Mining Software Engineering Data

Ahmed E. Hassan
Queen's University
www.cs.queensu.ca/~ahmed
ahmed@cs.queensu.ca

Tao Xie
North Carolina State University
www.csc.ncsu.edu/faculty/xie
xie@csc.ncsu.edu

Some slides are adapted from tutorial slides co-prepared by
Jian Pei from Simon Fraser University, Canada

An up-to-date version of this tutorial is available at
http://ase.csc.ncsu.edu/dmse/dmse-icse08-tutorial.pdf

Ahmed E. Hassan

- Assistant Professor at Queen's University, Canada
- Leads the SAIL research group at Queen's
- Co-chair for Workshop on Mining Software Repositories (MSR) from 2004-2006
- Chair of the steering committee for MSR

Tao Xie

- Assistant Professor at North Carolina State University, USA
- Leads the ASE research group at NCSU
- Co-presented tutorials on Mining Software Engineering Data at KDD 2006, ICSE 2007, & ICDM 2007
- Co-organizer of 2007 Dagstuhl Seminar on Mining Programs and Processes

Acknowledgments

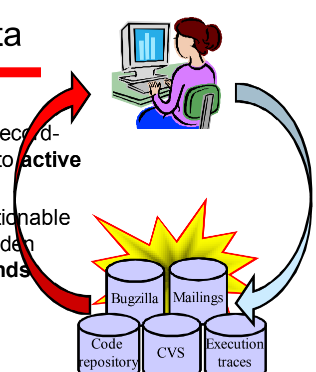
- Jian Pei, SFU
- Thomas Zimmermann, U. of Calgary
- Peter Rigby, U. of Victoria
- Sunghun Kim, MIT
- John Anvik, U. of Victoria

Tutorial Goals

- **Learn about:**
  - Recent and notable research and researchers in mining SE data
  - Data mining and data processing techniques and how to apply them to SE data
  - Risks in using SE data due to e.g., noise, project culture
- **By end of tutorial, you should be able:**
  - Retrieve SE data
  - Prepare SE data for mining
  - Mine interesting information from SE data

Mining SE Data

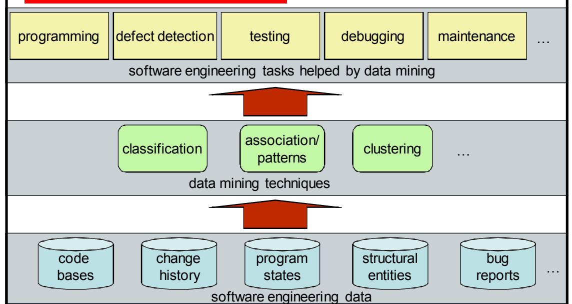
- **MAIN GOAL**
  - Transform static record-keeping SE data to **active** data
  - Make SE data actionable by uncovering hidden **patterns** and **trends**



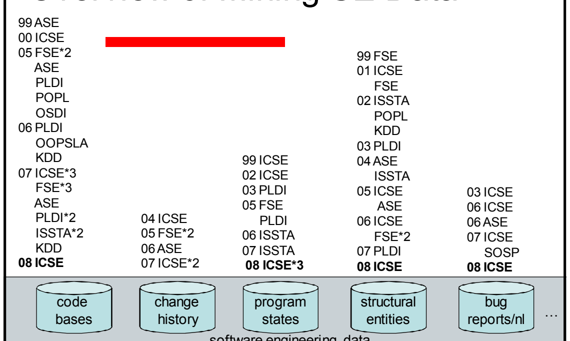
Mining SE Data

- SE data can be used to:
  - Gain empirically-based understanding of software development
  - Predict, plan, and understand various aspects of a project
  - Support future development and project management activities

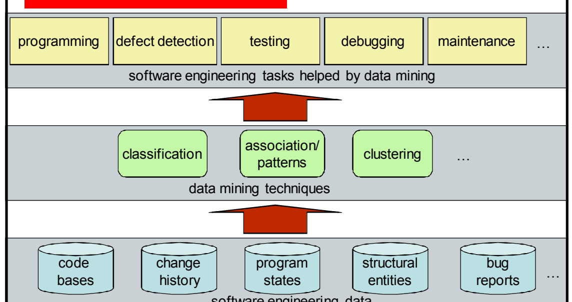
Overview of Mining SE Data



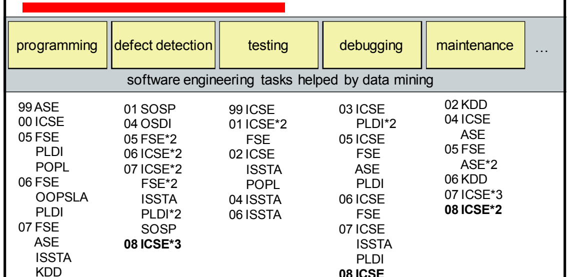
Overview of Mining SE Data



Overview of Mining SE Data



Overview of Mining SE Data



Tutorial Outline

- **Part I:** What can you learn from SE data?
  - A sample of notable recent findings for different SE data types
- **Part II:** How can you mine SE data?
  - Overview of data mining techniques
  - Overview of SE data processing tools and techniques

Types of SE Data

- **Historical data**
  - Version or source control: cvs, subversion, perforce
  - Bug systems: bugzilla, GNATS, JIRA
  - Mailing lists: mbox
- **Multi-run and multi-site data**
  - Execution traces
  - Deployment logs
- **Source code data**
  - Source code repositories: sourceforge.net, google code

Historical Data

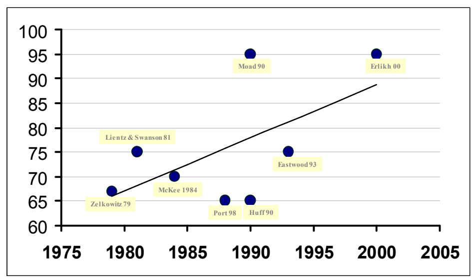
“History is a guide to navigation in perilous times. History is who we are and why we are the way we are.”

