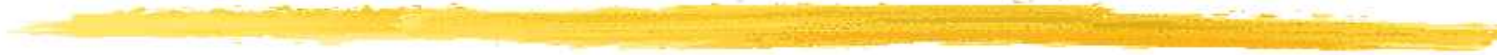


Using Software Changes to Understand and Improve Software Projects



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

Outline

- ❖ Background
 - ❖ Motivation
 - ❖ Software project repositories
 - ❖ How to use change data
- ❖ Software project issues
 - ❖ Developer learning in offshoring and outsourcing
 - ❖ 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

Basics: software changes

- ❖ Developers **create** software by changes
- ❖ All changes **are recorded**

Before:

```
int i = n;  
while(i++)  
    printf(" %d", i--);
```

After:

```
//print n integers  
int i = n;  
while(i++ && i > 0)  
    printf(" %d", i--);
```

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

Domain and method

❖ Science

- ❖ X is the study of *past human events and activities*
- ❖ Y is the study of human **cultures** through the *recovery, documentation and analysis of **material** remains*
- ❖ Z is the study of developer **cultures** and **behaviors** through the *recovery, documentation and analysis of **digital** remains*

❖ Method

- ❖ Tomography is image reconstruction from multiple projections
- ❖ Software change tomography is the reconstruction of behavior of developers from the digital traces they leave in the code and elsewhere

Basics: software quality

Software quality is the most:

- ✧ difficult
- ✧ expensive
- ✧ important

part of software development,

... even for Microsoft

Practice: how to compare software releases?

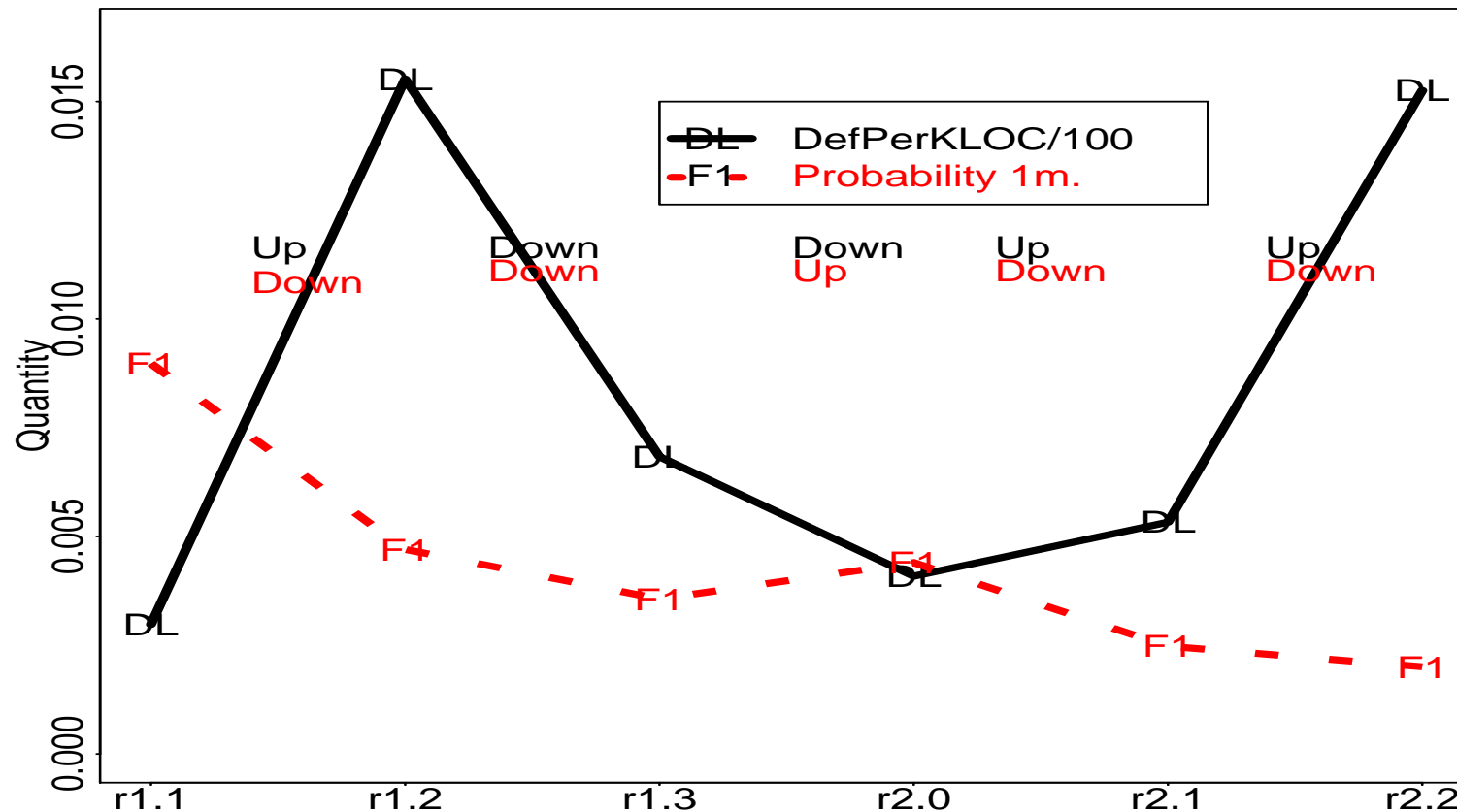
“**we tried to improve quality:** get most experienced team members to test, code inspections, root cause analysis, ...”

