Effects of Distributed Software Development and Virtual Teams

Audris Mockus

audris@avaya.com

Avaya Labs Research

Basking Ridge, NJ 07920

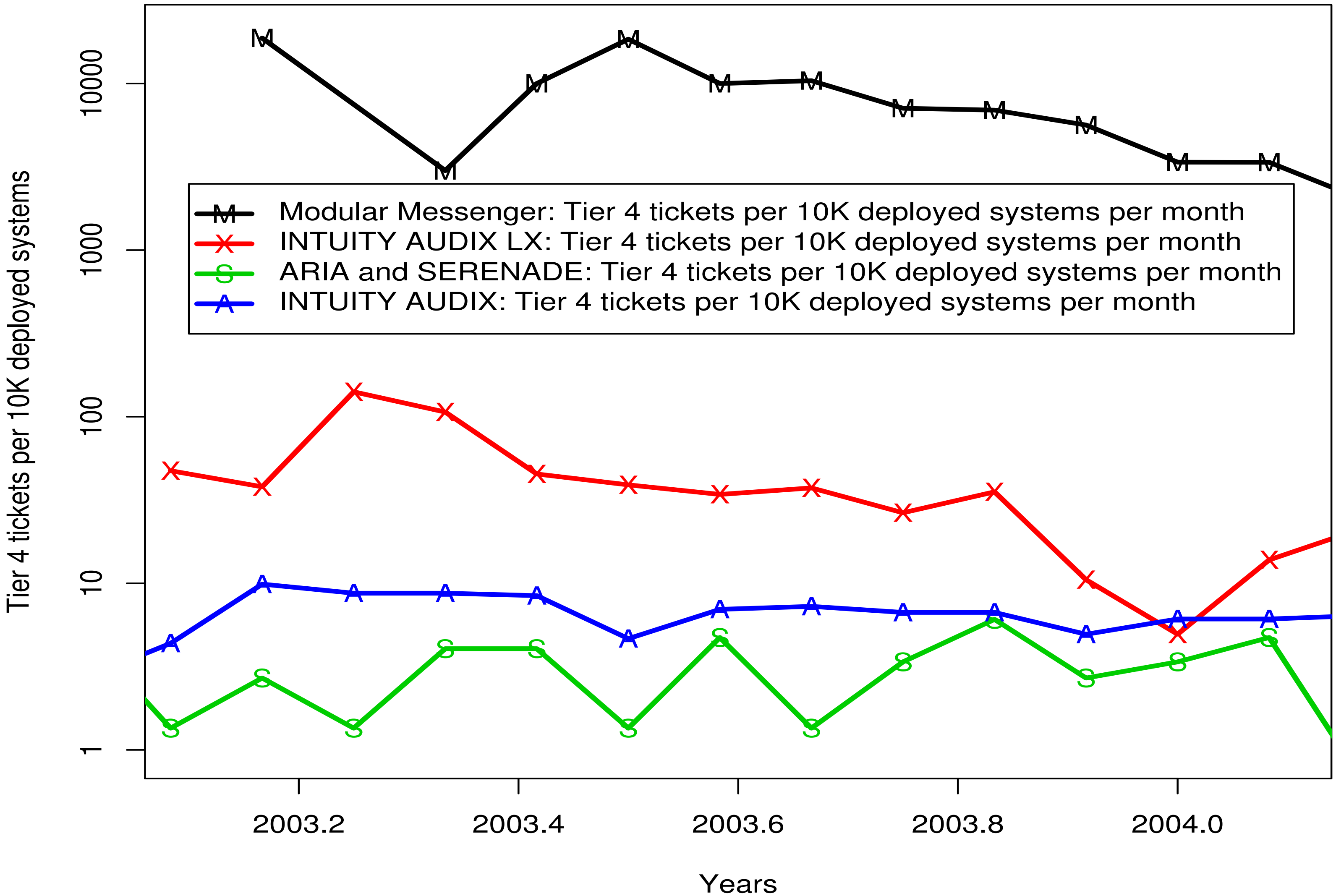
http://www.research.avayalabs.com/user/audris

Motivation

- Software project with a lack of common mental model
  - Sites: two primary sites with opposing views
  - Perceived user needs: reliability, maintainability, availability, backward compatibility, and cost versus get something out quick, fix it later, what are these -ilities anyway
  - Platform: a windows shop with no clue beyond windows versus embedded, unix/linux shop with concerns about portability and performance
  - User base: used to support tens of thousand of customers versus we'll make a patch for your system if you have problems
- Management by compromise
  - Two box system (a box for each team)
  - Constantly revisiting decisions, each party tries to prove others wrong