- David C. McCullough

Historical Data

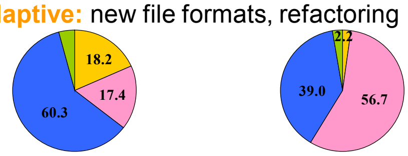
- Track the evolution of a software project:
  - *source control systems* store changes to the code
  - *defect tracking systems* follow the resolution of defects
  - *archived project communications* record rationale for decisions throughout the life of a project
- Used primarily for record-keeping activities:
  - checking the status of a bug
  - retrieving old code

Percentage of Project Costs Devoted to Maintenance



Survey of Software Maintenance Activities

- **Perfective:** add new functionality
- **Corrective:** fix faults
- **Adaptive:** new file formats, refactoring



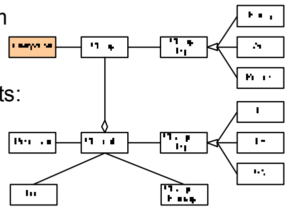
Lientz, Swanson, Tomhkins [1978]
Nosek, Palvia [1990]
MIS Survey

Schach, Jin, Yu, Heller, Offutt [2003]
Mining ChangeLogs
(Linux, GCC, RTP)

Source Control Repositories

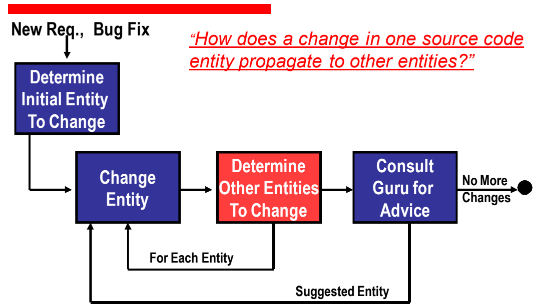
Source Control Repositories

- A source control system tracks changes to ChangeUnits
- Example of ChangeUnits:
  - File (most common)
  - Function
  - Dependency (e.g., Call)
- Each ChangeUnit:
  - Records the developer, change time, change message, co-changing Units



Change Propagation

"How does a change in one source code entity propagate to other entities?"



Measuring Change Propagation

$$\text{Precision} = \frac{\text{predicted entities which changed}}{\text{predicted entities}}$$

$$\text{Recall} = \frac{\text{predicted entities which changed}}{\text{changed entities}}$$

- We want:
  - High Precision to avoid wasting time
  - High Recall to avoid bugs

Guiding Change Propagation

- Mine association rules from change history
- Use rules to help propagate changes:
  - Recall as high as 44%
  - Precision around 30%
- High precision and recall reached in < 1mth
- Prediction accuracy improves prior to a release (i.e., during maintenance phase)

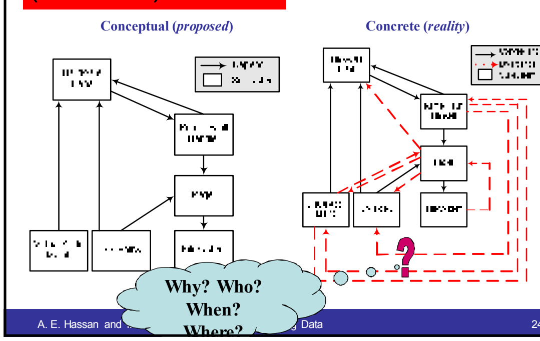
[Zimmermann et al. 05]

Code Sticky Notes

- Traditional dependency graphs and program understanding models usually do not use historical information
- Static dependencies capture only a static view of a system – not enough detail!
- Development history can help understand the current structure (architecture) of a software system

[Hassan & Holt 04]

Conceptual & Concrete Architecture (NetBSD)



Investigating Unexpected Dependencies Using Historical Code Changes

- Eight unexpected dependencies
- All except two dependencies *existed since day one*:
  - Virtual Address Maintenance → Pager
  - Pager → Hardware Translations

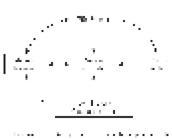
Which?	vm_map_entry_create (in src/sys/vm/Attic/vm_map.c) *depends on* pager_map (in /src/sys/uvm/uvm_pager.c)
Who?	cgd
When?	1993/04/09 15:54:59 Revision 1.2 of src/sys/vm/Attic/vm_map.c
Why?	from sean eric fagan: it seems to keep the vm system from deadlocking the system when it runs out of swap + physical memory. prevents the system from giving the last page(s) to anything but the referenced "processes" (especially important is the pager process, which should never have to wait for a free page).

Studying Conway's Law

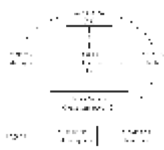
- Conway's Law:
  "*The structure of a software system is a direct reflection of the structure of the development team*"

[Bowman et al. 99]

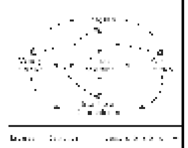
Linux: Conceptual, Ownership, Concrete



Conceptual Architecture



Ownership Architecture



Concrete Architecture

Source Control and Bug Repositories

Predicting Bugs

- Studies have shown that most complexity metrics correlate well with LOC!
  - Graves et al. 2000 on commercial systems
  - Herraiz et al. 2007 on open source systems
- Noteworthy findings:
  - Previous bugs are good predictors of future bugs
  - The more a file changes, the more likely it will have bugs in it
  - Recent changes affect more the bug potential of a file over older changes (*weighted time damp models*)
  - Number of developers is of little help in predicting bugs
  - Hard to generalize bug predictors across projects unless in similar domains [Nagappan, Ball et al. 2006]

Using Imports in Eclipse to Predict Bugs

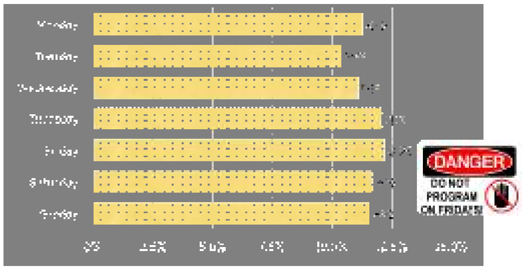
71% of files that import compiler packages, had to be fixed later on.

```
import org.eclipse.jdt.internal.compiler.lookup.*;
import org.eclipse.jdt.internal.compiler.*;
import org.eclipse.jdt.internal.compiler.ast.*;
import org.eclipse.jdt.internal.compiler.util.*;
...
import org.eclipse.pde.core.*;
import org.eclipse.jface.wizard.*;
import org.eclipse.ui.*;
```

14% of all files that import ui packages, had to be fixed later on.

