How to run empirical studies using project repositories

Audris Mockus

Avaya Labs

audris@mockus.org

Avaya Labs Research

Basking Ridge, NJ 07920

http://www.research.avayalabs.com/user/audris

Objective

- Difficulties in software measurement based on software change repositories
  - Report personal work/review experiences
  - Focus on a small subset of issues in-depth
  - No attempt to cover all or even the most important issues

Classes of Issues Discussed

- Irrelevant topic
- Overly specialized results
- Technical mistakes

Motivation

- What world needs
  - Understand and improve software practice
    - Informed (quantitative) tradeoffs between schedule, quality, cost
      - Understanding: where effort is spent, where defects are introduced
      - Acting: the impact of technologies/processes/organization
- Obstacles - lack of focus on software measurement
  - Low priority except in emergencies
  - Need for immediate results (short time horizon)
  - Lack of resources for measurement/improvement
  - Multiple stakeholders (developer/support/product management)

Background

- Software is a virtual product and does not exist outside developers' heads and sources or binaries in conjunction with development, maintenance, installation, configuration, and execution tools and environments
- Most tools and environment involved leave traces of development and maintenance activities in the form of event logs or state changes
- Therefore, more information may be recorded when producing software than in production of physical items
- Approach
  - Use project's repositories of change data to model (explain and predict) phenomena in software projects and to create tools that improve software productivity/quality/lead times

Systems commonly used in a typical organization

- Sales/Marketing: customer information, customer ratings, 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

Advantages of project repositories

- The data collection is non-intrusive (using only existing data minimizes overhead). *Requires in-depth understanding of project's development process*
- Long history on past projects enables historic comparisons, calibration, and immediate diagnosis in emergency situations. *It takes time and effort to get to that point.*
- The information is fine grained, at the MR/delta level. *Links to higher level (more sensible) attributes like features and releases is often tenuous.*
- The information is complete, everything under version control is recorded. *Except for fields, often essential, that are inconsistently or rarely filled in.*

Advantages of project repositories

- The data are uniform over time. *That does not imply that the process was constant over entire period.*
- Even small projects generate large volumes of changes making it possible to detect even small effects statistically. *As long as the relevant quantities are extractable.*
- The version control system is used as a standard part of the project, so the development project is unaffected by experimenter intrusion. *It is no longer true when the such data is used widely in organizational measurement.*

Irrelevant topic

- Model things that are easy to measure
  - Counts, trends, patterns
- Explore topics that are well formulated but of limited value
  - Which modules will get defects
- Overly fundamental laws
  - Power law distribution

Trends

- Number of changes and average lines added, deleted, and unchanged per modification over 12 year period. Each bar represents the underlying number
- Largest churn (added and deleted lines) can be observed early on and in 1992
- Average file size reaching peaks in 1993 and then decreasing somewhat

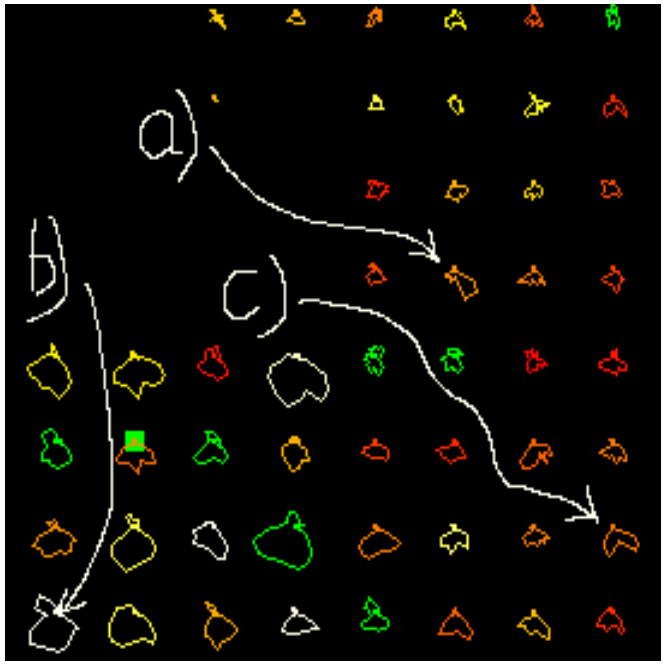
Cases	Count	Added	Deleted	Same
84	1	0	0	9
85	1855	26.3574	4.2814	176.177
86	11919	21.2353	9.56196	303.107
87	14633	32.0191	11.7993	655.749
88	10794	18.1634	7.94191	1009.2
89	19819	17.6601	6.80292	1717.51
90	12533	17.8687	5.85016	2609.18
91	12036	16.5215	5.32635	3336.27
92	12112	22.1412	8.92404	3338.02
93	10254	17.5703	5.08416	3470.49
94	15302	17.875	4.9719	3372.18
95	8385	17.226	4.59213	3088.31
96	2762	15.8856	5.10174	2664.84

