

Software Changes and Software Engineering: Why Not?

A. Mockus



audris@mockus.org

Avaya Labs Research

Basking Ridge, NJ 07920

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

The Specifications

- ❖ Provide a list of “don’t do it!” for software measurement based on software change repositories
- ❖ Influence a young generation of researchers

The Result

- ❖ It was too boring even to think about such task
- ❖ Smart people learn from other's mistakes, stupid learn from their own, the rest are ...
- ❖ Can I learn from my own experiences and you from other's?
 - ❖ Report personal work/review experiences
 - ❖ Critique own work (mostly)
 - ❖ To be fair annotated/expanded previous talks

Don't do it

- ❖ Irrelevant topic
- ❖ Overly specialized results
- ❖ Gross mistakes

Outline

- ❖ Background
- ❖ The no-nos
- ❖ The end

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 created by making changes to it
 - ❖ A delta is a single checkin (ci/commit/edput) representing an atomic modification of a single file with following attributes
 - ❖ File, Date, Developer, Comment
 - ❖ Other attributes that often can be derived:
 - ❖ Size (number of lines added,deleted)
 - ❖ Lead time (interval from start to completion)
 - ❖ Purpose (Fix/New)
- ❖ 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 rating, customer purchase patterns, 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

- ❖ It is tempting to model things that are easy to measure
 - ❖ Counts, trends, patterns
- ❖ It is tempting to try topics that are well formulated
 - ❖ Which modules will get defects

