Domain-Specific Defect Models

Audris Mockus

audris@avaya.com

Avaya Labs Research

Basking Ridge, NJ 07920

http://mockus.org/

Outline

- Fascination with defects
- Core issues in common approaches
- Assumptions used in defect models
- Domains and dimensions
- Costs and benefits
- Recommendations

Fascination with defects in SE

- How to not introduce defects?
  - Requirements and other process work
  - Modularity, high-level languages, type-checking and other LINT-type heuristics, garbage collection, . . .
  - 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?

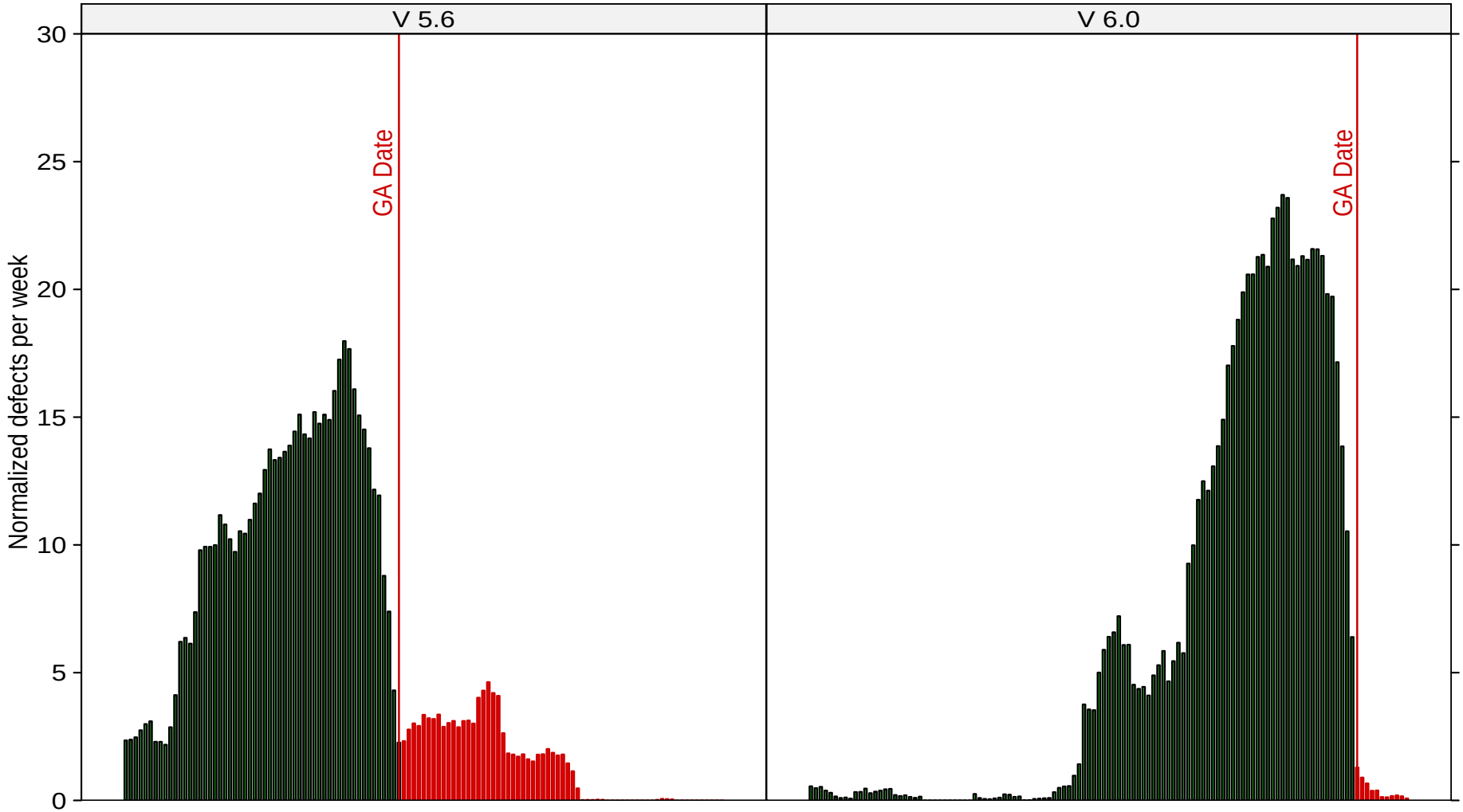
Some applications of defect models

- Faults remaining, e.g., [5], e.g., when to stop testing?
- Repair effort, e.g., development group will be distracted from new releases?
- **Focus QA on [where in the code] faults will occur**, e.g., [18, 6, 8, 19, 1, 17]
- Will a change/patch result in any faults [13]
  - such data are rare, may require identification of changes that caused faults [20]
- Impact of technology/practice on defects, e.g., [3, 2]
- Tools, e.g., [4, 21], benchmarking, e.g., [11], availability/reliability, e.g., [7, 16, 10]

