Using Software Changes to Understand and Improve Software Projects

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

Outline

- Background
  - Motivation
  - Software project repositories
  - How to use change data
- Software project issues
  - Appropriate levels of test coverage
  - Expertise transfer and distributed development
  - Customer-perceived quality
- Discussion

Motivation

- To quantify software production: make informed trade-offs among schedule, quality, and cost
  - Visibility: where/when effort is spent, defects introduced
  - Predictability: what will be the impact of choosing technology, processes, organization
  - Controllability: trade-offs between time to market, features, quality, and staffing

Approach

- Observe development through digital traces it leaves in source code changes, problem reporting/resolution, and recorded communications
  - Summarize actual software development (extract existing practices, effort, and quality)
  - Obtain tacit knowledge of successful individuals/teams/projects
  - Detect patters and relationships (laws) that can not be observed from by individual participants in software projects

Software changes

- Software is created by changes
- Changes are tracked

Before:

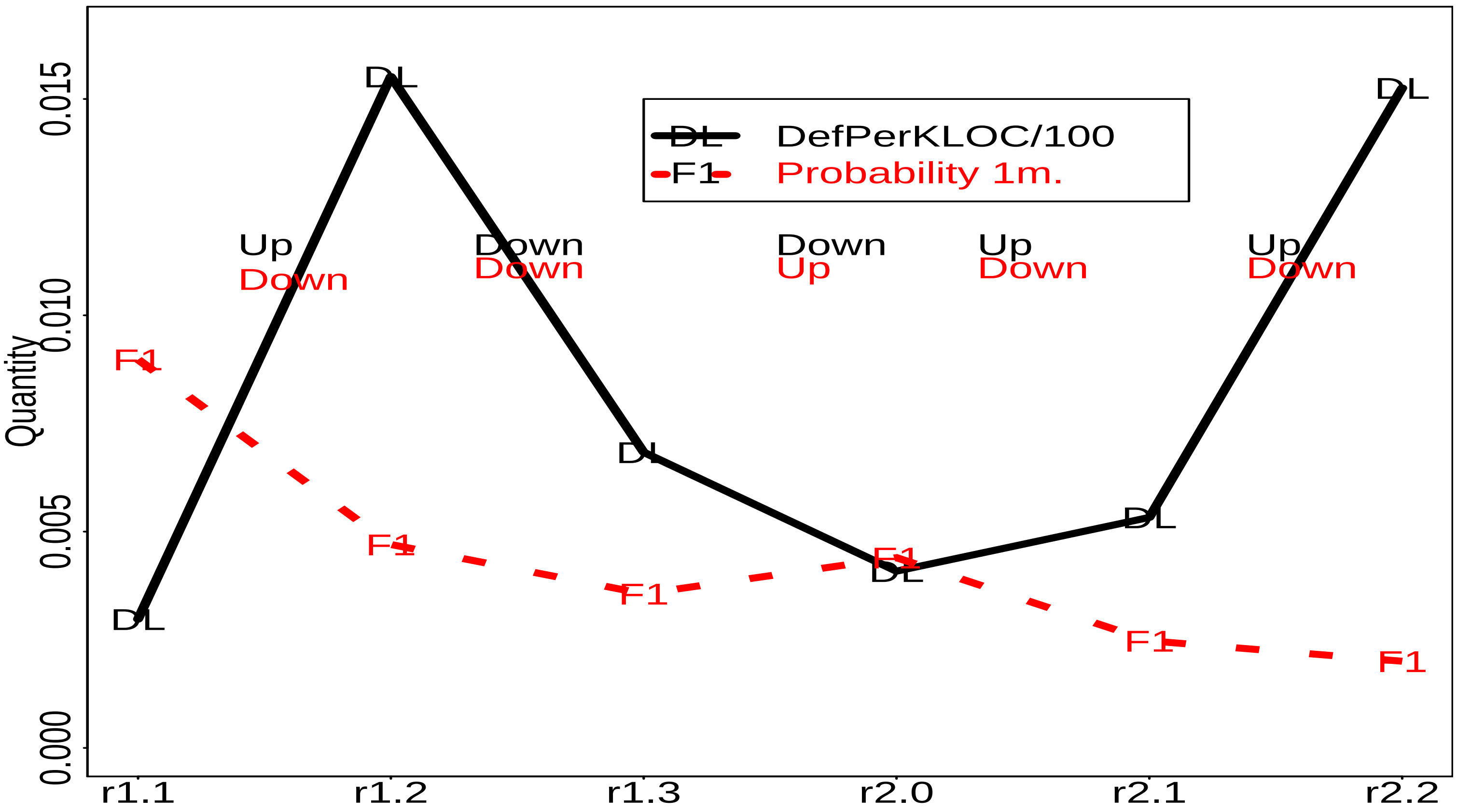
```


int i = n;
while(i++)
    prinf(” %d”, i−−);
```

