Software Changes and Software Engineering: Why Not?

A. Mockus

audris@mockus.org

Avaya Labs Research
Basking Ridge, NJ 07920
http://www.research.avayalabs.com/user/audris
The Specifications

- Provide a list of “don’t do it!” for software measurement based on software change repositories
- Influence a young generation of researchers
The Result

✦ It was too boring even to think about such task
✦ Smart people learn from other’s mistakes, stupid learn from their own, the rest are ...
✦ Can I learn from my own experiences and you from other’s?
  ✦ Report personal work/review experiences
  ✦ Critique own work (mostly)
    ✦ To be fair annotated/expanded previous talks
Don’t do it

- Irrelevant topic
- Overly specialized results
- Gross mistakes
Outline

- Background
- The no-nos
- The end
Motivation

✦ What world needs
  ✦ Understand and improve software practice
    ✦ Informed (quantitative) tradeoffs between schedule, quality, cost
      ✦ Understanding: where effort is spent, where defects are introduced
      ✦ Acting: the impact of technologies/processes/organization

✦ Obstacles - lack of focus on 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)
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 (explain and predict) phenomena in software projects and to create tools that improve software productivity/quality/lead times
Systems commonly used in a typical organization

- Sales/Marketing: customer information, customer rating, customer purchase patterns, customer needs: features and quality
- Accounting: Customer/system/software billing information and maintenance support level
- Maintenance support: Currently installed system, support level
- Field support: dispatching repair people, replacement parts
- Call center support: customer call/problem tracking
- Development field support: software related customer problem tracking, installed patch tracking
- Development: feature and development, testing, and field defect tracking, software change and software build tracking
Advantages of project repositories

✧ The data collection is non-intrusive (using only existing data minimizes overhead). Requires in-depth understanding of project’s development process.

✧ Long history on past projects enables historic comparisons, calibration, and immediate diagnosis in emergency situations. It takes time and effort to get to that point.

✧ The information is fine grained, at the MR/delta level. Links to higher level (more sensible) attributes like features and releases is often tenuous.

✧ The information is complete, everything under version control is recorded. Except for fields, often essential, that are inconsistently or rarely filled in.
Advantages of project repositories

✧ The data are uniform over time. *That does not imply that the process was constant over entire period.*

✧ Even small projects generate large volumes of changes making it possible to detect even small effects statistically. *As long as the relevant quantities are extractable.*

✧ The version control system is used as a standard part of the project, so the development project is unaffected by experimenter intrusion. *It is no longer true when the such data is used widely in organizational measurement.*
Irrelevant topic

✦ It is tempting to model things that are easy to measure
  ✦ Counts, trends, patterns

✦ It is tempting to try topics that are well formulated
  ✦ Which modules will get defects
## Trends

Number of changes, lines added, deleted, unchanged over the years

<table>
<thead>
<tr>
<th>Cases</th>
<th>Count</th>
<th>Added</th>
<th>Deleted</th>
<th>Same</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>85</td>
<td>1855</td>
<td>26.3574</td>
<td>4.2814</td>
<td>176.177</td>
</tr>
<tr>
<td>86</td>
<td>11919</td>
<td>21.2353</td>
<td>9.56196</td>
<td>303.107</td>
</tr>
<tr>
<td>87</td>
<td>14633</td>
<td>32.0191</td>
<td>11.7993</td>
<td>655.749</td>
</tr>
<tr>
<td>88</td>
<td>10794</td>
<td>18.1634</td>
<td>7.94191</td>
<td>1009.2</td>
</tr>
<tr>
<td>89</td>
<td>19819</td>
<td>17.6601</td>
<td>6.80292</td>
<td>1717.51</td>
</tr>
<tr>
<td>90</td>
<td>12533</td>
<td>17.8687</td>
<td>5.85016</td>
<td>2609.18</td>
</tr>
<tr>
<td>91</td>
<td>12036</td>
<td>16.5215</td>
<td>5.32635</td>
<td>3336.27</td>
</tr>
<tr>
<td>92</td>
<td>12112</td>
<td>22.1412</td>
<td>8.92404</td>
<td>3338.02</td>
</tr>
<tr>
<td>93</td>
<td>10254</td>
<td>17.5703</td>
<td>5.08416</td>
<td>3470.49</td>
</tr>
<tr>
<td>94</td>
<td>15302</td>
<td>17.875</td>
<td>4.9719</td>
<td>3372.18</td>
</tr>
<tr>
<td>95</td>
<td>8385</td>
<td>17.226</td>
<td>4.59213</td>
<td>3088.31</td>
</tr>
<tr>
<td>96</td>
<td>2762</td>
<td>15.8856</td>
<td>5.10174</td>
<td>2664.84</td>
</tr>
</tbody>
</table>
Patterns