State of defect prediction

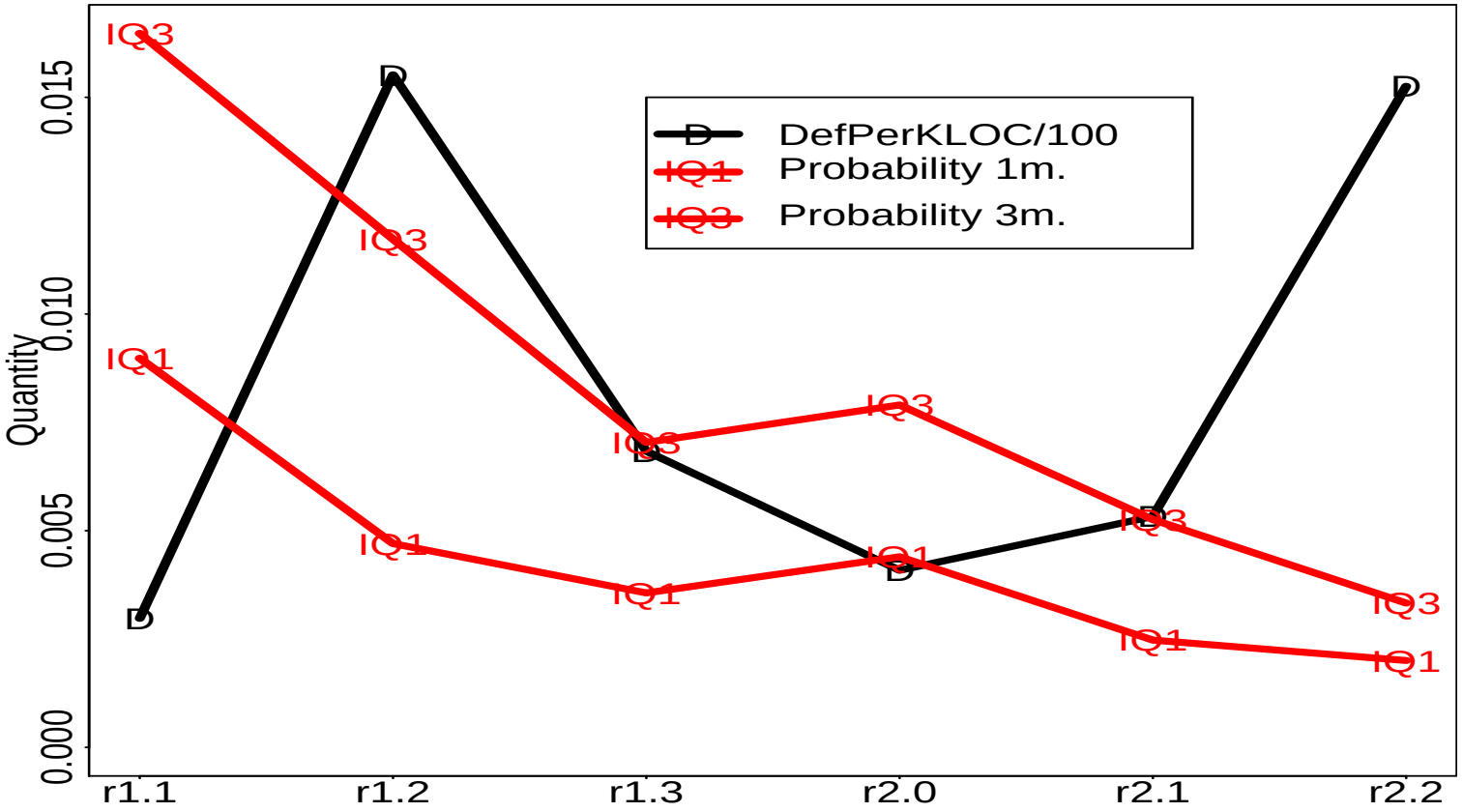
- Context: focus QA on modules that will experience faults post-release
- Almost impossible to beat past changes [6, 8]
- Use some measure of code size if change data is not available
- Other things that have been shown to matter: coupling (calls, MRs, organization, experience)
- What really matters tend to be outside the scope of software itself: the number of customers, configurations, installation date, release date, runtime [15, 12]
- Not clear if such models provide value
  - Even with perfect prediction the affected area is too large (exceeds release effort) for any meaningful QA activity

Post release defects for two releases



No defect prediction could handle these two releases!

