Measurement in software projects: taking advantage of version control repositories

Audris Mockus

Avaya Labs Research

Basking Ridge, NJ 07920, USA

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

Outline

- Background
  - Motivation and challenges
  - Software project repositories
- Advantages of using project repositories
- Pitfalls of using project repositories
- Models for common SE problems
- Process for using project data
- Discussion

Motivation and Challenges

- Needs
  - Understand and improve software practice
    - Need for quantitative estimates to understand limitations and to make informed tradeoffs between schedule, quality, cost.
      - Visibility: where effort is spent, where defects are introduced
      - Actions: the impact of technologies/processes/organization
- Key issue - lack of trust in software measurement
  - Low priority except in emergencies
  - Need for immediate results (short time horizon)
  - Lack of resources for measurement/improvement
  - Multiple stakeholders (developer/support/product management)
  - Accelerated turnover rate (“just in time employment”)

Background

- Software is created by making changes to it
  - A delta is a single checkin (ci/commit/edput) representing an atomic modification of a single file with following attributes
    - File, Date, Developer, Comment
  - Other attributes that often can be derived:
    - Size (number of lines added,deleted)
    - Lead time (interval from start to completion)
    - Purpose (Fix/New)
- Approach
  - Use project’s repositories of change data to model phenomena in software projects

Advantages of project repositories

- The data collection is non-intrusive (using only existing data minimizes overhead)
- Long history on past projects enables historic comparisons, calibration, and immediate diagnosis in emergency situations.
- The information is fine grained, at the 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 making it possible to detect even small effects statistically.
- The version control system is used as a standard part of the project, so the development project is unaffected by experimenter intrusion

Pitfalls of using project repositories

- Different process: how work is broken down into work items
- 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 models of key problems in software engineering based change data

Models of software changes

- Quality: how to keep customers happy with minimal resources [6]
- Globalization: move development where the resources are:
  - What parts of the code can be independently maintained [7]
  - Who are the experts to contact about any section of the code [5]
- Effort: estimate interval and benchmark process
  - What makes some changes hard and long [3]
  - What processes/tools work and why [1, 2]
  - How do you create a hybrid OSS/Commercial process [4]
- Estimation: predict project repair effort from planned new features
  - Plan for field problem repair after the release
  - Release readiness criteria

Project Data: Extraction

- Access 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 and automatic changes
  - Eliminate post-preprocessor changes

Project Data: Validation

- 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

How to foster experimentation in industry

- Volunteer help to projects in trouble
- Help with the top problem the project is faced, but collect information for the the future problems (the empirical work)
- Provide value for all parties involved (services vs development vs product vs CIO)
- Be sensitive about the privacy at individual and team level.
- Show something significant and unexpected about the project early on
  - Demonstrate immediate results
  - Gain credibility
- Get commitments for other studies

Discussion

- A vast amount of untapped resources for empirical work
- The usage of VCS/CM is rapidly increasing over time (startups than do not use them are rapidly disappearing)
- Immediate simple applications in project management: MR inflow/outflow
- It is already being used in more advanced projects
- Remaining challenges
  - Build and validate models to address all problems of practical/theoretical significance
  - What information developers would easily and accurately enter?
  - 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 David M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, April–June 2000.

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