Outcome

Tier 4 tickets per deployed system per month are 100 to 1000 times higher for MI



Outline

- Definitions and method
  - Virtual teams
  - Observing (estimating) interdependence
  - Observing commonness of the goals
- Some empirical evidence
  - Methodology
  - OSS vs mixed projects
  - Dealing with multiple people
  - Not complying with existing design

Virtual Teams

- Groups of people whose work is interdependent
  - Not necessarily collocated
  - Not necessarily interact
  - Not necessarily know each other
  - Not necessarily overlap in time
  - Not necessarily want to work together

Observing (Estimating) Interdependence (Teams)

- Work items implement decisions/choices that are tightly interdependent, therefore following form teams
  - People involved in the same work item
- The artifact (code) is the expression of all decisions taken by individuals and teams, therefore following form teams
  - People involved in work items that operate on the same artifacts (lines, files, modules, chunks)

Having common goals

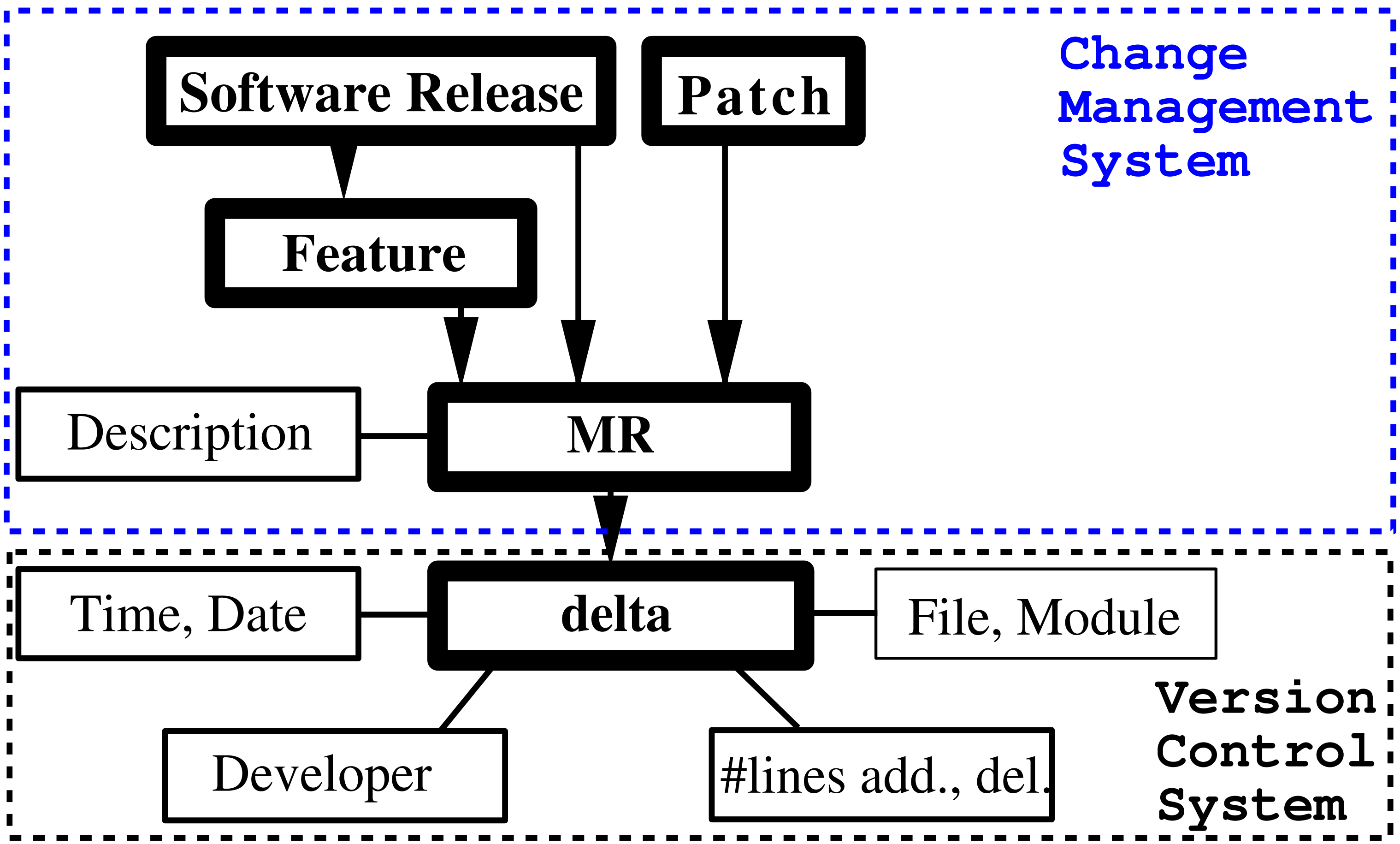
- Common mental model, common understanding of the relevant part of the world
  - Concept of the user and user needs and wants
  - Concept of the product, its behavior and its -ilities
  - Concept of development, delivery, and support processes