Defect density and customer experiences?



Even if predictions of defects were perfect they would not reflect software quality as perceived by end users

Defect prediction — *perpetum mobile* of SE

- Why predictors do not work?
  - Defects primarily depend on aspects that have little to do with code or development process
  - Therefore, such predictions are similar to astrology
  - Hope that AI can replace human experts is premature
- Why people engage in irrational behavior, e.g., defect prediction?
  - The promise to see the future is irresistible.
  - The promise is phrased in a way the absurdity is well concealed.

How the deception is perpetrated?

- By not comparing to naïve methods, e.g., locations with most changes
- By not verifying that it provides benefits to actual developers/testers — “we test features not files” or “we need to have at least some clues what the defect may be, not where”
- By selecting misleading evaluation criteria, e.g, focusing on 20% of the code that may represent more than release-worth of effort
- By comparing Type I,II errors of a product with 40% rate to a product with 0.5% rate
- By suggesting impractical solution, e.g., how many SW project managers can competently fit an involved AI technique?
- By selecting complicated hard-to-understand prediction method, e.g., BN models with hundreds of (mostly implicit) parameters

Then why do it?!?//1111one/

May be to summarize the historic data in a way that may be useful for expert developers/testers/managers to make relevant design, QA, and deployment decisions?

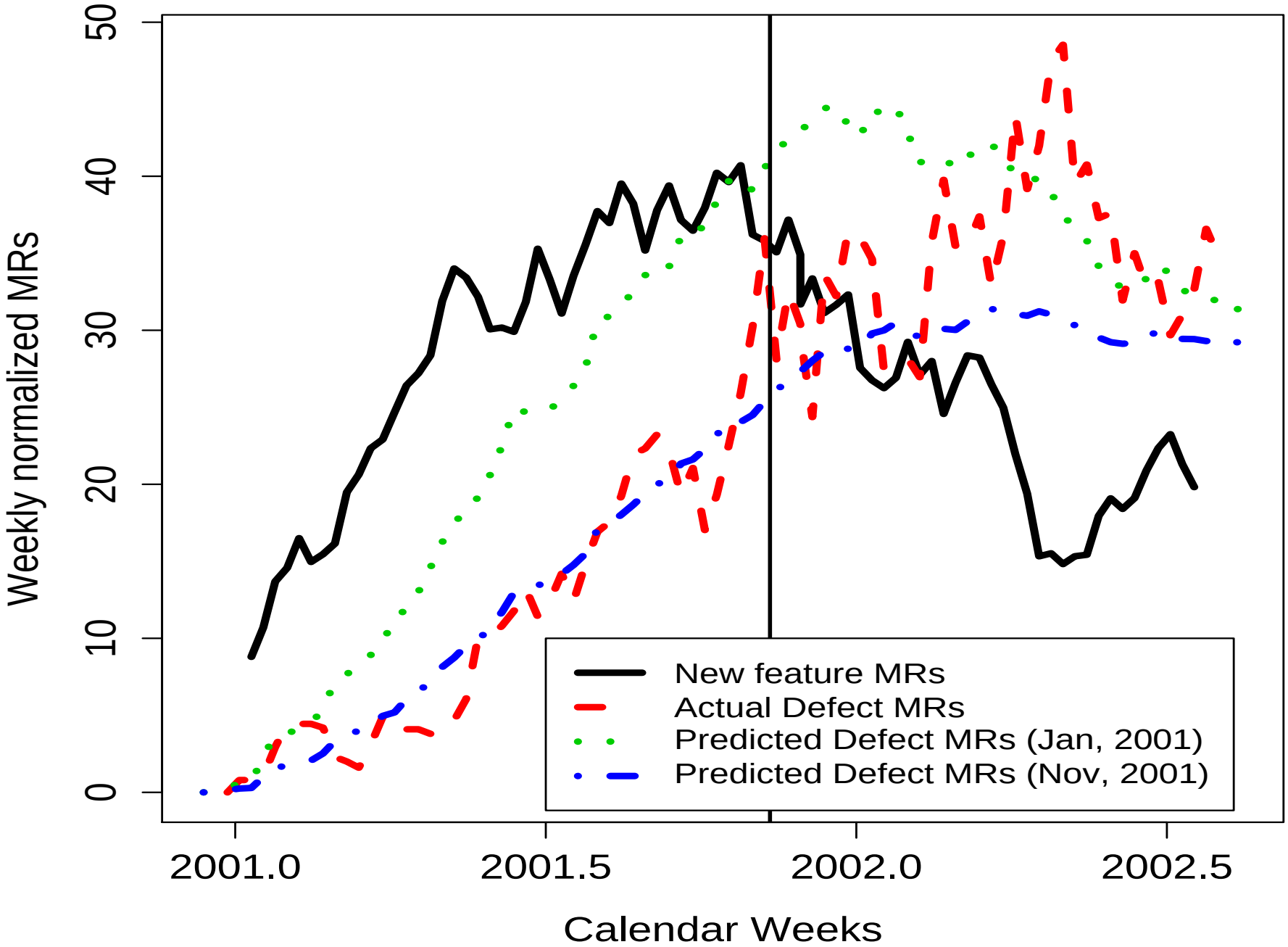
Some approaches used to model defects

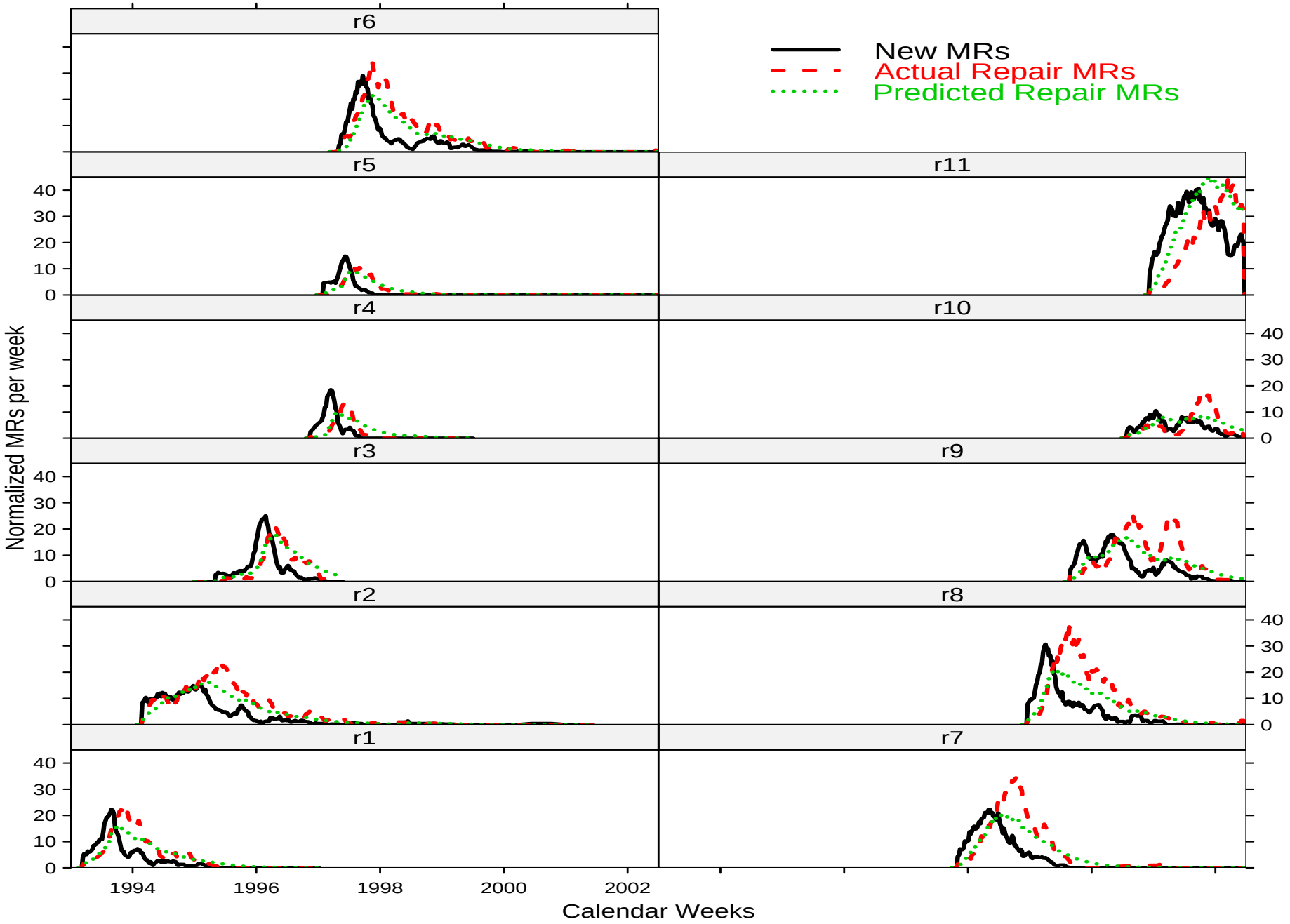
- Mechanistic: e.g., a change will cause a fault
- Invariants: e.g., ratio of post-SV defects to pre-SV changes is constant
- Data driven
  - All possible measures
  - principal components (measures tend to be strongly correlated),
  - fitting method
- Mixed: a mix of metrics from various areas that each has a reason to affect defects, but a regression or AI method are used to find which do