After:

```
//print n integers
int i = n;
while(i++ && i > 0)
    prinf(” %d”, i−−);
```

- one line deleted
- two lines added
- two lines unchanged
- Many other attributes: date, developer login, defect number, . . .

Defect density and probability that a customer will observe a defect?



How to compare releases?

Change Data Methodology: Extraction

- Get access to the systems
- Extract raw data
  - change table, developer table. (SCCS: prs, cleartool -lsh, cvs log, svn log, git log, hg 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

Change Data 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 [15], effort [1], risk [17]
- Validate and clean recorded and modeled data
- Iterate

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

Pitfalls of Using Project Repositories

- Different process: how work is broken down into work items may vary across projects
- Different tools: CVS, ClearCase, SCCS, svn, git, hg, bzr, ...
- 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

Change Data Methodology: Project Sample

- *Languages*: Java, C, SDL, C++, JavaScript, XML, ... *Platforms*: proprietary, unix'es, Windows, VXWorks, *Domains*: embedded, high-availability, network, user interface *Size*: from largest to small

Type	Added KLines	KDelta	Years	Developers	Locations
Voice switching software	140,000	3,000	19	6,000	5
Enterprise voice switching	14,000	500	12	500	3
Multimedia call center	8,000	230	7	400	3
Wireless call processing	7,000	160	5	180	3
Web browser	6,000	300	3	100/400	
OA&M system	6,000	100	5	350	3
Wireless call processing	5,000	140	3	340	5
Enterprise voice messaging	3,000	87	10	170	3
Enterprise call center	1,500	60	12	130	2
Optical network element	1,000	20	2	90	1
IP phone with WML browser	800	6	3	40	1
Web sever	200	15	3	15/300	