Patterns - Developer changes over 24 hours

Code submissions over 16 years

Star icon encodes the number of changes during one hour for a developer as distances from “star” center dot starting at midnight (top) and continuing clock-wise as a 24 hour clock (noon pointing down).

Developers are arranged according to similarity of these patterns.



a) Some have very regular pattern of checkins from 8AM (120 degrees) to 11AM (165 degrees)
b) Some work all day, have dinner break and check in before midnight (pan handle at the top)
c) Some never check in during the day (pie cut between 10AM and 5PM)

Software Releases

- Releases are the primary objects of interest requiring planning, design, implementation, and maintenance
- Unfortunately they are not always easy to associate with software changes or problem reports

Release Terminology

- Generic or base often refers to a major release and minor releases that are following it to fix problems and complete features. Version control files may be copied from a previous generic or a top level branch created. Many OSS projects simply have a development branch and maintenance branches for supported releases
- Load is an executable load of the product produced in development or later phases. Usually denoted via several numbers, e.g., 10.3.5.142
- Patch is a specialized load for an individual customer or a load supplanting a release
- Release is a load (most often several loads) that are delivered to customers under a single release name, e.g., r11.1, or r13.0.1

Identifying releases

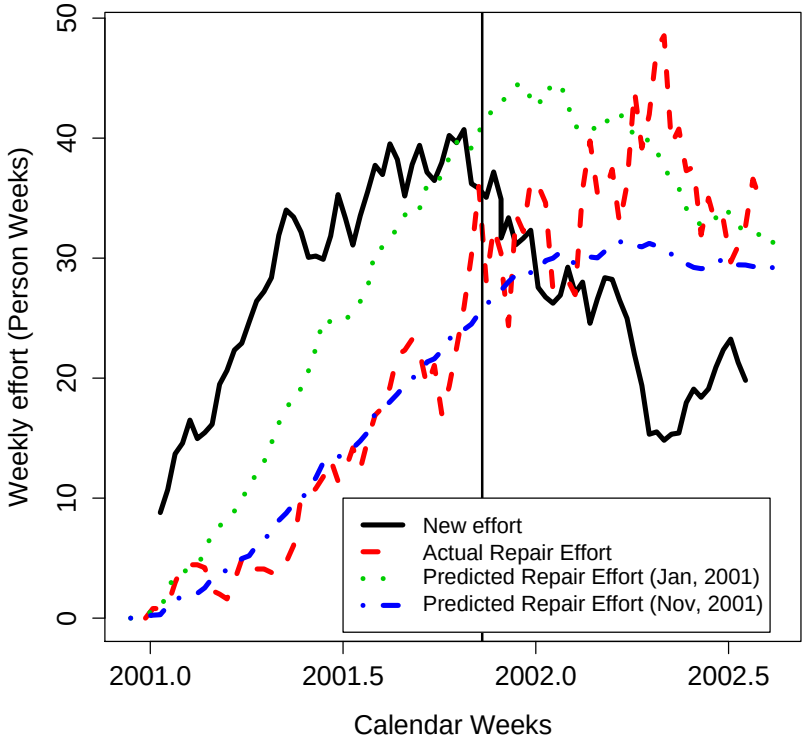
- Generic is often used to track problems. Each problem may be open, assigned, and resolved separately for each generic it may affect. Problem tracking systems that do not treat problems for each generic separately tend to have unusual resolution times for problems affecting several releases.
- Release and date reported identify the release and time when the problem was discovered. They are necessary to use when measuring the quality characteristics of a release. Sometimes date reported is referred to as the origination date

