Measurement in Science, Engineering, and Software

A. Mockus

---

audris@mockus.org

Avaya Labs Research

Basking Ridge, NJ 07920

http://mockus.org/

The requirements

Topics for day’s discussions

Topics on software engineering

The process

The complication

- The only thing I know for sure is that I don't know anything
  - How can I provide topics worthy of discussion?
  - How can I avoid embarrassing organizers?

The way out

Tell what I do not know

But how can I tell what I do not know?

by asking questions?!

The proposed solution

- I will ask questions, you will provide answers
  - "One fool can ask more questions than one hundred wise (wo)men can answer"
  - so please don't get frustrated by silly questions...

Basic questions

The purpose of human endeavor?

- If the works succeeds beyond the wildest dreams, will the results be useful?
  - For me?
  - For anyone else?
  - For people who do what?
    - How many?
    - For how long?

What are we doing in this room?

- What is MSA?
- Is it different from MSR?
- Is it data mining?
- Is it software engineering?
- Is it measurement?
- Is it science?

Data Mining?

- “the process of extracting patterns from data”
- What to do with patterns?
  - Use them to accomplish specific tasks?
    - Direct benefits: more revenue/less cost
      - Recommend a movie?
      - Pick the advertisement or advertiser?
      - In software - static analysis (e.g., klockwork), test generation?
    - Indirect: more reputation/trust?
      - Provide relevant information/service (search, news, reviews, people, ...)?
      - In software?

Statistics?

“the science of making effective use of numerical data, including not only the collection, analysis and interpretation of such data, but also the planning of the collection of data.”

Measurement?

- Why measure? Because without data, you only have opinions? or
  - to *characterize*, or gain understanding of your processes, products, resources, and environments?
  - to *evaluate*, to determine your status with respect to your plans?
  - to *predict*, by understanding relationships among processes and products so the values you observe for some attributes can be used to predict others?
  - to *improve*, by identifying roadblocks, root causes, inefficiencies, and other opportunities for improvement?
- Why analyze? Because the data you collect can’t help you if you **don’t understand** it and **use** it to shape your decisions?

Software Engineering?

- Characterize, understand, and improve software practice?
  - Inform and predict: (quantitatively) trade-offs among schedule, quality, cost?
    - Where effort is spent, where defects are introduced?
    - What is the impact of technologies/organization/individuals?
  - Act:
    - Introduce technologies?
    - Change organization?
    - Train individuals?

Science?

- Fundamental questions about human and collective nature?
  - 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*
- Is it X, Y, or Z?
  - *Tomography* is image reconstruction from multiple projections
  - What is the reconstruction of developer behavior from the digital traces they leave in the code and elsewhere?

Method: Software Tomography?

Software change?

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

Before:	*After:*
	//print n integers
int i = n;	int i = n;
while(i++)	while(i++ && i > 0)
prinf(" %d", i−−);	prinf(" %d", i−−);

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

Uniform Theory of Everything?

- Sales/Marketing: customer information, rating, purchase patters, needs: features and quality
- Accounting: customer/system/software/billing
- Maintenance support: installed system, support level, warranty
- Field support: dispatching repairmen, replacement parts
- Call center support: customer/agent/problem tracking
- Development field support: software related customer problem tracking, installed patch tracking
- Development: feature and development, testing, and field defects, software change and software build, process WIKI

Context, data, and software — D-Ware?

- Data have meaning without context?
- Data have meaning without knowing how it was obtained?
- Data have meaning without knowing how it was processed?
- Data have “bugs” beyond bugs in the analysis software?

SW tomography: all D-Ware has bugs?

- Bugs in the phenomena under study — randomness?
- Bugs in data recording — people (longitudinal), process, tool interface and schema (bias)?
- Bugs in data processing — software, schema, no “classical” randomness?
- Bugs in interpretation — method?

Any bugs here?

Priority	Tot. Prj A	Tot. Prj B	Tot. Prj C	% A	% B	% C
Critical	10	62	0	0	0	0
High	201	1642	16	5	13	5
Medium	3233	9920	659	84	76	85
Low	384	344	1	10	3	1
Total	3828	12968	676	100	100	100

Any bugs here?

- Question: Reliability of SD Flash cards (used to boot the system)?
- Answers:
  - Lets count the number of cases where customer tickets mention flash card and divide by the number of all systems/run-time?
  - Lets count the number of flash card replacement shipments?

Any bugs here?

- **Lets count the number of flash card replacement shipments?**
  - Unneeded replacements (the card was fine)?
  - Missed replacements (the card was obtained through other sources)?

Any bugs here?

- **Lets count the number of cases where customer tickets mention flash card?**
  - What if the ticket just mentions the flash card, but there is no problem with it?

Any bugs here?

- Lets count the number of cases where customer tickets mention flash card?
  - **What if the ticket just mentions the flash card, but there is no problem with it?**
  - If we eliminate these false matches, what about the rest?

Any bugs here?

- Lets count the number of cases where customer tickets mention flash card?
  - What if the ticket just mentions the flash card, but there is no problem with it?
  - **If we eliminate these false matches, what about the rest?**
  - Interview people who worked on the problem — (ground truth - Terra Verita)?

Any bugs here?

- Lets count the number of cases where customer tickets mention flash card?
  - What if the ticket just mentions the flash card, but there is no problem with it?
  - If we eliminate these false matches, what about the rest?
  - **Interview people who worked on the problem — (ground truth - Terra Verita)?**
  - What if we cant trust them? E.g., “the first action in a case of a problem with a reboot is to replace the card.”

SW tomography: bugs in the phenomena?

“We get the notions theories are right because we keep talking about them. Not only are most theories wrong, but most **data** are also **wrong** at first subject to glaring uncertainties. The recent history of X is full of promising discoveries that disappeared because they could not be repeated.”

- Statistical methods take **variability** into account to support making informed decisions based on quantitative studies designed to answer specific questions.
- Visual displays and summary statistics condense the information in data sets into *usable knowledge.*
- **Randomness** is the foundation for using statistics to draw conclusions when testing a claim or estimating plausible values for a population characteristic.
- The design of an experiment or sample survey is of critical importance to analyzing the data and drawing conclusions.