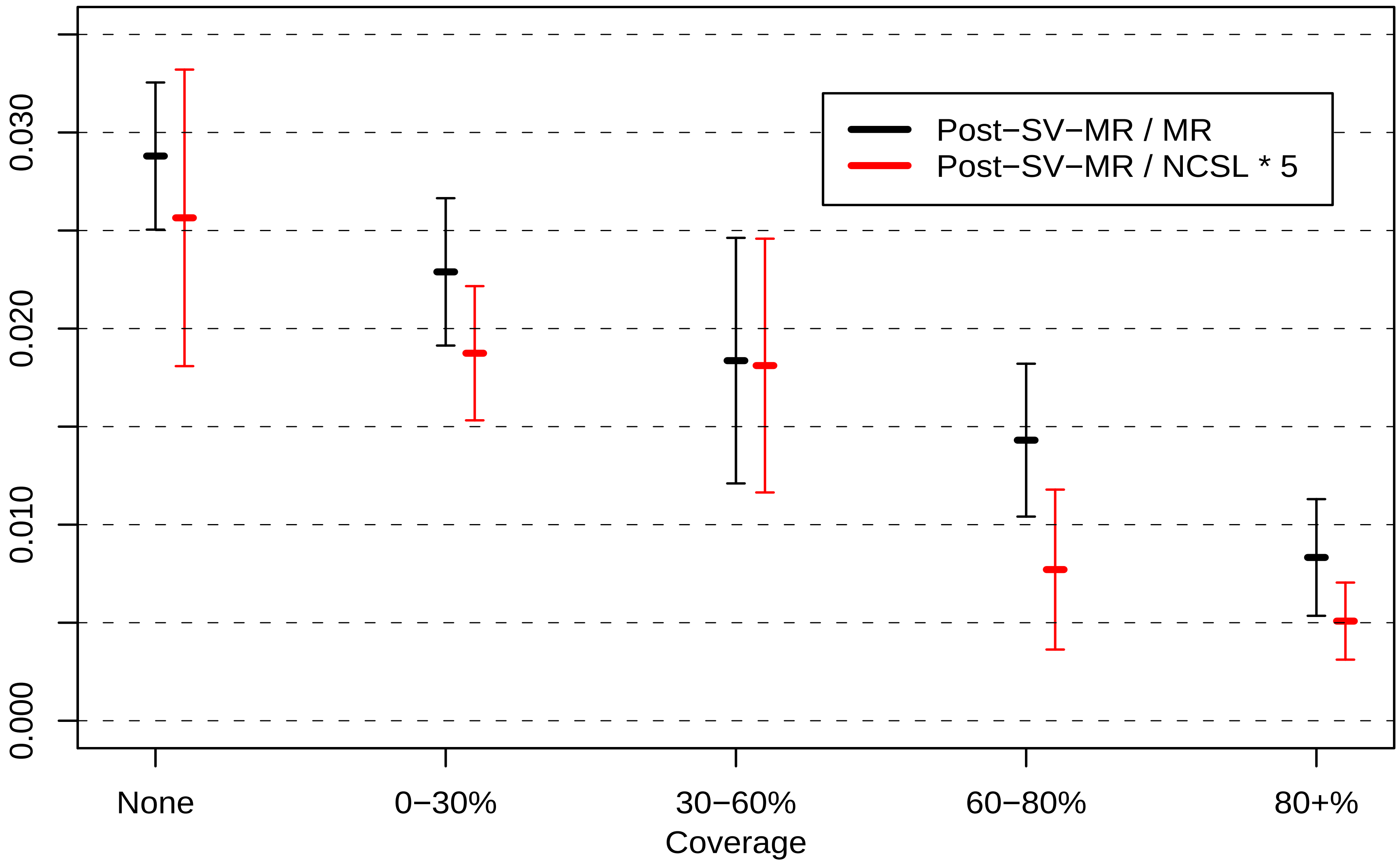
Existing Models

- Predicting the quality of a patch [17]
- Globalization: move development where the resources are:
  - What parts of the code can be independently maintained [18]
  - Who are the experts to contact about any section of the code [13]
  - Mentorship and learning [11, 21]
- Effort: estimate MR effort and benchmark process
  - What makes some changes hard [7, 6, 10]
  - What processes/tools work [1, 2, 4, 14]
  - What are OSS/Commercial process differences [12]
- Project models
  - **Release schedule** [8, 19, 5]
  - Release quality/availability [3, 16, 9, 20]

