Evidence Engineering[1]

Audris Mockus
University of Tennessee
audris@utk.edu

[2015-03-13]

[1]Presented at the Innovative Computing Laboratory

Outline

Premises

Definition (Evidence)

Accurate data, properly interpreted

Definition (Knowledge)

A useful model, i.e., simplification of reality

Definition (Evidence Engineering)

Methods to produce accurate and actionable evidence from operational data

Definition (Data Science)

The study of the generalizable extraction of knowledge from data

Why not Science?

`Science extracts knowledge from experiment data`

Definition (Operational Data (OD))

Digital traces produced in the regular course of work or play (i.e., data generated or managed by operational support (OS) tools)

- no carefully designed measurement system

Science: Temperature Experiment Data

Meteorology

- Weather stations
  - Known locations everywhere

Science: Temperature Experiment Data

Meteorology

- Weather stations
  - Known locations everywhere
  - Calibrated sensor, 5 ± 1 ft above the ground, shielded from sun, freely ventilated by air flow ...

Science: Temperature Experiment Data

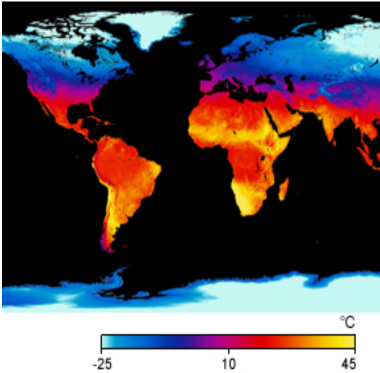
Meteorology

- Weather stations
  - Known locations everywhere
  - Calibrated sensor, 5 ± 1 ft above the ground, shielded from sun, freely ventilated by air flow ...
  - Measures collected at defined times

Science: Temperature Experiment Data

Meteorology

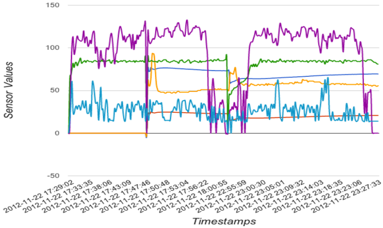
- Weather stations
  - Known locations everywhere
  - Calibrated sensor, 5 ± 1 ft above the ground, shielded from sun, freely ventilated by air flow ...
  - Measures collected at defined times
- Use measures directly in models



Data Science: Operational Data

Mobile Phones

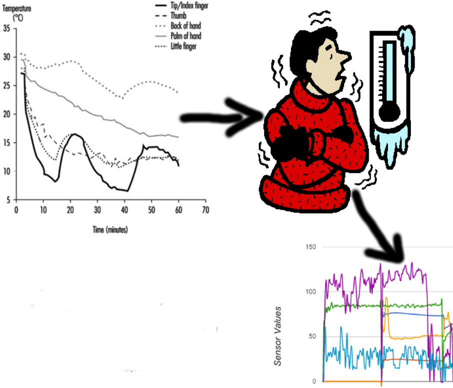
- Location, accelerometer, no temperature
  - No context: indoors/outside
  - Locations/times missing
  - Incorrect values



Data Science: Operational Data

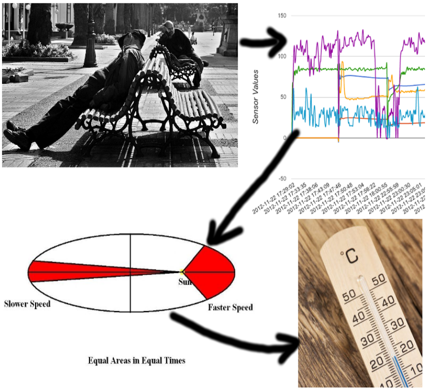
Mobile Phones

- Data Laws, e.g,
  - Temperature $\rightarrow$ sensor?
  - When outside?



Data Science: Operational Data

Mobile Phones



- Use Data Laws
  - Recover context, correct, impute missing
  - Map sensor output into temperature

Example SE Tools Producing OD

- Version control systems (VCS)
  - SCCS, CVS, ClearCase, SVN, Bzr, Hg, Git
- Issue tracking and customer relationship mgmt
  - Bugzilla, JIRA, ClearQuest, Siebel
- Code editing
  - Emacs, Eclipse, Sublime
- Communication
  - Twitter, IM, Forums
- Documentation
  - StackOverflow, Wikies
- Execution
  - GoogleAnalytics, AB testing, performance logs

Metaphor

Example (Piloting a Drone)

- Feed sensor data to the pilot
- Wrong data/misinterpretation $\rightarrow$ crash
- Accuracy is relatively well defined

In most cases accuracy is not easy to measure or even define

- Domains
  - Search
  - Entertainment
  - Social life
  - Software development
- How to navigate without crashing?

What are main challenges?