SW tomography: Debugging

- 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

SW tomography: Levels [0-2]

- Level 0 — actual project. Learn about the project, make copies of 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

SW Tomography: Testing/Debugging

Takes up 9[5-9]% of all effort

- 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

Version control D-Ware to aid “data debugging”?

Keep raw data/systems and version control processing scripts?

Why software tomography?

- Non-intrusive, minimizes overhead? *What about in-depth understanding of project's development process?*
- Historic calibration, immediate diagnosis? *It takes time and effort to get to that point?*
- Fine-grain, at the delta level? *But aren't links to more sensible attributes like features and releases often tenuous?*
- Everything is recorded? *What about entries that are inconsistently or rarely filled in?*
- Uniform over time? *But process may have changed?*
- Small effects can be detected with a lot of data? *Are the relevant quantities extractable?*
- No observer effect? *Even when the such data are used widely in organizational measurement?*

Why not Software Tomography?

- Apples and oranges:
  - Do projects use the same rules to divide work (MRs)?
  - How to compare data from: CVS, ClearCase, SCCS, svn, git, hg, bzr?
  - Does every project uses the same tool in the same way: under what circumstances the change is submitted, when the MR is created?
- Easy to get lost analyzing irrelevant things?
- Are there change back-projection models of key software engineering problems?

Software Tomography: reconstructing the image

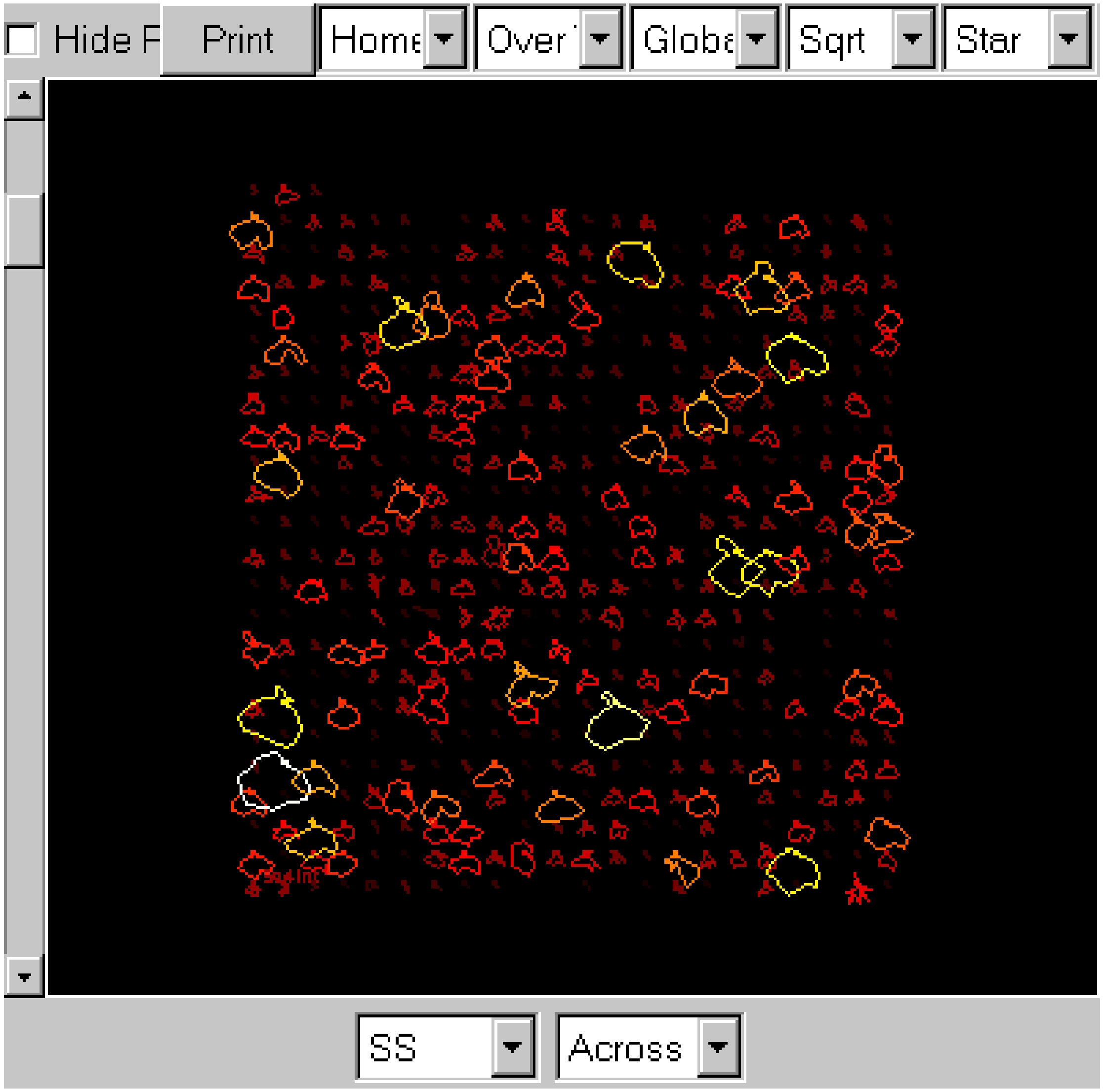
- Predicting the quality of a patch [16]
- Globalization: move development where the resources are:
  - What parts of the code can be independently maintained [17]
  - Who are the experts to contact about any section of the code [13]
  - Mentorship and learning [11, 20]
- Effort: estimate MR effort and 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, 18, 5]
  - Release quality/availability [3, 15, 9, 19]

Questions of style and productivity

Why do easy things?

- Counts, trends, patterns?
- Open source, popular projects, VCS?
- Topics that are well formulated?
  - Which modules will get defects?

Patterns: Developer changes over 24 hours — isn't it beautiful?