Code coverage

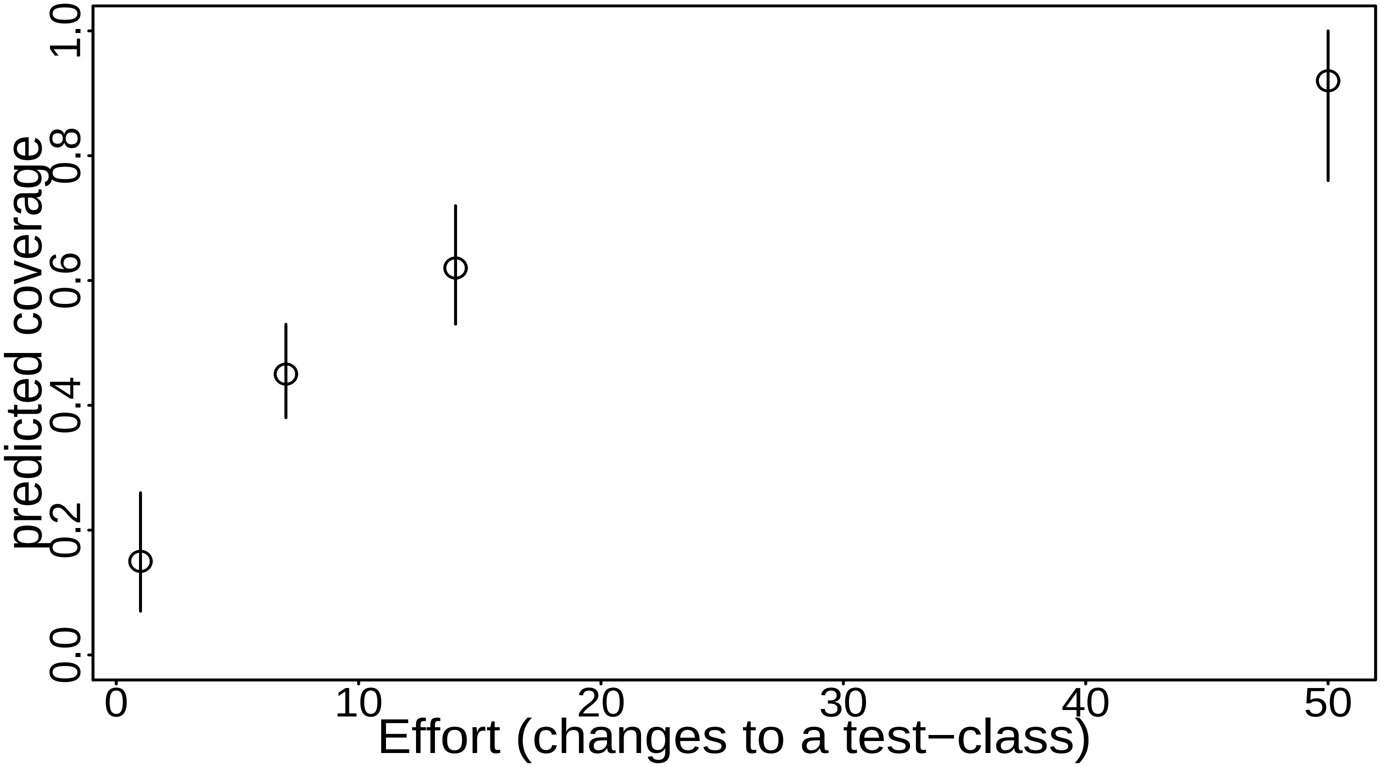
- How much code coverage is needed?
  - Increasing the level of coverage would decrease the defect rate.
  - Progressively more effort is needed to increase the coverage by the same amount for higher levels of coverage.
- Project
  - New real-time call center reporting system (1M lines)
  - Mostly in Java, uses Cobertura to report coverage
  - Post-release defects used as a measure of quality
  - ClearCase used for version control, ClearQuest (CQ) for problem tracking, each checkin comment includes MR number
  - Customer reported CQ's (Post-SV-MRs) can be identified

Increasing coverage decreases defect rate



Increasing coverage demands exponentially increasing effort

Predicted levels of coverage for different numbers of changes to the test class and median Fan-Out of 7.



Code as functional knowledge: implications

- Developers are *transient*, but the code is *everlasting*
- Developers can *only* leave a lasting impact
  - through *changes* to the code and
  - through *training* developers who succeed them
- Traces developers leave in the code shed light on how software is created and how developers interact
- **What happens in a succession: when developers change, but code stays?**

Context and data sources

- All software projects in Avaya (85 MNCSL)
- Version control systems
  - Chronology: 1990 until present, varies with project
  - Attributes: login, date, file
- People: organizational Directory (LDAP) snapshots
  - Attributes: personal ID, supervisor ID, department, location, phone, email
  - Chronology: 2001 until present
- People to login maps

Succession

- *Definition: Implicit* or *virtual teams* are relationships among individuals based on the affinity to the parts of the product they are working or have worked on.
- *Definition: Succession* is a relationship between individuals within the implicit teams reflecting the transfer of responsibilities to maintain and enhance the product. There are various types of succession: here we are concerned with offshoring and refer to receiving party as *followers* and to the transferring party as *mentors*. Note that, in general, followers and mentors do not need to communicate with each other.
  - The chronological order of engagements by *mentors* and *followers* should be reflected in the temporal order of changes
- **How succession affects developer productivity?**