[Schröter et al. 06]

Don't program on Fridays ;-)

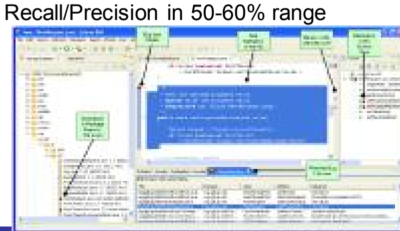


Percentage of bug-introducing changes for eclipse

[Zimmermann et al. 05]

Classifying Changes as Buggy or Clean

- Given a change can we warn a developer that there is a bug in it?
  - Recall/Precision in 50-60% range



[Sung et al. 06]

Project Communication – Mailing lists

Project Communication (Mailinglists)

- Most open source projects communicate through mailing lists or IRC channels
- Rich source of information about the inner workings of large projects
- Discussions cover topics such as future plans, design decisions, project policies, code or patch reviews
- Social network analysis could be performed on discussion threads

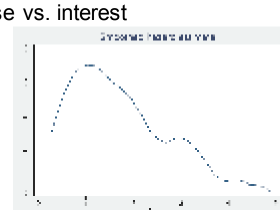
Social Network Analysis

- Mailing list activity:
  - strongly correlates with code change activity
  - moderately correlates with document change activity
- Social network measures (in-degree, out-degree, betweenness) indicate that committers play a more significant role in the mailing list community than non-committers

[Bird et al. 06]

Immigration Rate of Developers

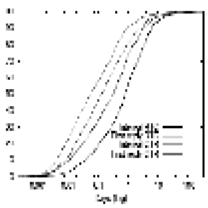
- When will a developer be invited to join a project?
  - Expertise vs. interest



[Bird et al. 07]

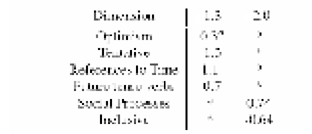
The Patch Review Process

- Two review styles
  - RTC: Review-then-commit
  - CTR: Commit-then-review
- 80% patches reviewed within 3.5 days and 50% reviewed in <19 hrs



[Rigby et al. 06]

Measure a team's morale around release time?



Dimension	1.3	2.0
Optimism	[illegible]	*
Tentative	1.3	*
References to Time	1.1	*
Future tense verbs	0.7	*
Social Processes	*	[illegible]
Inclusive	*	-0.64

Table 4. Mean differences for Apache 1.3 and 2.0 releases. (* p > 0.05, otherwise p < 0.05)

- Study the content of messages before and after a release
- Use dimensions from a psychometric text analysis tool:
  - After Apache 1.3 release there was a drop in optimism
  - After Apache 2.0 release there was an increase in sociability

[Rigby & Hassan 07]

Program Source Code

Code Entities

Source data	Mined info
Variable names and function names	Software categories [Kawaguchi et al. 04]
Statement seq in a basic block	Copy-paste code [Li et al. 04]
Set of functions, variables, and data types within a C function	Programming rules [Li&Zhou 05]
Sequence of methods within a Java method	API usages [Xie&Pei 05]
API method signatures	API Jungloids [Mandelin et al. 05]

Mining API Usage Patterns

- How should an API be used correctly?
  - An API may serve multiple functionalities
  - Different styles of API usage
- "I know what type of object I need, but I don't know how to write the code to get the object" [Mandelin et al. 05]
  - Can we synthesize jungloid code fragments automatically?
  - Given a simple query describing the desired code in terms of input and output types, return a code segment
- "I know what method call I need, but I don't know how to write code before and after this method call" [Xie&Pei 06]

Relationships btw Code Entities

- Mine framework reuse patterns [Michail 00]
  - Membership relationships
    - A class contains membership functions
  - Reuse relationships
    - Class inheritance/ instantiation
    - Function invocations/overriding
- Mine software plagiarism [Liu et al. 06]
  - Program dependence graphs

[Michail 99/00] http://codeweb.sourceforge.net/ for C++

Program Execution Traces

Method-Entry/Exit States

- Goal: mine specifications (pre/post conditions) or object behavior (object transition diagrams)
- State of an object
  - Values of transitively reachable fields
- Method-entry state
  - Receiver-object state, method argument values
- Method-exit state
  - Receiver-object state, updated method argument values, method return value

[Ernst et al. 02] http://pag.csail.mit.edu/daikon/
[Xie&Notkin 04/05][Dallmeier et al. 06] http://www.st.cs.uni-sb.de/models/

Other Profiled Program States

- Goal: detect or locate bugs
- Values of variables at certain code locations [Hangal&Lam 02]
  - Object/static field read/write
  - Method-call arguments
  - Method returns
- Sampled predicates on values of variables [Liblit et al. 03/05][Liu et al. 05]

[Hangal&Lam 02] http://diduce.sourceforge.net/
[Liblit et al. 03/05] http://www.cs.wisc.edu/cbi/
[Liu et al. 05] http://www.ews.uiuc.edu/~chaoliu/sober.htm

Executed Structural Entities

- Goal: locate bugs
- Executed branches/paths, def-use pairs
- Executed function/method calls
  - Group methods invoked on the same object
- Profiling options
  - Execution hit vs. count
  - Execution order (sequences)

[Dallmeier et al. 05] http://www.st.cs.uni-sb.de/ample/
More related tools: http://www.csc.ncsu.edu/faculty/xie/research.htm#related

Q&A and break

Part I Review

- We presented notable results based on mining SE data such as:
  - Historical data:
    - Source control: predict co-changes
    - Bug databases: predict bug likelihood
    - Mailing lists: gauge team morale around release time
  - Other data:
    - Program source code: mine API usage patterns
    - Program execution traces: mine specs, detect or locate bugs

Data Mining Techniques in SE

Part II: How can you mine SE data?

