Software Changes: from Insights to Solutions

Audris Mockus

Avaya Labs Research

Basking Ridge, NJ 07920

http://www.research.avayalabs.com/user/audris

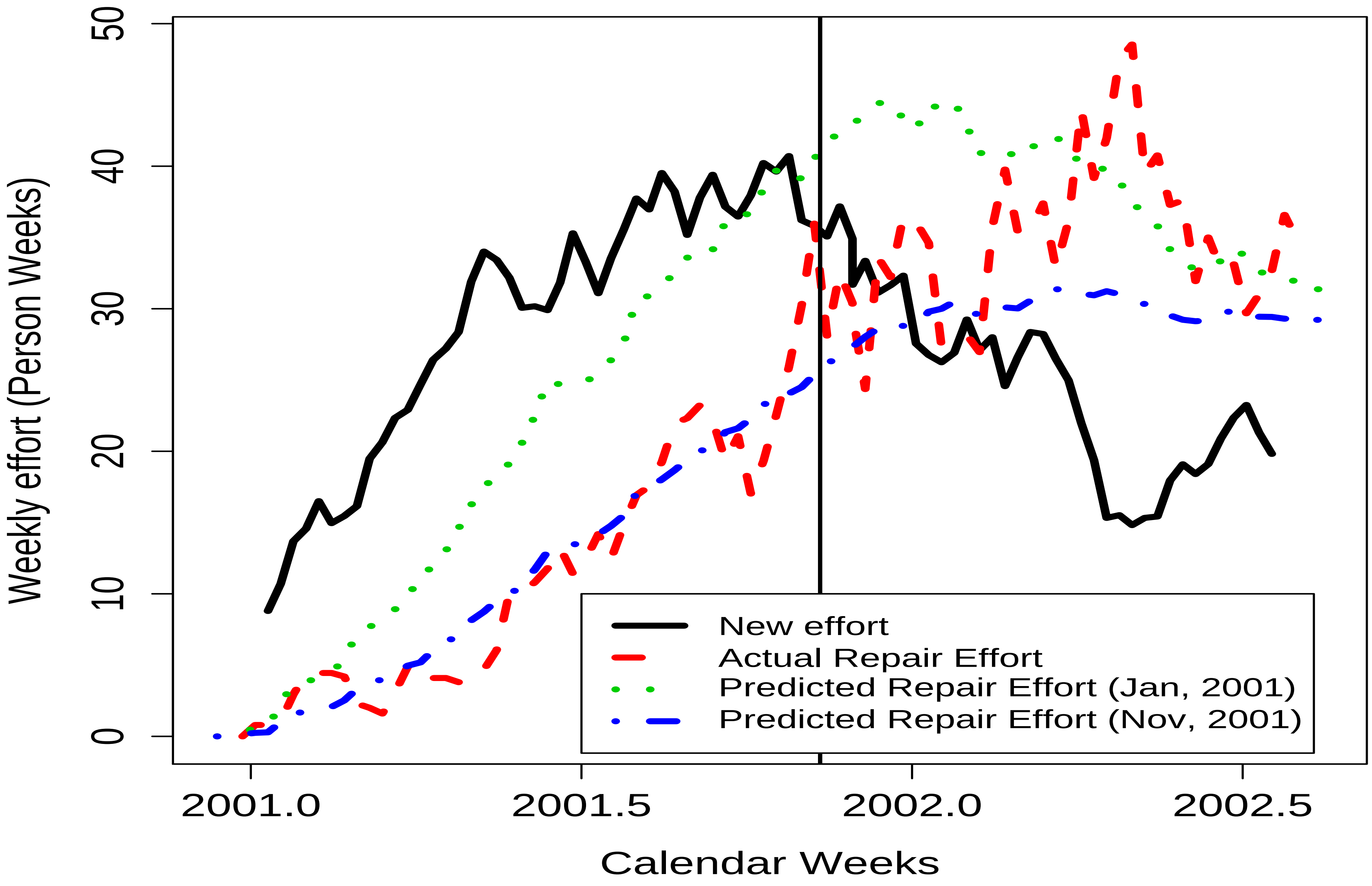
Outline

- Background
  - Motivation
  - Software project repositories
  - How to use change data
- A model of software project
  - Predicting schedule
  - Predicting post-release defects
- Discussion