The ratio of follower and mentor productivity

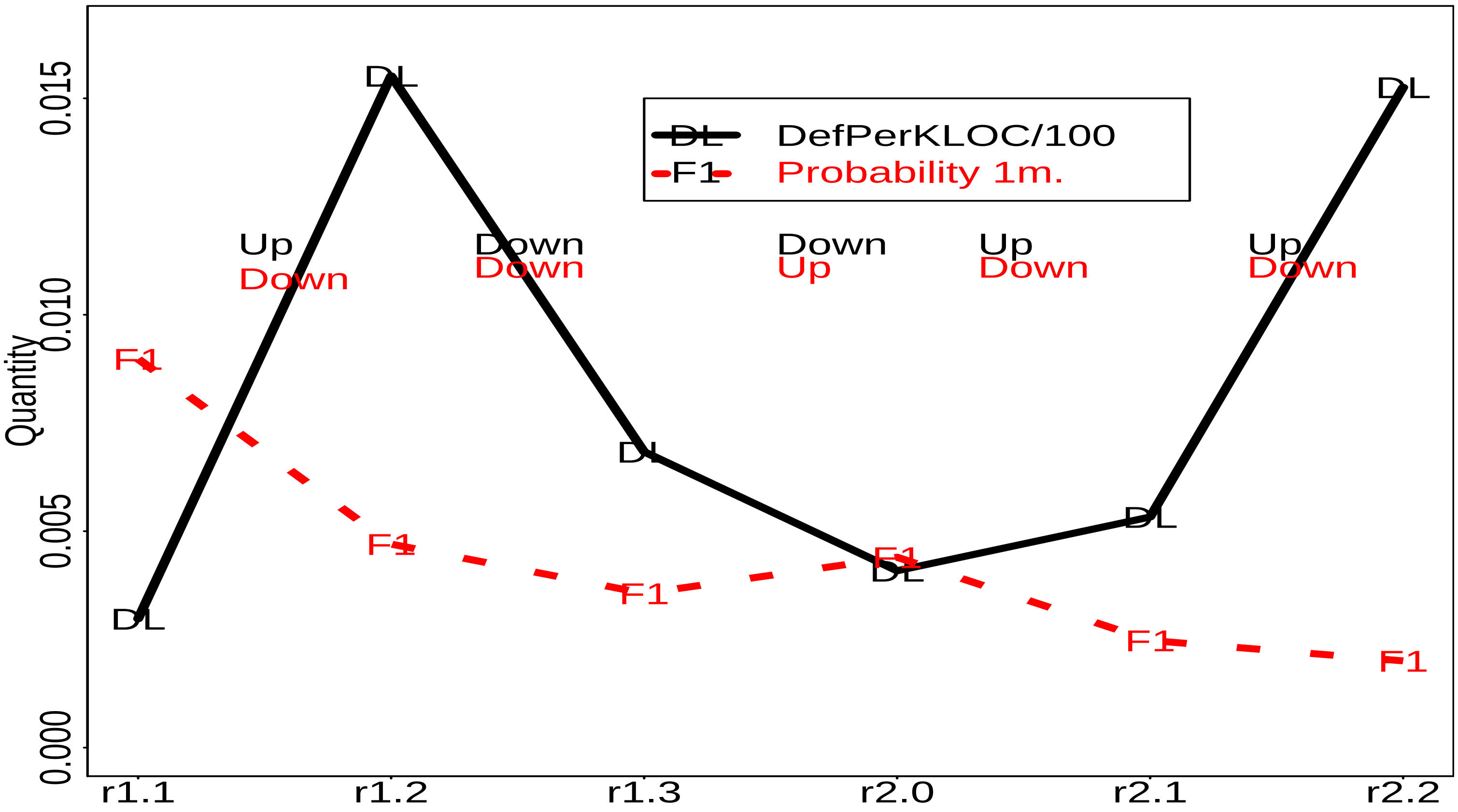
	Estimate	95%CI
Offshoring	$\frac{1}{2}$	$[0.42, 0.67]$
Primary expertise	$\frac{1}{2}$	$[0.40, 0.64]$
Expertise breadth	$\frac{1}{2}$	$[0.38, 0.64]$
Large prod.	$\frac{1}{3}$	$[0.21, 0.42]$
Medium prod.	$\frac{2}{3}$	$[0.52, 0.77]$
$\ln(No.Followers)$	$\frac{1}{\sqrt{No.Followers}}$	

