

Interval Quality: Relating Customer-Perceived Quality To Process Quality

Audris Mockus and David Weiss



{audris,weiss}@avaya.com

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

Motivation: bridge the gap between developer and user and measure *in vivo*

- ❖ A key software engineering objective is to improve quality via practices and tools that support requirements, design, implementation, verification, and maintenance
- ❖ Needs of a user: installability, reliability, availability, backward compatibility, cost, and features
- ❖ Primary objectives
 - ❖ Can we measure user-perceived-quality *in vivo*?
 - ❖ Can we communicate it to the development team?
 - ❖ Is the common wisdom about software quality correct?

Outline

- ❖ History of quality in communications systems
- ❖ How to observe quality *in vivo*
- ❖ Questions
 - ❖ Can we compare quality among releases?
 - ❖ Which part of the life-cycle affects quality the most?
 - ❖ Can we approximate quality using easy-to-obtain measures?
 - ❖ Does hardware or software have more impact on quality?
- ❖ Answers
 - ❖ Yes, service, no, it depends
- ❖ Discussion

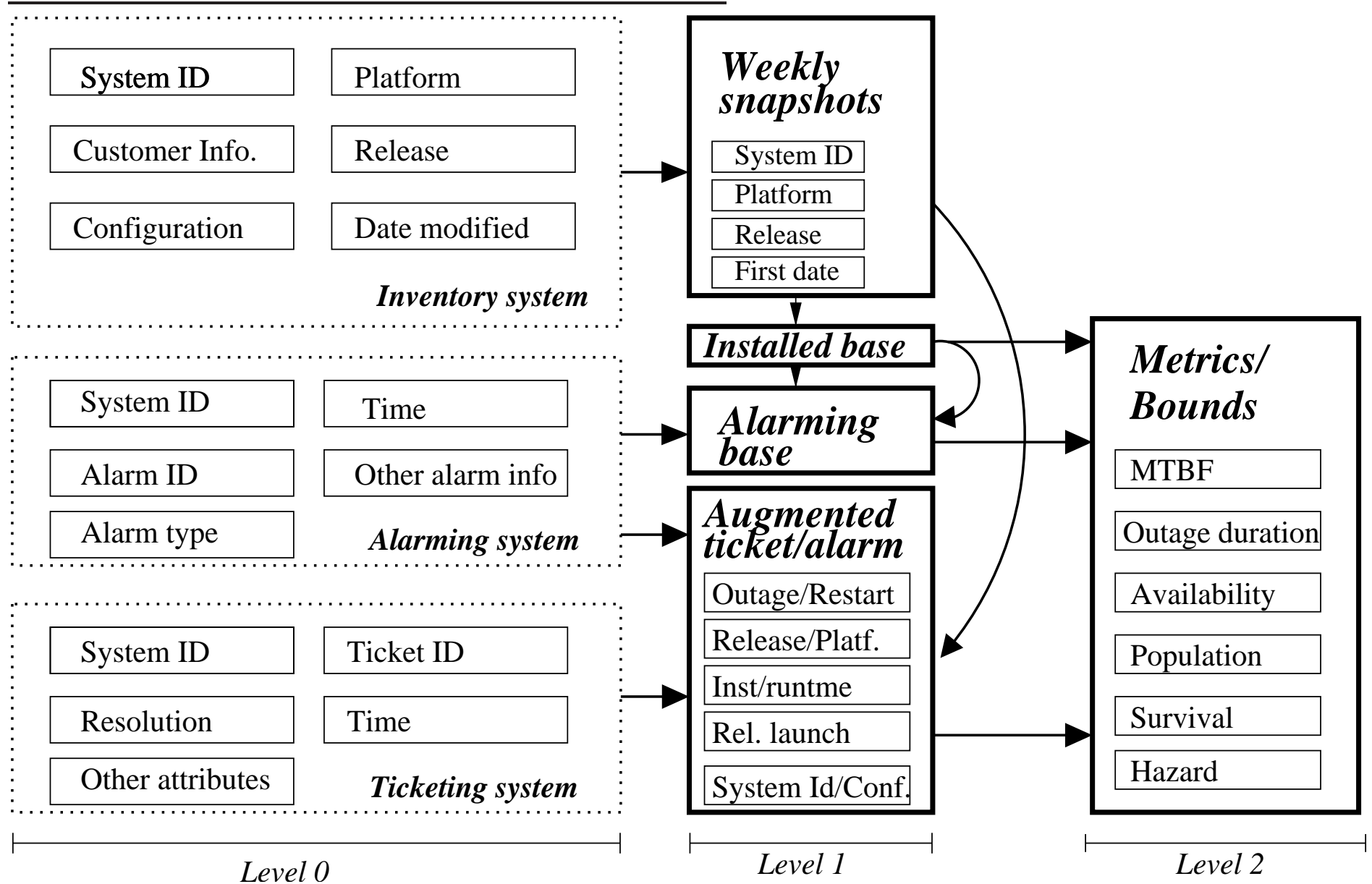
Approaches to measure quality

- ❖ Theoretical models [16]
- ❖ Simulations (*in silico*)
- ❖ Observing indirectly
 - ❖ Test runs, load tests, stress tests, SW defects and failures
- ❖ **Observing directly *in vivo*** via recorded user/system actions (not opinion surveys) has following benefits:
 - ❖ is more realistic,
 - ❖ is more accurate,
 - ❖ provides higher level of confidence,
 - ❖ is more suited to observe an overall effect than *in vitro* research,
 - ❖ is more relevant in practice.

History of Communications Quality [6]

- ❖ Context: military and commercial communication systems, 1960-present
- ❖ Goals: system outage, loss of service, degradation of service
 - ❖ Downtime of 2 hours over 40 yr, later “5 nines” (or 5 min per year)
 - ❖ Degradation of service, e.g., $< .01\%$ calls mishandled
 - ❖ Faults per line per time unit, e.g., errors per 100 subscribers per year