“**Did it work?** I.e., is this release better than previous one?”

Everyone uses **defect density** (e.g., customer reported defects per 1000 changes or lines of code), but “it **does not reflect** feedback from customers.”

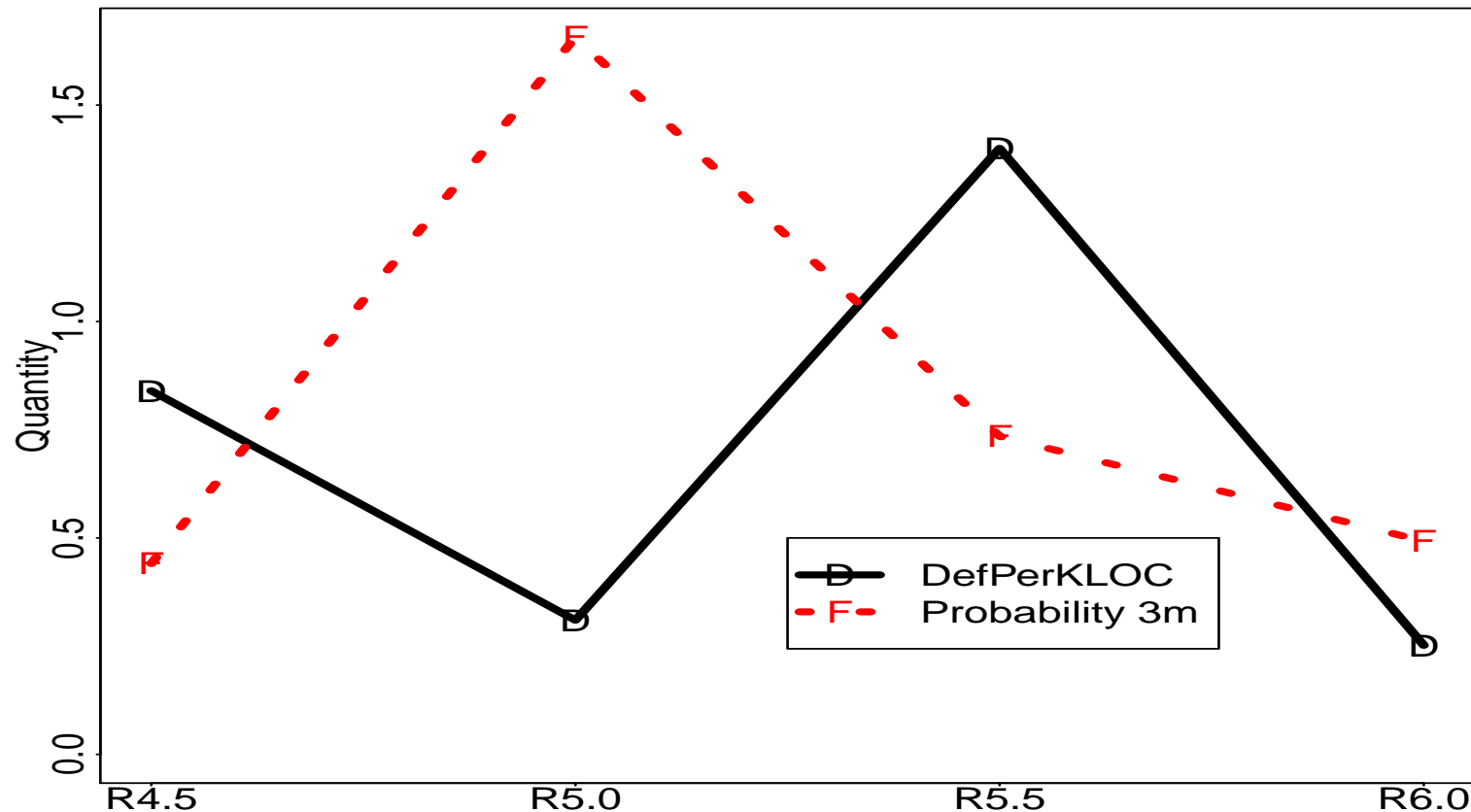
Ok, then lets measure the probability that *a customer will report a software defect*

A paradox: large telecom software



Does the **increase** in defect density make customers **more satisfied** and **decrease less satisfied**?

