Towards Understanding of Software Changes

Audris Mockus

audris@avaya.com

Avaya Labs Research
Basking Ridge, NJ 07920
http://mockus.org/

Goals

- To understand software evolution
  - by understanding changes
- Universe is built from atoms
  - and software is built from changes
- To get the principles of software evolution
  - by finding fundamental properties of changes

Why study changes

- Reflect relationships between
  - requirements and design
  - technology and implementation
  - personel (organization)
  - time (evolution of the system)
- Practical
  - changes are tracked to enable multiple people to work on them
  - always documented by version control systems
  - results have wide applicability

Model of a Version Control System

- Change record
  - Textual description, often including bug numbers
  - Date, Time, and, often, the size of change
  - Change (file before and after change)
- Hierarchies:
  - File/Module
  - Developer/Organization
  - Feature/Project

Goal: Version Control Census

- Gather all public VCS
- Create Universal Version History
- Determine
  - global extent of code reuse
  - authorship (succession)
  - study innovation, defect propagation, compare properties of widely reused code and non-reused code

Steps for Version Control Census

- Discover VCS repositories
- Copy/clone repositories
- Extract and index all versions of each file

Discover VCS repositories

- Forges, e.g., SourceForge, GoogleCode, Savannah
- Miniforges, e.g., FreeBSD, NetBSD, OpenBSD, OpenSolaris, Gnome, KDE, Eclipse, RubyForge, OpenSolaris, NetBeans, OpenJDK, and Mozilla, common-lisp.net, cvs.kaffe.org, cvs.xemacs.org, freedesktop.org, and sourceware
- Large/well-known, e.g., Mysql, Perl, Wine, Postgres, and GCC
- Most widely used — use distributions Gentoo, Debian, Slackware, OpenSuse, and RedHat
- 3G VCS: repo.or.cz, github.com, gitorious.org, git.debian.org
- Published surveys of projects
- Directories: RawMeat and FSF
- Verify: search for common filenams on Google Code Search to see if new files are discovered

How to automate VCS discovery?

- CVS, Subversion, Git, Mercurial, and Bazaar all have a specific pattern for the URLs pointing to a project repository
  - cvs:, svn:, git: or cvs., svn., git., hg., bzr.
- Creatie a spider utilizing a search engine, and seeded by project directories (RawMeat, FSF) could grab these URLs from projects' home page

Copy/clone

- if copy/clone is possible
  - rsync CVS repositories, e.g,
    - rsync -avu rsync://cvs.savannah.gnu.org/sources/”$PRJ” .
  - Mirror subversion:
    - svnadmin create $SVMREPOS; svm init $PRJ
    - svm sync $PRJ svn://svn.forge.objectweb.org/svnroot/$PRJ
  - Clone others
    - git clone git://perl5.git.perl.org/perl.git perl
    - hg clone ssh://anon@hg.opensolaris.org/hg/fuse/fusefs
    - bzr branch lp:mysql-server
- If copy/clone not possible, extract over internet

Extract/Index: list revisions

- List revisions
  - CVS: find $PRJ -name '*,v' | rlog "$REPLY" | perl extr.perl
  - SVN:
    - svn log -v ––non-interactive svn://svn.forge.objectweb.org/svnroot/"$PRJ" | perl extrsvn.perl
  - GIT:
    - git log ––numstat -M -C ––diff-filter=ACMR ––full-history ––pretty=tformat:"STARTOFTHECOMMIT%n%H;%T;%P;%an;%ae | perl extrgit.perl
  - Mercurial: hg log -v $PRJ — perl extrhg.perl
  - Bazaar: bzr log -v ––long $PRJ — perl extrbzr.perl

Extract/Index: get content

- For each revision (obtained above) extract content:
  - CVS: rcs -p$REV $FILE
  - SVN: svn cat -r$REV $URL/$FILE$REV
  - GIT: git show $REV:$FILE
  - Mercurial: hg cat -r$REV $FILE
  - Bazaar: bzr cat -r$REV $FILE

Extract/Index: store/index content

- Compress: $ccon = compress $con
- Insert into a hashtable (DBFile): $clones{$ccon} = $idx
  - where $idx is a new integer if content is not in the table
  - $clones{$ccon} if such content already exists
- print log: $idx;size;$FILE/$VERSION