 - ❖ MTBF for service or equipment, e.g, exchange MTBF, % subscribers with $MTBF > X$
 - ❖ Duplication levels, e.g., standby HW for systems with > 64 subscribers

Observing *in vivo* — architecture



Observing *in vivo* — primary data sources

- ❖ Service tickets
 - ❖ Represent requests for action to remedy adverse events: outages, software and hardware issues, and other requests
 - ❖ Manual input \implies not always accurate
 - ❖ Some issues may be unnoticed and/or unreported \implies missing
- ❖ Software alarms
 - ❖ Complete list for the systems configured to generate them
 - ❖ Irrelevant events may be included, e.g, experimental, misconfigured systems that are not in production use at the time
- ❖ Inventory
 - ❖ Type, size, configuration, install date for each release
- ❖ **Link between deployment dates and tickets/alarms**

Issues with commonly available data and published analyses

❖ Present

- ❖ Problem reports by month (hopefully grouped by release)
- ❖ Sales (installations) by month (except for freely downloadable SW)

❖ Absent

- ❖ No link between install time and problem report \implies
 - ❖ no way to get accurate estimates of hazard function (probability density of observing a failure conditional on the absence of earlier failures)
- ❖ No complete list of software outages \implies
 - ❖ no way to get rough estimates of the underlying rate

Data Remedies

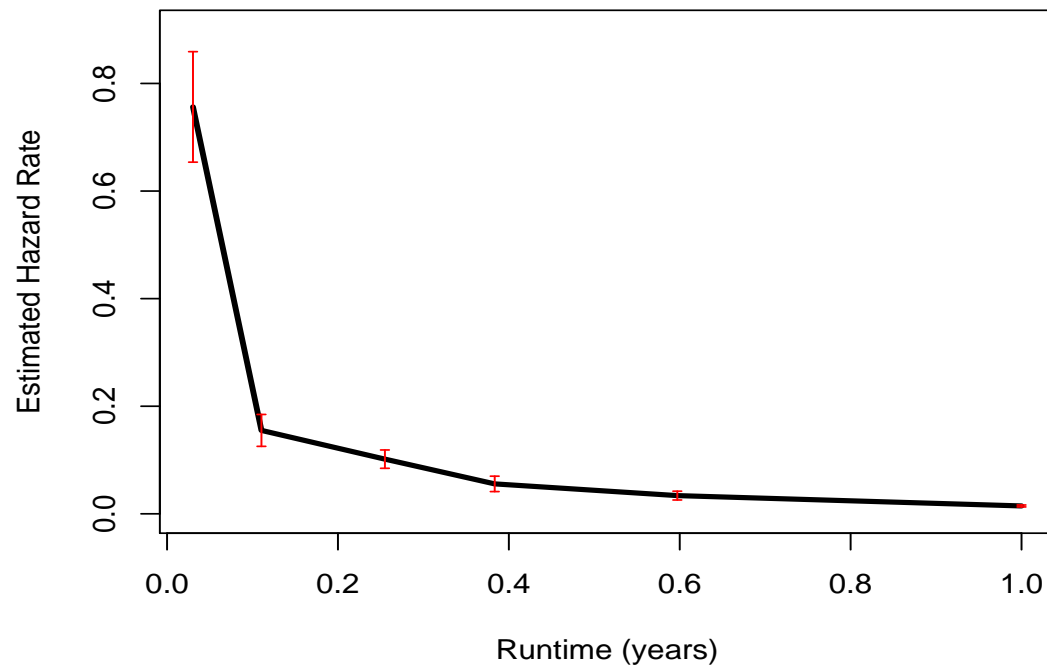
- ❖ Only present state of inventory is kept \implies collect snapshots to reconstruct history
- ❖ The accounting aggregation (by solution, license) is different from service (by system) or production (by release/patch) aggregation \implies remap to the finest common aggregation
- ❖ Missing data
 - ❖ Systems observed for different periods \implies use survival curves
 - ❖ Reporting bias \implies divide into groups according to service levels and practices
- ❖ Quantity of interest not measured \implies design measures for upper and lower bounds