–Overview of data mining techniques

–Overview of SE data processing tools and techniques

Data Mining Techniques in SE

- Association rules and frequent patterns
- Classification
- Clustering
- Misc.

Frequent Itemsets

- Itemset: a set of items
  - E.g., acm={a, c, m}
- Support of itemsets
  - Sup(acm)=3
- Given min_sup = 3, acm is a frequent pattern
- Frequent pattern mining: find all frequent patterns in a database

Transaction database TDB

TID	Items bought
100	f, a, c, d, g, I, m, p
200	a, b, c, f, l, m, o
300	b, f, h, j, o
400	b, c, k, s, p
500	a, f, c, e, l, p, m, n

Association Rules

- (Time $\in$ {Fri, Sat}) $\wedge$ buy(X, diaper) $\rightarrow$ buy(X, beer)
  - Dads taking care of babies in weekends drink beer
- Itemsets should be frequent
  - It can be applied extensively
- Rules should be confident
  - With strong prediction capability

A Simple Case

- Finding highly correlated method call pairs
- Confidence of pairs helps
  - Conf(<a,b>)=support(<a,b>)/support(<a,a>)
- Check the revisions (fixes to bugs), find the pairs of method calls whose confidences have improved dramatically by frequent added fixes
  - Those are the matching method call pairs that may often be violated by programmers

[Livshits&Zimmermann 05]

Conflicting Patterns

- 999 out of 1000 times `spin_lock` is followed by `spin_unlock`
  - The single time that `spin_unlock` does not follow may likely be an error
- We can detect an error without knowing the correctness rules

[Li&Zhou 05, Livshits&Zimmermann 05, Yang et al. 06]

Detect Copy-Paste Code

- Apply closed sequential pattern mining techniques
- Customizing the techniques
  - A copy-paste segment typically does not have big gaps – use a maximum gap threshold to control
  - Output the instances of patterns (i.e., the copy-pasted code segments) instead of the patterns
  - Use small copy-pasted segments to form larger ones
  - Prune false positives: tiny segments, unmappable segments, overlapping segments, and segments with large gaps

[Li et al. 04]

Find Bugs in Copy-Pasted Segments

- For two copy-pasted segments, are the modifications consistent?
  - Identifier *a* in segment S1 is changed to *b* in segment S2 3 times, but remains unchanged once – *likely a bug*
  - The heuristic may not be correct all the time
- The lower the unchanged rate of an identifier, the more likely there is a bug

[Li et al. 04]

Mining Rules in Traces

- Mine association rules or sequential patterns S → F, where S is a statement and F is the status of program failure
- The higher the confidence, the more likely S is faulty or related to a fault
- Using only one statement at the left side of the rule can be misleading, since a fault may be led by a combination of statements
  - Frequent patterns can be used to improve

[Denmat et al. 05]

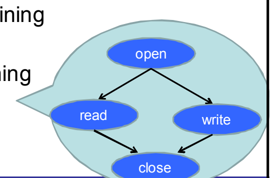
Mining Emerging Patterns in Traces

- A method executed only in failing runs is likely to point to the defect
  - Comparing the coverage of passing and failing program runs helps
- Mining patterns frequent in failing program runs but infrequent in passing program runs
  - Sequential patterns may be used

[Dallmeier et al. 05, Denmat et al. 05]

Types of Frequent Pattern Mining

- Association rules
  - open → close
- Frequent itemset mining
  - {open, close}
- Frequent subsequence mining
  - open → close
- Frequent partial order mining
  Frequent graph mining
  Finite automaton mining



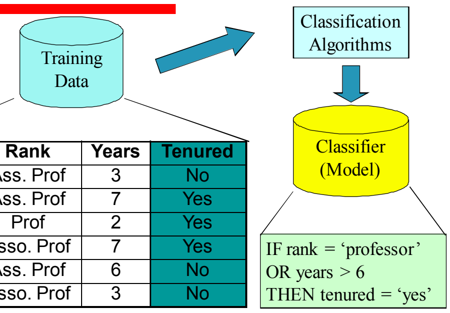
Data Mining Techniques in SE

- Association rules and frequent patterns
- Classification
- Clustering
- Misc.

Classification: A 2-step Process

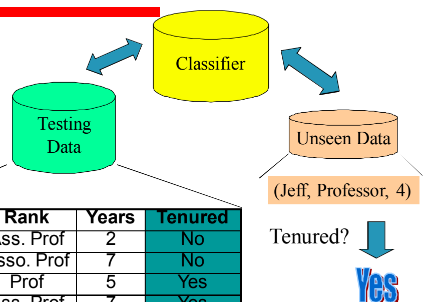
- Model construction: describe a set of predetermined classes
  - Training dataset: tuples for model construction
    - Each tuple/sample belongs to a predefined class
  - Classification rules, decision trees, or math formulae
- Model application: classify unseen objects
  - Estimate accuracy of the model using an independent test set
  - Acceptable accuracy → apply the model to classify tuples with unknown class labels

Model Construction



Name	Rank	Years	Tenured
Mike	Ass. Prof	3	No
Mary	Ass. Prof	7	Yes
Bill	Prof	2	Yes
Jim	Asso. Prof	7	Yes
Dave	Ass. Prof	6	No
Anne	Asso. Prof	3	No

