Innovation, Reuse, Reproducibility, and Transparency are Correlated

Audris Mockus
University of Tennessee, audris@utk.edu

[2017-06-21]

Outline

Objective: A simple message

- Changes are critical to understanding the final product
- Document changes, open, automate!

Speaking notes:
Paradata is necessary to understand how respondents react to survey questions. Documenting the changes to code, data, or documentation provides invaluable record of the reasons for and context in which decisions to make these changes were made (transparency). Design rationale, thus, is preserved for posterity and does not disappear as experts who made these decisions move on, retire, or die.
It makes no sense to keep such paradate private. Making it open (transparency) serves several purposes:
a) Improves the quality of the work through potential of later scrutiny of the decisions made
b) Allows others to learn about the domain by learning how issues are addressed and design decisions are made without direct contact. c) Improves quality of the result through "many eyes make all defects shallow."
Automation forces encoding tacit knowledge used to solve problems. It forces that tacit knowledge transparent by expressing it as code

The second major benefit of automation is that tests can be run after even a minor change at no cost. This major milestone in reputability also has tremendous impact on delivery times and quality.

Tools and practices

- Version Control for everything
- Good practices:
  - Document changes well
    - Bad: misc fixes and cleanups
    - Good: atomic, etc,
      `https://chris.beams.io/posts/git-commit/`
- Inspect changes
  - To improve quality
  - To spread knowledge
  - To foster innovation
- Automate: build, test, deploy, virtualize

Speaking notes:
A primary tool to document changes is the version control system. As with any tool it can be used well or ineffectively. Therefore good practices are needed in conjunction with tools.

1. Changes need to be explained well.

2. Documents/data/code need peer review to be accepted (pull-requests)

3. After each change the whole process including tests needs to be run automatically. Defects can be found much earlier and easier that way and even large teams do not get bogged down in conflict resolution.

What is version control?

- Each change is recorded
- Changes can be undone
- Changes can be grouped into branches (releases)
- Allows collaboration
- Illustrate

History

- Story: Capability Maturity Model (CMM) [4]
  - Five levels of maturity that determine quality/predictability
- State of Software Report [2, 3]
  - VCS records critical for coverage/cost
  - Surveys/Interviews support
- Open source ecosystems [1]
- Current state of public VCS in Federal Statistcal Agencies
  - `https://bitbucket.com/TranspRepr/study`

Speaking notes:
Projects that used version control tended to deliver on time, spend less effort, and have better quality of their products. That lead to CMM.
State of Software Report in Avaya was instrumental to adopt best practices in reproducibility and to unlock full business value of the intellectual property embedded in software code. If one wanted to do "state of federal statistics" report it would be very helpful if all the agencies were using version control for their code and data.
Software domain has progressed incredibly fast and now every aspect of society relies on massive open source infrastructure supported by volunteers. It demonstrates the benefits of transparency and reproducibility.
Perhaps one outcome of this workshop could be an initiative to create a report on the state of transparency and reproducibility in Federal Statistical Agencies.

Summary

- Document changes
- Document changes well
- Make everything public
- R&T Maturity Model (TRRMM)?

Bibliography

Bian Fitzgerald, Audris Mockus, and Minghui Zhou.

Towards engineering free/libre open source software (floss) ecosystems for impact and sustainability, June 2017.

R. Hackbarth, A. Mockus, J. Palframan, and D. Weiss.

Assessing the state of software in a large enterprise: A twelve year retrospective.
In *The Art and Science of Analyzing Software Data*, volume 1, chapter 15. Morgan Kaufmann, 2015.

Randy Hackbarth, Audris Mockus, John Palframan, and David Weiss.

Assessing the state of software in a large enterprise.
Journal of Empirical Software Engineering, 10(3):219–249, 2010.

Mark C Paulk.

A history of the capability maturity model for software.
ASQ Software Quality Professional, 12(1):5–19, 2009.