Practical questions

- ❖ Can we compare quality among releases to evaluate the effectiveness of QA practices?
- ❖ Which part of the production/deployment/service life-cycle affects quality the most?
- ❖ Can quality be approximated with easy-to-obtain measures, e.g., defect density?
- ❖ Does hardware or software have more impact on quality?

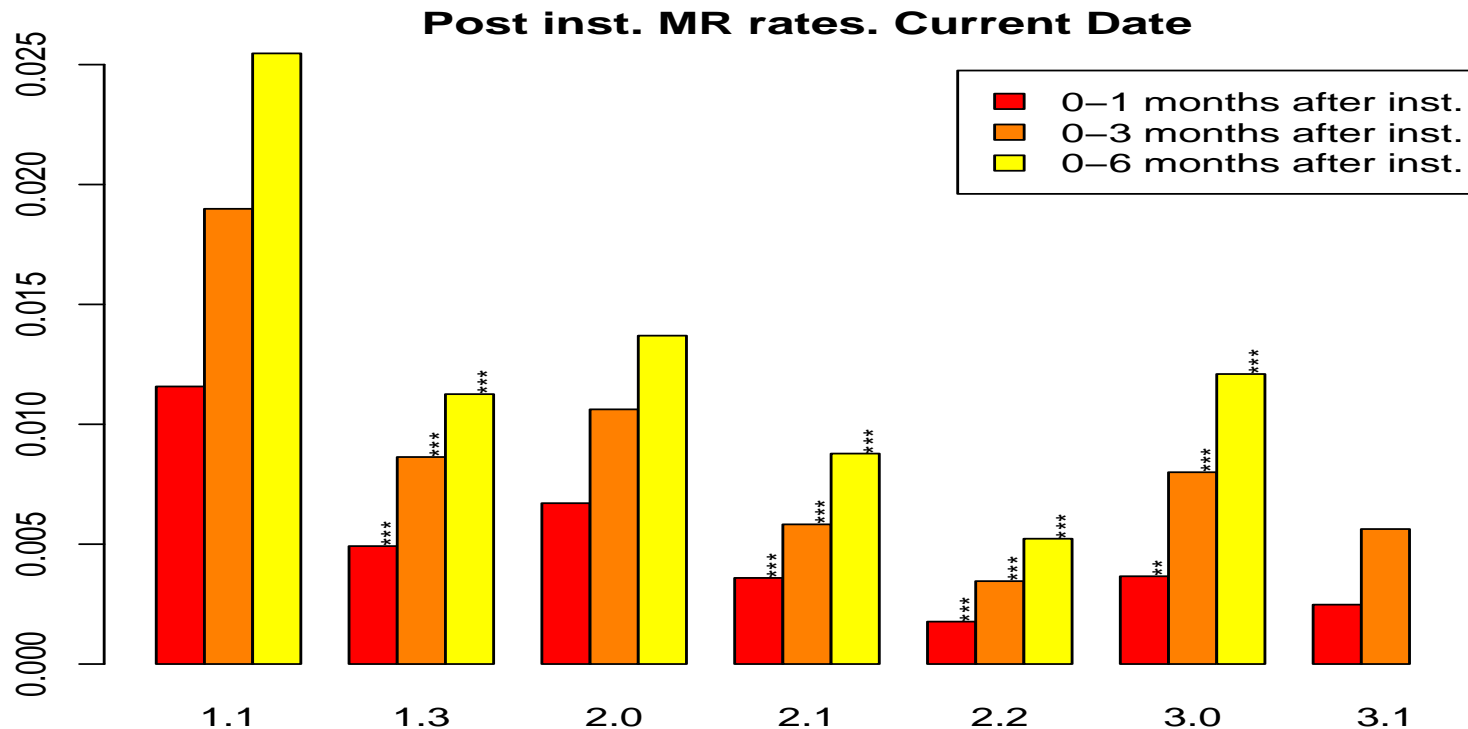
Hazard function

(Probability density of observing a failure conditional on the absence of earlier failures)