Motivation

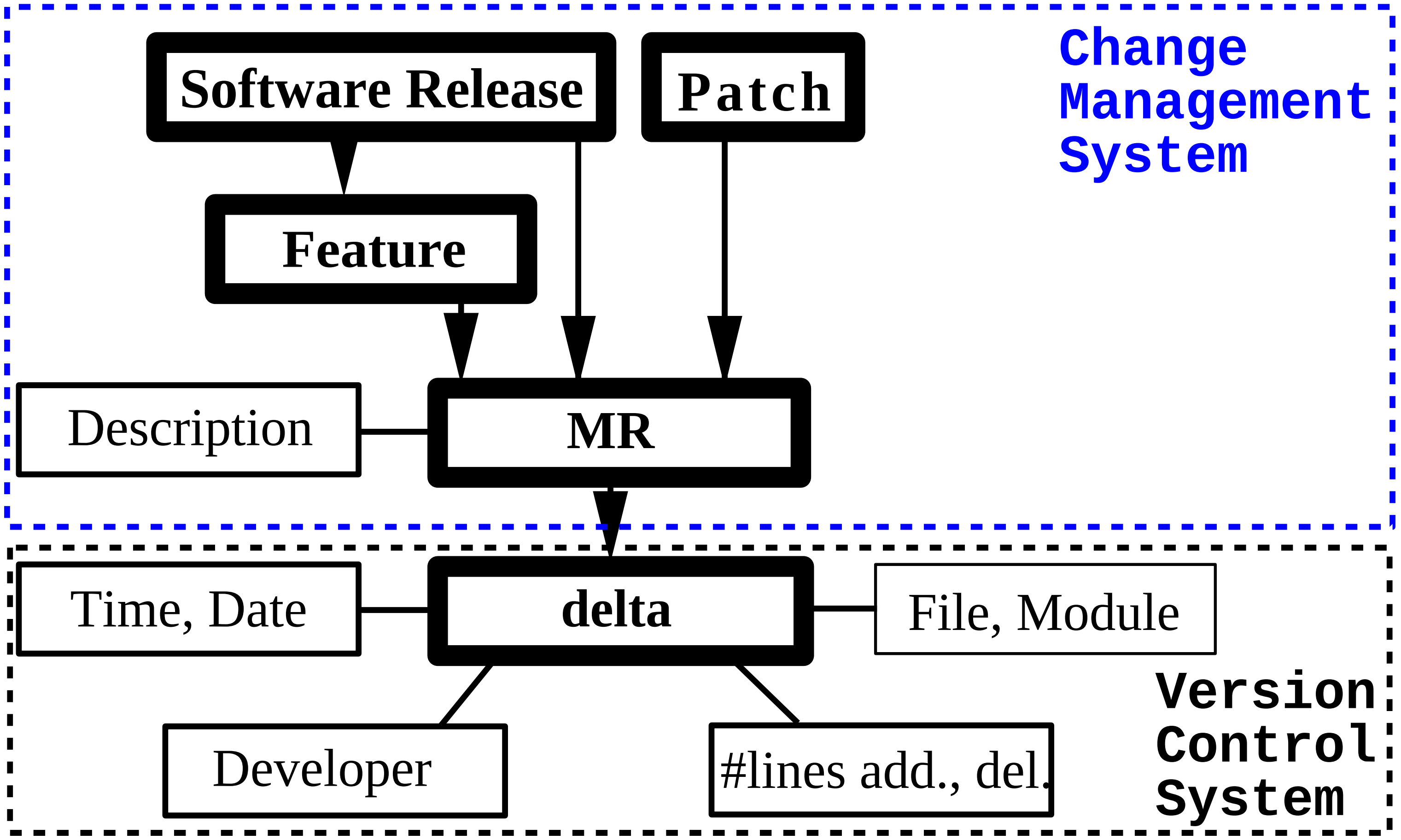
- To quantify software production: make informed trade-offs between schedule, quality, cost.
  - Visibility: where/when effort is spent, defects introduced
  - Predictability: what will be the impact of choosing technology, processes, organization
  - Controllability: trade-offs between time to market,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