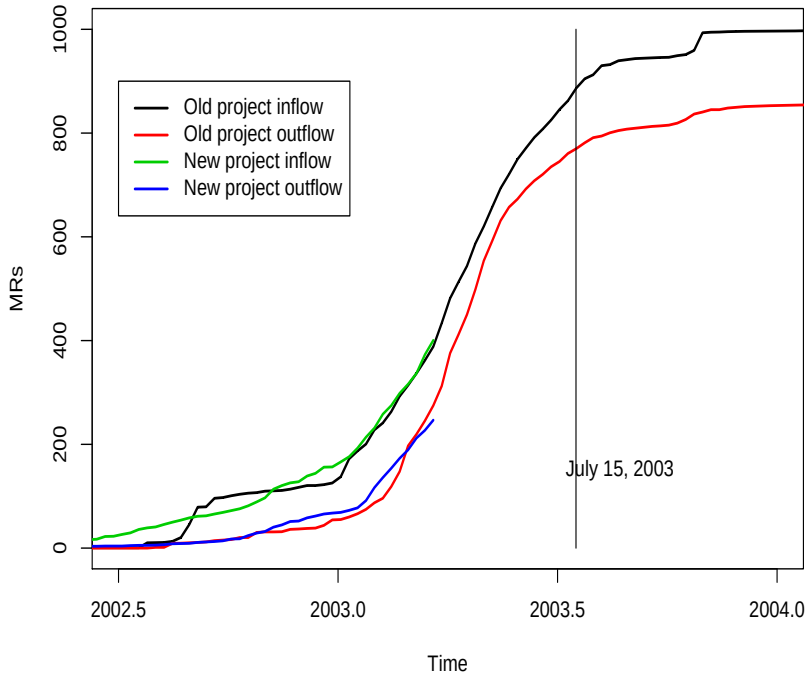
Identifying releases — II

- Release resolved identifies the release in which the problem resolution was implemented. Often, load number is also available. A problem (especially discovered in a maintenance branch) is often resolved in several releases, often at widely different times and sometimes by different developers. It is necessary to identify the effort spent by developer and effort related to a particular release
- Date submitted indicates time when a resolution was provided (different for each release involved). If the resolution is “nochange” or “deferred”, dates for nochange or deferral decisions should be used instead

Phase detected

- Important for quality assessment
- Can be identified by development and maintenance branches in OSS
- The values may not be readily available - use other sources, i.e. field tracking systems





Where faults occur?

- Assume the best possible outcome, i.e., we can predict exactly!
  - This can be evaluated by, for example, looking at actual occurrence after the fact
  - e.g., 50% of the faults occur in 20% of the modules
  - Unfortunately, these 20% of the modules contain 60% of the code!?

order	%files	% faults	% lines	% changes
most faults	10	47	14	20
	19	63	25	29
density (per line)	10	32	7	12
	19	59	17	27
density (per change)	10	41	10	13
	19	54	21	20

Table 1: Fraction of field faults in frequently changed modules

Some models of software changes

- Predicting the quality of a patch [13]
- Work coordination:
  - What parts of the code can be independently maintained [14]
  - Who are the experts to contact about any section of the code [12]
  - How to measure organizational dependencies [6]
- Effort: estimate interval and benchmark process
  - What makes some changes hard and long [7]
  - What processes/tools work and why [1, 2, 5]
  - How do you create a hybrid OSS/Commercial process [11, 4]
- Project models
  - Plan for field problem repair after the release [15, 16]
  - Release readiness criteria [9]
  - Quality: model of customer experience [10, 16]

Some development support tools

- Finding relevant people [12]
- Finding related defects [3]
- Finding related changes [17, 18, 8]
- Finding independently maintainable pieces of code [14]

Identifying Real Problems

- Ask two question:
  - Suppose the questions I am posing can be answered beyond the wildest optimistic projections - what difference will it make?
  - Suppose I will get some handle on these questions - what difference will it make?