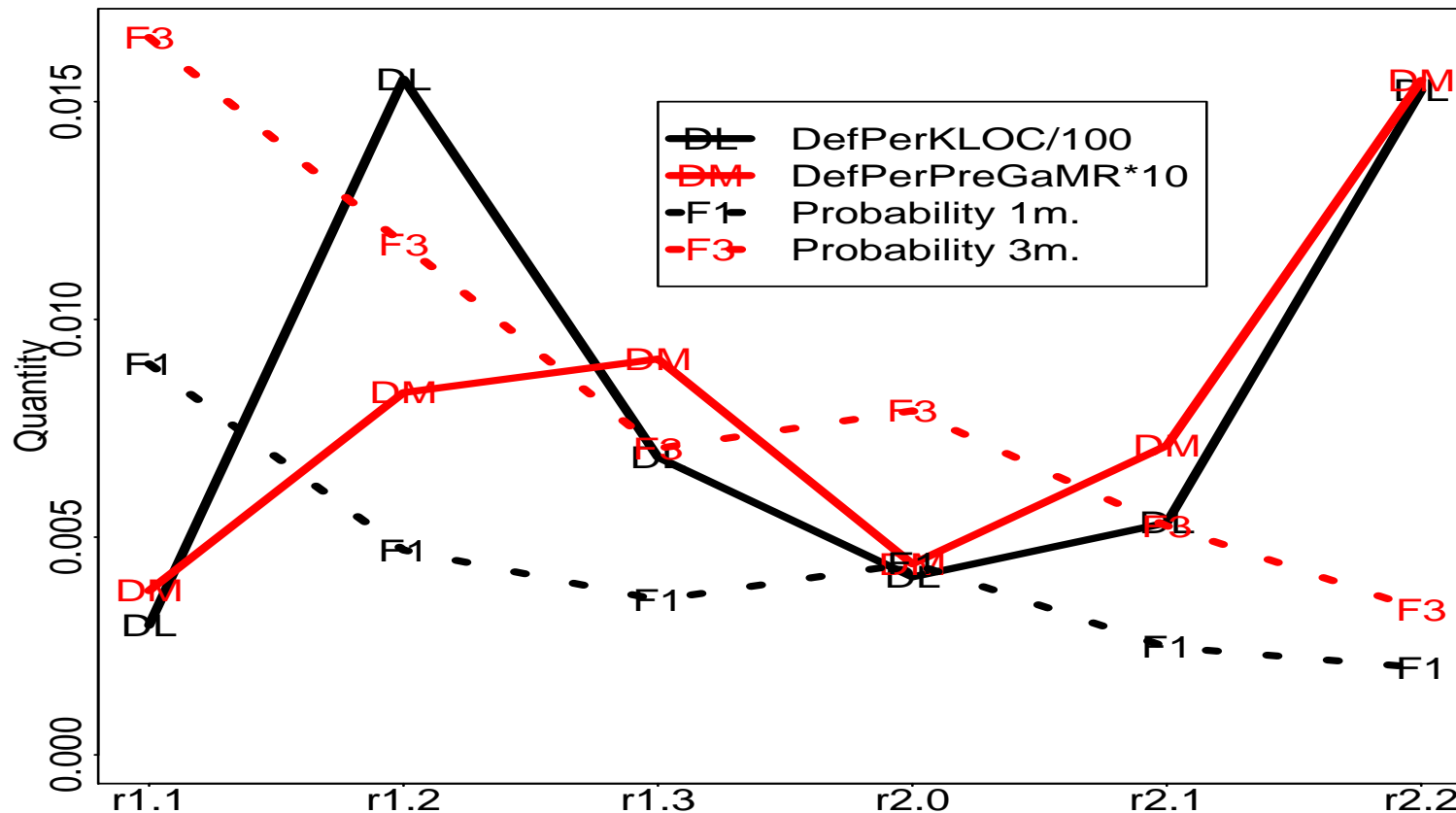
- ❖ Have to adjust for runtime and separate by platform or the MTBF will characterize the currently installed base, not release quality
- ❖ Therefore, how to compare release quality?

Interval 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”

Can we use easy-to-obtain defect density?