Mechanism to the extreme

- **Axiom 1:** a change will cause an average number of μ faults with average delay of λ [14]
  - Empirical relationship between changes and defects is well established
  - New features can only be predicted based on the business needs: use them as a predictor of fixes
  - The $-log(\text{Likelihood})$ is

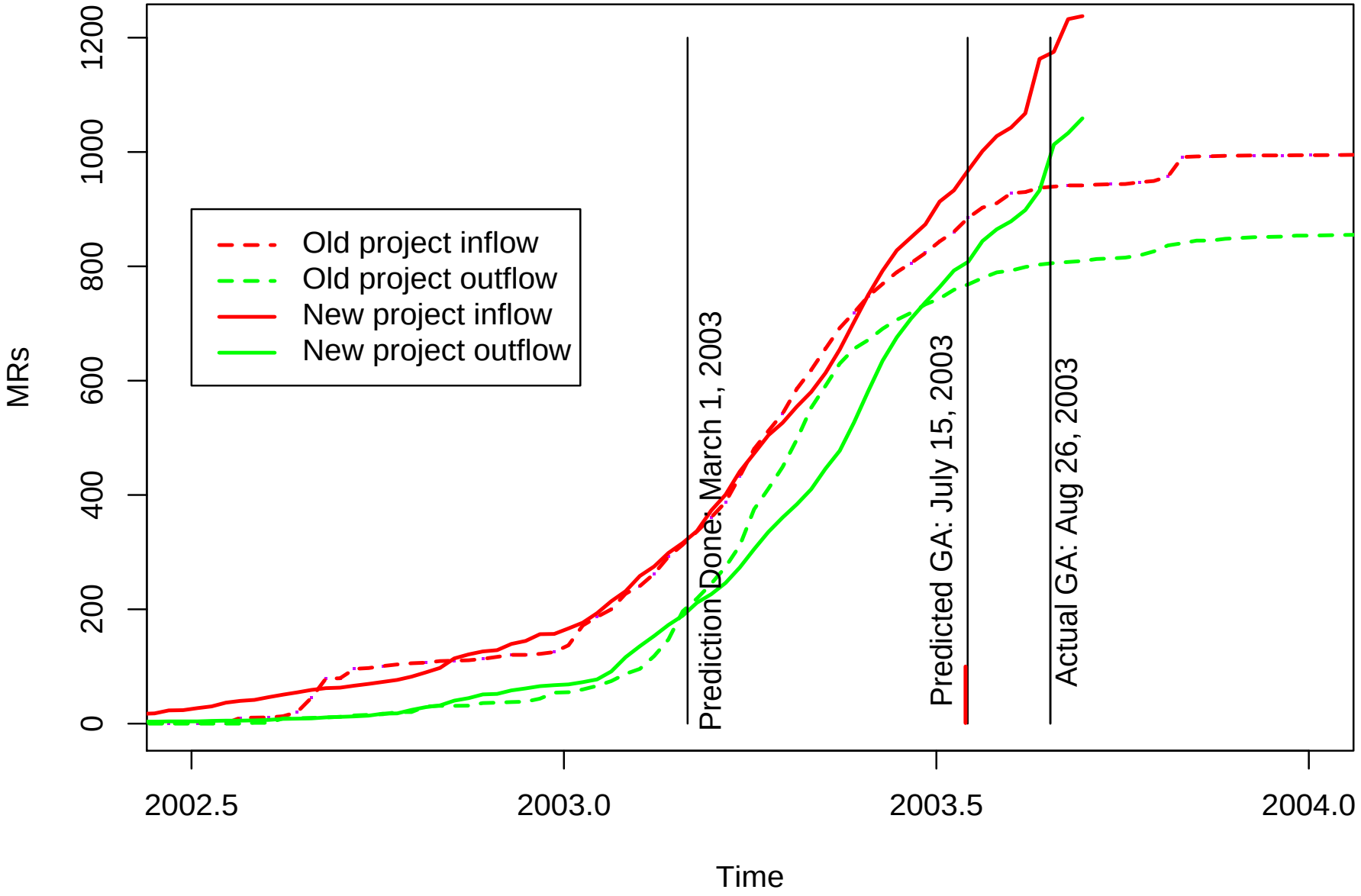
$$\sum_i \mu N_{t_i} \left(1 - e^{-\lambda(t-t_i)}\right) - B_{[0,t]} \log(\mu\lambda) - \sum_{s_k} B_{s_k} \log\left(\sum_{i:t_i<s_k} e^{-\lambda(s_k-t_i)}\right)$$