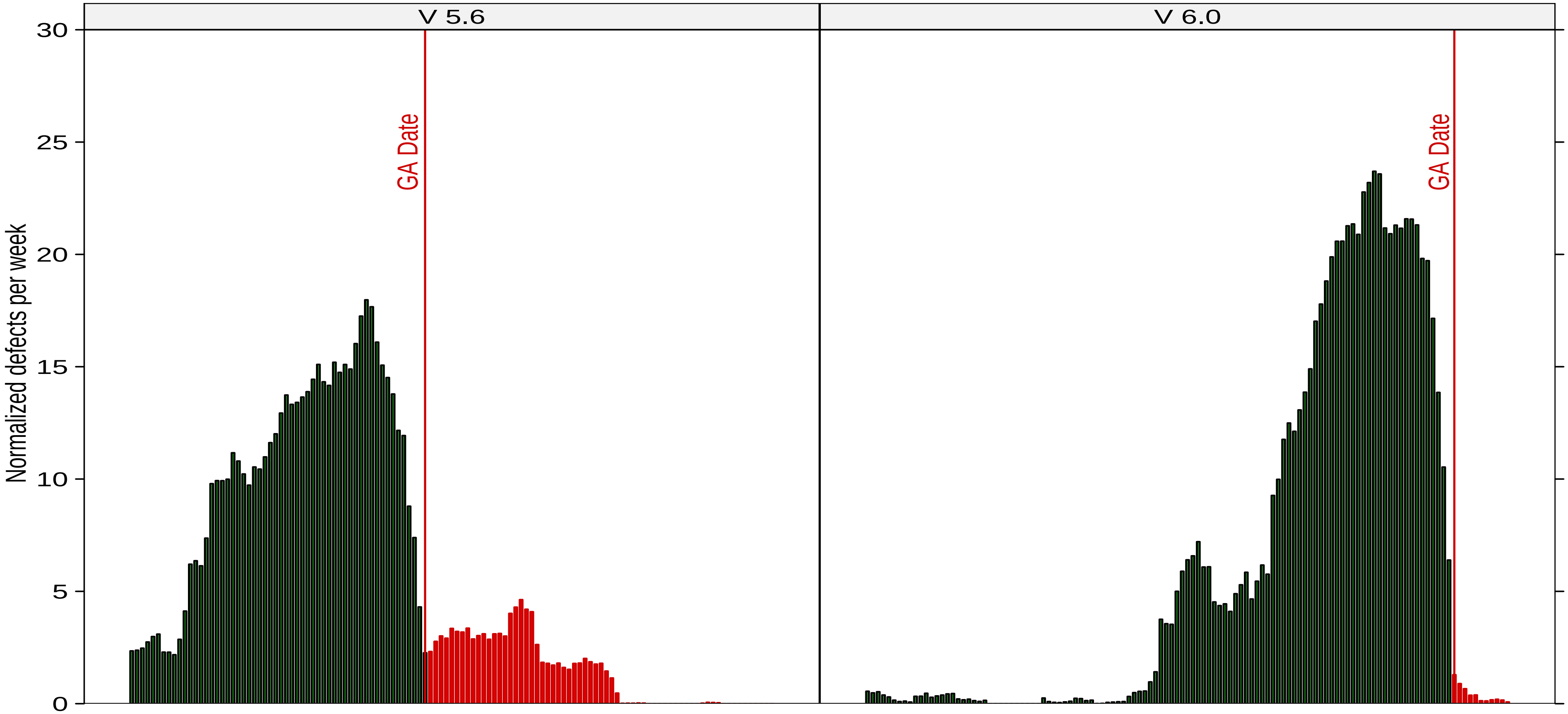
Fascination with defects

- How to not introduce defects?
  - Improve requirements and other process?
  - Improve modularity, increase language level, smarter static type-checking, LINT-type heuristics, . . . ?
  - Verification of software models?
- How to find/eliminate defects?
  - Inspections?
  - Testing?
  - Debugging?
- **How to predict defects?**
  - When to stop testing and release?
  - **What files, changes will have defects?**
  - **How customers will be affected?**

Where faults will occur?

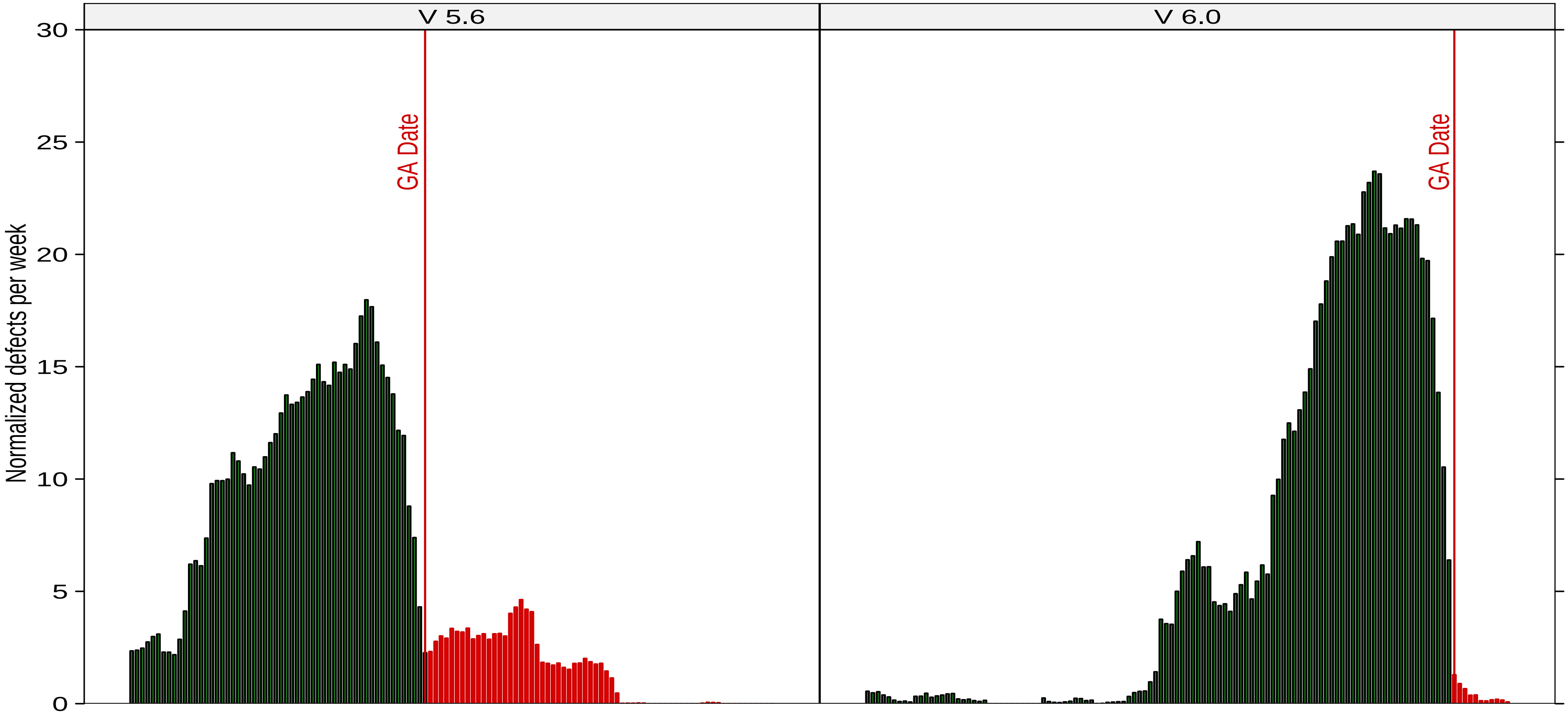
- Assume the best possible outcome, i.e., we can predict exactly!
- Does it help?
  - “We look at defects for features, not for files”
  - Most defects discovered by static-analysis tools are not fixed?
  - “often it’s better to leave a known defect unresolved, than fix it and [by doing that] introduce a defect you don’t know about”
  - Effort needed to investigate predictions exceeds all QA resources?