Is the paradox unique for this product?



A large product from another company:

Why does the **increase** in defect density make customers **satisfied**?

Change Tomography: Projections

- ❖ 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 Tomography: reconstructing the image

- ❖ 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]

Why Change Tomography?

- ❖ 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 Change Tomography

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

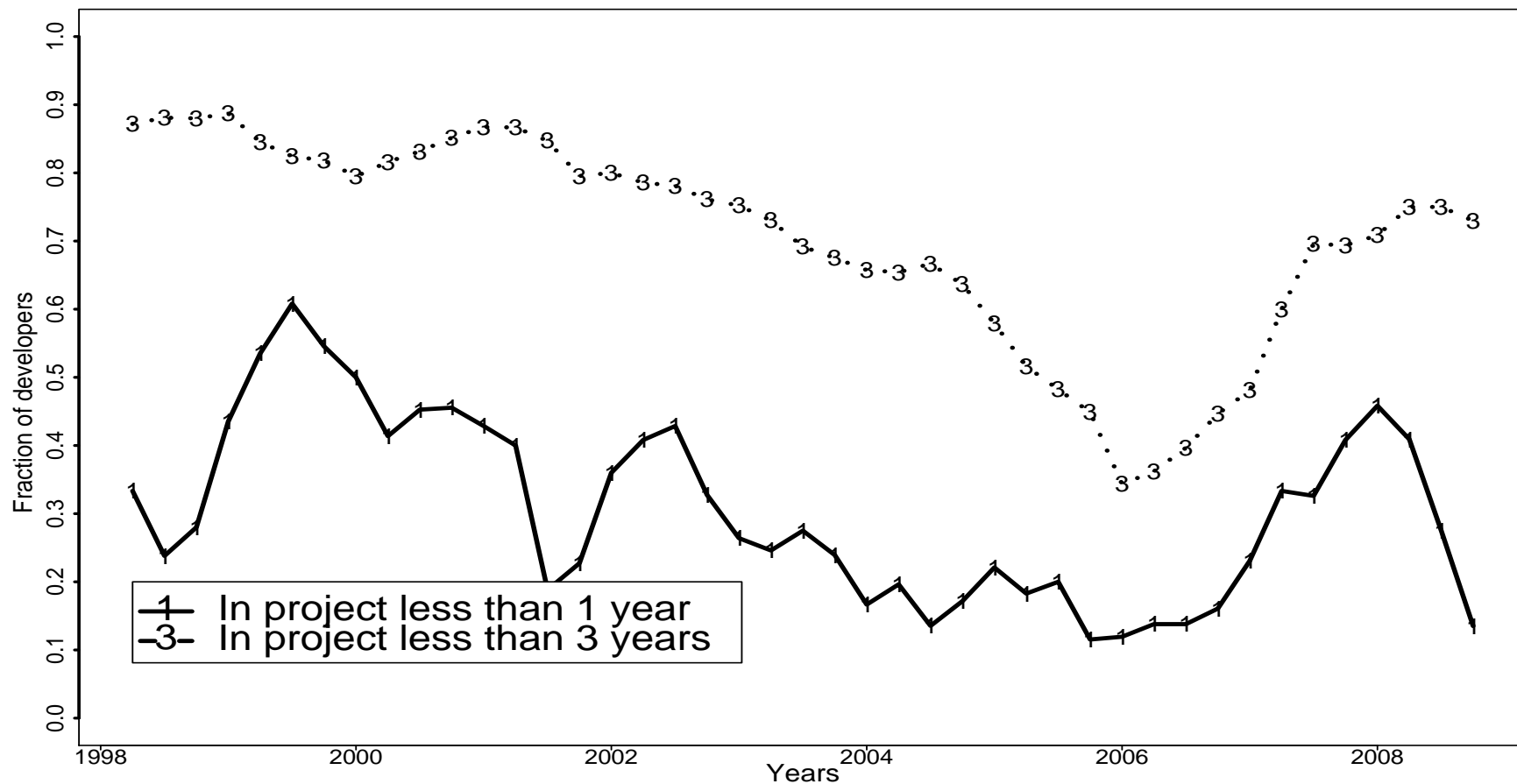
Change Tomography: Project Sample

❖ *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	

How fast developers learn?