Weekly normalized MRs
0
10
20
30
40
50
2001.0
2001.5
2002.0
2002.5
Calendar Weeks
New feature MRs
Actual Defect MRs
Predicted Defect MRs (Jan, 2001)
Predicted Defect MRs (Nov, 2001)


r6
r5
r11
r4
r10
r3
r9
r2
r8
r1
r7
New MRs
Actual Repair MRs
Predicted Repair MRs
Normalized MRs per week
0
10
20
30
40
1994
1996
1998
2000
2002
Calendar Weeks

Invariance to the extreme

- **Axiom 2:** The history of MRs for release n will be a scaled and shifted version of the history of MRs for releases $n-1, n-2, \ldots$ [9]
  - Anything can be predicted: inflow, resolution, test defects, customer reported defects, number of people on the project, release date, effort ...


MRs
0
200
400
600
800
1000
1200
2002.5
2003.0
2003.5
2004.0
Time
Old project inflow
Old project outflow
New project inflow
New project outflow
Prediction Done: March 1, 2003
Predicted GA: July 15, 2003
Actual GA: Aug 26, 2003

Most common approach

- **Axiom 3:** $\exists f : \forall l, f(\mathbf{m}, l) = d(l)$ that given measures $\mathbf{m}$ will produce the number of defects $d(l)$ at location l
- $\hat{f}(\mathbf{m}, l) = \arg_f \min \sum_l \left(f(\mathbf{m}, l) - d(l)\right)^2$
- Common measures $\mathbf{m}$
  - Code measures: structural, OO, call/data flow
  - Process measures: change properties, age, practices, tools
  - Organization measures: experience, location, management hierarchy
  - Interactions: coupling, cohesion, inflow, outflow, social network measures for call/data flow, MR touches, workflow, ...

Locations l

- Lines, functions, **files**, packages/subsystems, entire system
- Functionality (features)
- Chunks — groups of files changed together
- Changes — MRs/work items and their hierarchy
- Geographic locations
- Organizational groups
- Tool/practice users

Defects d

- **Customer reported defects**
- Alpha/Beta defects
- Customer requested enhancements
- System test reported
- Found in integration/unit test/development
- Higher severity levels

What predictors may contribute?

- The value may not be in seeing the future but in understanding the past: gain insights
  - Formulate hypotheses
  - Create theories
  - Suggest ideas for tools or practices
- Focus QA
  - Instead of telling what files will fail, tools that help experts assess situation and evaluate actions may prove more useful
  - Need to find sufficiently small set and type of locations to match resources that could be devoted for QA
- Domain specific questions/analysis based on the cost-benefit analysis

Utility function: costs to repair

- What value will prevention bring?
  - Reduces costs to repair:
    - Domain: low cost for web service, high cost for embedded, heavy/large consumer products, aerospace
    - Number of customers: few customers can be served by the development group itself
  - Reduce cost of outage/malfunction:
    - Domain: low for desktop apps, high for aerospace, medical, or large time-critical business systems (banking, telephony, Amazon, Google)
    - Number/size of customers: fewer/smaller customers $\Longrightarrow$ less cost
  - Improve vendor reputation: personal reputation for internal products