Developer changes over 24 hours
Patterns
Patterns II

Numbers of changes and lines added by hour and type
Module Summaries

The subsystem **SYS**

**Demographics**

<table>
<thead>
<tr>
<th>modules</th>
<th>files</th>
<th>lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>512</td>
<td>168100</td>
</tr>
</tbody>
</table>

Average lengths of files of various types:

<table>
<thead>
<tr>
<th></th>
<th>md</th>
<th>h</th>
<th>L</th>
<th>G</th>
<th>es</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files</td>
<td>150</td>
<td>10</td>
<td>50</td>
<td>300</td>
<td>10</td>
</tr>
<tr>
<td>Lines</td>
<td>150000</td>
<td>5000</td>
<td>2500</td>
<td>10000</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>1000.25</td>
<td>500</td>
<td>50.6</td>
<td>40.9</td>
<td>87</td>
</tr>
</tbody>
</table>

**Change Summaries**

There are 4000 MR-module combinations. The modules are described in the following view:

**Regression models**

**Deltas per MR:** $\text{deltas} = 4.0 \text{ mrs} + 1.9 \times \text{mrs} \times \text{epsilon}$

<table>
<thead>
<tr>
<th>sys/module3</th>
<th>sys/module4</th>
<th>sys/module1</th>
<th>sys/module5</th>
<th>sys/module6</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.4</td>
<td>-1.2</td>
<td>-0.88</td>
<td>-0.7</td>
<td>-0.59</td>
</tr>
</tbody>
</table>

**Developers per line:** $(1 + \log(\text{mrs})) = 0.068 \times (1 + \text{lines}) \times 0.75 \times \exp(0.6 \times \text{epsilon})$

<table>
<thead>
<tr>
<th>sys/module11</th>
<th>sys/module9</th>
<th>sys/module3</th>
<th>sys/module10</th>
<th>sys/module7</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.55</td>
<td>-0.43</td>
<td>-0.36</td>
<td>-0.33</td>
<td>-0.24</td>
</tr>
</tbody>
</table>

**Fault Potential**

<table>
<thead>
<tr>
<th>sys/module8</th>
<th>sys/module5</th>
<th>sys/module4</th>
<th>sys/module1</th>
<th>sys/module6</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.33</td>
<td>-0.24</td>
<td>0.36</td>
<td>0.37</td>
<td>1.2</td>
</tr>
</tbody>
</table>

**Total number of modules involved:** 8

**Total number of (fault ln, module) combinations:** 250

**Deviance for naive model:** 9

**Null deviance:** 260 on 7 degrees of freedom

**Residual deviance:** 30 on 5 degrees of freedom

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
<th>Std. Error</th>
<th>t value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>-0.38</td>
<td>0.40</td>
<td>-0.98</td>
</tr>
<tr>
<td>xldeltas</td>
<td>0.67</td>
<td>0.05</td>
<td>13.60</td>
</tr>
<tr>
<td>age</td>
<td>-0.03</td>
<td>0.10</td>
<td>-0.30</td>
</tr>
</tbody>
</table>
Where faults occur?