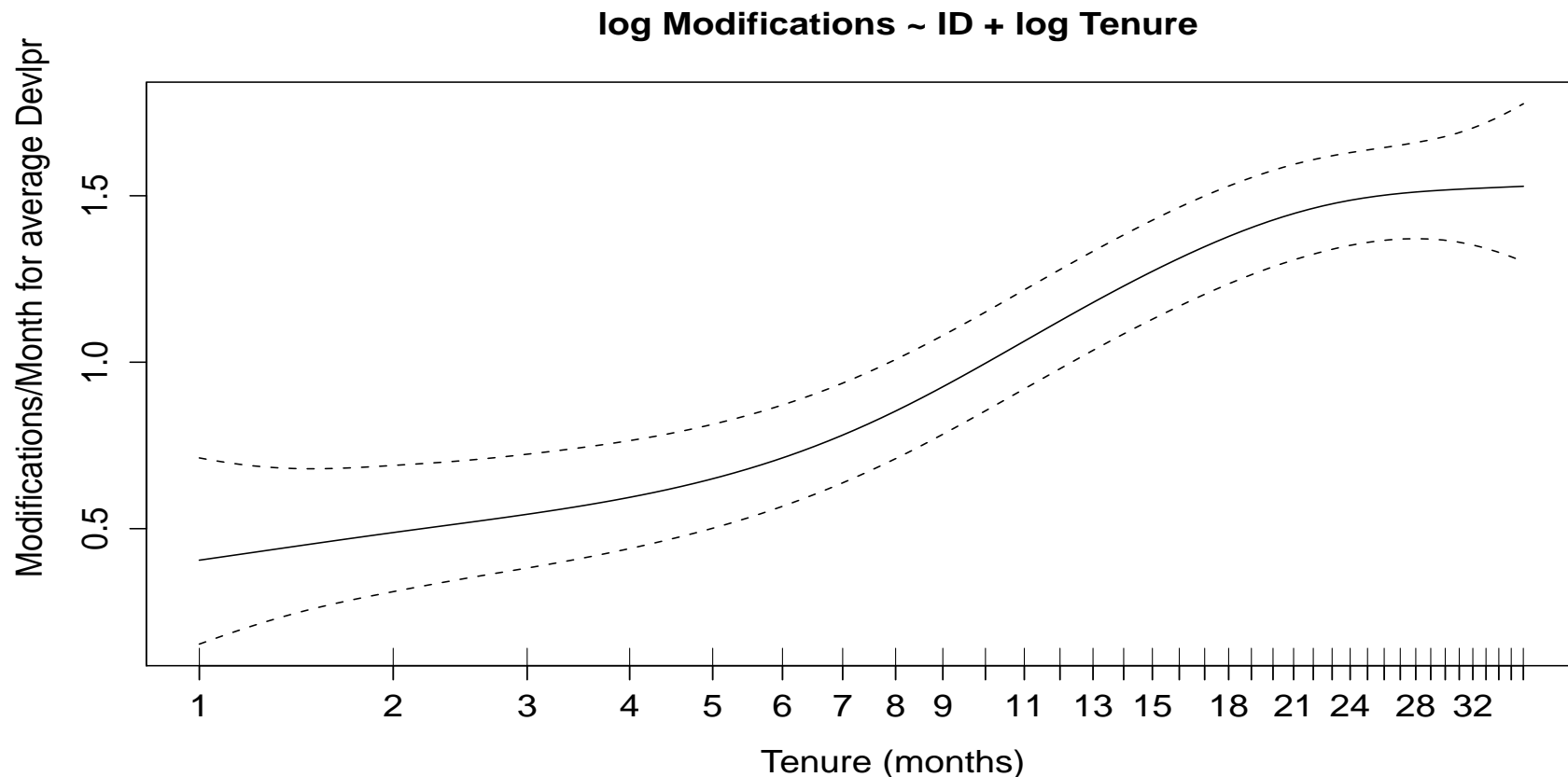
OFFSHORING/OUTSOURCING/RETIREMENT CHURN



A plateau?

“developers reach **full** productivity in **few** months.”

— a common response from managers and developers



Modifications per month versus Tenure

Fully productive, but...