Audience that is too narrow

- “Simulating the process of simulating the process”
- Similarly the tools that support software project data analysis

SoftChange

- http://sourceforge.net/projects/sourcechange
- The SoftChange project will create software to summarize and analyze software changes in CVS repositories and defect tracking systems
- Requirements
  - retrieve the raw data from the web or the underlying system via archive downloads, CVS logs, and processing Bugzilla web pages;
  - verify completeness and validity of different change records by cross-matching changes from CVS mail, CVS log, and ChangeLog files; matching changes to PR reports and identities of contributors;
  - construct meaningful measures that can be used to assess various aspects of open source projects.
- Road map at:

`http://sourceforge.net/docman/display_doc.php?docid=15813&group_id=58432P`

Gross Errors

- Lack of validation
  - Limited understanding of the process
  - Insufficient data cleaning
  - Eliminating missing/default/auto values
  - Predictors and responses

Missing data

- MCAR — missing completely at random: never happens
- MAR — missing at random: missingness is random conditional on non-missing values
- Other — missingness depends on the value itself: most common

Example

- Two projects are compared
  - First has 30% of the cases where the attribute is missing
  - Second has 60% of the cases where the attribute is missing
  - Comparison is performed by doing a two-sample t-test on the attributes that are not missing
- Potential problem - the value of the response is likely to depend on whether or not the attribute is missing, i.e., extreme values of an attribute make it more likely to be missing and affect the response

Example: “the right way”

- Sample cases with missing attributes and interview relevant people to determine:
  - Do actual values for missing cases differ from values for non-missing cases
  - Is the difference the same for both projects
  - Can the difference be explained by other non-missing/default values
- If there is no possibility for validation assess the impact of non-random missingness
- And: don’t forget to take logs before doing non-rank based tests

What is the problem with this data?

Priority	project A	Projet B	Project C
1	10	62	
2	201	1642	16
3	3233	9920	659
4	384	344	1

Model of MR interval

- Current MR backlog for a project and developer may affect the time it takes to resolve current MR as developers may have to spread their time among such MRs

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	5.16	0.19	27.82	0.00
log(Dev. backlog)	0.46	0.04	12.84	0.00
log(Proj. backlog)	0.73	0.04	17.73	0.00
many	3.72	0.08	45.54	0.00
log(Lines)	0.03	0.01	2.00	0.05

Table 2: Regression of interval, adj-R^2 =0.57

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	5.23	0.19	27.96	0.00
log(Dev. backlog + 1)	0.41	0.04	11.39	0.00
log(Proj. backlog + 1)	0.64	0.04	16.67	0.00
many	3.97	0.08	52.87	0.00
log(Lines + 1)	0.04	0.01	3.02	0.00

Table 3: Regression of model-generated interval, adj-R^2 =0.61. Model used to generate the data involves only developers and lines and no backlog.

Methodology: Main Principles

Main focus on supporting the 9[5-9]% of the work related to extraction/cleaning/validation

- Use levels and pipes, a la satellite image processing
- Validation tools (regression, interactive) for each level/transition
  - Traceability to sources from each level
  - Multiple operationalizations within/across levels
  - Comparison against invariants
  - Detecting default values
  - Handling missing values

Project Data: Levels [0-2]

- Level 0 — actual project. Learn about the project, access its systems
- Level 1 — 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)
- Level 2 — Do basic cleaning
  - Eliminate administrative and automatic artifacts
  - Eliminate post-preprocessor artifacts

Project Data: Validation

- Learn the real process
  - Interview key people: architect, developer, tester, field support, project manager
    - Go over recent change(s) the person was involved with
      - to illustrate the actual process (What is the nature of this work item, why/where it come to you, who (if any) reviewed it, ...)
      - to understand what the various field values mean: (When was the work done in relation to recorded fields, ...)
      - to ask additional questions: effort spent, information exchange with other project participants, ...
      - to add experimental questions
  - Apply relevant models
  - Validate and clean recorded and modeled data
  - Iterate

