Mining Software Repositories for Intelligent Software Maintenance

Thomas Weibel <weibelt@ethz.ch>

SETLabs, Infosys Tech. Ltd., Bangalore

December 1, 2009

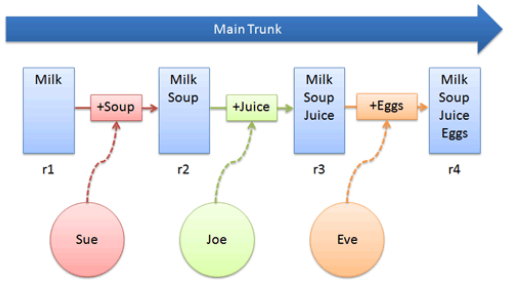
Executive Summary

- 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

- First I will introduce change management with version control systems.
- Then I will show you how you can improve software maintenance through software repository mining
- After this, I'm going present you a framework for preventive maintenance I developed.
- As a novelty, we devised a metrics for localization.
- And did a 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, ...

- Version control is the management of changes to documents, programs, and other information stored as computer files. It is most commonly used in software development, where a team of people may be changing the same files.
- Changes are usually identified by a number or letter code, termed the "revision". For example, an initial set of files is "revision 1". When the first change is made, the resulting set is "revision 2", and so on.
- Each revision is associated with a timestamp and the person making the change.
- Examples of version control systems are CVS, Subversion or 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
```


- The working copy is the local copy of files from a repository, at a specific time or revision.
- A commit occurs when a copy of the changes made to the working copy is written or merged into the repository.
- On version control systems with atomic multi-change commits, a change set identifies the set of changes made in a single commit.
- Explain commit!

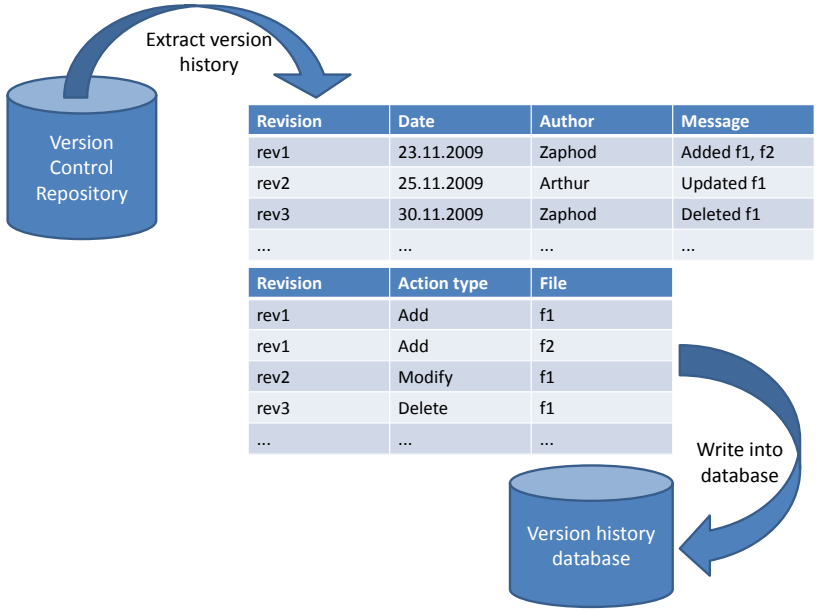
Outline

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

- Version control systems contain large amounts of historical information that can give deep insight into the evolution of a software project: "*Who* changed *what*, *why* and *when*."
- When mining software archives, we want to learn from the past to shape the future.
- The mining of software repositories is concerned with the automated extraction, collection, and abstraction of information from available software development data.

Populating Version History Database



- First we extract version history from the version control repository.
- This gives us information about who changes what, why and when.
- This information is written into a version history database for easier mining.

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

- Finding frequent item sets in a set of transactions is a popular method for so-called market basket analysis. It aims at finding regularities in the shopping behaviour of customers of supermarkets, mail-order companies, online-shops and so forth.
- In particular, it tries to identify sets of products that are frequently bought together:
- A famous example was the discovery that people who buy diapers also frequently buy beer. Those are probably exhausted fathers of small children.
- Therefore nowadays one finds frequently beer close to diapers - and of course also chips close to beer - in supermarkets.
- In our framework, we apply frequent item set mining to the version history of software repositories.
- The goal is to find out, which code files have been frequently changed together.

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
```


- A change set identifies the set of changes made in a single commit to the version control system. We take change sets as our transactions in frequent item set mining.
- In the example below, the transaction consist of four files.

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

- The transaction database contains all transactions of the version control system.
- Members of transactions are called items.
- And and item set is a subset of possible items
- The support of an item set i is the number of transactions t that support - this means 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 $R : X \to Y$: $\text{conf}(R) = \text{conf}(X \to Y) = \text{sup}(X \cup Y)/\text{sup}(X)$

- Association rule mining gives options to finish the sentence "Customers who bought this item also bought..."
- For example "If a customer buys bread and wine, then she will probably also buy cheese".
- The problem is decomposed into two subproblems:
  - Finding frequent item sets with minimum support
  - Generate association rules with minimal confidence
- Instead of going into details about confidences, I will give you an example for frequent item set mining and association rules.