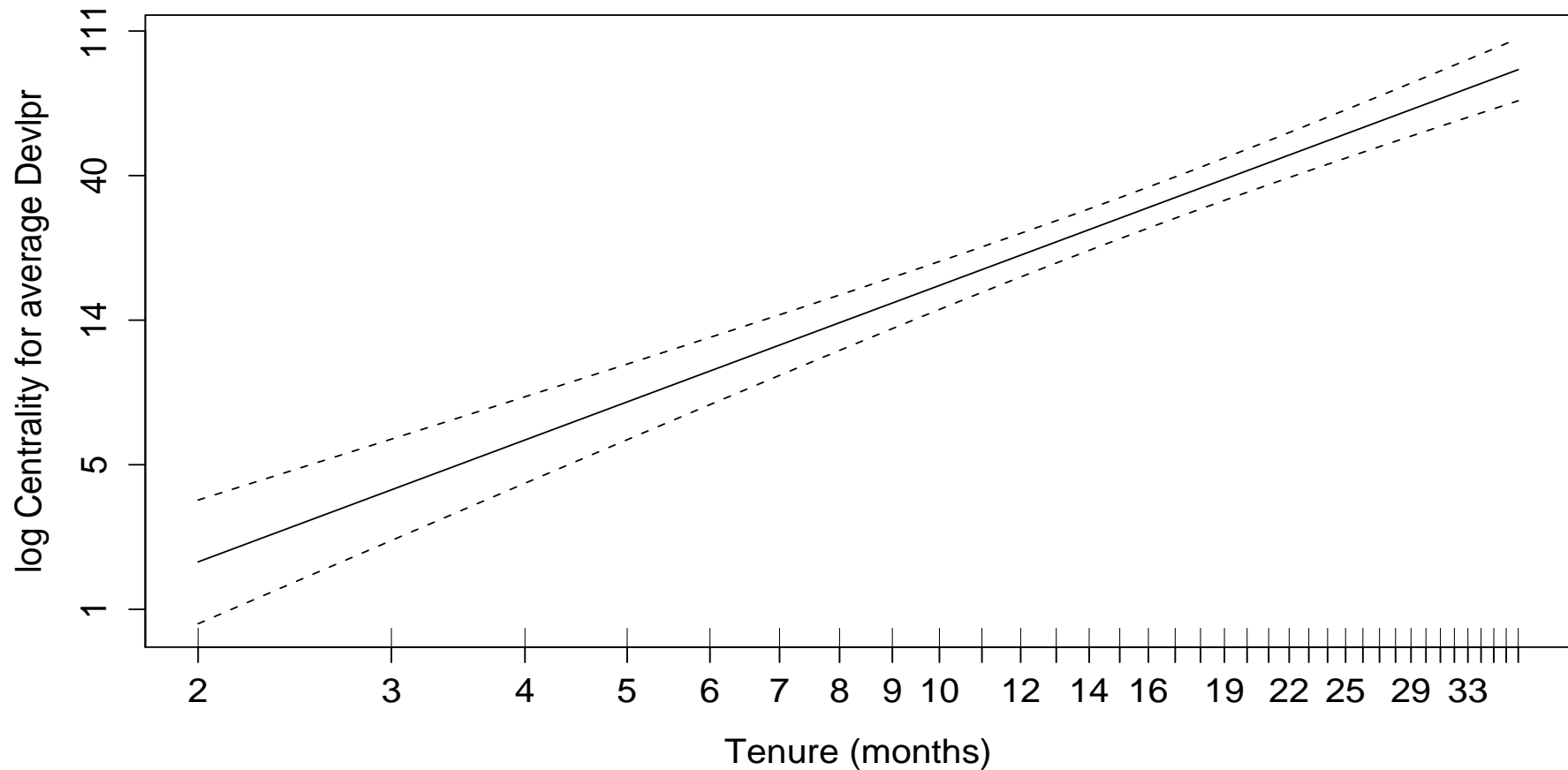
“We do not assign important tasks for developers that have been less than three years on a project.”

“We tried to do that after two years, but it **did not work** well.”

— Senior architect

Tasks importance keeps increasing!