Model Application



Name	Rank	Years	Tenured
Tom	Ass. Prof	2	No
Merlisa	Asso. Prof	7	No
George	Prof	5	Yes
Joseph	Ass. Prof	7	Yes

Supervised vs. Unsupervised Learning

- Supervised learning (classification)
  - Supervision: objects in the training data set have labels
  - New data is classified based on the training set
- Unsupervised learning (clustering)
  - The class labels of training data are unknown
  - Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data

GUI-Application Stabilizer

- Given a program state ***S*** and an event ***e***, predict whether ***e*** likely results in a bug
  - Positive samples: past bugs
  - Negative samples: “not bug” reports
- A k-NN based approach
  - Consider the k closest cases reported before
  - Compare Σ 1/d for bug cases and not-bug cases, where d is the similarity between the current state and the reported states
  - If the current state is more similar to bugs, predict a bug

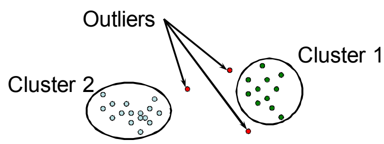
[Michail&Xie 05]

Data Mining Techniques in SE

- Association rules and frequent patterns
- Classification
- Clustering
- Misc.

What is Clustering?

- Group data into clusters
  - Similar to one another within the same cluster
  - Dissimilar to the objects in other clusters
  - Unsupervised learning: no predefined classes



Clustering and Categorization

- Software categorization
  - Partitioning software systems into categories
- Categories predefined – a classification problem
- Categories discovered automatically – a clustering problem

Software Categorization - MUDABlue

- Understanding source code
  - Use Latent Semantic Analysis (LSA) to find similarity between software systems
  - Use identifiers (e.g., variable names, function names) as features
    - "`gtk_window`" represents some window
    - The source code near "`gtk_window`" contains some GUI operation on the window
- Extracting categories using frequent identifiers
  - "`gtk_window`", "`gtk_main`", and "`gpointer`" → GTK related software system
  - Use LSA to find relationships between identifiers

[Kawaguchi et al. 04]

Data Mining Techniques in SE

- Association rules and frequent patterns
- Classification
- Clustering
- Misc.

Other Mining Techniques

- Automaton/grammar/regular expression learning
- Searching/matching
- Concept analysis
- Template-based analysis
- Abstraction-based analysis

http://ase.csc.ncsu.edu/dmse/miningalgs.html

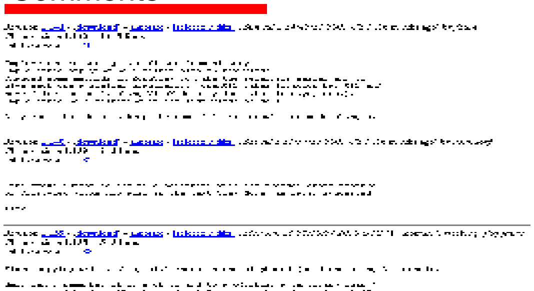
How to Do Research in Mining SE Data

How to do research in mining SE data

- We discussed results derived from:
  - Historical data:
    - Source control
    - Bug databases
    - Mailing lists
  - Program data:
    - Program source code
    - Program execution traces
- We discussed several mining techniques
- We now discuss how to:
  - Get access to a particular type of SE data
  - Process the SE data for further mining and analysis

Source Control Repositories

Concurrent Versions System (CVS) Comments



[Chen et al. 01] http://cvssearch.sourceforge.net/

CVS Comments

- **cvs log** – displays for all revisions and its comments for each file
- **cvs diff** – shows differences between different versions of a file
- Used for program understanding

```
RCS files:/repository/file.h,v
Working file: file.h
head: 1.5
...
description:
----------------------------
Revision 1.5
Date: ...
cvs comment ...
----------------------------
...
```

```
…
RCS file: /repository/file.h,v
…
9c9,10
< old line
---
> new line
> another new line
```

[Chen et al. 01] http://cvssearch.sourceforge.net/

Code Version Histories

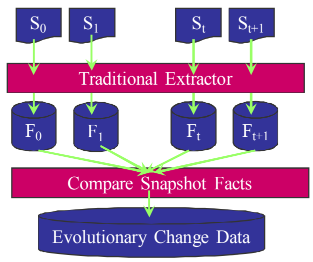
- CVS provides file versioning
  - Group individual per-file changes into **individual transactions**: checked in by the same author with the same check-in comment within a short time window
- CVS manages only files and line numbers
  - Associate **syntactic entities** with line ranges
- Filter out long transactions not corresponding to meaningful atomic changes
  - E.g., features and bug fixes vs. branch and merging
- Used to mine co-changed entities

[Hassan& Holt 04, Ying et al. 04]
[Zimmermann et al. 04] http://www.st.cs.uni-sb.de/softevo/erose/

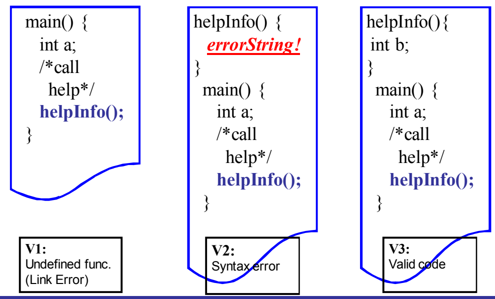
Getting Access to Source Control

- These tools are commonly used
  - **Email:** ask for a local copy to avoid taxing the project's servers during your analysis and development
  - **CVSup:** mirrors a repository if supported by the particular project
  - **rsync:** a protocol used to mirror data repositories
  - **CVSsuck:**
    - Uses the CVS protocol itself to mirror a CVS repository
    - The CVS protocol is not designed for mirroring; therefore, CVSsuck is not efficient
    - Use as a last resort to acquire a repository due to its inefficiency
    - Used primarily for dead projects