Serious Issues with the Approach

- Data cleaning and validation takes at least 95% effort - analysis only 1 to 5 percent
- It is very tempting to model easy-to-obtain yet irrelevant measures
- Need to understand implications of missing data
- Using such data will change developer behavior and, therefore, the meaning such data may have

Pitfalls of using project repositories

- A lot of work — try something simpler first
- Easy to study irrelevant phenomena or tool generated artifacts
- Different process: how work is broken down into work items
- 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 models of key problems in software engineering based on repository data
  - Easy to compute a lot of irrelevant numbers
  - Interesting phenomena are often not captured even in software project data

Discussion

- A vast amount of untapped resources for empirical work
- The usage of VCS/CM is rapidly increasing over time (startups than do not use them are rapidly disappearing)
- Immediate simple applications in project management: MR inflow/outflow
- It is already being used in more advanced projects
- Remaining challenges
  - Build and validate models to address all problems of practical/theoretical significance
  - What information developers would easily and accurately enter?
  - What is the “sufficient statistic” for a software change?

Abstract

Over the last few years software project support tool repositories are increasingly being used for empirical studies of software. This has been primarily driven by the interest in open source projects and the wide availability of their repositories. Unfortunately, the project support tools are not designed as data sources for empirical studies and, therefore, many pitfalls and problems await. This mini-tutorial will discuss and illustrate some of the common things to avoid when analyzing software repositories. The topics range from the high level objectives of empirical study having an irrelevant topic or a too narrow audience to common mistakes related to the impact of systematically missing, default, or tool-generated data.

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] D. Cubranic and G.C Murphy. Hipikat: A project memory for software development. *TSE*, 31(6), 2005.

[4] T Dinh-Trong and Bieman J.M. Open source software development: A case study of freebsd. *IEEE Transactions of Software Engineering*, 31(6), 2005.

[5] Birgit Geppert, Audris Mockus, and Frank Rößler. Refactoring for changeability: A way to go? In *Metrics 2005: 11th International Symposium on Software Metrics*, Como, September 2005. IEEE CS Press.

[6] 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.

[7] 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.

[8] Miryung Kim and David Notkin. Using a clone genealogy extractor for understanding and supporting evolution of code clones. In *International Workshop on Mining Software Repositories*, 2005.

[9] 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.

[10] Audris Mockus. Empirical estimates of software availability of deployed systems. In *2006 International Symposium on Empirical Software Engineering*, page to appear, Rio de Janeiro, Brazil, September 21-22 2006. ACM Press.

[11] 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.

[12] 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.

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

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

[15] 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.

[16] Audris Mockus, Ping Zhang, and Paul Li. Drivers for customer perceived software quality. In *ICSE 2005*, St Louis, Missouri, May 2005. ACM Press.

[17] Annie Ying, Gail Murphy, Raymond Ng, and Mark Chu-Carroll. Predicting source code changes by mining change history. *IEEE Transactions of Software Engineering*, 30(9), 2004.

[18] Thomas Zimmermann, Peter Weissgerberv, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. *IEEE Transactions of Software Engineering*, 30(9), 2004.

Bio

Audris Mockus
Avaya Labs Research
233 Mt. Airy Road
Basking Ridge, NJ 07920
ph: +1 908 696 5608, fax:+1 908 696 5402
http://mockus.org, mailto:audris@mockus.org,
picture:http://mockus.org/images/small.gif

Audris Mockus conducts research of complex dynamic systems. He designs data mining methods to summarize and augment the system evolution data, interactive visualization techniques to inspect, present, and control the systems, and statistical models and optimization techniques to understand the systems. Audris Mockus received B.S. and M.S. in Applied Mathematics from Moscow Institute of Physics and Technology in 1988. In 1991 he received M.S. and in 1994 he received Ph.D. in Statistics from Carnegie Mellon University. He works at Software Technology Research Department of Avaya Labs. Previously he worked at Software Production Research Department of Bell Labs.