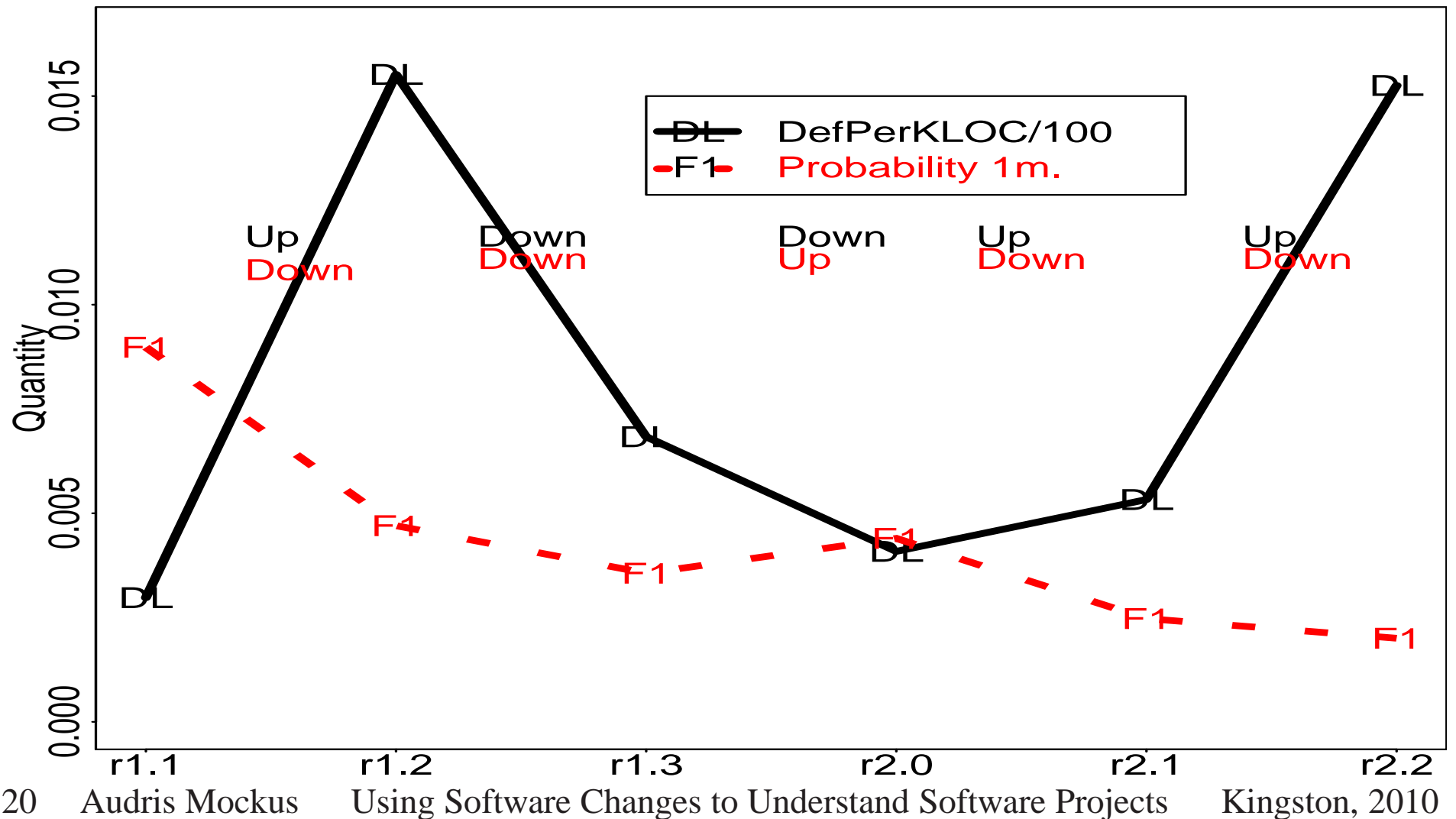
log Centrality \sim ID + log Tenure



Average task centrality versus Tenure

Back to the first paradox..

High defect density satisfies customers?



High defect density leads to satisfied customers?

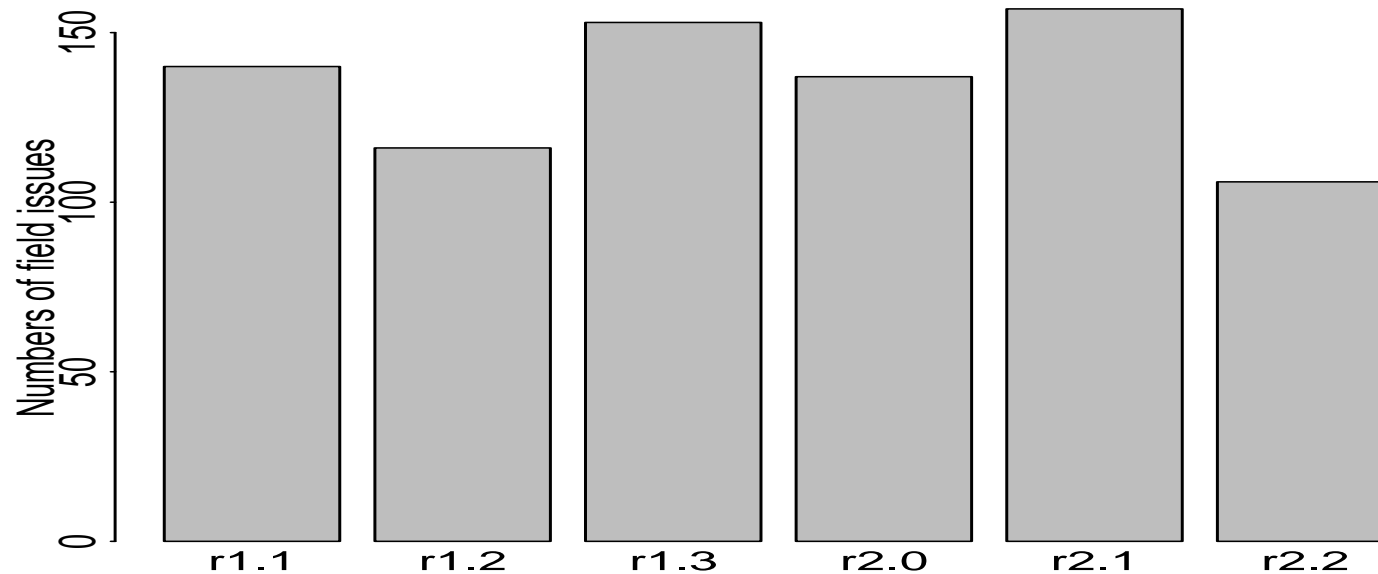
What does any organization strive for?

