.c
A       include/linux/mmu_context.h
M       mm/Makefile
A       mm/mmu_context.c
```

Outline

Software Repository Mining

- Version control systems contain large amounts of historical information: "*Who* changed *what*, *why* and *when*."
- Learn from the past to shape the future
- Automated extraction, collection, and abstraction of information from software development data

Populating Version History Database



Extract version history

Revision	Date	Author	Message
rev1	23.11.2009	Zaphod	Added f1, f2
rev2	25.11.2009	Arthur	Updated f1
rev3	30.11.2009	Zaphod	Deleted f1
...	...	...	...

Revision	Action type	File
rev1	Add	f1
rev1	Add	f2
rev2	Modify	f1
rev3	Delete	f1
...	...	...

Write into database

Frequent Item Set Mining

- Popular method for market basket analysis
- Identify sets of products frequently bought together: Beer and diapers
- Framework applies frequent item set mining to the version history of software repositories
- Identify which code files have been frequently changed together

Source: `http://research.nii.ac.jp/~uno`

Frequent Item Set Mining: Transactions

- Transactions: Change sets
- Example:

  { fs/aio.c, include/linux/mmu_context.h,
  mm/Makefile, mm/mmu_context.c }

```
commit 3d2d827f5ca5e32816194119d5c980c7e04474a6

M       fs/aio.c
A       include/linux/mmu_context.h
M       mm/Makefile
A       mm/mmu_context.c
```

Frequent Item Set Mining: Definitions

- Transaction database contains all change sets
- Members of transactions are items
- Item set is a subset of possible items
- Support of an item set i: $\sup(i) :=$ number of transactions t that contain i

Frequent Item Set Mining: Association Rules

Source: `amazon.com`

- If a customer buys bread and wine, then she will probably also buy cheese
- Problem decomposed into two subproblems:
  - Finding frequent item sets with minimum support
  - Generate association rules with minimal confidence
- Confidence for association rule $R : X \rightarrow Y$:
  $\text{conf}(R) = \text{conf}(X \rightarrow Y) = \text{sup}(X \cup Y)/\text{sup}(X)$

Frequent Item Set Mining: Example

Transaction IDs	Transactions (Files)
1	$\{1, 2, 3, 4\}$
2	$\{2, 3, 4\}$
3	$\{2, 3\}$
4	$\{1, 2, 4\}$
5	$\{1, 2, 3, 4\}$
6	$\{2, 4\}$

Maintenance Challenges

- Predicting changes
  - Incomplete changes
- Traceability links
  - "Cross-language" changes
- Predicting faults
- Understanding software evolution
  - Measure change localization

Predicting Changes

A. Ying, G. Murphy et al., *Predicting Source Code Changes by Mining Change History*

- Determines change patterns from change history of the code base
- Uses association rule mining for identifying implicit dependencies
- Change patterns can be used to recommend potentially relevant source code to a developer performing a modification task

Predicting Changes: Incomplete Change

- Comments in modification task report of Mozilla:
  - 2002-06-12 14:14: Patch to gtk/nsFontMetricsGTK.cpp, limiting the size of fonts to twice the display height.
  - 2002-06-12 14:37: Patch misses the Xlib version.