Table 1: $\log(Ratio) = Offshore + Primary + Breadth + Size + \log(NFollow)$. 1012 mentor-follower pairs. Adj-R^2 $= 59$.

Customer and developer perceptions of quality

- Development team found that defect density does not reflect customer perception of quality in subsequent software releases
  - What measure reflects customer perceptions of software quality?
  - Can it be communicated in terms that are meaningful to the development team?
- Context: major projects in Avaya
- Version control: SCCS, ClearCase, SubVersion
- Problem tracking: various
- Customer deployment and issue reporting systems

Defect density and probability that a customer will observe a defect?



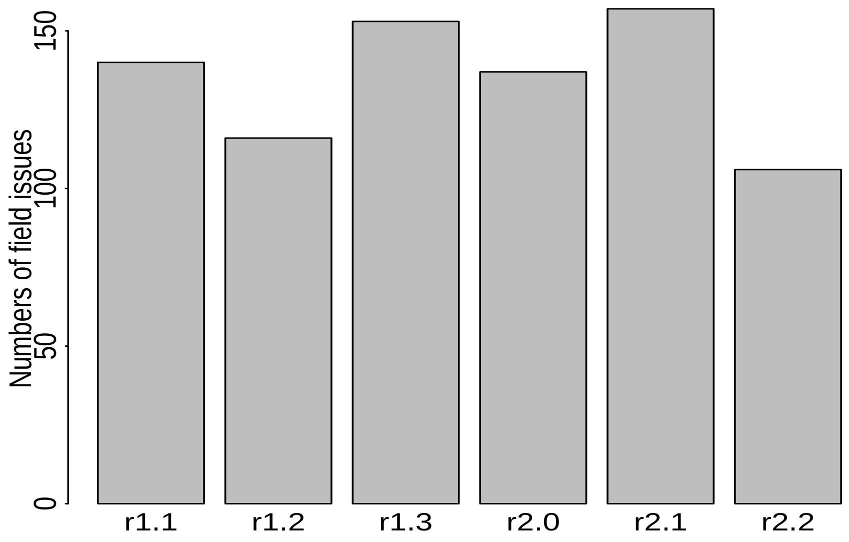
Anti-correlated?!

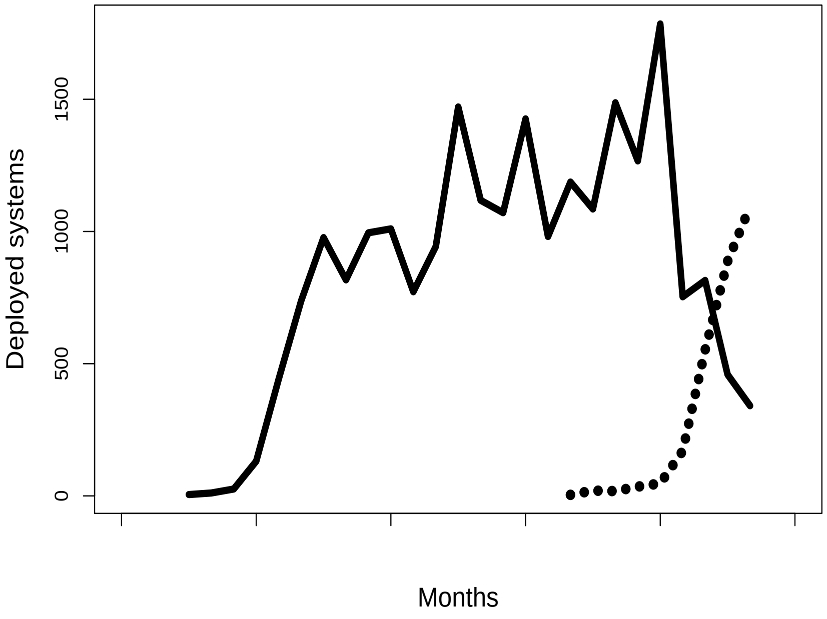
High defect density leads to satisfied customers?