Can bugs be predicted reliably?



Why such huge improvement in quality?

How many customers got each release?



$$V5.6 \approx 300, V6.0 \approx 0$$

Questions of practice

Practice: how to compare software releases?

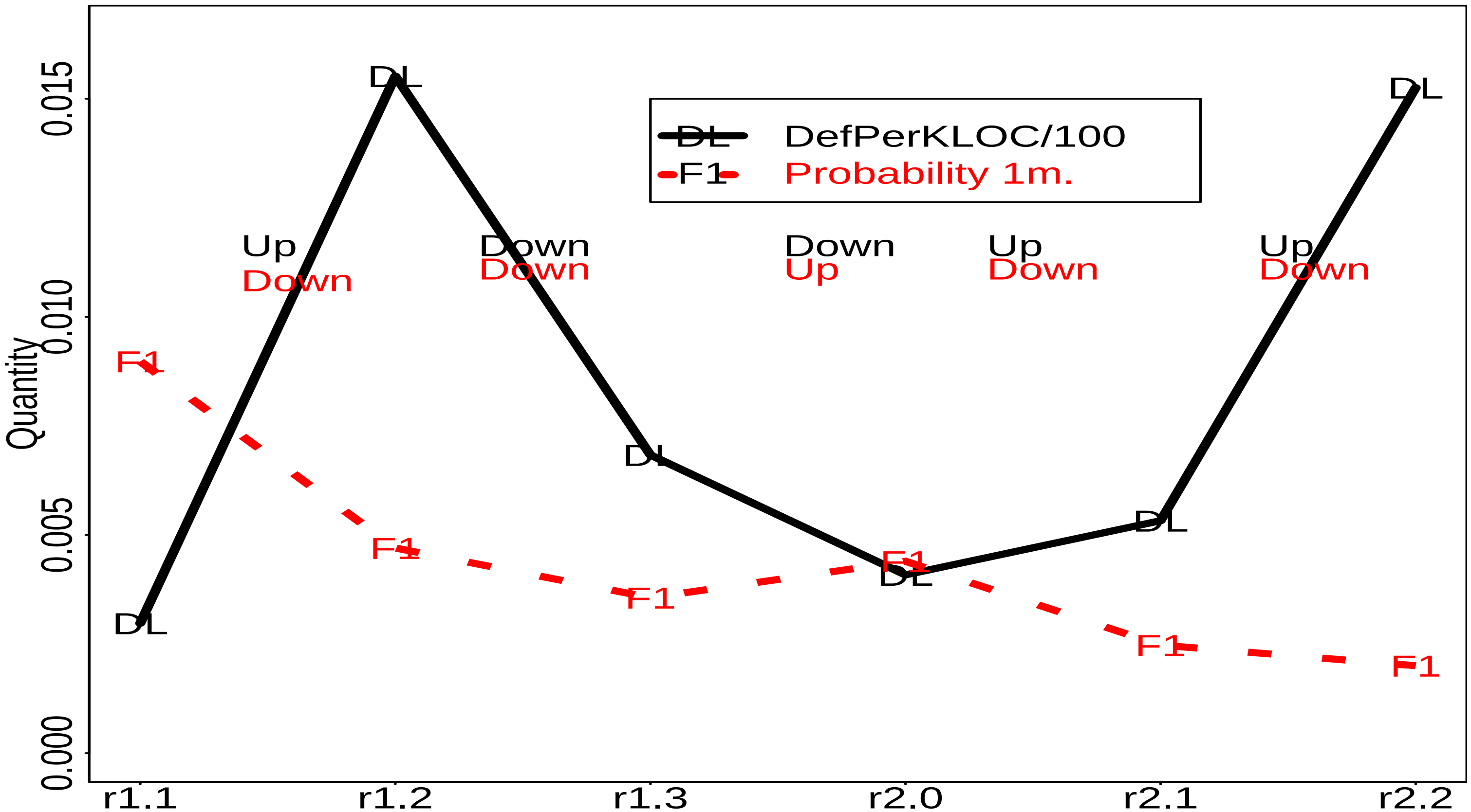
"**we tried to improve quality**: get most experienced team members to test, code inspections, root cause analysis, ..."

"**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."