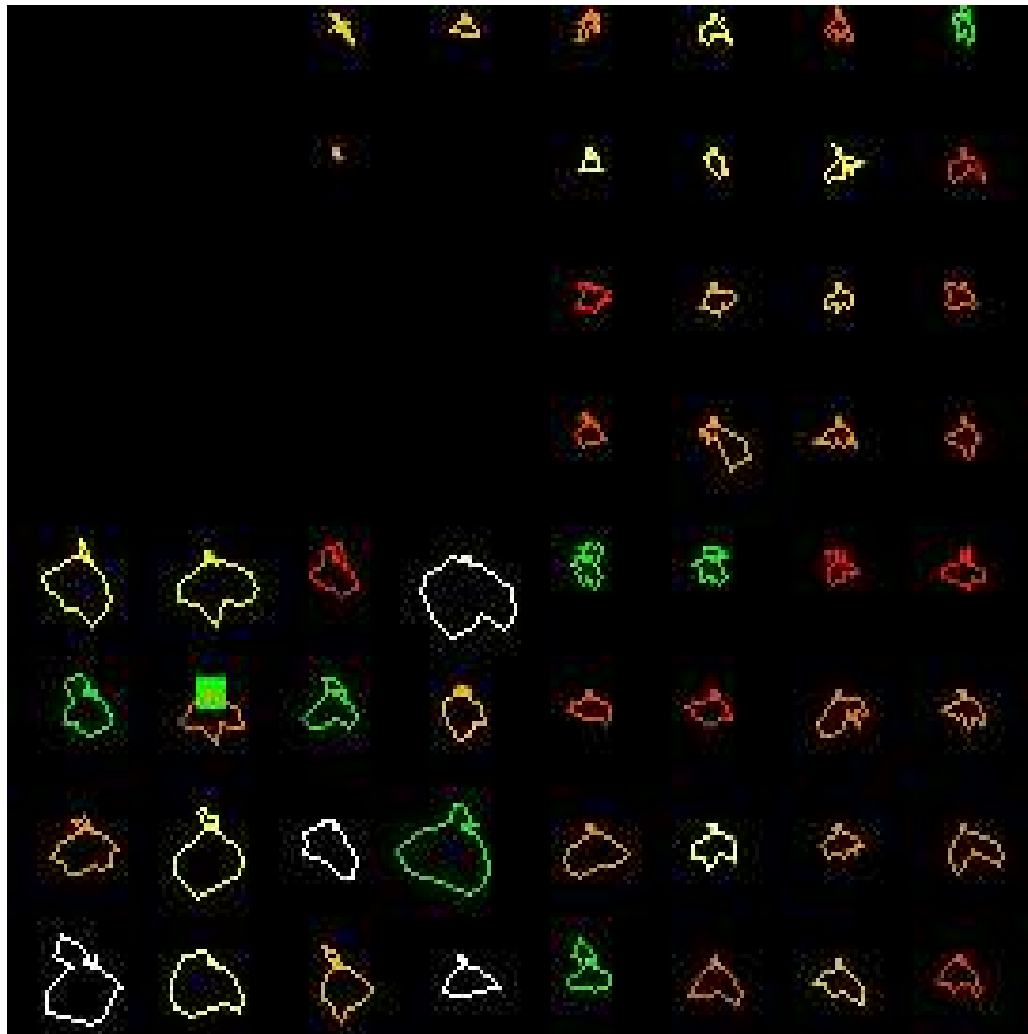
Trends

Number of changes, lines added, deleted, unchanged over the years

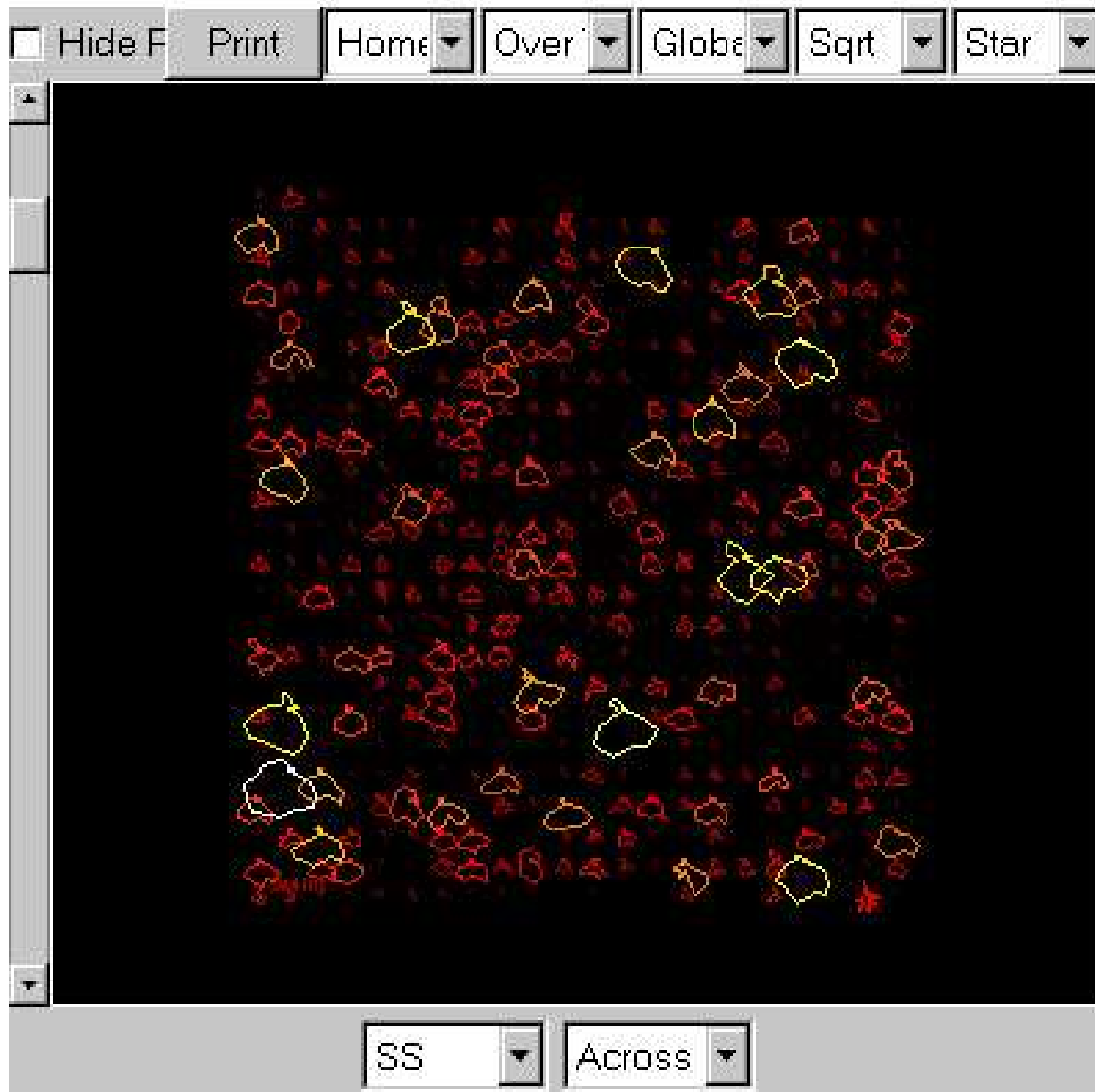
Cases	Count	Added	Deleted	Same
84	1	0	0	9
85	1855	26.3574	4.2814	176.177
86	11919	21.2353	9.56196	303.107
87	14633	32.0191	11.7993	655.749
88	10794	18.1634	7.94191	1009.2
89	19819	17.6601	6.80292	1717.51
90	12533	17.8687	5.85016	2609.18
91	12036	16.5215	5.32635	3336.27
92	12112	22.1412	8.92404	3338.02
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