- What does any organization strive for?

Stability $\Longrightarrow$ Predictability!

- The rate at which customer problems get to Tier IV is almost constant despite highly varying deployment and failure rates

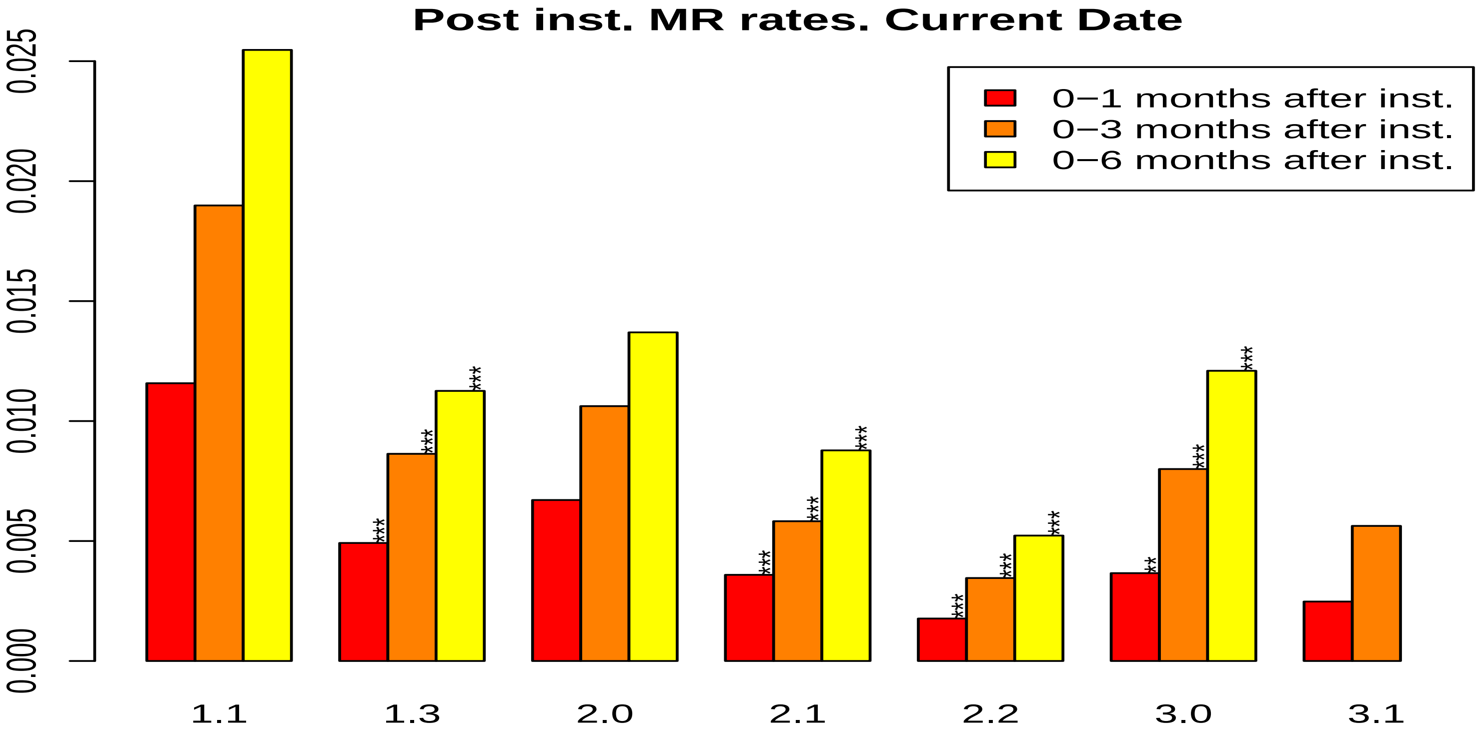




Major versus Minor releases

- Defect density numerator is about the same as for IQ because
  - Major releases are deployed more slowly to fewer customers
  - For minor releases a customer is less likely to experience a fault so they are deployed faster and to more customers
- The denominator diverges because
  - Major releases have more code changed and fewer customers
  - Minor releases have less code and more customers