Example: Release Dates



Background: Illustration

- Software is created by changes
- Changes are tracked



Background: Details

- Software is created by changes
- Changes are tracked by version control/configuration management systems (VCS/CMS)
  - A delta is a single checkin (ci/commit/edput) representing an atomic modification of a single file with following attributes
    - File, Date, Developer (resolver), Comment
  - Other attributes:
    - Size (number of lines added,deleted)
    - Lead time (interval from start to completion)
    - Purpose (Fix/New)
    - Reporter, date reported
- Modification Request (MR): a group delta from a logical change
- Patch, Release: a group of MRs released to users

Approach

- Use properties and relationships among changes to model phenomena in software projects
  - Obtain change properties from project repositories (VCS/CMS)
  - Model staffing/schedule/quality relationships to decide upon future changes
  - The product/code is simply a dynamic superposition of changes, and is not of particular interest otherwise

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, ...
- 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

Existing Models

- Predicting the quality of a patch [7]
- Globalization: move development where the resources are:
  - What parts of the code can be independently maintained [8]
  - Who are the experts to contact about any section of the code [5]
- Effort: estimate MR effort and benchmark process
  - What makes some changes hard [3]
  - What processes/tools work [1, 2]
  - What are OSS/Commercial process differences [4]
- Project models
  - **Release schedule** [9]
  - Release readiness criteria
  - Release quality

Change Data Methodology: Extraction

- Get access to the systems
- 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)
- 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 [6], effort [1], risk [7]
- Validate and clean recorded and modeled data
- Iterate

Change Data Methodology: Project Sample

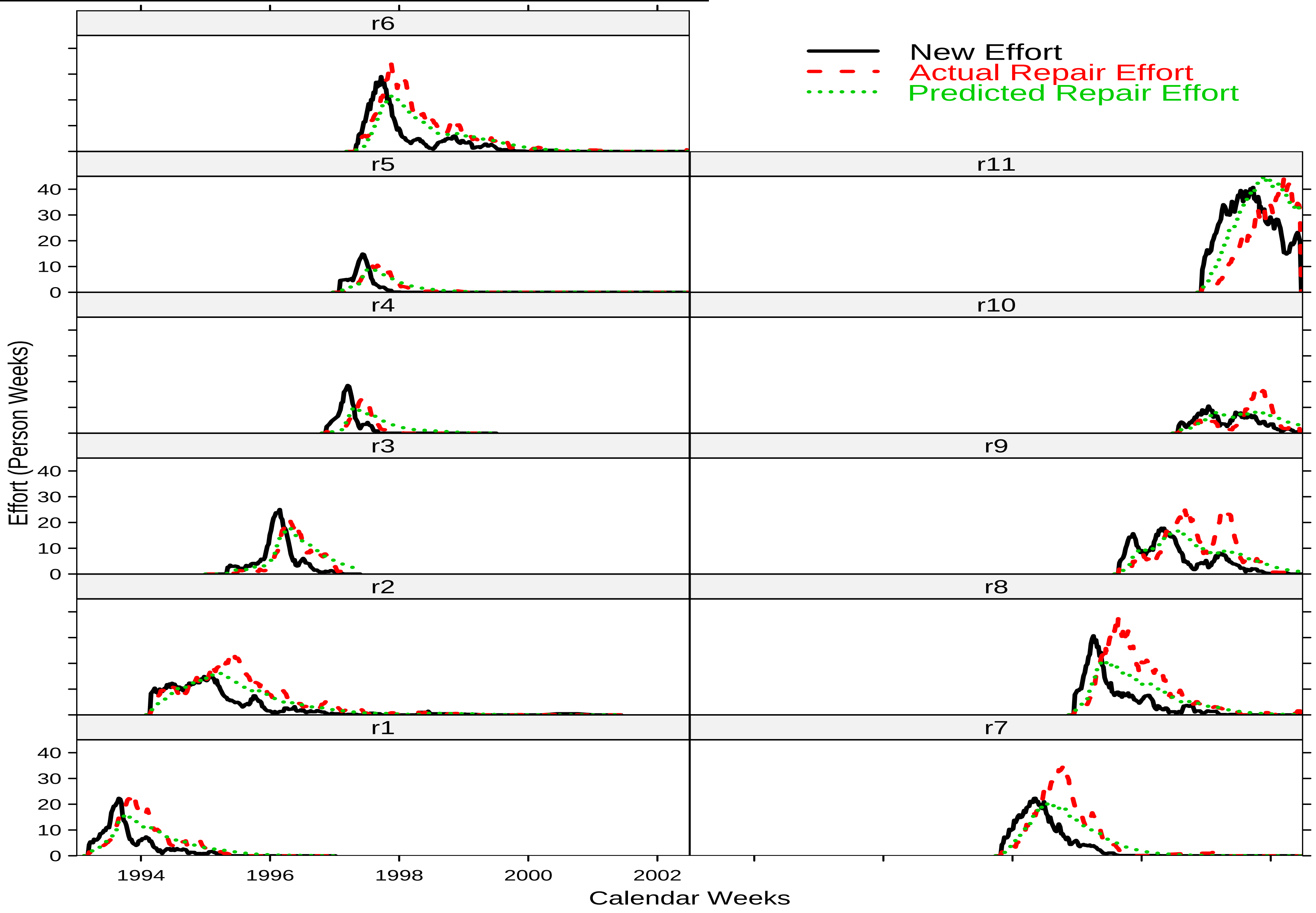
- *Languages*: Java, C, SDL, C++, JavaScript, XML, ... *Platforms*: proprietary, unix'es, Windows, VXWorks, *Domains*: embedded, high-availability, network, user interface *Size*: from largest to small