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	$\{2, 4\}$

Show example on the whiteboard

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

- Determines change patterns from the change history of the code base. Change patterns are sets of files that were changed together frequently in the past. This means, they are frequent item sets.
- The method 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

- The Mozilla Web browser includes a Web content layout engine responsible for Web pages. There was a bug that caused the consumption of all available memory when a Web page with very large fonts was displayed. The comments in the modification task report include the following:
  - 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
- The source code in gtk/nsFontMetricsGTK.cpp does not reference the code in xlib/nsFontMetricsXlib.cpp because the code in the gtk version and the code in xlib version are used in different configurations of Mozilla.
- However, an analysis of the change history for Mozilla indicates that these two files were changed 41 times together.
- When we applied our approach to Mozilla, we extracted a change pattern. Changing the gtk/nsFontMetricsGTK.cpp could trigger a recommendation for the other file.

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 → 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

- If files of different types are co-changed with a high frequency over multiple versions, then such files potentially have a traceability link between them
- Traceability links are 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. Another approach would be frequent item set mining.
- Sequential-pattern mining produces ordered lists of co-changing files whereas frequent item set mining produces sets.
- The ordering information can be used to infer directionality of traceability links

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`

- Mining change sets in the Wine repository
- Changes are tested
- Identifies cross language changes

Predicting Faults

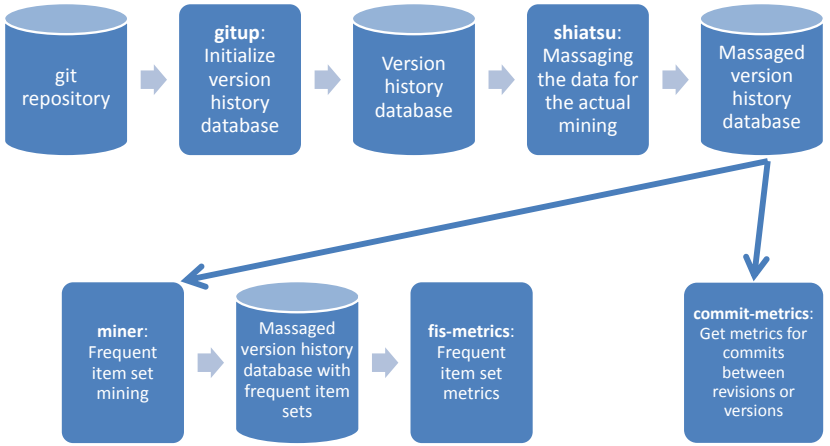
S. Kim, T. Zimmermann et al., *Predicting Faults from Cached History*

- 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

- The basic assumption is that faults do not occur in isolation, but rather in bursts of several related faults.
- Therefore, one can cache locations that are likely to have faults: starting from the location of a known (fixed) fault, cache the location itself, any locations changed together with the fault, recently added locations, and recently changed locations.
- By consulting the cache at the moment a fault is fixed, a developer can detect likely fault-prone locations. This is useful for prioritizing verification and validation resources on the most fault prone files or entities.
- Identifying bug fixes by mining change log messages: Searching for keywords such as "Fixed" or "Bug" and references to bug reports like "42"

Outline

Architecture



- We built a framework for software repository mining
- Explain framework!

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

- We chose Git as the base of our framework.
- Git is a free distributed version control system.
- It was initially designed and developed by Linus Torvalds for Linux kernel development.
- Every Git working directory is a full-fledged repository with complete history and full revision tracking capabilities, not dependent on network access or a central server.
- Repositories of other version control systems like CVS or Subversion can easily be converted into Git repositories.
- This means we 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

- Gitup generates logfile and initializes versions history database.
- Shiatsu massages the data:
  - It sets modularization according to specified directory depth.
  - Shiatsu removes deleted files from the database and set the number of modifications.
  - It also uses heuristics for file moves.
- The massaged version history database is then 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

Our framework can help with solving all mentioned maintenance challenges

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

- 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
- Our hypothesis is that changes in frequent item sets in well modularized software systems are localized
- This means that frequent item sets have a high localization

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`

Do the calculation after the next slide!

- $1 - 0 = 1$
- $1 - \frac{4}{5} = \frac{1}{5}$
- $1 - \frac{3}{3} = 1 - 1 = 0$

Change Localization Metrics

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

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

Go back to the previous slide to show localization!