Whats there

Forge	Type	VCSs	Files	File/Versions	Disk Space
git.kernel.org	Git	595	12,974,502	97,585,997	205GB
SourceForge	CVS	121,389	26,095,113	81,239,047	820GB
netbeans	Mercurial	57	185,039	23,847,028	69GB
github.com	Git	29,015	5,694,237	18,986,007	154GB
repo.or.cz	Git	1,867	2519529	11,068,696	43GB
Kde	Subversion	1	2,645,452	10,162,006	50GB
code.google	Subversion	33,292[a]	4,936,428	8,787,662	remote
gitorious.org	Git	1,098	1,229,185	4,896,943	20GB
Gcc	Subversion	1	3,758,856	4,803,695	14GB
Debian	Git	1662	1,058,120	4,741,273	19GB
gnome.org	Subversion	566	1,284,074	3,981,198	1GB
Savannah	CVS	2,946	852,462	3,623,674	25GB
forge.objectweb.org	Subversion	93	1,778,598	2,287,258	17GB
Eclipse	CVS	9	729,383	2,127,009	11GB
SourceWare	CVS	65	213,676	1,459,220	10GB
OpenSolaris	Mercurial	98	77,469	1,108,338	9.7GB
rubyforge.org	Subversion	3,825	456,067	807,421	4.9GB
Freedesktop	CVS	75	139,225	784,021	4GB
OpenJDK	Mercurial	392	32,273	747,861	15GB
Mysql-Server	Bazaar	1	10,786	523,383	6GB
FreeBSD	CVS	1	196,988	360,876	2.5GB
ruby-lang	Subversion	1	163,602	271,032	0.6GB
Mozilla	Mercurial	14	58,110	210,748	1.6GB
PostgreSQL	CVS	1	6,967	108,905	0.5GB
Perl	Git	1	11,539	103,157	0.2GB
Python	Subversion	1	8,601	89,846	0.8GB

Questions about changes

- Basic descriptive
  - What are the types of changes?
  - What are the best change profiles over/for time, file, developer, project?
- Comparison
  - comparing two or more projects
  - Prediction
  - what type/how many/where/when/who?

Example: Why code is changed?

- Primary reasons for maintenance activities
  - corrective: fix faults
  - adaptive: add features
- How those reasons relate to:
  - interval, effort, quality developer, size location, time

Why code is changed?

- How to obtain the purpose?
  - Look for bug/new field
    - may not be there, unreliable, only two values
  - Ask developers
    - too much overhead - small coverage
  - Read change abstracts
    - great idea - but 2M abstracts
  - Let computer read abstracts
    - but how?

An algorithm

- Use change description line
- extract frequent keywords
- classify keywords (fix, new, add, etc.)
- discover new types
  - perfective - code cleanup
  - inspection - code inspection suggestions
- verify on sample abstracts
  - keyword -¿ purpose of the change
  - iterate