Recovering Information from CVS



Challenges in recovering information from CVS

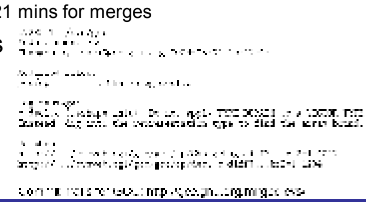


CVS Limitations

- CVS has limited query functionality and is slow
- CVS does not track co-changes
- CVS tracks only changes at the file level

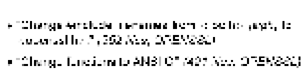
Inferring Transactions in CVS

- Sliding Window:
  - Time window: [3-5mins on average]
    - min 3mins
    - as high as 21 mins for merges
- Commit Mails

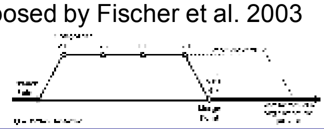


Noise in CVS Transactions

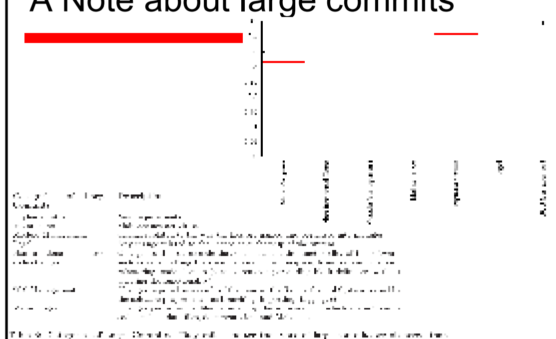
- Drop all transactions above a large threshold



- For Branch merges either look at CVS comments or use heuristic algorithm proposed by Fischer et al. 2003

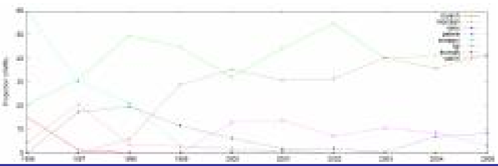


A Note about large commits



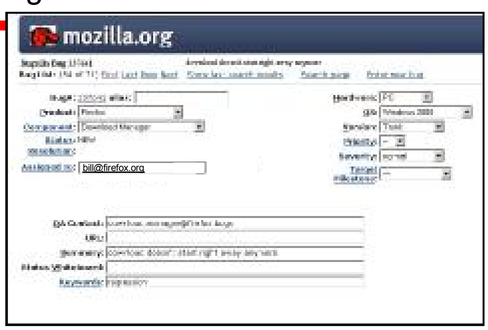
Noise in detecting developers

- Few developers are given commit privileges
- Actual developer is usually mentioned in the change message
- One must study project commit policies before reaching any conclusions

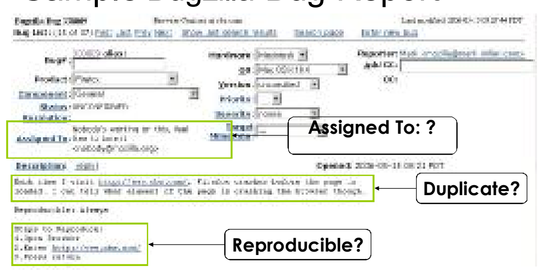


Source Control and Bug Repositories

Bugzilla



Sample Bugzilla Bug Report



Bugzilla: open source bug tracking tool
http://www.bugzilla.org/
[Anvik et al. 06]
http://www.cs.ubc.ca/labs/spl/projects/bugTriage.html

Acquiring Bugzilla data

- Download bug reports using the XML export feature (in chunks of 100 reports)
- Download attachments (one request per attachment)
- Download activities for each bug report (one request per bug report)

Using Bugzilla Data

- Depending on the analysis, you might need to rollback the fields of each bug report using the stored changes and activities
- Linking changes to bug reports is more or less straightforward:
  - Any number in a log message could refer to a bug report
  - Usually good to ignore numbers less than 1000. Some issue tracking systems (such as JIRA) have identifiers that are easy to recognize (e.g., JIRA-4223)

So far: Focus on fixes

teicher   2003-10-29 16:11:01

fixes issues mentioned in bug 45635: [hovering] rollover hovers
- mouse exit detection is safer and should not allow for loopholes any more, except for shell deactiviation
- hovers behave like normal ones:
    - tooltips pop up below the control
    - they move with subjectArea
    - once a popup is showing, they will show up instantly

Fixes give only the location of a defect, not when it was introduced.

Bug-introducing changes

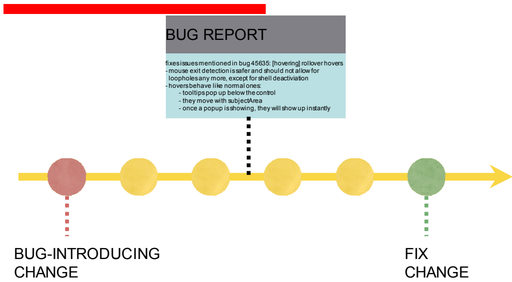
BUG-INTRODUCING
...
if (foo==null) {
  foo.bar();
...

later fixed

FIX
...
if (foo!=null) {
  foo.bar();
...

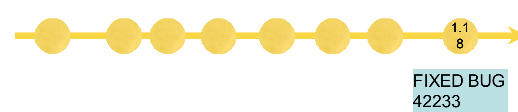
Bug-introducing changes are changes that lead to problems as indicated by later fixes.