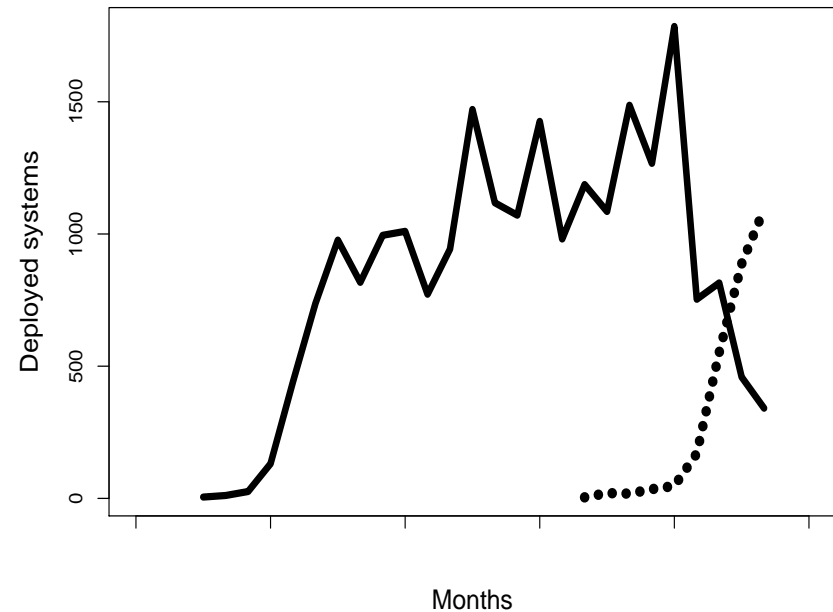
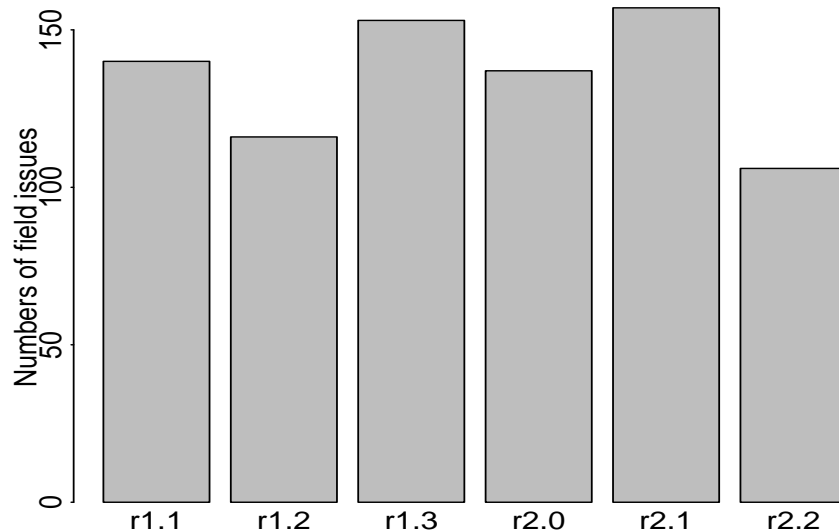
Anti-correlated?!

High defect density leads to satisfied customers?

- ❖ What does any organization strive for?

Stability \implies Predictability!

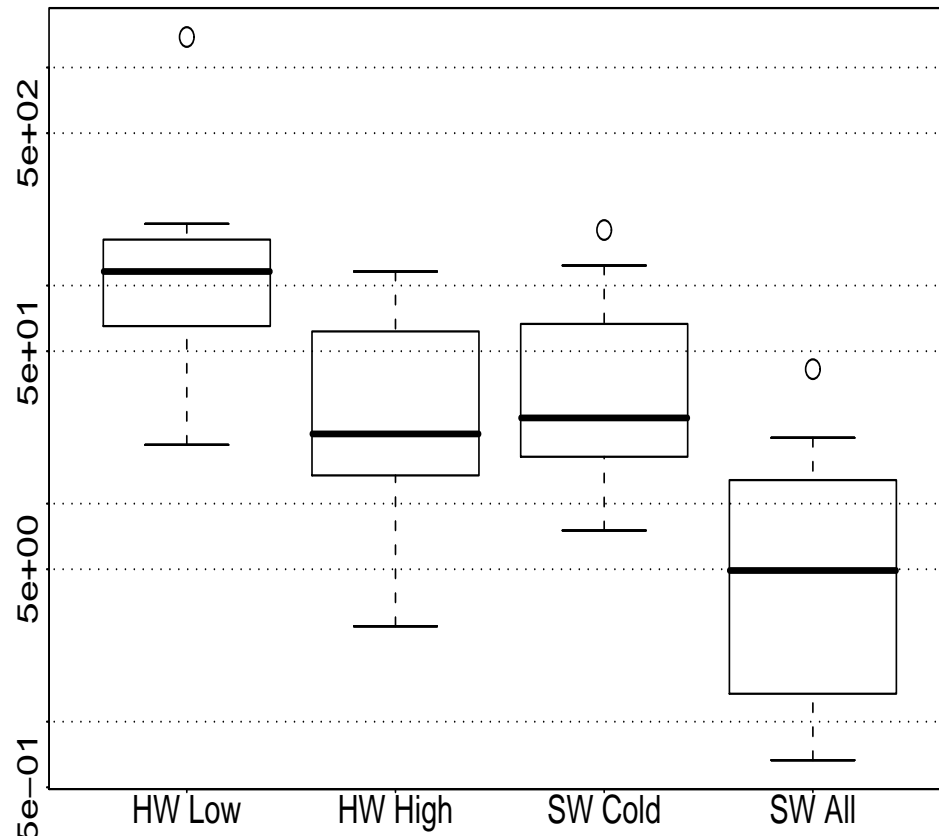
- ❖ The rate at which customer problems get to Tier IV is almost constant despite highly varying deployment and failure rates