Example keywords

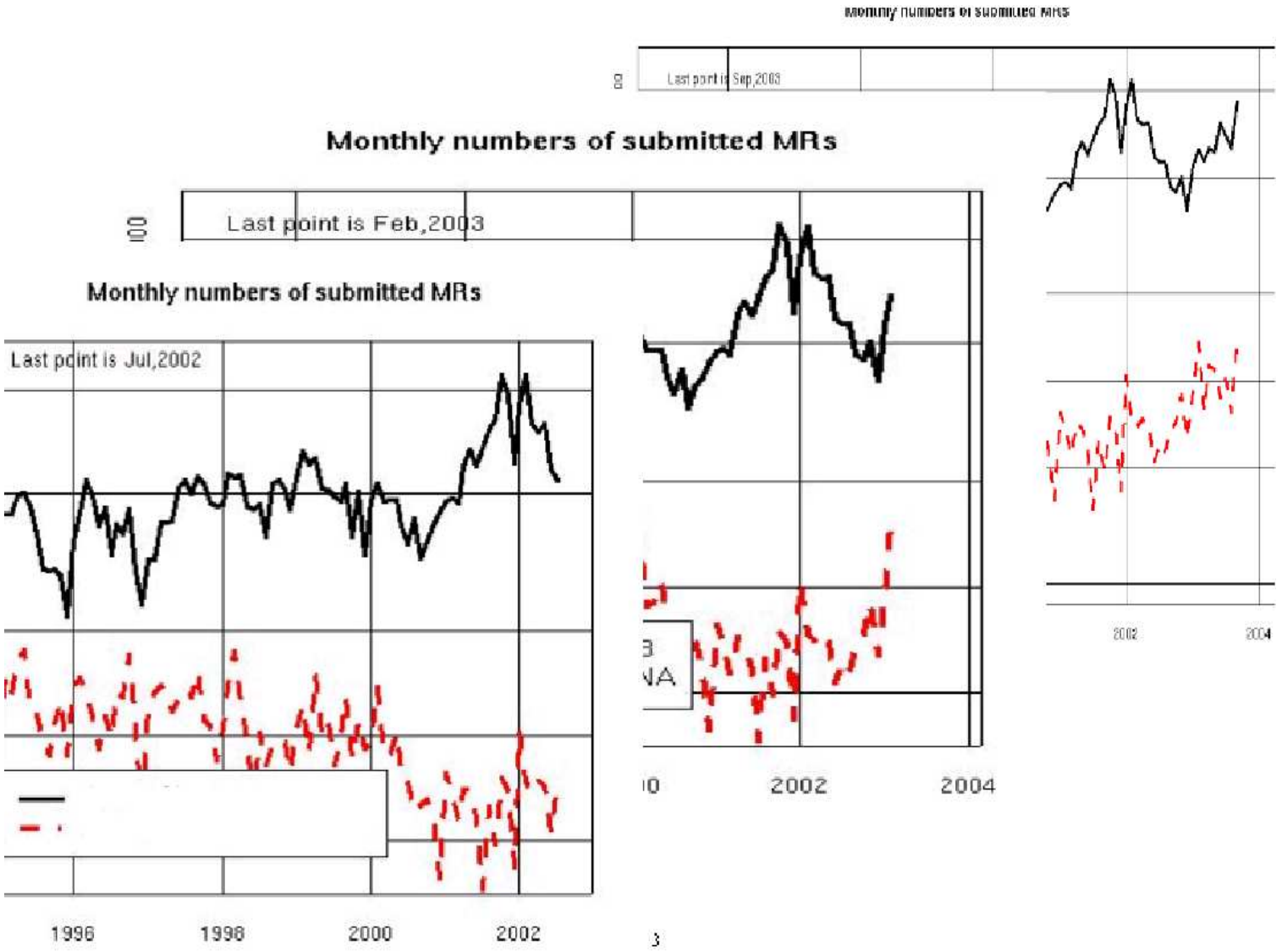
Proportions

- add new functionality - 45%
- fix faults (bug) - 34%
- cleanup/restructure - 4%
- code inspection - 5%
- unclassified - 12%

Application: Software Dashboards

- A set of web pages or ”dashboards” is created to summarize the current state development in AVAYA in order to support decision making based on quantitative information
  - Make comparisons to what happened the past
  - To anticipate future needs
  - Estimate project completion dates
  - Estimate the amount of resources needed
  - Summarize aspects of quality and productivity in individual projects

Example: predicting quality



Tools for Software Mining

- Absent experimental tools - build your own
  - SoftChange: http://sourcefourge.net/projects/sourcechange
- Related tools
  - R - high-end/extensible data analysis/statistics (google:R language)
  - SW data cleaning, text analysis, CM/VC interfaces
    - Cleaning: regular expressions (Perl, Tcl, sed)
    - Tabulation: hashtables, arrays, sorting (Perl, Shell)
    - Text analysis: word stemming, word semantics (google: WordNet), sentence semantics ???

Systems used in a typical organization

- Sales/Marketing: customer information, customer rating, customer purchase patters, customer needs: features and quality
- Accounting: Customer/system/software billing information and maintenance support level
- Maintenance support: Currently installed system, support level
- Field support: dispatching repair people, replacement parts
- Call center support: customer call/problem tracking
- Development field support: software related customer problem tracking, installed patch tracking
- Development: feature and development, testing, and field defect tracking, software change and software build tracking

How to foster experimentation in industry

- Volunteer help to projects in trouble
- Help with the top problem the project is faced, but collect information for the the future problems (the empirical work)
- Provide value for all parties involved (services vs development vs product vs CIO)
- Be sensitive about the privacy at individual and team level.
- Show something significant and unexpected about the project early on
  - Demonstrate immediate results
  - Gain credibility
- Get commitments for other studies