High defect density leads to satisfied customers?

What does any living being strive for?

Stability \implies Predictability!

The **rate** at which customer problems get to developers is

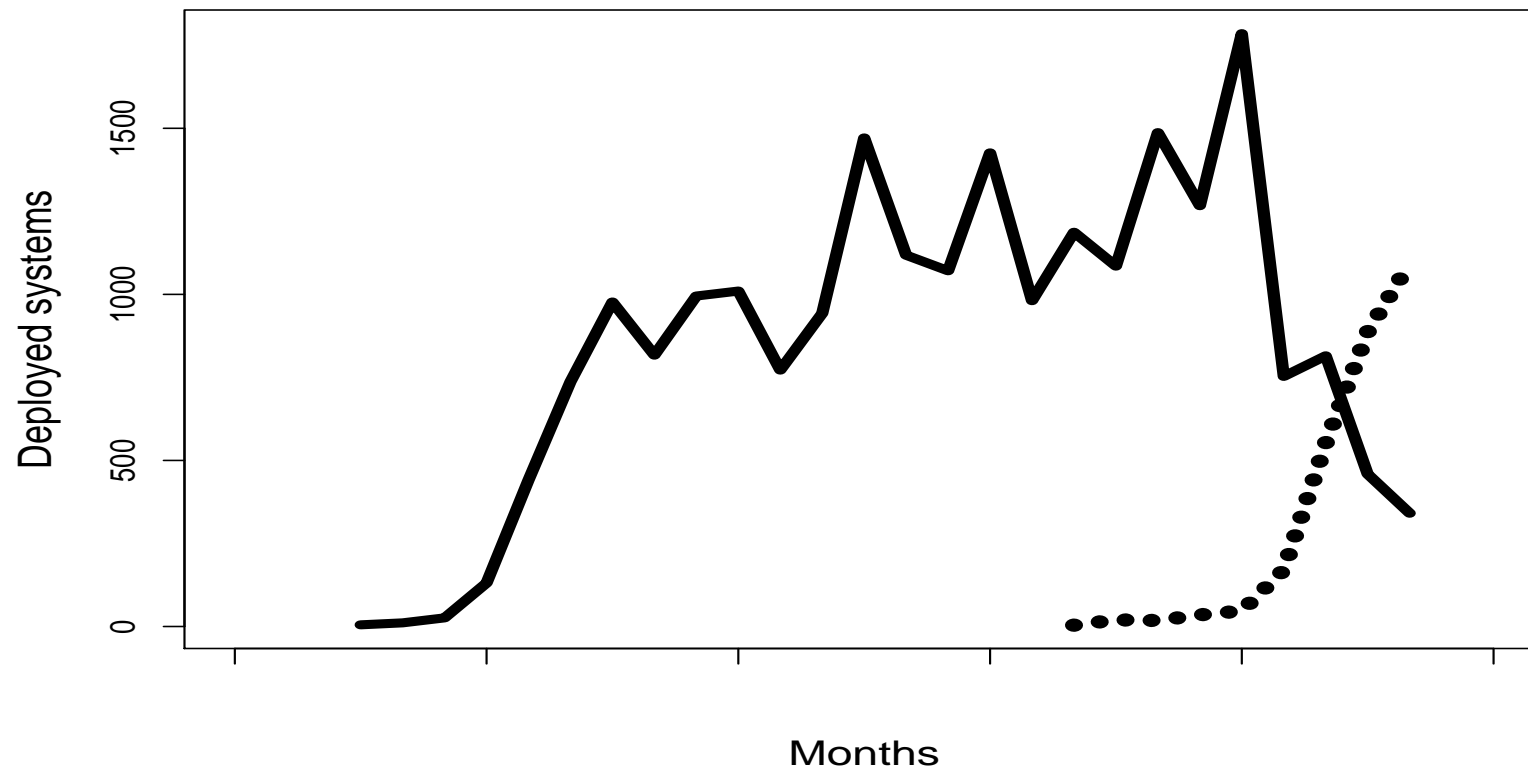


almost **constant**

but

Variability \implies unpredictability!