Life-cycle of a "bug"



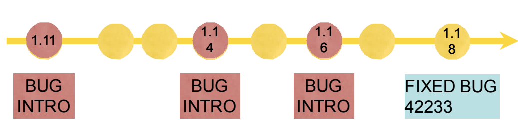
The SZZ algorithm

$ cvs annotate -r 1.17 Foo.java
...
20: 1.11 (john 12-Feb-03):   return i/0;
...
40: 1.14 (kate 23-May-03):   return 42;
...
60: 1.16 (mary 10-Jun-03):   int i=0;

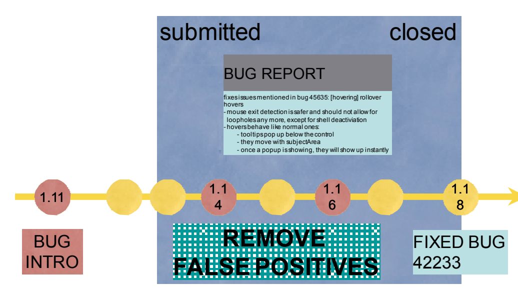


The SZZ algorithm

$ cvs annotate -r 1.17 Foo.java
...
20: 1.11 (john 12-Feb-03):   return i/0;
...
40: 1.14 (kate 23-May-03):   return 42;
...
60: 1.16 (mary 10-Jun-03):   int i=0;



The SZZ algorithm



Project Communication – Mailing lists

Acquiring Mailing lists

- Usually archived and available from the project's webpage
- Stored in mbox format:
  - The mbox file format sequentially lists every message of a mail folder

Challenges using Mailing lists data I

- Unstructured nature of email makes extracting information difficult
  - Written English
- Multiple email addresses
  - Must resolve emails to individuals
- Broken discussion threads
  - Many email clients do not include "In-Reply-To" field

Challenges using Mailing lists data II

- Country information is not accurate
  - Many sites are hosted in the US:
    - Yahoo.com.ar is hosted in the US
- Tools to process mailbox files rarely scale to handle such large amount of data (years of mailing list information)
  - Will need to write your own

Program Source Code

Acquiring Source Code

- Ahead-of-time download directly from code repositories (e.g., Sourceforge.net)
  - Advantage: offline perform slow data processing and mining
  - Some tools (Prospector and Strathcona) focus on framework API code such as Eclipse framework APIs
- On-demand search through code search engines:
  - E.g., http://www.google.com/codesearch
  - Advantage: not limited on a small number of downloaded code repositories

Prospector: http://snobol.cs.berkeley.edu/prospector
Strathcona: http://lsmr.cs.ucalgary.ca/projects/heuristic/strathcona/

Processing Source Code

- Use one of various static analysis/compiler tools (McGill Soot, BCEL, Berkeley CIL, GCC, etc.)
- But sometimes downloaded code may not be compliable
  - E.g., use Eclipse JDT http://www.eclipse.org/jdt/ for AST traversal
  - E.g., use exuberant ctags http://ctags.sourceforge.net/ for high-level tagging of code
- May use simple heuristics/analysis to deal with some language features [Xie&Pei 06, Mandelin et al. 05]
  - Conditional, loops, inter-procedural, downcast, etc.

Program Execution Traces

Acquiring Execution Traces

- Code instrumentation or VM instrumentation
  - Java: ASM, BCEL, SERP, Soot, Java Debug Interface
  - C/C++/Binary: Valgrind, Fjalar, Dyninst
- See Mike Ernst's ASE 05 tutorial on "Learning from executions: Dynamic analysis for software engineering and program understanding"

http://pag.csail.mit.edu/~mernst/pubs/dynamic-tutorial-ase2005-abstract.html

More related tools: http://ase.csc.ncsu.edu/tools/

Processing Execution Traces

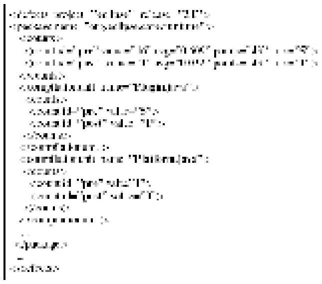
- Processing types: online (as data is encountered) vs. offline (write data to file)
- May need to group relevant traces together
  - e.g., based on receiver-object references
  - e.g., based on corresponding method entry/exit
- Debugging traces: view large log/trace files with V-file editor: http://www.fileviewer.com/

Tools and Repositories

Repositories Available Online

- Promise repository:
  - http://promisedata.org/
- Eclipse bug data:
  - http://www.st.cs.uni-sb.de/softevo/bug-data/eclipse/
- iBug
  - http://www.st.cs.uni-sb.de/ibugs/
- MSR Challenge (data for Mozilla & Eclipse):
  - http://msr.uwaterloo.ca/msr2007/challenge/
  - http://msr.uwaterloo.ca/msr2008/challenge/
- FLOSSmole:
  - http://ossmole.sourceforge.net/
- Software-artifact infrastructure repository:
  - http://sir.unl.edu/portal/index.html

Eclipse Bug Data



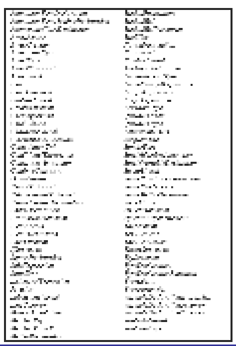
- Defect counts are listed as **counts** at the plug-in, package and compilation unit levels.
- The **value** field contains the actual number of pre- ("**pre**") and post-release defects ("**post**").
- The average ("**avg**") and maximum ("**max**") values refer to the defects found in the compilation units ("**compilationunits**").