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

Hardware vs Software



❖ Limitations

- ❖ Durations of SW Warm, SW Cold, HW differ by orders of magnitude
- ❖ Warm rst. don't drop calls
- ❖ High/Critical cfg. may be unaffected
- ❖ HW-High ultra conservative
- ❖ Variability for each estimate may be high

❖ Distribution of MTBF for 15 platform/release combinations

Which part of the software production and delivery contributes most to quality?

- ❖ Development perspective - fraction of MRs removed per stage
 - ❖ Development → features, bugs introduced, and resolved
 - ❖ Verification → 40% of development stage MRs (post unit-test)
 - ❖ α/β trials → 7% of development stage MRs
 - ❖ Deployment → 5% in major and 18% in minor releases
- ❖ Customer perspective - probability of observing a failure
 - ❖ may drop up to 30 times in the first few months post-launch [15]

In vivo investigation \implies new insights

❖ Methodology

- ❖ Service support systems provide *in vivo* capability \implies new insights
- ❖ Results become an integral part of development practices — continuous feedback on production changes/improvements

❖ Quality insights

- ❖ Maintenance — the most important quality improvement activity
- ❖ Development process view does not represent customer views
- ❖ Software tends to be a bigger reliability issue with a few exceptions

❖ Measurement hints

- ❖ Pick the right measure for the objective — no single “quality” exists
- ❖ Adjust for relevant factors to avoid measuring demographics
- ❖ Bound objective, navigate around missing, biased, irrelevant data

Thank You.

Limitations

- ❖ Different characteristics of the project including numbers of customers, application domain, software size, quality requirements are likely to affect most of the presented values
- ❖ Many projects may not have as detailed and homogeneous service repositories

Methodology: Validation

- ❖ Interview a sample of individuals operating and maintaining relevant systems
 - ❖ Go over recent cases the person was involved with
 - ❖ to illustrate the practices (what is the nature of the work item, why you got it, who reviewed it)
 - ❖ to understand/validate the meaning of 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 data via relevant models [8, 11, 1, 12]
- ❖ Validate and clean retrieved and modeled data
- ❖ Iterate