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	

Software Project Expressed through Changes

- Project consists of two types of changes
  - business driven changes — planned new feature/platform changes
  - consequences — repair changes due to incorrect implementation of new features or unanticipated interaction or novel exercise of “base” functionality
- Assume “modification $\Longrightarrow$ repairs later”
  - A unit of effort spent on new “planned” changes generates B units of repair effort with delay T
  - Choose appropriate distribution for B and T
    - $B \sim \mathrm{Poisson}(\mu)$
    - Times until each unit of repair effort is spent are IID $T \sim \mathrm{Exponential}(\lambda)$

Model Fit: 11 Releases



Model Details

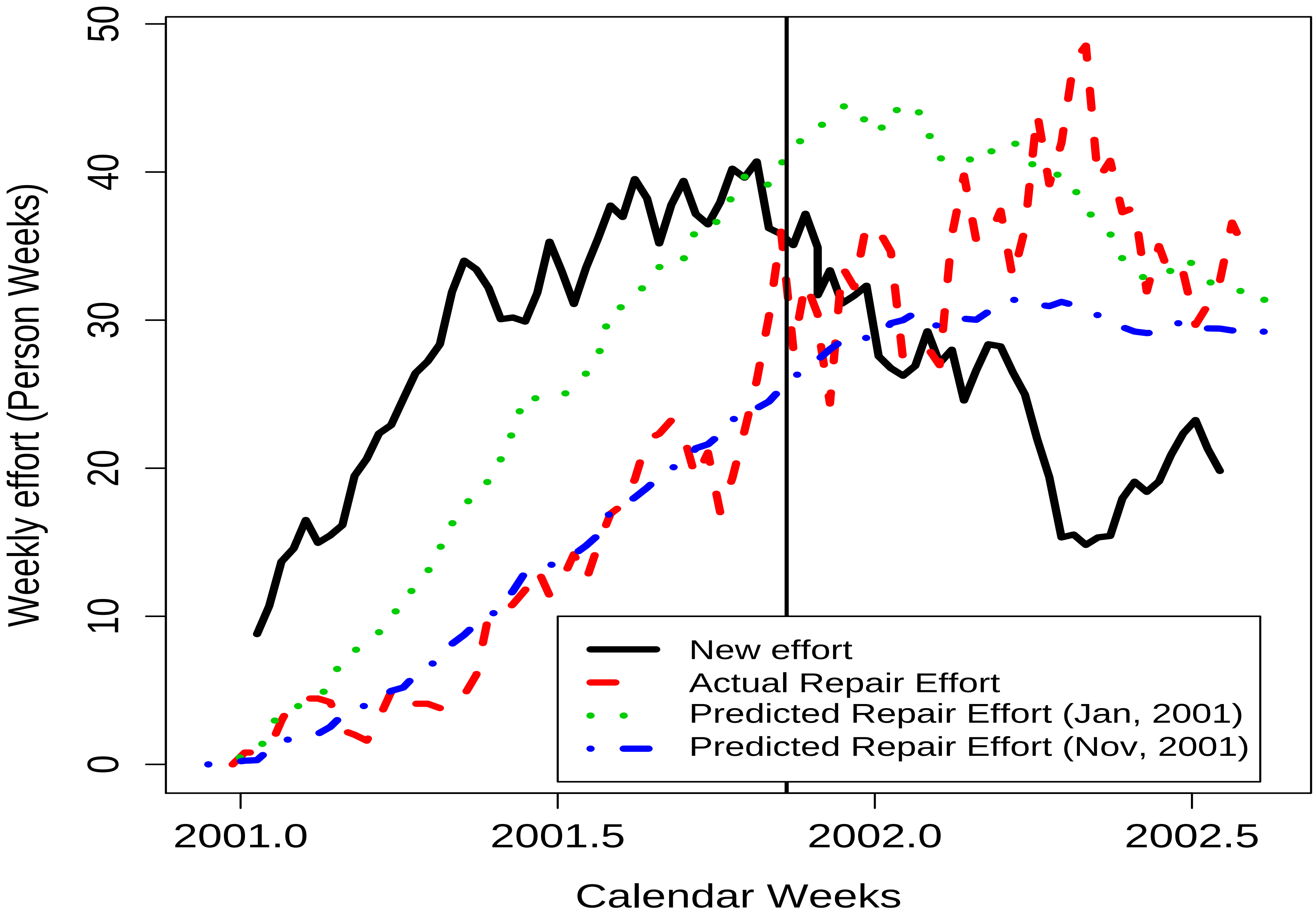
- Notation
  - Denote N_{t_i} the number of new feature effort units at time t_i, and, similarly B_{s_k} for fixes. Denote $B_{[a,b]}$ to be repair effort units on interval $[a, b]$
  - Project data (N_{t_i} and B_{s_k}) are observed until time t
  - No direct links between changes are observed
- The $-log$(Likelihood) is