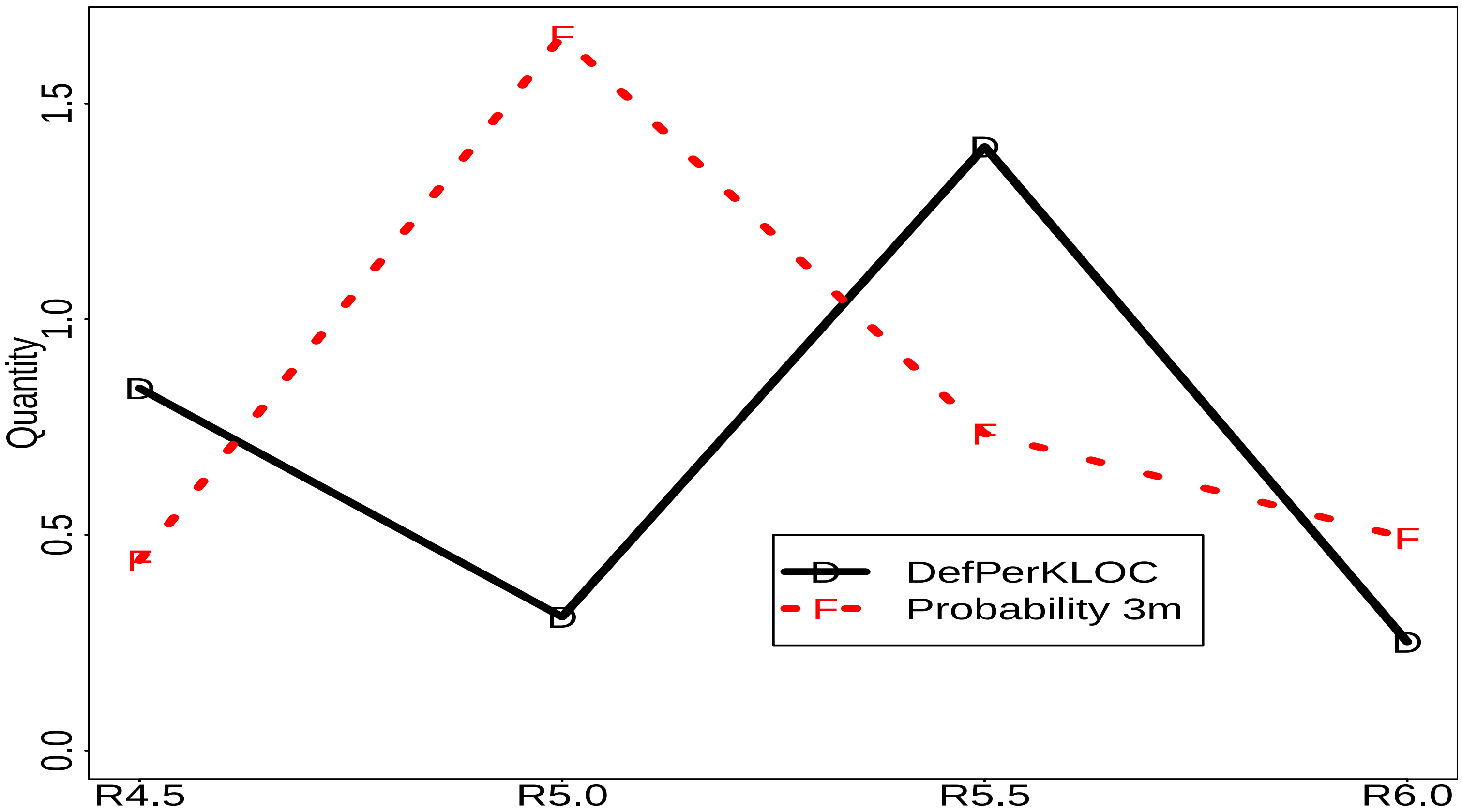
Ok, then lets measure the probability that *a customer will report a software defect*

A paradox: large telecom software



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

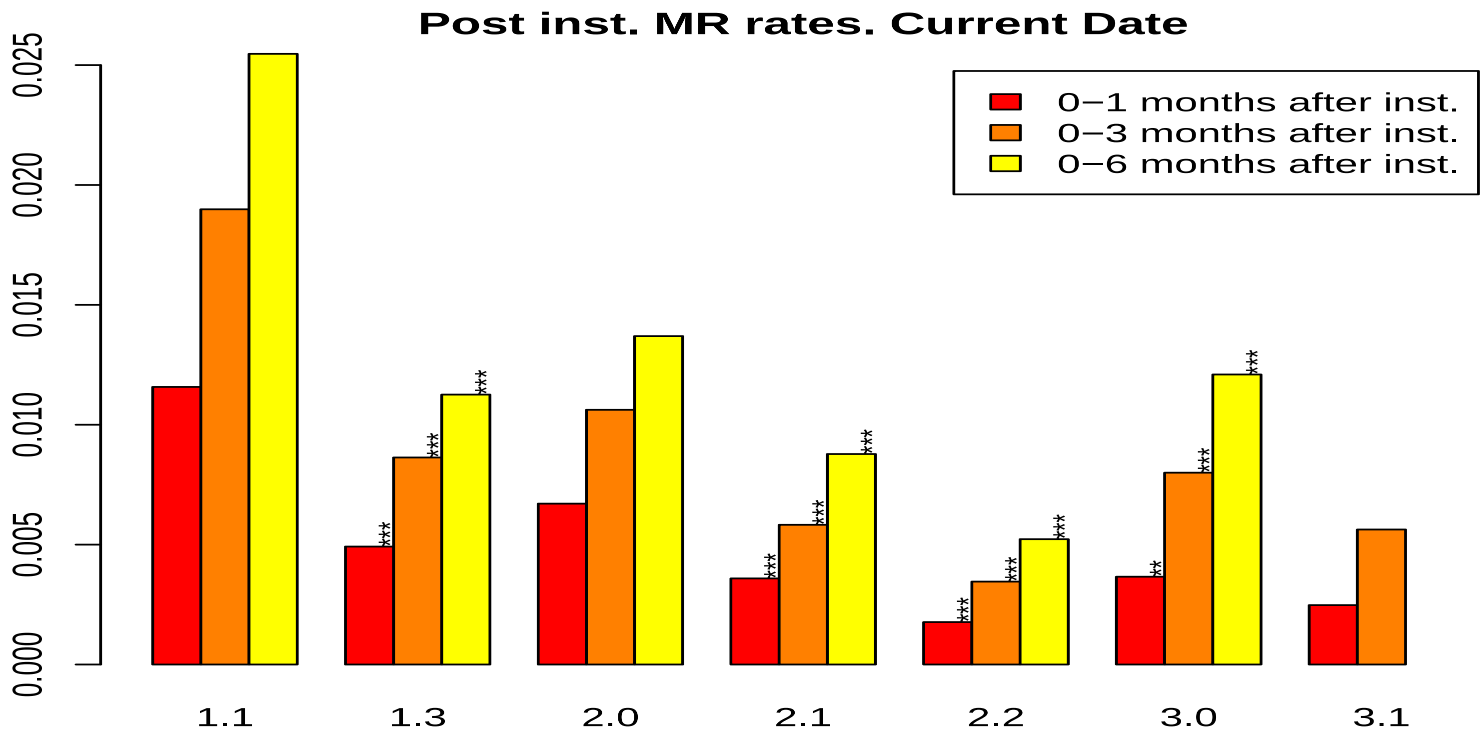
Is the paradox unique for this product?