Methodology: Existing Models

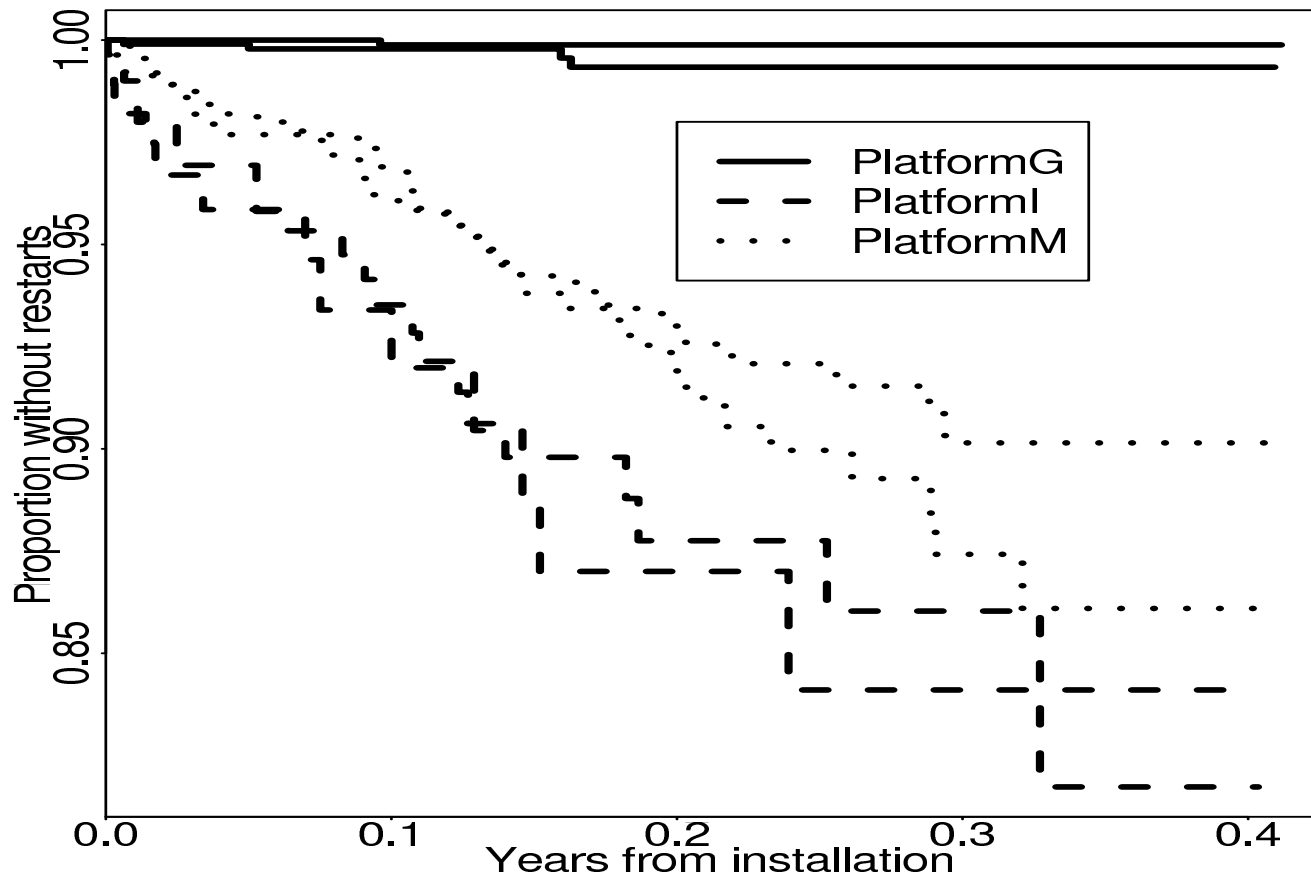
- ❖ Predicting the quality of a patch [12]
- ❖ Work coordination:
 - ❖ What parts of the code can be independently maintained [13]
 - ❖ Who are the experts to contact about any section of the code [10]
 - ❖ How to measure organizational dependencies [4]
- ❖ Effort: estimate MR effort and benchmark practices
 - ❖ What makes some changes hard [5]
 - ❖ What practices and tools work [1, 2, 3]
 - ❖ How OSS and Commercial practices differ [9]
- ❖ Project models
 - ❖ Release schedule [14]
 - ❖ Release readiness criteria [7]
 - ❖ Consumer perceived quality [15, 8]

Naive reliability estimates

- ❖ Naive estimate: $\frac{\text{calendar time} \times \text{installed base}}{\# \text{ software restarts}}$
- ❖ Naive+ estimate: $\frac{\text{runtime} | \text{simplex systems}}{\# \text{ restarts} | \text{simplex}}$
- ❖ Alarming syst. estimate: $\frac{\text{runtime} | \text{simplex, generating alarms}}{\# \text{ restarts} | \text{simplex}}$

