On the Limits of Software Repository Mining

Jim Whitehead

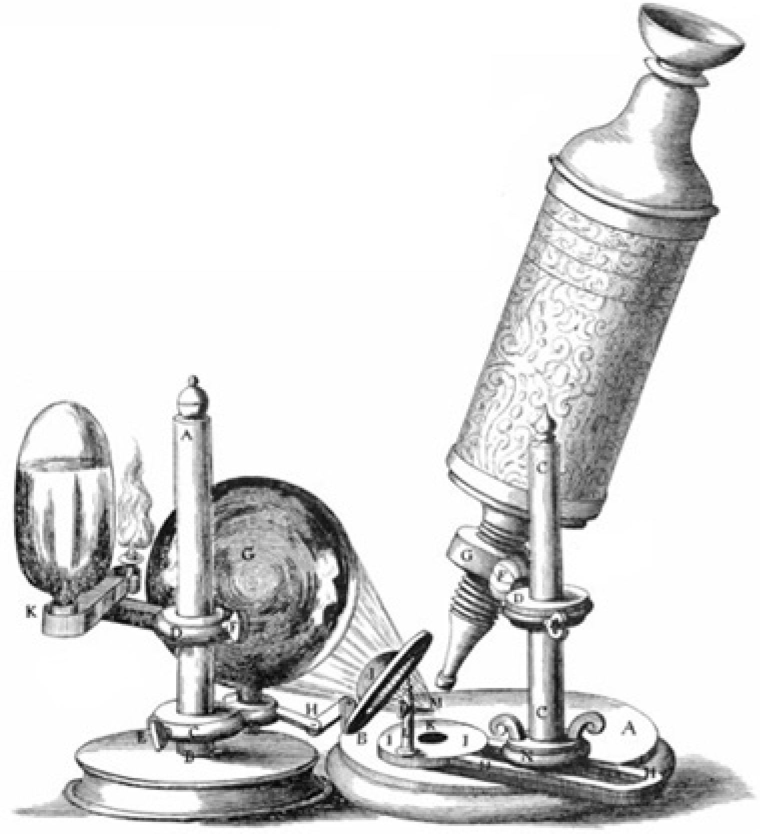
Software Introspection Laboratory
Department of Computational Media
University of California, Santa Cruz

Presented at NG MSR Workshop, Nov. 16, 2014

In the beginning…

- Emergence of large repositories of software project data
  - Open source movement + Forges

- Hmm, is it possible to do anything useful with this data?



Can we construct instruments to extract this data?

YES!

MSR in a Nutshell

Limited Data Sources

- Software engineering projects **create few data sources**
  - Source code repositories
  - Mailing list archives
  - Bug tracking database
  - Test case databases
  - Test execution data

  - … plus many special case repositories (app stores, Q&A sites, etc.)

- These are **traces of the actual activity**

- **A rich but biased** view of what happened on a project

Typical Kinds of Findings

- Descriptive
  - Quantitative understandings of what happened in a project
    - Lines of code, bug counts and locations, clone counts and locations, patterns and locations, identifiers
    - Static code properties and relationships
    - Timings (bug fix time)
    - Frequency distributions of all kinds
    - Visualizations
    - Relationships between data types (bug tossing)
    - … and many more
- Predictive
  - Using time series information to make predictions
    - Bug prediction, code size growth, …

What's wrong with that?

Core Limitation: Incomplete Picture

- Trace data is insufficient to fully understand software project phenomena

- Missing information:
  - Cost, political backdrop, project goals over time, marketplace expectations, interpersonal interactions, physical setting, deadline pressures, why participants find project to be meaningful, etc.

Bugs are costly
Sipho Mabona, The Plague, 2012

  - Messy! Mostly not capable of quantitative representation

Chair Story

Software Evolution

- To really understand how a software project has evolved, need to consider more than trace data
  - Need to perform a deep dive
- Historical analysis
  - Careful collection of information sources, interviews with key participants
  - Development of explanatory frameworks
- Social analysis
  - Understanding of social dynamics within team, relationship of team to broader organization
- **Evolution as the interplay of multiple technical and social forces and influences**

Core Limitation: Answering Why Questions

- MSR approach: **well suited to descriptive science**
- Good at answering **what** and **what happened** questions
- Not so ideal for answering **why** questions

Why Power Law Distributions?

- We know many phenomena in software have power law frequency distributions
  - Source file size, change size, number of subclasses, etc.

- Why?

- As in, why power law, instead of linear, or normal, etc.?

- No one ever explicitly instructs a developer to add code to files to maintain a power law size distribution!
  - An emergent phenomena

Power Law Generative Processes

- Newman (2005) surveys multiple generative models that can produce power law distributions:

- Preferential attachment (Yule process)
  - Entities get random increments of a property in proportion to their current value of that property
  - Example: Big cities attract more people than small cities

- Self-organized criticality
  - Entities added to system at a constant rate
  - Occasional reduction process
  - Example: adding sand to a sand pile, which occasionally slumps

- Can any of these processes explain power laws in software?

Simulate Software Generative Processes

- Use simulation as a tool to explore theories of power law formation
- Since power laws are emergent phenomena, need to execute basic processes to create emergence
- Focus of Zhonpeng Lin PhD work
- See also work by Turnu et al. 2011

Game Dev Story – a playable simulation

- Incomplete Picture
  - Software repositories are one (easily processed) source of project information, among many

- Answering Why Questions
  - Use simulation as a way theories of observed phenomena