  - A patch was later submitted with the correct changes in the X-windows font handling code in the file xlib/nsFontMetricsXlib.cpp
- gtk/nsFontMetricsGTK.cpp does not reference xlib/nsFontMetricsXlib.cpp $\rightarrow$ used in different configurations
- Files were changed 41 times together $\rightarrow$ change pattern
- Changing the gtk/nsFontMetricsGTK.cpp could trigger a recommendation for xlib/nsFontMetricsXlib.cpp

Traceability Links

H. Kadgi et al., *Mining Software Repositories for Traceability Link*

- If files of difference types are co-changed with a high frequency over multiple versions $\rightarrow$ potential traceability link
- Traceability links derived from the actual changes to files by mining software repositories
- Uses sequential-pattern mining to identify and analyze sets of files that are committed together
- Sequential-pattern mining produces ordered lists of co-changing files
- Ordering information can be used to infer directionality

Traceability Links: Example

- Mining change sets in the Wine repository
- Changes are tested:
  - `./dlls/gdiplus/graphicspath.c ->`
    `./dlls/gdiplus/tests/graphicspath.c`
  - `./dlls/inetmib1/main.c ->`
    `./dlls/inetmib1/tests/main.c`
- Cross language changes:
  - `./dlls/rpcrt4/tests/server.c ->`
    `./dlls/rpcrt4/tests/server.idl`
  - `./dlls/dxgi/dxgi_private.h ->`
    `./include/wine/winedxgi.idl`

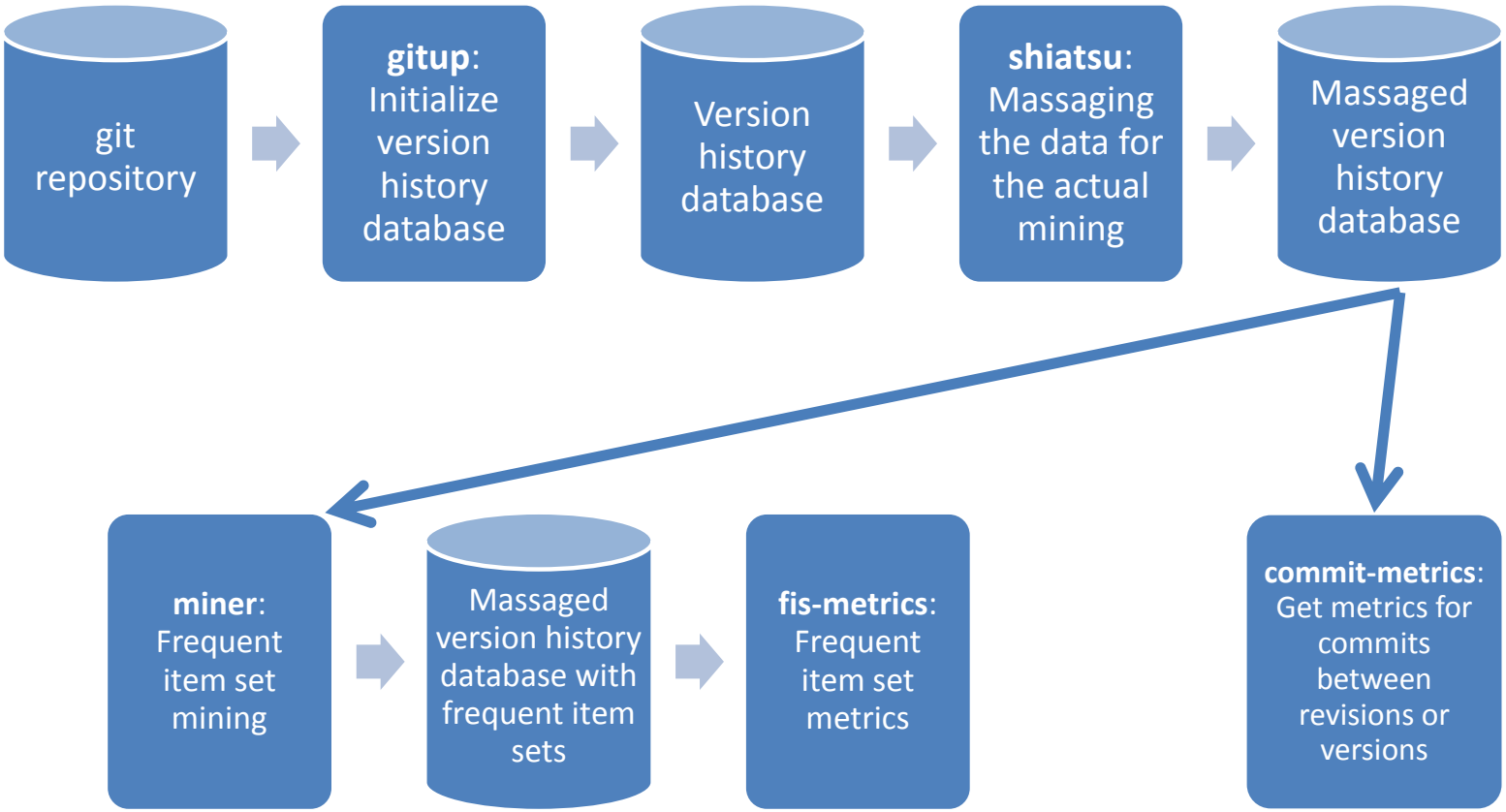
Predicting Faults

S. Kim, T. Zimmermann et al., *Predicting Faults from Cached History*

- Assumption: faults do not occur in isolation, but rather in bursts of several related faults
- Identifying bug fixes by mining commit messages: Searching for keywords such as "Fixed" or "Bug" and references to bug reports like "42"
- Cache locations that are likely to have faults
- By consulting the cache at the moment a fault is fixed, a developer can detect likely fault-prone locations

Outline

Architecture

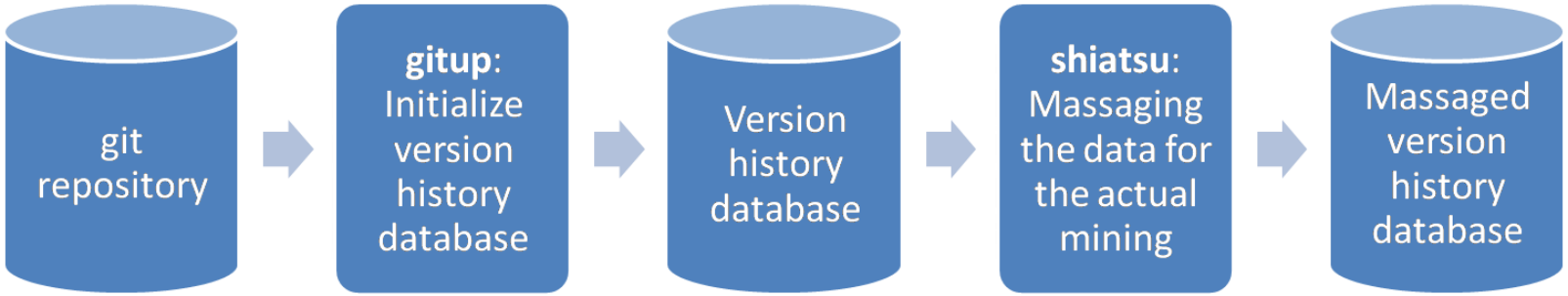


Git

- Free distributed version control system
- Initially designed and developed for Linux kernel development
- Every working directory is a full-fledged repository:
  - Complete history
  - Full revision tracking capabilities
  - Not dependent on network access or a central server
- Easily convert repositories of other version control systems like Subversion into Git repositories
- Only need to write mining and analysis tools for one format rather than many