Observing (Estimating) the Commonness of Goals

- Many factors affect it
  - The nature of participants' motivation (compensation, pleasure, etc.)
  - The size of the team
  - Common code base, version control, problem tracking, process/decision making
  - Multiple sites (language/culture/communication)
- Self selection by their common goals in some OSS (no other motivation is apparent)
- Product/process may be affected by the lack of it
  - Moderated by the degree of interdependence

Empirical Methodology: Assumptions

- Software is created by by work items or changes
- Changes are tracked to enable multiple people to work on them



Methodology: 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

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

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 [8], effort [1], risk [9]
- Validate and clean recorded and modeled data
- Iterate

Methodology: 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

Methodology: 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

Methodology: Existing Models

- Predicting the quality of a patch [9]
- Work coordination:
  - What parts of the code can be independently maintained [10]
  - Who are the experts to contact about any section of the code [7]
  - How to measure organizational dependencies [3]
- Effort: estimate MR effort and benchmark process
  - What makes some changes hard [4]
  - What processes/tools work [1, 2]
  - What are OSS/Commercial process differences [6]
- Project models
  - Release schedule [11]
  - Release readiness criteria [5]
  - Consumer perceived quality

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	

Evidence 1: Receiving work from multiple people decreases productivity

More people may imply dissimilarity of goals

Variable	Coeff.	Std. Error	p-val
Intercept	6.4	0.96	.001
self	0.47	0.18	.01
in	1.16	0.32	.001
out	0.42	0.82	.6
inDegree	-2.1	0.68	.006
outDegree	-1.1	1.4	.41

Dependent Variable: productivity, defined as MRs/week [3]

Evidence 2: Making changes across chunk boundaries takes longer

Each chunk implies a module, change across modules indicates unanticipated goals

Variable	Coeff.	Std. Error	p-val
Intercept	11.3	0.24	.001
Other	2.46	0.10	.001
nReleases	1.04	0.11	.001
NFiles	0.18	0.05	.001
Multi-chunk	0.41	0.19	.027

Dependent Variable: MR elapsed time: first change to last change [10, 3]

Apache/Mozilla Development Process

- Apache (most common goals)
  - No external motivation (self selection by goals)
  - Small core team
- Mozilla (somewhat less common goals)
  - Most developers compensated
  - Large core team
- Commercial projects (varies)
  - All developers compensated, although there is some self selection based on expertise
  - Typically larger teams, involving non developers
  - In case of multiples sites often different tools/process apply

Evidence 3: Productivity

- Compare sets of developers that produced 80% of the code in each application
- A-E: similar-sized commercial projects

	Ap./Moz	A	B	C	D	E
KMR/developer/year	$.11 \pm .5$	.03	.03	.09	.02	.06
KLOC/developer/year	4.3/6-16	38.6	11.7	6.1	5.4	10

Evidence 4: Defect Density

- Measures
  - Post release and post-feature test
  - Per KLOC added and per thousand Delta

	Ap./Moz.	A	C	D	E
Post-release Defects/KLOCA	2.6±1.4 / 1	.11 / 24	0.1 / 2.6	0.7 / 3.8	0.1 / 26
Post-release Defects/KDelta	40±20 / 1	4.3 / 9.5	14 / 2.9	28 / 1.5	10 / 5
Post-feature test Defects/KLOCA	2.6±1.4 / 1	*	5.7 / .5	6.0 / .4	6.9 / .4
Post-feature test Defects/KDelta	40±20 / 1	*	164 / .25	196 / .2	256 / .16

Discussion

- Virtual team: people whose work is interdependent
  - Two methods to identify such teams
- Common goals
  - In certain organization (some OSS projects), people may self select based on what they want to accomplish
  - Different sites tend to have dissimilar goals
  - Product/process should be negatively affected when there is a lack of common goals

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 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

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

[11] 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.

Abstract

Software development by distributed teams often results in delays, inefficiencies, and misunderstanding. Proposed explanations range from differences in cultural background to the lack of face-to-face and informal communication needed to coordinate interdependent tasks. Analyzing software project repositories we reconstruct virtual teams and investigate how the lack of common goals and development infrastructure within these teams lead to problems in distributed software development. More specifically, we consider a range of traditional and open source projects where the commonality of goals and infrastructure varies across teams and relate that to measures of productivity, interval, and quality.

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.