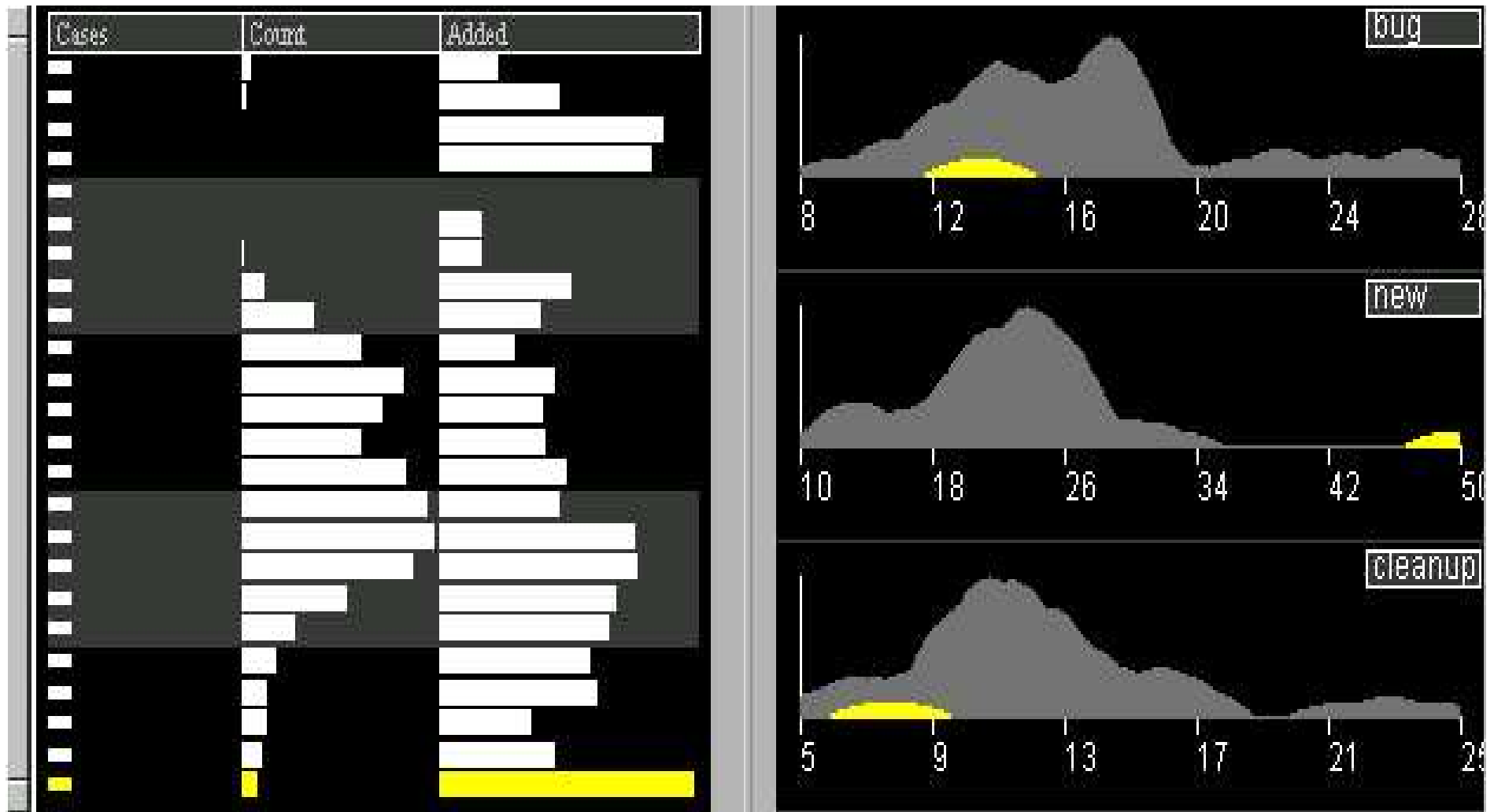


Patterns



Patterns II

Numbers of changes and lines added by hour and type



Module Summaries

The subsystem **sys**

Demographics

modules	files	lines	Average lengths of files of various types											
12	512	168100	c	md	h	L	G	ees	c	md	h	L	G	ees
Files	150	10	50	300	10	2	1000.25	500	50.6	40.9	87	29.5		
Lines	150000	5000	2500	10000	500	100								