The software deployment and failure rates

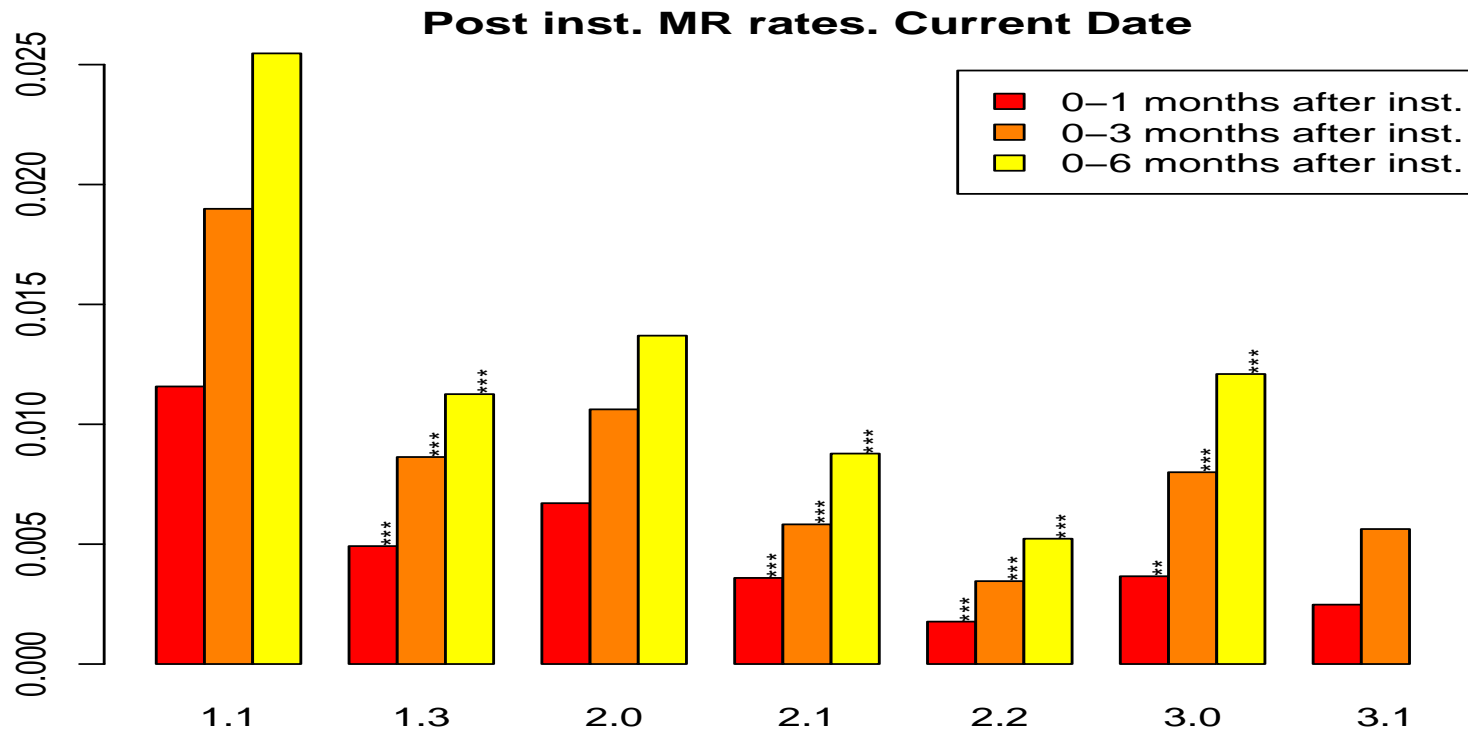


vary a lot!

Resolution: major vs minor

- ❖ Compare Defect Density and Customer Quality
 - ❖ The numerators: approximately the same because
 - ❖ Major releases are deployed more slowly to fewer customers
 - ❖ Customers (correctly) do not expect a fault in minor releases and deploy more rapidly
 - ❖ The denominator: diverges because
 - ❖ Major releases have more code modified but fewer customers
 - ❖ Minor releases have less code modified but more customers

Customer Quality



- ❖ Fraction of customers reporting software failures within months of installation
- ❖ Does not account for proximity to launch, platform mix
- ❖ Significant differences marked with “*”
- ❖ “We live or die by this measure.”
— executive for product quality

Change tomography \implies insights

- ❖ Methodology
 - ❖ Changes provide traces of developer and organizational behavior
 - ❖ Insights become an integral part of development practices — continuous feedback on production changes/improvements
- ❖ Insights
 - ❖ Development process view does not represent customer view
 - ❖ Learning proceeds not just by increasing the number of tasks performed, but mostly by their importance to the organization

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 *23rd 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. The talk describes methods and tools to retrieve, process, and model data from ubiquitous change management systems and uses them to understand common problems facing a software project. In particular, the approach is illustrated using three industry applications. The quantification of the relationship between the test coverage and customer reported defects shows that increases in test coverage are related to lower defect density, but reaching high levels of coverage requires exponentially more effort. In another application of the approach the transfer of code ownership substantially reduces developer productivity, especially in cases of offshoring. Finally, the customer and developer views of software quality diverge. The 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 Avaya Labs Research. Previously he worked at Software Production Research Department of Bell Labs.