- Assume the best possible outcome, i.e., we can predict exactly!
  - This can be evaluated by, for example, looking at actual occurrence after the fact
  - 50% of the faults occur in 20% of the modules
  - Unfortunately, these 20% of the modules contain 60% of the code!
Some models of software changes

✦ Quality: model of customer experience [11, 14]
✦ Effort: estimate interval and benchmark process
  ✧ What makes some changes hard and long [6]
  ✧ What processes/tools work and why [2, 3]
  ✧ How do you create a hybrid OSS/Commercial process [9, 5]
✦ Estimation: predict project repair effort from planned new features
  ✧ Plan for field problem repair after the release [13, 14]
  ✧ Release readiness criteria [13, 8]
Some development support tools

- Finding relevant people [10]
- Finding related defects [4]
- Finding related changes [1, 15, 7]
- Finding independently maintainable pieces of code [12]
Real-Real Problems?

✧ Ask two question:

✧ Suppose the questions I am posing can be answered beyond the wildest optimistic projections - what difference will it make?
✧ Suppose I will get some handle on these questions - what difference will it make?
Audience that is too narrow

- “Simulating the process of simulating the process”
- Similarly the tools that support software project data analysis
SoftChange

- http://sourceforge.net/projects/sourcechange
- The SoftChange project will create software to summarize and analyze software changes in CVS repositories and defect tracking systems
- Requirements
  - retrieve the raw data from the web or the underlying system via archive downloads, CVS logs, and processing Bugzilla web pages;
  - verify completeness and validity of different change records by cross-matching changes from CVS mail, CVS log, and ChangeLog files; matching changes to PR reports and identities of contributors;
  - construct meaningful measures that can be used to assess various aspects of open source projects.
- Road map at:
  http://sourceforge.net/docman/display_doc.php?docid=15813&group_id=58432P
Gross Errors

- Lack of validation
  - Limited understanding of the process
  - Insufficient data cleaning
  - Eliminating missing/default/auto values
Missing data

- MCAR — missing completely at random: never happens
- MAR — missing at random: missingness is random conditional on non-missing values
- Other — missingness depends on the value itself: most common
Example

- Two projects are compared
- First has 30% of the cases where the attribute is missing
- Second has 60% of the cases where the attribute is missing
- Comparison is performed by doing a two-sample t-test on the attributes that are not missing
**Example: “the right way”**

- Sample cases with missing attributes and interview relevant people to determine:
  - Do actual values for missing cases differ from values for non-missing cases
  - Is the difference the same for both projects
  - Can the difference be explained by other non-missing/default values
- If there is no possibility for validation assess the impact of non-random missingness
- And: don’t forget to take logs before doing non-rank based tests
What is the problem?

<table>
<thead>
<tr>
<th>Priority</th>
<th>project A</th>
<th>Projet B</th>
<th>Project C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>201</td>
<td>1642</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>3233</td>
<td>9920</td>
<td>659</td>
</tr>
<tr>
<td>4</td>
<td>384</td>
<td>344</td>
<td>1</td>
</tr>
</tbody>
</table>
Methodology: Main Principles

Main focus on supporting the 9[5-9]% of the work related to extraction/cleaning/validation

✦ Use levels and pipes, a la satellite image processing
✦ Validation tools (regression, interactive) for each level/transition
  ◇ Traceability to sources from each level
  ◇ Multiple operationalizations within/across levels
  ◇ Comparison against invariants
  ◇ Detecting default values
  ◇ Handling missing values
Project Data: Levels [0-2]

✧ Level 0 — actual project. Learn about the project, access its systems

✧ Level 1 — 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)

✧ Level 2 — Do basic cleaning
  ✧ Eliminate administrative and automatic artifacts
  ✧ Eliminate post-preprocessor artifacts
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
Serious Issues with the Approach

- Data cleaning and validation takes at least 95% effort - analysis only 1 to 5 percent
- It is very tempting to model easy-to-obtain yet irrelevant measures
- Need to understand implications of missing data
- Using such data will change developer behaviour and, therefore, the meaning such data may have
Pitfalls of using project repositories

- A lot of work — try something simpler first
- Easy to study irrelevant phenomena or tool generated artifacts
- 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 on repository data
  - Easy to compute a lot of irrelevant numbers
  - Interesting phenomena are often not captured even in software project data
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


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.

A. Mockus Software Changes and Software Engineering Dagstuhl, 2005