Customer Quality



- Fraction of customers that report software failures within the first few months of installation
- Does not account for proximity to launch, platform mix
- Significant differences marked with "*"
- "We live or die by this measure"

Change data $\Longrightarrow$ new insights

- Methodology
  - Change data analysis brings new insights
  - Results become an integral part of development practices — continuous feedback on production changes/improvements
- Insights
  - Development process view does not represent customer views
  - A manner of succession affects productivity: start offshoring with simpler projects, pick appropriate mentors
  - While increasing coverage decreases defects, effort needed to achieve very high levels may be prohibitive
- Measurement hints
  - Pick the right measure for the objective — no single "quality" exists
  - Adjust for relevant factors to avoid measuring demographics

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] Marcelo Cataldo, Audris Mockus, Jeffrey A. Roberts, and James D. Herbsleb. Software dependencies, the structure of work dependencies and their impact on failures. *IEEE Transactions on Software Engineering*, 2009.

[4] 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.

[5] J. D. Herbsleb and A. Mockus. An empirical study of speed and communication in globally-distributed software development. *IEEE Transactions on Software Engineering*, 29(6):481–494, June 2003.

[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] 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. Empirical estimates of software availability of deployed systems. In *2006 International Symposium on Empirical Software Engineering*, pages 222–231, Rio de Janeiro, Brazil, September 21-22 2006. ACM Press.

[10] Audris Mockus. Organizational volatility and developer productivity. In *ICSE Workshop on Socio-Technical Congruence*, Vancouver, Canada, May 19 2009.

[11] Audris Mockus. Succession: Measuring transfer of code and developer productivity. In *2009 International Conference on Software Engineering*, Vancouver, CA, May 12–22 2009. ACM Press.

[12] 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.

[13] 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.

[14] Audris Mockus, Nachiappan Nagappan, and T Dinh-Trong, Trung. Test coverage and post-verification defects: A multiple case study. In *International Conference on Empirical Software Engineering and Measurement*, Lake Buena Vista, Florida USA, October 2009. ACM.

[15] 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.

[16] Audris Mockus and David Weiss. Interval quality: Relating customer-perceived quality to process quality. In *2008 International Conference on Software Engineering*, pages 733–740, Leipzig, Germany, May 10–18 2008. ACM Press.

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

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

[19] 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.

[20] Audris Mockus, Ping Zhang, and Paul Li. Drivers for customer perceived software quality. In *ICSE 2005*, pages 225–233, St Louis, Missouri, May 2005. ACM Press.

[21] Minghui Zhou, Audris Mockus, and David Weiss. Learning in offshored and legacy software projects: How product structure shapes organization. In *ICSE Workshop on Socio-Technical Congruence*, Vancouver, Canada, May 19 2009.

Abstract

Software systems are created and maintained by making changes to their source code. Therefore, understanding the nature and relationships among changes and their effects on the success of software projects is essential to improve software engineering. Using methods and tools to retrieve, process, and model data from ubiquitous change management systems we have gained insights into the relationships among properties of the software product, the way it is constructed, and outcomes, including quality, effort, and interval. We illustrate how measures and models of changes can lead to a better understanding of a software project. In particular, we analyze the relationship between the test coverage and customer reported defects, the transfer of code and its impact on developer productivity, and the divergence of developer and customer view of software quality. We find that increases in test coverage are related to lower defect density, but reaching high levels of coverage requires exponentially more effort. Transfer of code ownership reduces developer productivity and in larges projects the productivity may be reduced up to three times while in offshoring up to two times as compared to the original developers. Finally, customer perception of software quality represented by the fraction of customers that experience and report software defects, is not related to a simple-to-compute measure of defect density commonly used to assess the quality of a software projects.

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.