Utility function: costs to prevent

- How costly to prevent?
  - Utility of the prediction in prevention
  - Ability to test/verify all inputs for all configurations
  - The size and complexity of l
  - Less cost for internal customer (more control over environment), web services (few installations), harder for a component of a complex real-time multi-vendor system with a large customer base
- Other considerations
  - Will quick repair of field problems count as prevention?
  - Cost of alpha/beta trials
  - Cost of testing
  - Cost of better requirements/design/inspections

Ultimately

Will prediction reduce prevention costs below the repair costs?

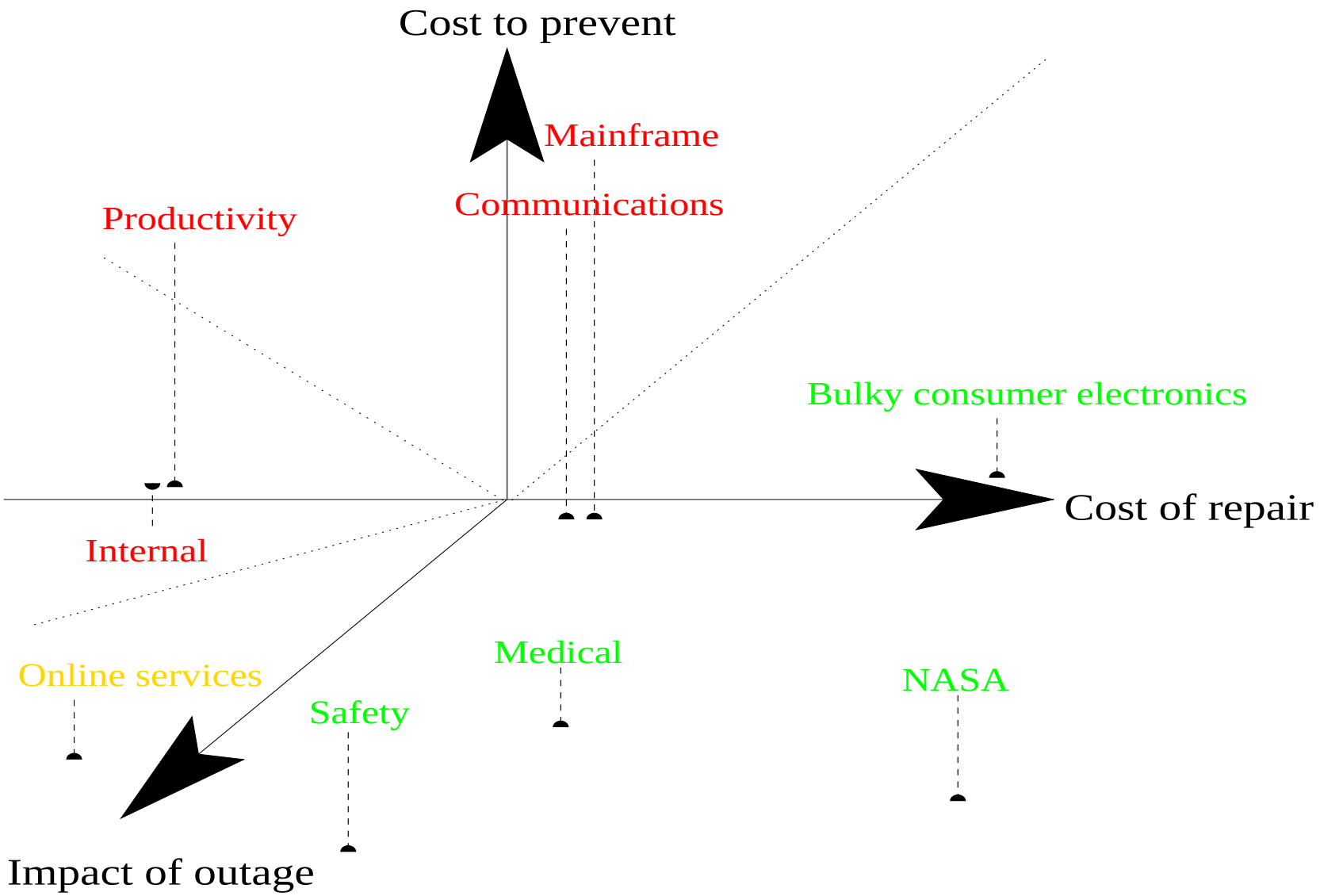
From domains to dimensions

- NASA: single use, limited replacement/update, errors critical, often completed by contractors
- Online services: few installations, many users, costly downtime
- Consumer devices: many users, expensive to replace somewhat alleviated by Internet connectivity
- Internal projects: single user, no difference between testing and post-SV
- Office productivity: many users, error cost absorbed by end users
- Switching/servers: downtime costly, but software easy to update
- Mainframes - availability, no downtime