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

Release Planning

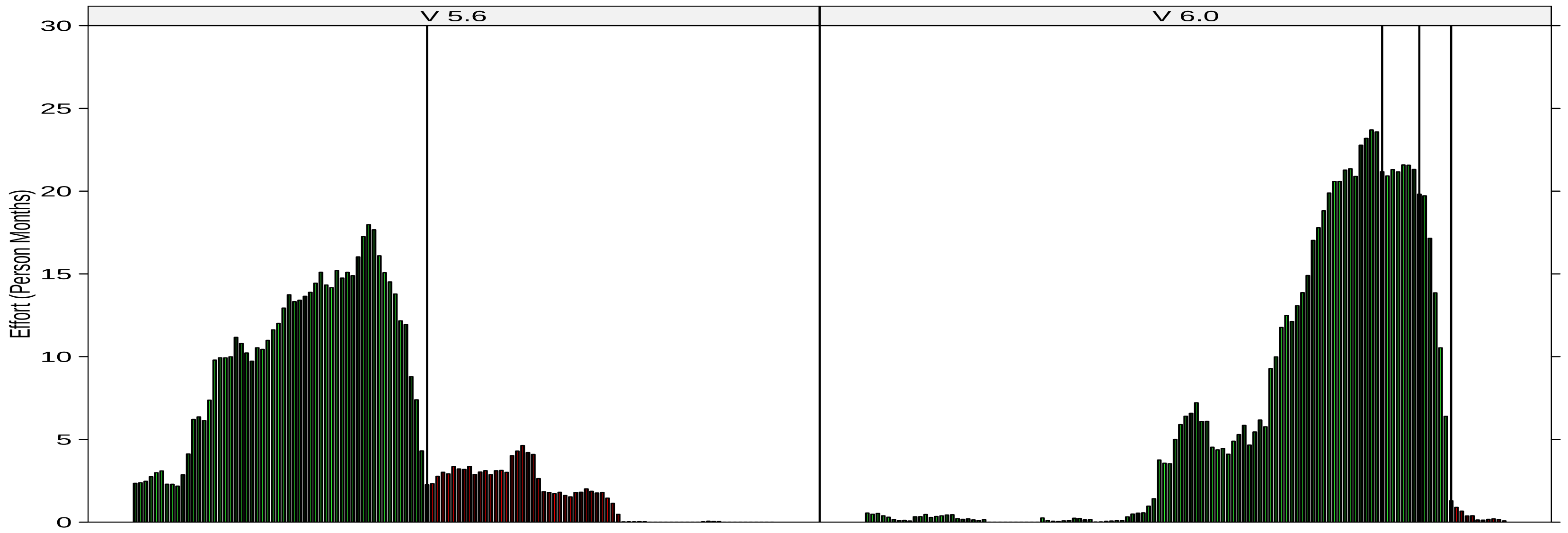
- Goal: Model tradeoffs between release feature content, schedule, staffing needs, and quality
  - stakeholders provide new feature content, release dates, staffing, and quality goals
  - repair (and total) schedule is predicted
  - new feature content, release dates, staffing, and quality goals are revised
- Results
  - $\hat{\mu}$ shows the fraction of fix to new effort.
  - $\frac{1}{\hat{\lambda}}$ shows mean time until fix.
  - $(\hat{\mu}_{regular}, 1/\hat{\lambda}) = (1.4, 19) \pm (0.2, 2)$
  - $(\hat{\mu}_{field}, 1/\hat{\lambda}) = (1.7, 41) \pm (0.3, 7)$