A large product from another company:

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

What fraction of customers are affected (IQ)?

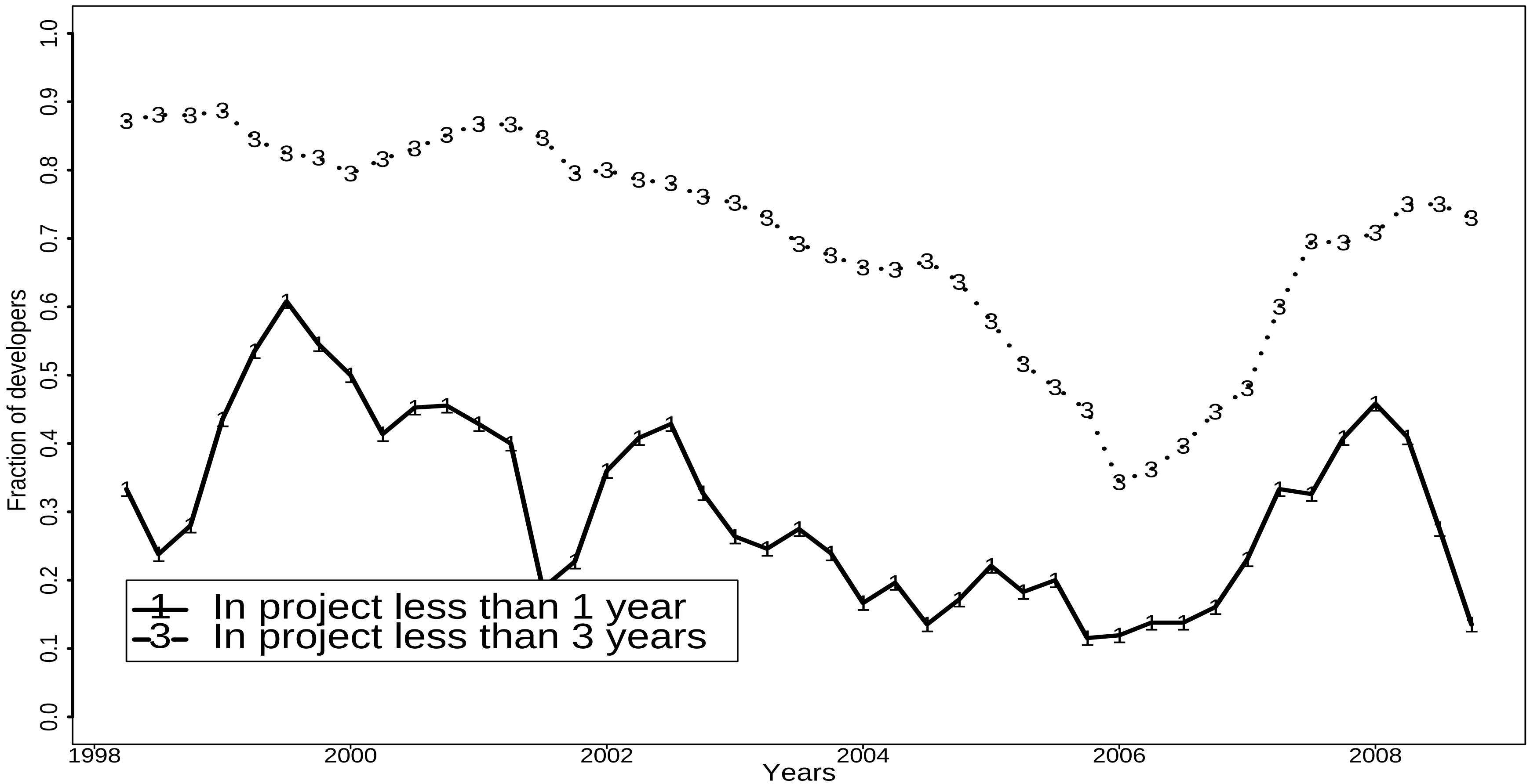


- Fraction of customers reporting software failures within months of installation
- Significant differences from prior releases marked by “*”
- “We live or die by this measure.”
  — executive for product quality

Can we move software production to the cheapest location?