Change Summaries

There are 4000 MR-module combinations. The modules are described in the following view:

name	files	filenames	deltas	lines	files.c	files.md	files.h
...

Regression models

Deltas per MR: $deltas = 4.0 mrs + 1.9 * mrs * epsilon$

Standard error of slope: 0.993. Extreme residuals:

sys/module3	sys/module4	sys/module1	sys/module5	sys/module6
-1.4	-1.2	-0.88	-0.7	-0.59
sys/module7	sys/module8	sys/module9	sys/module10	sys/module11
-0.7	-0.59	-0.49	0.2	5.6

Developers per line: $(1 + \log_{10} lines) = 0.068 (1 + lines)^{(0.75)} \exp(0.6 * epsilon)$

$\exp(\text{Standard error of } \log(\text{slope}))$: 2.5

Standard error of power: 0.10. Extreme residuals:

sys/module11	sys/module9	sys/module3	sys/module10	sys/module7
-0.55	-0.43	-0.36	-0.33	-0.24
sys/module8	sys/module5	sys/module4	sys/module1	sys/module6
-0.33	-0.24	0.36	0.37	1.2

Fault Potential

Total number of modules involved: 8

Total number of (fault imr, module) combinations: 250

Deviance for naive model: 9

Null deviance: 360 on 7 degrees of freedom

Residual deviance: 30 on 5 degrees of freedom

	Value	Std. Error	t value
(Intercept)	-0.35	0.40	-0.88
α deltas	0.67	0.05	13.60
age	-0.03	0.10	-0.30

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
 - ❖ 50% of the faults occur in 20% of the modules
 - ❖ Unfortunately, these 20% of the modules contain 60% of the code!?

Some models of software changes

- ❖ Quality: model of customer experience [11, 14]
- ❖ Effort: estimate interval and benchmark process
 - ❖ What makes some changes hard and long [6]
 - ❖ What processes/tools work and why [2, 3]
 - ❖ How do you create a hybrid OSS/Commercial process [9, 5]
- ❖ Estimation: predict project repair effort from planned new features
 - ❖ Plan for field problem repair after the release [13, 14]
 - ❖ Release readiness criteria [13, 8]

Some development support tools

- ❖ Finding relevant people [10]
- ❖ Finding related defects [4]
- ❖ Finding related changes [1, 15, 7]
- ❖ Finding independently maintainable pieces of code [12]

Real-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

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

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?

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

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 behaviour 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?

References

- ~~[1] Raymond Ng, Annie Ying, Gail Murphy and Mark Chu-Carroll. Predicting source code changes by mining change history. *IEEE Transactions of Software Engineering*, 30(9), 2004.~~
- [2] 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.
- [3] 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.
- [4] D. Cubranic and G.C Murphy. Hipikat: A project memory for software development. *TSE*, 31(6), 2005.
- [5] T Dinh-Trong and Bieman J.M. Open source software development: A case study of freebsd. *IEEE Transactions of Software Engineering*, 31(6), 2005.
- [6] James D. Herbsleb, Audris Mockus, Thomas A. Finholt, and Rebecca E. Grinter. An empirical study of global software development: Distance and speed. In *23rd International Conference on Software Engineering*, pages 81–90, Toronto, Canada, May 12-19 2001.
- [7] 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.
- [8] 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.

- [9] 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.
- [10] 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.
- [11] Audris Mockus and David M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, April–June 2000.
- [12] Audris Mockus and David M. Weiss. Globalization by chunking: a quantitative approach. *IEEE Software*, 18(2):30–37, March 2001.
- [13] 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.
- [14] Audris Mockus, Ping Zhang, and Paul Li. Drivers for customer perceived software quality. In *ICSE 2005*, St Louis, Missouri, May 2005. ACM Press.
- [15] Thomas Zimmermann, Peter Weissgerberv, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes.

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.