Predicted Schedule: Ongoing Project



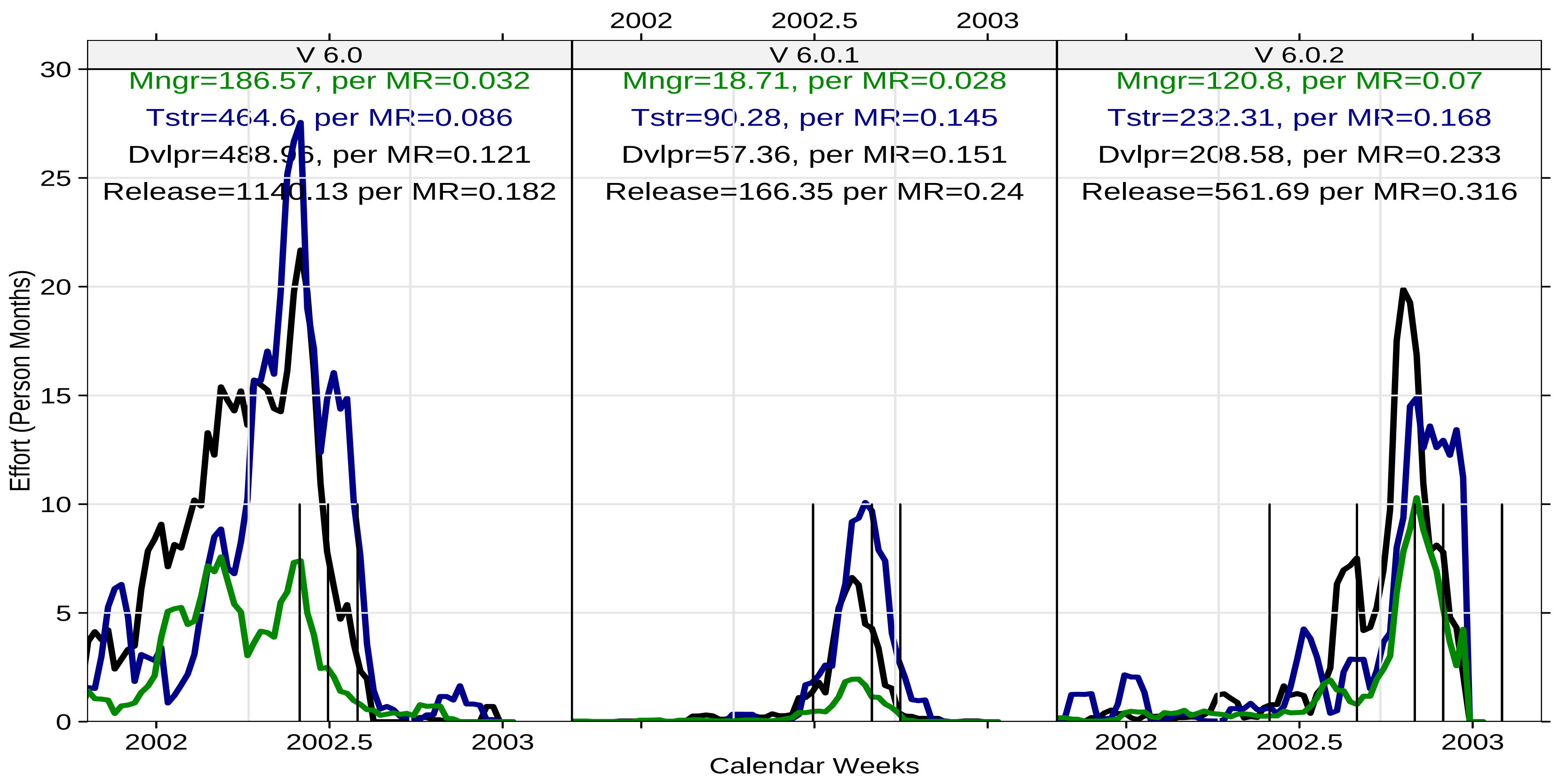
Application: Release Quality Criteria

- Goal: what criteria need to be satisfied for a release to match user expectations?
- Inputs
  - Schedule of defects and release dates in previous projects
  - Number of unsolved MRs in previous and current project at GA date
  - Schedule of deployment



Application: Testing Effort

- Goal: predict the amount and schedule of testing effort
  - CM data is used to identify testers, management
  - Relationships between development and testing from historic projects are used for planning new projects



Discussion

- Unified project model: predicting schedule, quality, and effort
  - Input: information that is known or should be known in advance
  - Output: likely consequences
- Change data represents a vast amount of untapped resources
- Remaining challenges
  - Broader application and validation of existing models
  - New models to address other problems of practical/theoretical significance
  - What information developers would easily and accurately enter in a CM systems?
  - What is the “sufficient statistic” for a software change?

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] 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.

[4] 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.

[5] 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.

[6] 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.

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

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

[9] Audris Mockus, David M. Weiss, and Ping Zhang. Understanding and predicting effort in software projects. In *2003 International Conference on Software Engineering*, Portland, Oregon, May 3-10 2003. ACM Press. Accepted.

Abstract

Software systems are changed constantly throughout their lifetime. Understanding relationships between different types of changes and the effects of these changes on the success of software projects is essential to make progress in Software Engineering. By using novel methods and tools to retrieve, process, and model data from ubiquitous change management databases at the granularity of Modification Requests (individual changes to software) we have gained insights regarding the relationships between process/product factors and key outcomes, such as, quality, effort, and interval. Here we introduce ways to use changes to understand and predict the state of an entire software project. We propose a model based on the premise that each modification to software will cause changes later and investigate its theoretical properties and applications to several software projects. The model presents a unified framework to investigate and predict effort, schedule, and defects of a software project. The results of applying the model confirm a fundamental relationship between the new feature and defect repair changes and demonstrate model's predictive capabilities in large 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.