Offshoring/Outsourcing/Retirement

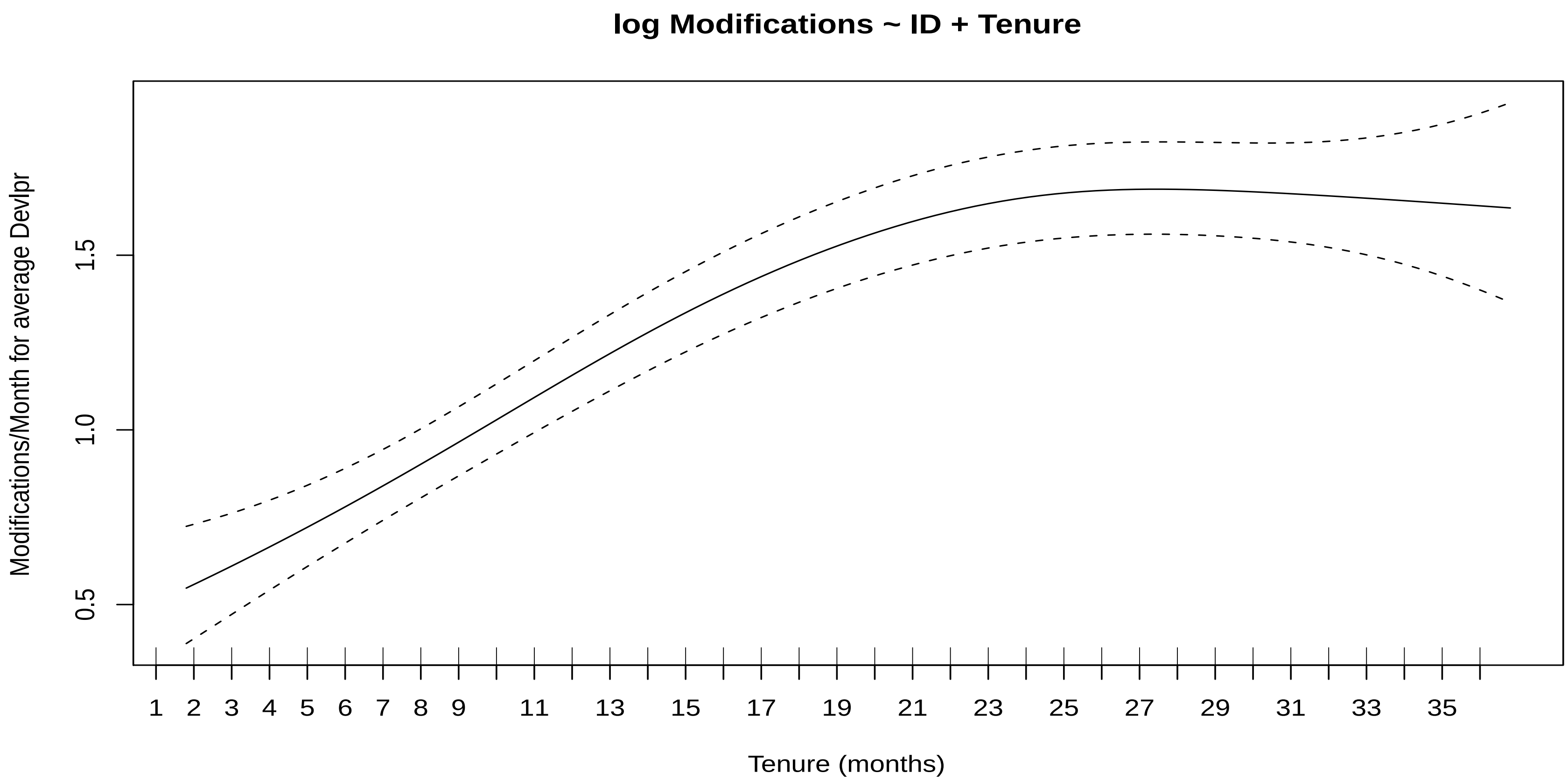
Developer Churn



A plateau?

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

— a common response from managers and developers



Modifications per month versus Tenure

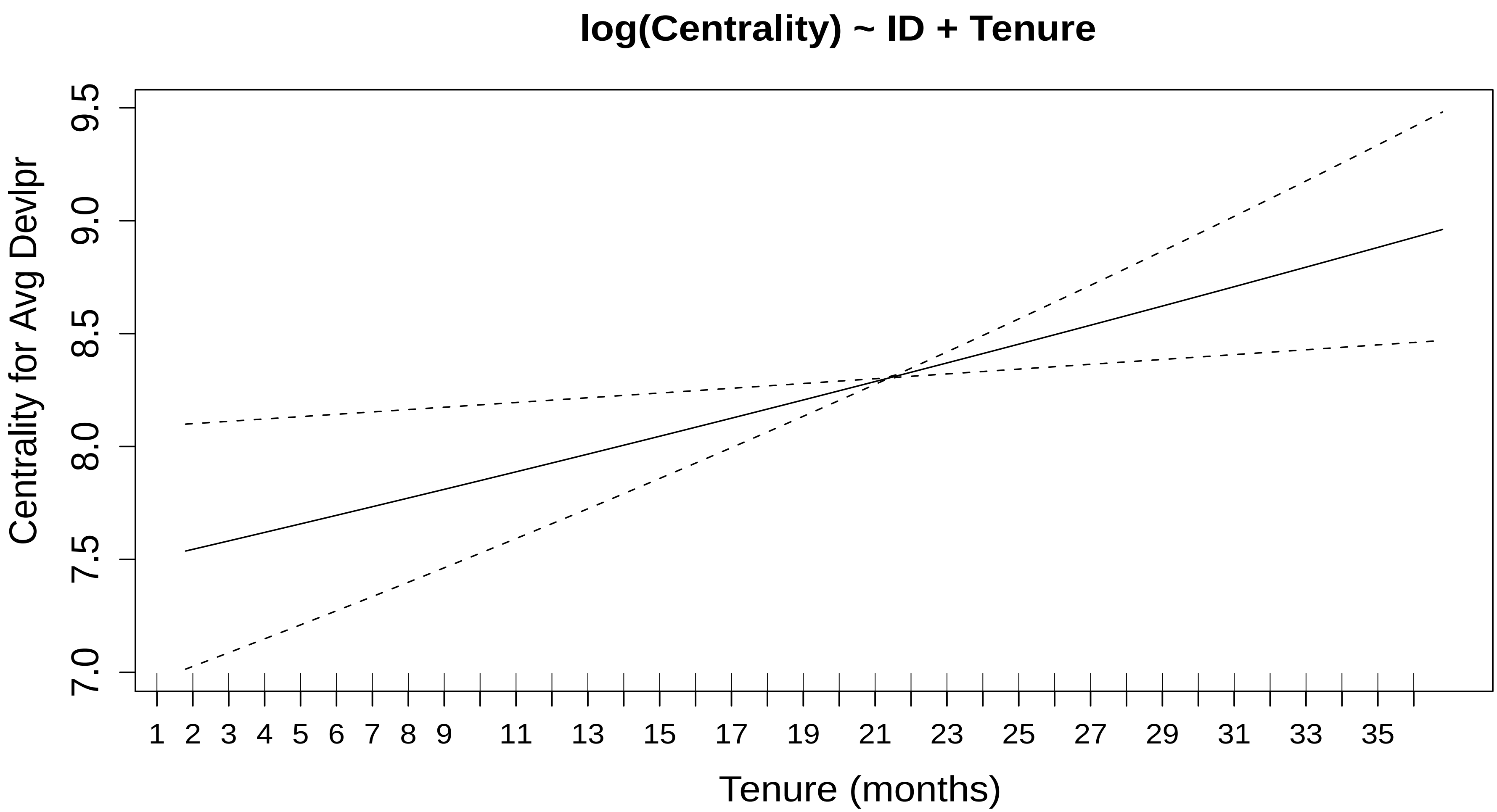
Fully productive, but...