Methodology: Extraction

- Get access to the systems
- Extract raw data
  - change table, developer table. (SCCS: prs, ClearCase: cleartool -lsh, CVS:cvs log), write/modify drivers for other CM/VCS/Directory systems
  - Interview the tool support person (especially for home-grown tools)
- Do basic cleaning
  - Eliminate administrative, automatic, post-preprocessor changes
  - Assess the quality of the available attributes (type, dates, logins)
  - Eliminate un- or auto-populated attributes
  - Eliminate remaining system generated artifacts

Methodology: Validation

- Interview a sample of developers, testers, project manager, tech. support
  - Go over recent change(s) the person was involved with
    - to illustrate the actual process (what is the nature of the work item, why you got it, who reviewed it)
    - to understand/validate the meaning various attribute values: (when was the work done, for what purpose, by whom)
    - to gather additional data: effort spent, information exchange with other project participants
    - to add experimental/task specific questions
- Augment MR properties via relevant models: purpose [8], effort [1], risk [9]
- Validate and clean recorded and modeled data
- Iterate

Methodology: Why Use Project Repositories?

- The data collection is non-intrusive (using only existing data minimizes overhead)
- Long history of past projects enables historic comparisons, calibration, and immediate diagnosis in emergency situations.
- The information is fine grained: at MR/delta level
- The information is complete: everything under version control is recorded
- The data are uniform over time
- Even small projects generate large volumes of changes: small effects are detectable.
- The version control system is used as a standard part of a project, so the development project is unaffected by observer

Methodology: Pitfalls of Using Project Repositories

- Different process: how work is broken down into work items may vary across projects
- Different tools: CVS, ClearCase, SCCS, ...
- Different ways of using the same tool: under what circumstances the change is submitted, when the MR is created
- The main challenge: create change based models of key problems in software engineering

Methodology: Existing Models

- Predicting the quality of a patch [9]
- Work coordination:
  - What parts of the code can be independently maintained [10]
  - Who are the experts to contact about any section of the code [7]
  - How to measure organizational dependencies [3]
- Effort: estimate MR effort and benchmark process
  - What makes some changes hard [4]
  - What processes/tools work [1, 2]
  - What are OSS/Commercial process differences [6]
- Project models
  - Release schedule [11]
  - Release readiness criteria [5]
  - Consumer perceived quality

References

[1] D. Atkins, T. Ball, T. Graves, and A. Mockus. Using version control data to evaluate the impact of software tools: A case study of the version editor. *IEEE Transactions on Software Engineering*, 28(7):625–637, July 2002.

[2] D. Atkins, A. Mockus, and H. Siy. Measuring technology effects on software change cost. *Bell Labs Technical Journal*, 5(2):7–18, April–June 2000.

[3] James Herbsleb and Audris Mockus. Formulation and preliminary test of an empirical theory of coordination in software engineering. In *2003 International Conference on Foundations of Software Engineering*, Helsinki, Finland, October 2003. ACM Press.

[4] James D. Herbsleb, Audris Mockus, Thomas A. Finholt, and Rebecca E. Grinter. An empirical study of global software development: Distance and speed. In *23nd International Conference on Software Engineering*, pages 81–90, Toronto, Canada, May 12-19 2001.

[5] Audris Mockus. Analogy based prediction of work item flow in software projects: a case study. In *2003 International Symposium on Empirical Software Engineering*, pages 110–119, Rome, Italy, October 2003. ACM Press.

[6] Audris Mockus, Roy T. Fielding, and James Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):1–38, July 2002.

[7] Audris Mockus and James Herbsleb. Expertise browser: A quantitative approach to

identifying expertise. In *2002 International Conference on Software Engineering*, pages 503–512, Orlando, Florida, May 19-25 2002. ACM Press.

[8] Audris Mockus and Lawrence G. Votta. Identifying reasons for software change using historic databases. In *International Conference on Software Maintenance*, pages 120–130, San Jose, California, October 11-14 2000.

[9] Audris Mockus and David M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, April–June 2000.

[10] Audris Mockus and David M. Weiss. Globalization by chunking: a quantitative approach. *IEEE Software*, 18(2):30–37, March 2001.

[11] Audris Mockus, David M. Weiss, and Ping Zhang. Understanding and predicting effort in software projects. In *2003 International Conference on Software Engineering*, pages 274–284, Portland, Oregon, May 3-10 2003. ACM Press.