[Schröter et al. 06] http://www.st.cs.uni-sb.de/softevo/bug-data/eclipse/

Metrics in the Eclipse Bug Data

		Metric	File level	Package level
methods	FOUT	Number of method calls (fan out)	avg, max, total	avg, max, total
	MLOC	Method lines of code	avg, max, total	avg, max, total
	NBD	Nested block depth	avg, max, total	avg, max, total
	PAR	Number of parameters	avg, max, total	avg, max, total
	VG	McCabe cyclomatic complexity	avg, max, total	avg, max, total
classes	NOF	Number of fields	avg, max, total	avg, max, total
	NOM	Number of methods	avg, max, total	avg, max, total
	NSF	Number of static fields	avg, max, total	avg, max, total
	NSM	Number of static methods	avg, max, total	avg, max, total
files	ACD	Number of anonymous type declarations	value	avg, max, total
	NOI	Number of interfaces	value	avg, max, total
	NOT	Number of classes	value	avg, max, total
	TLOC	Total lines of code	value	avg, max, total
packages	NOCU	Number of files (compilation units)	N/A	value

Abstract Syntax Tree Nodes in Eclipse Bug Data



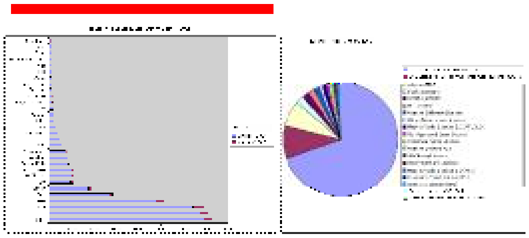
- The AST node information can be used to calculate various metrics

FLOSSmole

- **FLOSSmole**
  - provides raw data about open source projects
  - provides summary reports about open source projects
  - integrates donated data from other research teams
  - provides tools so you can gather your own data
- **Data sources**
  - Sourceforge
  - Freshmeat
  - Rubyforge
  - ObjectWeb
  - Free Software Foundation (FSF)
  - SourceKibitzer

http://ossmole.sourceforge.net/

Example Graphs from FlossMole



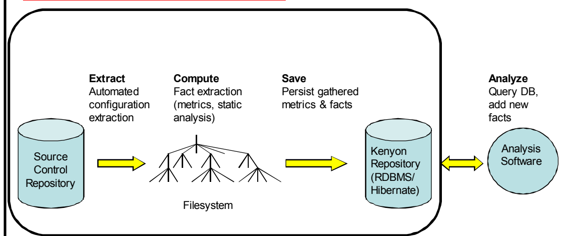
Analysis Tools

- **R**
  - http://www.r-project.org/
  - R is a free software environment for statistical computing and graphics
- **Aisee**
  - http://www.aisee.com/
  - Aisee is a graph layout software for very large graphs
- **WEKA**
  - http://www.cs.waikato.ac.nz/ml/weka/
  - WEKA contains a collection of machine learning algorithms for data mining tasks
- **RapidMiner (YALE)**
  - http://rapidminer.com/
- More tools: http://ase.csc.ncsu.edu/dmse/resources.html

Data Extraction/Processing Tools

- **Kenyon**
  - http://dforge.cse.ucsc.edu/projects/kenyon/
- **MyIn/Mylar** (comes with API for Bugzilla and JIRA)
  - http://www.eclipse.org/myln/
- **Libresoft toolset**
  - Tools (cvsanaly/mlstats/detras) for recovering data from cvs/svn and mailinglists
  - http://forge.morfeo-project.org/projects/libresoft-tools/

Kenyon



[Adapted from Bevan et al. 05]

Publishing Advice

- Report the statistical significance of your results:
  - Get a statistics book (one for social scientist, not for mathematicians)
- Discuss any limitations of your findings based on the characteristics of the studied repositories:
  - Make sure you manually examine the repositories. Do not fully automate the process!
  - Use random sampling to resolve issues about data noise
- Relevant conferences/workshops:
  - main SE conferences, ICSM, ISSTA, MSR, WODA, …

Mining Software Repositories

- Very active research area in SE:
  - MSR is the most attended ICSE event in last 5 yrs
    - http://msrconf.org
  - Special Issue of IEEE TSE on MSR:
    - 15 % of all submissions of TSE in 2004
    - Fastest review cycle in TSE history: 8 months
  - Special Issue Empirical Software Engineering (late 08)
  - Upcoming Special Issues:
    - Journal of Empirical Software Engineering
    - Journal of Soft. Maintenance and Evolution
    - IEEE Software (July 1st 2008)

Q&A

Mining Software Engineering Data Bibliography
http://ase.csc.ncsu.edu/dmse/
- What software engineering tasks can be helped by data mining?
- What kinds of software engineering data can be mined?
- How are data mining techniques used in software engineering?
- Resources

Example Tools

- **MAPO**: mining *API usages* from *open source repositories* [Xie&Pei 06]
- **DynaMine**: mining *error/usage patterns* from *code revision histories* [Livshits&Zimmermann 05]
- **BugTriage:** learning *bug assignments* from *historical bug reports* [Anvik et al. 06]

Demand-Driven Or Not

	Any-gold mining	Demand-driven mining
Examples	DynaMine, …	MAPO, BugTriage, …
Advantages	Surface up only cases that are applicable	Exploit demands to filter out irrelevant information
Issues	How much gold is good enough given the amount of data to be mined?	How high percentage of cases would work well?

Code vs. Non-Code

	Code/ Programming Langs	Non-Code/ Natural Langs
Examples	MAPO, DynaMine, …	BugTriage, CVS/Code comments, emails, docs
Advantages	Relatively stable and consistent representation	Common source of capturing programmers' intentions
Issues		What project/context-specific heuristics to use?

Static vs. Dynamic

	Static Data: code bases, change histories	Dynamic Data: prog states, structural profiles
Examples	MAPO, DynaMine, …	Spec discovery, …
Advantages	No need to set up exec environment; More scalable	More-precise info
Issues	How to reduce false positives?	How to reduce false negatives? Where tests come from?

Snapshot vs. Changes

	Code snapshot	Code change history
Examples	MAPO, …	DynaMine, …
Advantages	Larger amount of available data	Revision transactions encode more-focused entity relationships
Issues		How to group CVS changes into transactions?

Characteristics in Mining SE Data

- Improve quality of source data: data preprocessing
  - MAPO: inlining, reduction
  - DynaMine: call association
  - BugTriage: labeling heuristics, inactive-developer removal
- Reduce uninteresting patterns: pattern postprocessing
  - MAPO: compression, reduction
  - DynaMine: dynamic validation
- Source data may not be sufficient
  - DynaMine: revision histories
  - BugTriage: historical bug reports

SE-Domain-Specific Heuristics are important