Relevant dimensions

- Cost of defects
  - Quality requirements: medical, mainframe, productivity
  - Numbers of users
- Cost of prevention
  - Scale/complexity of software
  - Complexity of the operating environment: e.g., multi-vendor
  - Resources needed to test/inspect/fix
- Cost of repair
  - Few, internal users/installations
  - Easy/inexpensive to upgrade
- Other considerations
  - How accurate (similar releases/configurations/custumer set)
  - What data is available: contractors may not share change data

Which domains are likely to benefit?



Recommendations

- Resist the urge to be an astrologer, no matter how sophisticated the technique
  - Use defect models to quantify the effect of methods/tools on quality improvements
  - Consider relevant dimensions and the utility if predicting
  - Compare to naïve/simple predictors
  - Stay, if possible, with a transparent evaluation criteria
- Summarize the historic data in a way that may be useful for expert developers/testers/managers to make relevant design, QA, and deployment decisions

References

[1] Erik Arisholm and Lionel C. Briand. Predicting fault-prone components in a java legacy system. In *International Symposium on Empirical Software Engineering*, pages 8 – 17, 2006.

[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] V.R. Basili, L.C. Briand, and W.L. Melo. A validation of object-oriented design metrics as qualityindicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, Oct 1996.

[4] D. Cubranic and G.C Murphy. Hipikat: A project memory for software development. *TSE*, 31(6), 2005.

[5] S. R. Dalal and C. L. Mallows. When should one stop testing software? *Journal of American Statist. Assoc*, 83:872–879, 1988.

[6] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering*, 26(2), 2000.

[7] J. Jelinski and P. B. Moranda. Software reliability research. In W. Freiberger, editor, *Probabilistic Models for Software*, pages 485–502. Academic Press, 1972.

[8] T. M. Khoshgoftaar and N. Seliya. Comparative assessment of software quality classification techniques: An empirical case study. *Empirical Software Engineering*, 9(3):229–257, September 2004.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[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] John D. Musa, Anthony Iannino, and Kazuhira Okumoto. *Software Reliability: Measurement, Prediction, Application.* McGraw-Hill Book Company, 1987. ISBN: 0-07-044093-X.

[17] Nachiappan Nagappan, Brendan Murphy, and Victor R. Basili. The influence of organizational structure on software quality: an empirical case study. In *ICSE 2008*, pages 521–530, 2008.

[18] N. Ohlsson and H. Alberg. Predicting fault-prone software modules in telephone switches. *IEEE Trans. on Software Engineering*, 22(12):886–894, December 1996.

[19] Thomas J. Ostrand, Elaine J. Weyuker, and Robert M. Bell. Predicting the location and number of faults in large software systems. *IEEE Trans. Software Eng.*, 31(4):340–355, 2005.

[20] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *Proceedings of the 2005 international workshop on Mining software repositories*, pages 1 – 5, 2005.

[21] Annie Ying, Gail Murphy, Raymond Ng, and Mark Chu-Carroll. Predicting source code changes by mining change history. *IEEE Transactions of Software Engineering*, 30(9), 2004.

Abstract

Defect prediction has always fascinated researchers and practitioners. The promise of being able to predict the future and acting upon that knowledge is hard to resist. Complex models used to perform the predictions and the lack of fair comparisons to what may happen in practice obscure the core assumption that quantitative methods using generic measures can improve upon decisions made by people with intimate knowledge of the project. We consider how defect analysis techniques may be beneficial in a domain-specific context and argue that more explicit and more realistic objectives that address practical questions or further deeper understanding of software quality are needed to realize the full potential of defect modeling. This can be achieved by focusing on issues specific to a particular domain, including the scale of software and of user base, economic, contractual, or regulatory quality requirements, and business models of software providers.

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.