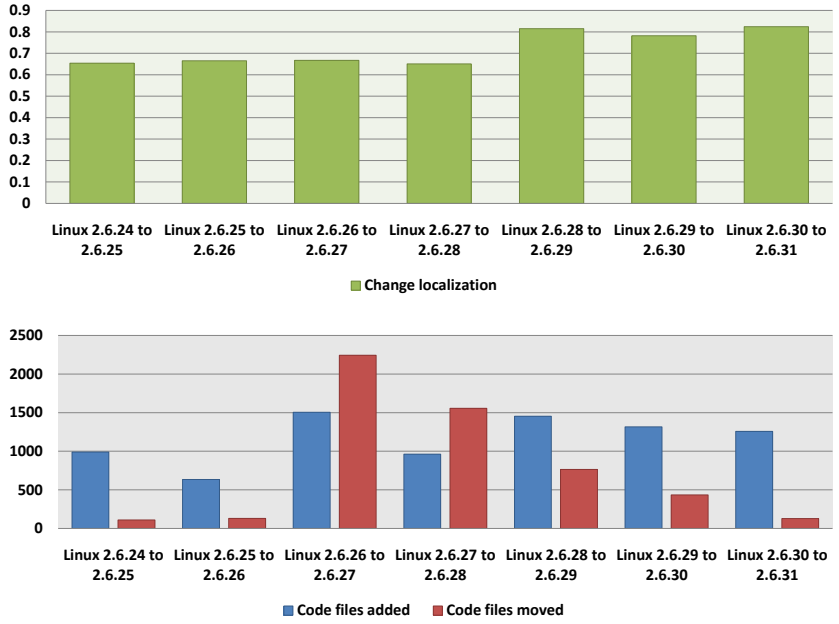
Outline

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 is a Unix-like operating system kernel.
- I checked out the repository of the mainline on November 19, 2009
- It has 25,277 code files and consists of 168,800 commits
- Settings used for the frequent item set mining are:
  - Minimum number of commits (modifications) a code file has to have to be added to the transactions file: 4
  - Minimum support: 4
  - Maximum size of commits (number of code files): 50

Linux 2.6: Frequent Item Set Metrics



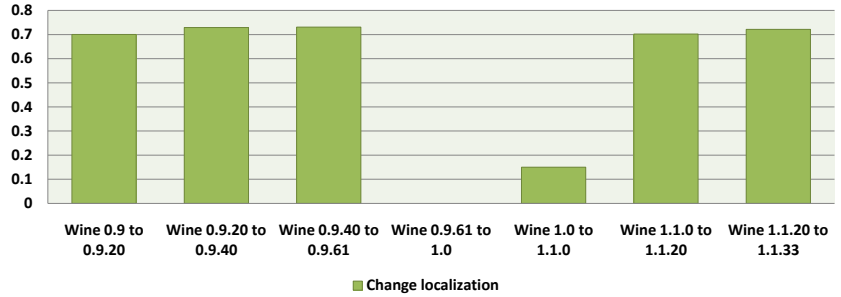
Explain how data was gathered: Log between tags (versions) in the repository.

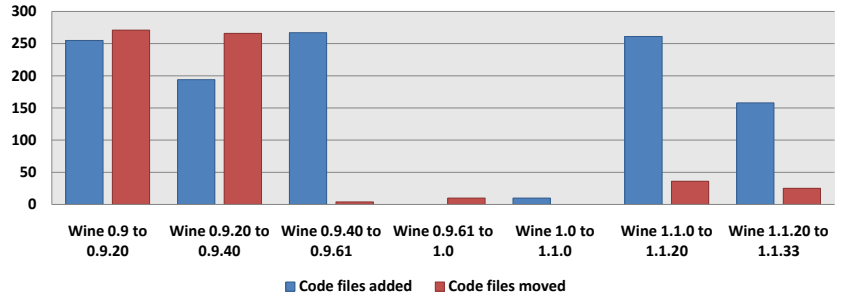
Wine

- 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 allows execution of Microsoft Windows programs on Unix-like operating systems
- I checked out the repository on November 20, 2009
- It has 3,479 code files and consists of 63,864 commits
- We use the same settings for the frequent item set mining as with Linux 2.6

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

- We got the following insights by examining Open Source software repositories:
- 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
  this means they have a low localization and only few moves and adds
- Unstable versions contain mostly new features
  this means they have a high localization and many moves and adds

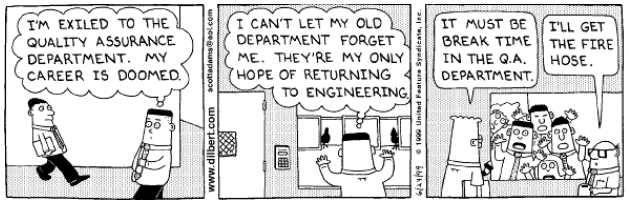
Future Work

- 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

- We want to use our framework to mine software repositories of commercial systems.
- This would allow us to compare localization metrics of Open Source and Closed Source systems.
- Now after we introduced a way to measure change localization, we want to use the frequent item sets extracted from the version history to come up with a better modularization.
- Another goal is to publish the result of our 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



- I presented mining software repositories for intelligent software maintenance.
- And have shown applications of frequent item set mining in improving software maintainability.
- I implemented framework for preventing software maintenance.
- And we Introduced new metrics for change localization.
- We used our framework and metrics to measure and analyze localization of frequent item sets of Open Source projects.