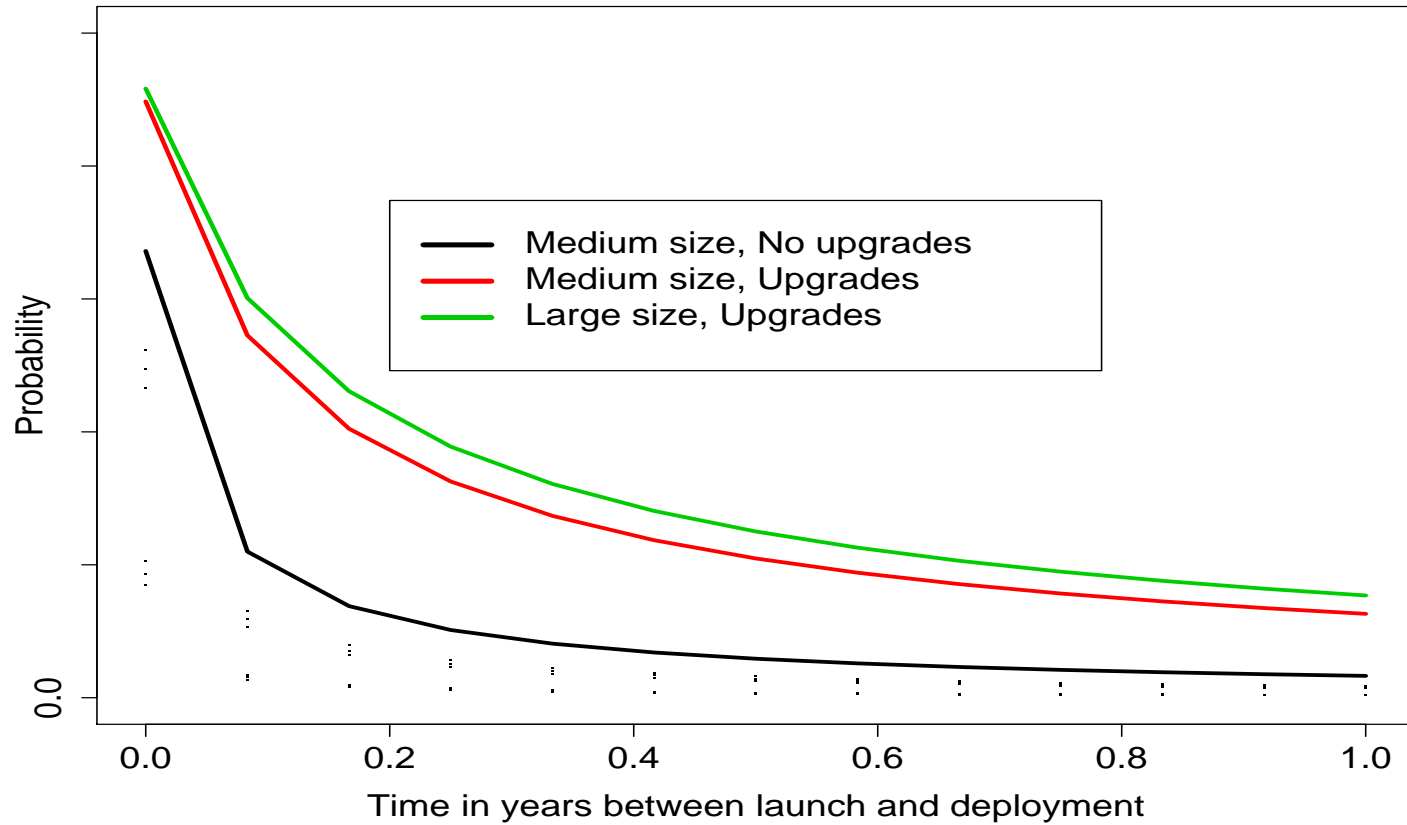
	Naive	Naive+	Alarming
Systems	80000	1011	761
Restarts	14000	32	32
Period	.5	.25	.25
MTBF (years)	3	7.9	5.9

What affects restart rates?



- ❖ Kaplan-Meier estimates of the survival curves for three platforms and two releases
- ❖ Differences between releases dwarfed by differences among platforms [8]

Probability of observing SW issue in the first 3 months



Quality:

- ◆ ↑ with time after the launch
- ◆ ↓ with Size
- ◆ ↑ for new installs

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] Birgit Geppert, Audris Mockus, and Frank Röbler. Refactoring for changeability: A way to go? In *Metrics 2005: 11th International Symposium on Software Metrics*, Como, September 2005. IEEE CS Press.
- [4] 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.
- [5] 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.
- [6] H.A. Malec. Communications reliability: a historical perspective. *IEEE Transactions on Reliability*, 47(3):333–345, Sept. 1998.
- [7] 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.
- [8] 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.
- [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 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.

- [12] Audris Mockus and David M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, April–June 2000.
- [13] Audris Mockus and David M. Weiss. Globalization by chunking: a quantitative approach. *IEEE Software*, 18(2):30–37, March 2001.
- [14] 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.
- [15] 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.
- [16] J. D. Musa, A. Iannino, and K. Okumoto. *Software Reliability*. McGraw-Hill Publishing Co., 1990.

Abstract

We investigate relationships among software quality measures commonly used to assess the value of a technology, and several aspects of customer perceived quality measured by Interval Quality (IQ): a novel measure of the probability that a customer will observe a failure within a certain interval after software release. We integrate information from development and customer support systems to compare defect density measures and IQ for six releases of a major telecommunications system. We find a surprising negative relationship between the traditional defect density and IQ. The four years of use in several large telecommunication products demonstrates how a software organization can control customer perceived quality not just during development and verification, but also during deployment by changing the release rate strategy and by increasing the resources to correct field problems rapidly. Such adaptive behavior can compensate for the variations in defect density between major and minor releases.

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>



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 the Software Technology Research Department of Avaya Labs. Previously he worked in the Software Production Research Department of Bell Labs.