Populating the Version History Database

- Gitup generates logfile and initializes versions history database
- Shiatsu massages the data to be used by the metrics applications
  - Set modularization according to specified directory depth
  - Remove deleted files
  - Set number of modifications
  - Heuristics for file moves
- Massaged version history database is used to generate frequent item sets and calculate metrics



Preventing Maintenance

Our framework can help with solving all mentioned maintenance challenges:

- Predicting changes
  - Incomplete changes
- Traceability links
  - "Cross-language" changes
- Predicting faults
- Understanding software evolution
  - Measure change localization

Outline

Change Localization

- A change is well localized, if it touches only one or very few modules
- A change is not well localized, if it touches many modules
- Apply change localization for frequent item sets

Hypothesis

Changes in frequent item sets in well modularized software systems are localized

Change Localization: Example

- Well localized: Touches only one module

  `dlls/ntdll/signal_i386.c`
  `dlls/ntdll/thread.c`

- Badly localized: Touches four modules out of five possible

  `if1632/thunk.c`
  `include/process.h`
  `loader/task.c`
  `scheduler/process.c`
  `scheduler/thread.c`

- Not localized at all: Touches all possible modules

  `files/dos_fs.c`
  `scheduler/syslevel.c`
  `tools/winapi-check`

Change Localization Metrics

- Value between 0 and 1
- 0: Not localized at all
- 1: Fully localized

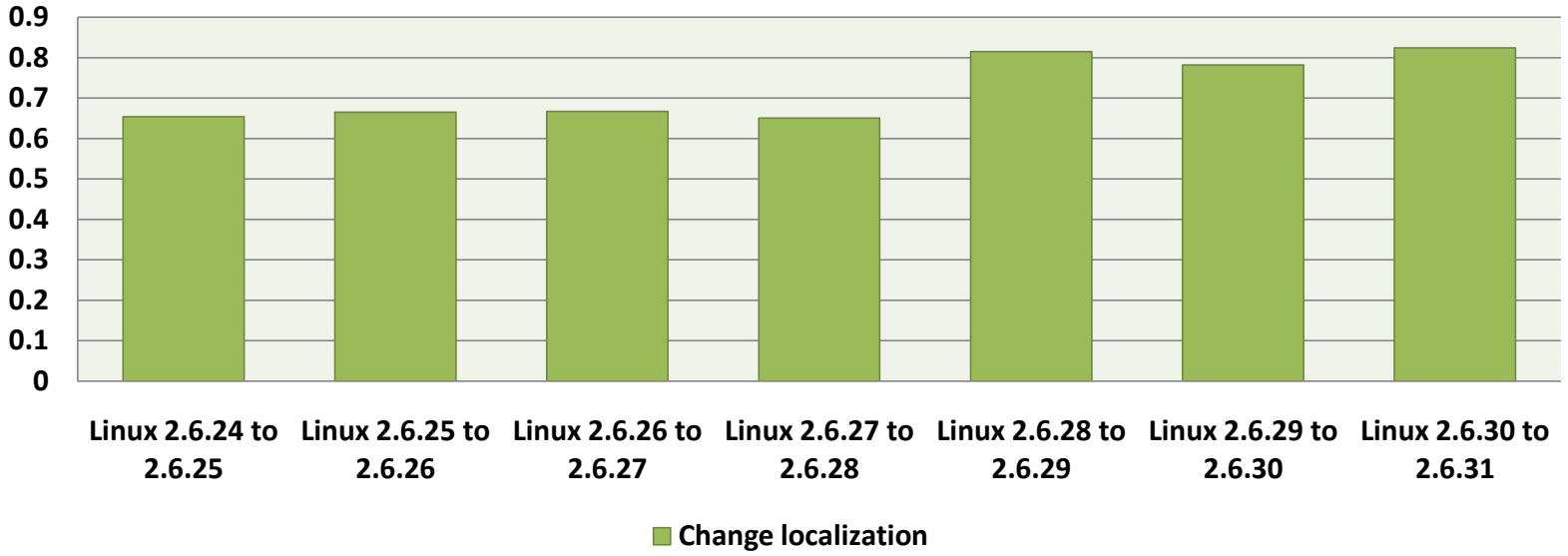
$$\frac{\sum_{i=\mathrm{FIS}_1}^{\mathrm{FIS}_n} 1 - \left(\mathrm{if}\left(i.\mathrm{modules\_touched} = 1, 0, \frac{i.\mathrm{modules\_touched}}{i.\mathrm{files\_touched}}\right)\right)}{n}$$

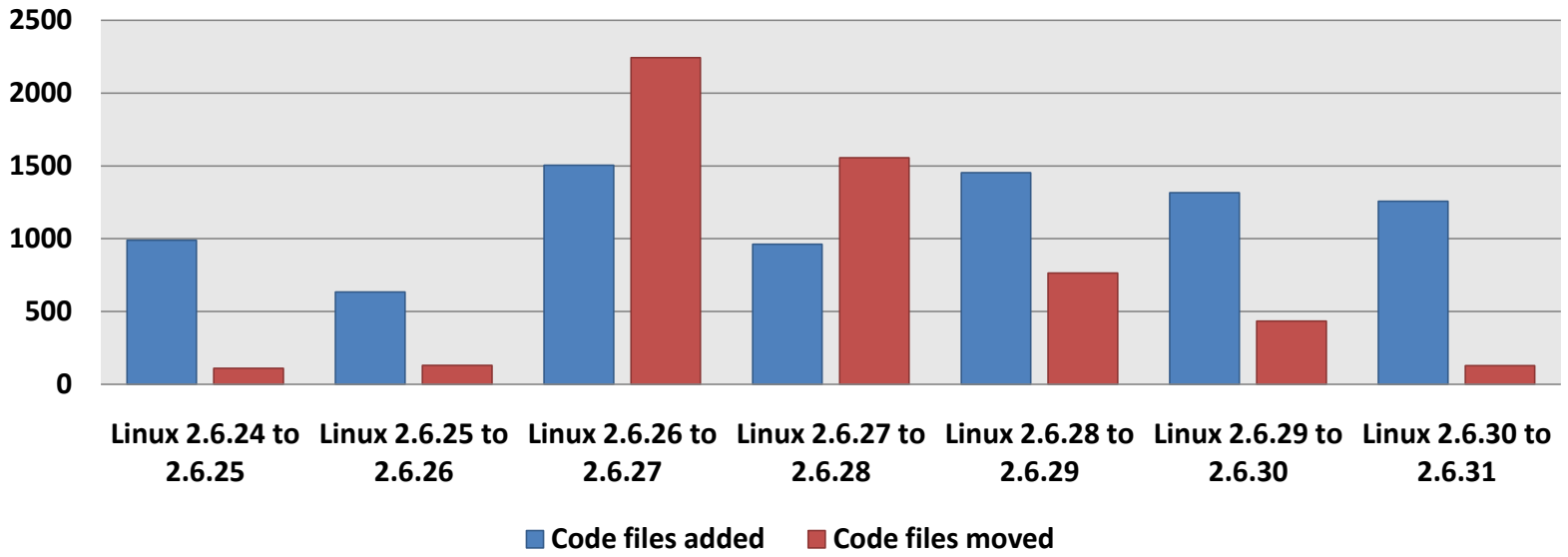
Outline

Linux 2.6

- Unix-like operating system kernel
- Repository checked out on November 19, 2009
- 25,277 code files
- 168,800 commits
- Frequent item set mining:
  - Minimum number of commits (modifications) for code files: 4
  - Minimum support: 4
  - Maximum size of commits (number of code files): 50

Linux 2.6: Frequent Item Set Metrics

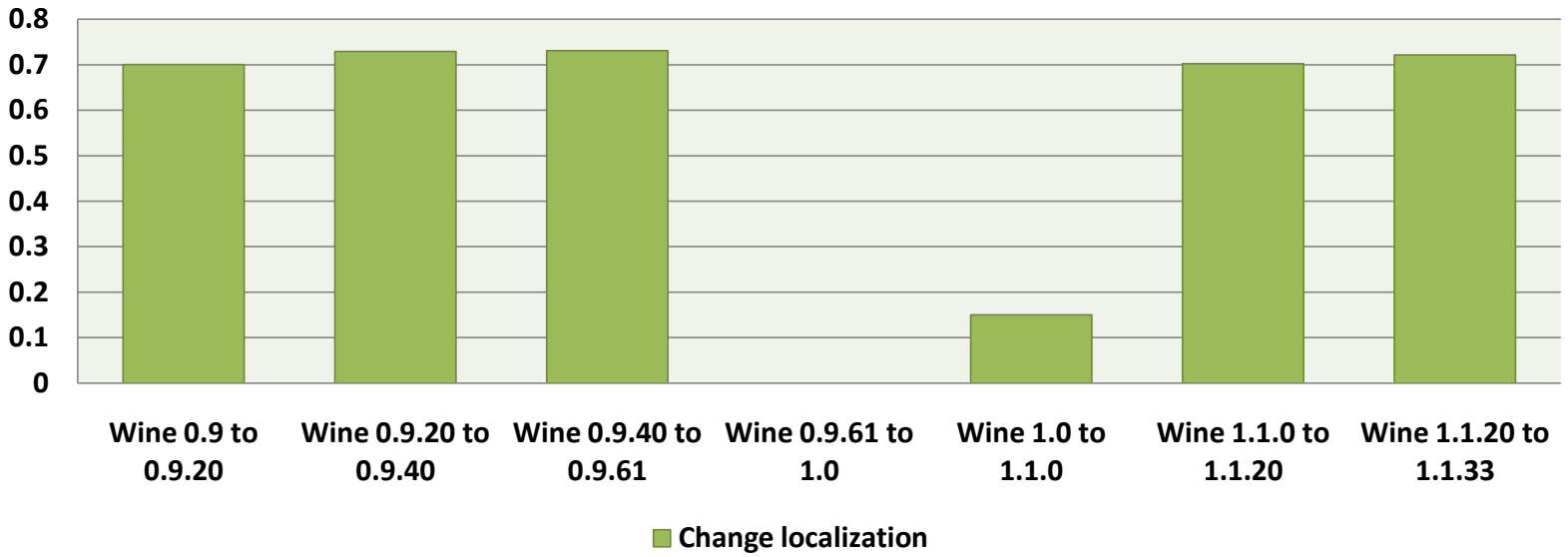


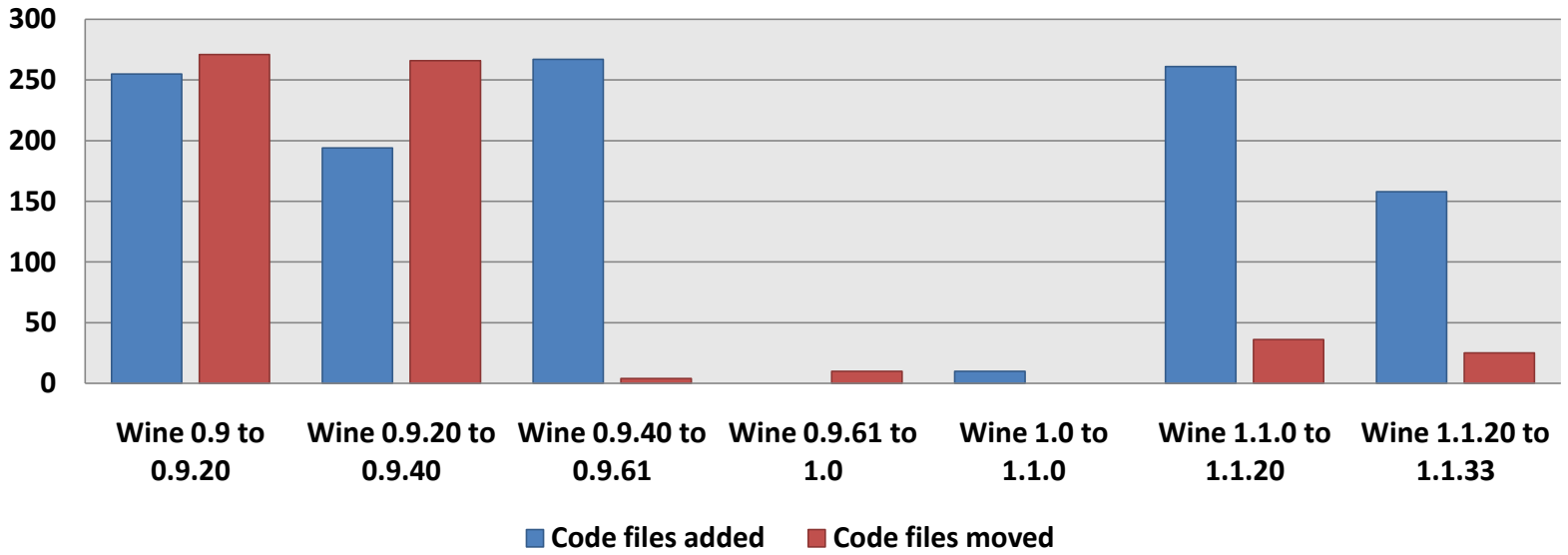


Wine

- Allows execution of Microsoft Windows programs on Unix-like operating systems
- Repository checked out on November 20, 2009
- 3,479 code files
- 63,864 commits
- Frequent item set mining:
  - Minimum number of commits (modifications) for code files: 4
  - Minimum support: 4
  - Maximum size of commits (number of code files): 50

Wine: Frequent Item Set Metrics





Insights

- Code file moves increase localization
- Adding of many code files decrease localization
- Adding of code files can clear effect of moves on localization
- Stable versions contain mostly bug fixes
  $\Rightarrow$ low localization, only few moves and adds
- Unstable versions contain mostly new features
  $\Rightarrow$ high localization, many moves and adds

Future Work

- Use framework to mine software repositories of commercial systems
- Compare localization metrics of Open Source and Closed Source systems
- Use the frequent item sets extracted to come up with a better modularization
- Publish research in the form of a paper

Summary

- Mining software repositories for intelligent software maintenance
- Applications of frequent item set mining in improving software maintainability
- Framework for preventing software maintenance
- New metrics for change localization
- Localization of frequent item sets of Open Source projects