"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."

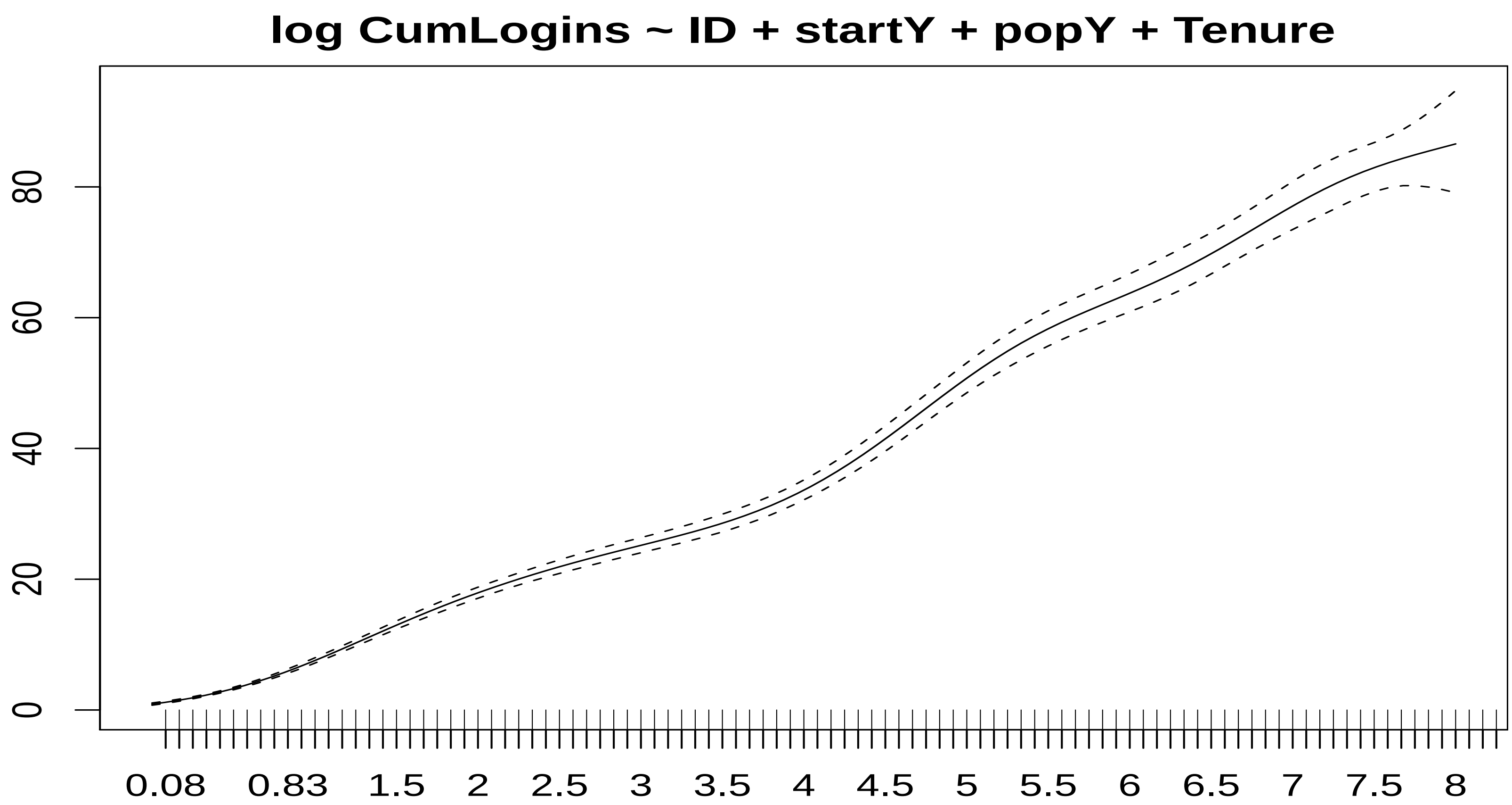
— Senior architect

Task's importance keeps increasing?



Average task's centrality (average centrality of modules modified by the task) versus Tenure

Social learning



Accelerates after three to four years?

Discussion

- Entire social and business life is digitally recorded: infinite resources and opportunities for Software Tomography?
  - Multiple dimensions of human activity are recorded?
  - Multiple models (reconstructions) from various fields are (re)invented?
- D-Ware bugs: phenomena, UI, software, data processing, and interpretation?
  - Statistical and software “randomness/bugs”?
- But how to use these digital projections of human endeavor to get results relevant to
  - Yourself?
  - Someone else?
  - Many people?
  - For eternity...

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 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.

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

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

[18] 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.

[19] 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.

[20] 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

Measurement is the essence of science: ”To measure is to know”. In engineering the data can’t help if you don’t understand it and use it to make decisions. As many professional and social activities are moving online and rely on software tools, a vast amount of data becomes available. Practical applications in business intelligence, and sciences have been demonstrated that use various models and methods to solve a particular problem in the corresponding domain. It is, therefore, tempting to apply these techniques on software engineering data often without the adequate adaptations to the domain with the completely different needs. Furthermore, as the field of Computer Science matures, it requires more rigorous empirical approaches and the same can be said about rapidly maturing fields of Mining Software Archives/Repositories. Therefore, we discuss common issues facing researchers with Computer Science background as they move into empirical areas that require several fundamentally different concepts: variation, reproducibility, and human factors. In addition to methodological issues, we also look at the future challenges posed by the need to integrate more and more disparate sources of data, the tradeoffs between using the most easily available and the more meaningful measures, and the need to address core software engineering concerns.

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 is interested in quantifying, modeling, and improving software development. He designs data mining methods to summarize and augment software change data, interactive visualization techniques to inspect, present, and control the development process, and statistical models and optimization techniques to understand the relationships among people, organizations, and characteristics of a software product. 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 in Avaya Labs Research. Previously he worked in the Software Production Research Department of Bell Labs.