Historic analysis of MSR

Audris Mockus
University of Tennessee
audris@utk.edu

MSR'23 *[2023-05-15 Mon]*

MSR is a branch of history/archeology

"... my history will ... be judged useful ... as an aid to the interpretation of the future, which in the course of `human things` must `resemble` if it does not `reflect` it."
Thucydides

MSR is workflow

- A typical MSR workflow[8]:
  - Identify relevant sample
  - Collect historic data
  - Fit/use a model to interpret (or shape) the future
- Aims of this talk
  - Highly simplified history of MSR
  - Potential lessons to interpret/shape future MSR

How MSR differs from traditional empirical approaches?

- General scientific workflow
  - measure inputs prospectively
  - model/predict outputs
- MSR
  - measure inputs from pre-existing data
  - model/predict outputs

Determinants of scientific productivity?

- What slows you down?

Determinants of scientific productivity?

- What slows you down?
- Answer: Measurement

Determinants of scientific productivity

- Reduction of measurement effort
- Improved measurement accuracy
- Increased ability to measure, i.e, measure new phenomena
- Will grow population of scientists (MSRers)

Decreasing measurement costs

- Science: interviews, observations, manually records
  - Tiny scale
  - Extremely disruptive and costly
  - But, measures exactly what is desired
- AI: large-curated community datasets in certain domains
- MSR: operational data in VCS/ITS/Mailing lists
  - Free to use
  - Massive scale (in OSS)
  - But, highly problematic accuracy

Lesson 1

a) Look for domains with free-to-use, abundant data
b) Use, help curate, and construct community datasets

Increasing accuracy

- Create a custom measurement instrument
  - Expensive for human behavior data: self reports, interviews, surveys
- Operational data techniques ([5])
  - Recovering context (e.g., [6])
  - Correcting data errors (e.g., [9])
  - Adjusting interpretation based on missing data (e.g., [3, 1])

Lesson 2

- Increase accuracy via operational data techniques to
  - Contextualize (auto-label)
  - Correct
  - Understand why data is missing
- "Data programming" [7]
  - Use heuristics+domain knowledge to label pragmatically, i.e.,
    - has(comment,"bug") $\rightarrow$ *type* = *fix*
  - Use modeling to correct labels

Ability to measure

- Input(Program code)
  - McCabe, Halstead, goto
  - OO complexity/smells
- Input (VCS changes, ITS issues)
  - Developers, e.g., productivity
  - Code and process quality and lead time
  - Code, process, expertise, identity embeddings

Lesson 3?

- What are the new enablers?

Historic modes of software production

- Direct hardware: wires/assembly language
- Programming languages, tools, styles
- Development process/coordination in large industry projects
- Massive collaboration, code copy/reuse, interdependencies, knowledge transfer
  - 60M contributors (43M aliased)
  - 173M projects (107M deforked)
  - 22 percent copied (to 14 projects on average)
  - Largest communities
    - Collaboration: 6M projects+contributors (75M cliques)

Lesson 3

- Investigate new modes of production
  - Input: software supply chains
  - Output: risk, sustainability
- Use effective representations (embeddings) for the nodes, links, and the network itself
- Exploit existing infrastructure

SSC of the 1st kind

- Technical dependencies among projects with change effort as product flow
- Primary risks: unknown vulnerabilities, breaking changes, lack of maintenance, lack of popularity

Examples of SSC of the first kind

- Python: import re
- Java: import java.util.Collection;
- JavaScript: package.json

SSC of the 2nd kind

- Copying of the source code from project to project as product
  flow
- Primary risks: license compliance, unfixed vulnerabilities/bugs, missing updated functionality

Examples of SSC of the second kind

- Implementation of a complex algorithm
- Useful template
- Build configuration

SSC of the 3rd kind

- Knowledge (product) flow through code changes as developers learn from and impart their knowledge to the source code
- Primary risks: developers may leave, companies may discontinue support

Examples of SSC of the third kind

- Developers gaining skills with tools/packages/practices
- Developers spreading practices, e.g., testing frameworks

SSCs Measurement: Key Needs

- Completeness: the entirety of OSS
- Autocuration: address data quality at scale
- Cross-referencing: to make analysis run in minutes not months

Ready-to-use infrastructure

- World of Code Infrastructure[4, 2]:
  - Complete, Current, Curated, Cross-referenced
  - "Research Ready" for Supply Chain Research
  - How to use: github.com/woc-hack/tutorial, worldofcode.org
  - This Oct, hybrid hackathon to try out

Lesson 4

- Big data slows you (and computer down)
- Use stratified sampling
- Focus not just on projects, but also
  - Authors
  - APIs
  - Source code
- Reconstruct past states of the world

Summary (lesson 5)

- To increase productivity
  - Expand measurement scope to new modes of production (SSC)
  - Keep effort low by
    - Using existing or standard datasets
    - Stratified sampling from complete collections
  - Increase accuracy via operational data "programming"

Bio

Audris Mockus worked at AT&T, then Lucent Bell Labs and Avaya Labs for 21 years. Now he is the Ericsson-Harlan D. Mills Chair professor in the Department of Electrical Engineering and Computer Science of the University of Tennessee.
He specializes in the recovery, documentation, and analysis of digital remains left as traces of collective and individual activity. He would like to reconstruct and improve the reality from these projections via methods that contextualize, correct, and augment these digital traces, modeling techniques that present and affect the behavior of teams and individuals, and statistical models and optimization techniques that help understand the nature of individual and collective behavior. His work has improved the understanding of how teams of software engineers interact and how to measure their productivity.
Dr. Mockus received a B.S. and an M.S. in Applied
Mathematics from Moscow Institute of Physics and Technology in 1988. In 1991 he received an M.S. and in 1994 he received a Ph.D. in Statistics from Carnegie Mellon University.

Abstract

The desire to better understand software development lead to numerous attempts to quantify it. Easy-to-measure artifacts, such as source code, could provide only the most basic understanding of the entire development process and the attempts to directly measure quality and effort were cost-prohibitive, error-prone, and rarely shared with researchers or made public. The rise of open source not only provided a reliable software infrastructure but also the rich data source for the software engineering community to finally measure aspects of software development never seen before. Over time and with increased use of open source software, actual developers increasingly need to deal not just with their own project but with projects upstream, downstream, or sideways in the huge open source software supply chain. Measures derived from the entire software supply chain are now likely to bring about the next software engineering revolution.

References

R. Hackbarth, A. Mockus, J. Palframan, and R. Sethi.

Customer quality improvement of software systems.
Software, IEEE, 33(4):40–45, 2016.

Yuxing Ma, Chris Bogart, Sadika Amreen, Russell Zaretzki, and Audris Mockus.

World of code: An infrastructure for mining the universe of open source vcs data.
In *IEEE Working Conference on Mining Software Repositories*, May 26 2019.

Audris Mockus.

Missing data in software engineering.
In J. Singer et al., editor, *Guide to Advanced Empirical Software Engineering*, pages 185–200. Springer-Verlag, 2008.

Audris Mockus.

Amassing and indexing a large sample of version control systems: towards the census of public source code history.
In *6th IEEE Working Conference on Mining Software Repositories*, May 16–17 2009.

Audris Mockus.

Engineering big data solutions.
In *ICSE'14 FOSE*, 2014.

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.

Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré.

Data programming: Creating large training sets, quickly.
Advances in neural information processing systems, 29, 2016.

Adam Tutko, Austin Z Henley, and Audris Mockus.

How are software repositories mined? a systematic literature review of workflows, methodologies, reproducibility, and tools