- Operational data are treacherous - unlike experimental data
  - Multiple contexts
  - Missing events
  - Incorrect, filtered, or tampered with
- Continuously changing
  - Systems and practices are evolving
- Challenges measuring or defining accuracy
- Potential for misinterpretation

OD: Multi-context, Missing, and Wrong

- Example issues with commits in VCS
  - Context:
    - Why: merge/push/branch, fix/enhance/license
    - What: e.g, code, documentation, build, binaries
    - Practice: e.g., centralized vs distributed
  - Missing: e.g., private VCS, no links to defect
  - Incorrect: tangled, incorrect comments
  - Filtered: small projects, import from CVS
  - Tampered with: `git rebase`

Goals and Method of Evidence Engineering

Goals

- Create methods that increase the integrity of evidence

Method

- Discover by studying existing approaches
  - Ratings, workflow systems, crowd-sourcing (e.g., Google maps)
- Suitable techniques from other domains
  - Software engineering, databases, statistics, HCI, …
- Create novel approaches, e.g., Data Laws

Data Laws

- Segment, impute, and correct (see, e.g., ~[1, 3, 5, 4])
  - Based on the way OS tools are used
  - Based on the physical and economic constraints
  - Are empirically validated

How are Defects Observed?

Context

Enterprise software products, highly configurable, sophisticated users, many releases of software

Definition (Platonic Defect)

An error in coding or logic that causes a program to malfunction or to produce incorrect/unexpected results

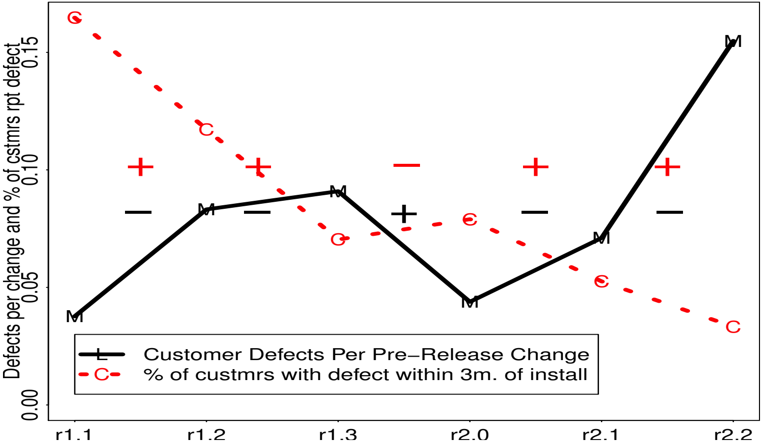
Definition (Customer Found Defect (CFD))

A user found (and reported) program behavior (e.g., failure) that results in a code change.

Using OD to Count CFDs

- CFDs are observed/measured, not defects
  - CFDs are introduced by users
- Lack of use hides defects
  - A mechanism by which defects are missing
- Not CFDs
  - (Small) issues users don't care to report
  - (Serious) issues that are too difficult to reproduce or fix
- More CFDs $\rightarrow$ more use $\rightarrow$ a better product
  - Smaller chances of discovering a CFD by later users

Example: CFDs per change and % of users with CFD



Data Laws for CFDs (Mechanisms and Good Practices)

Laws

- Law I: Code Change Increase Odds of CFDs
- Law II: More Users will Increase Odds of CFDs
- Law III: More Use will Increase Odds of CFDs

Essential Practices

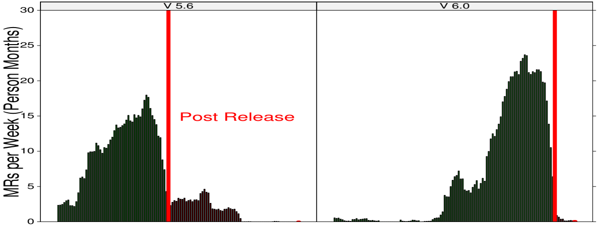
- Commandment I: Don't Be the First User
- Commandment II: Don't Panic After Install
- Cmdmnt III: Keep a Steady Rate of CFDs

Law II: Deploying to More Users will Increase Odds of CFDs

Mechanism

- New use profiles
- Different environments

Evidence



Release with no users

has no CFDs

Commandment I: Don't Be the First User

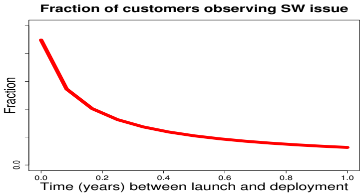
Formulation

`Early users are more likely to encounter a CFD`

Mechanism

- Later users get builds with patches
- Services team learns how to install/configure
- Workarounds for many issues are discovered

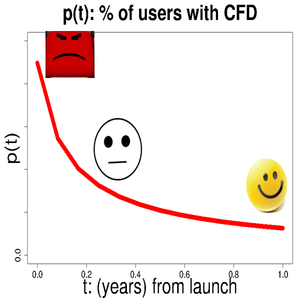
Evidence



- Quality $\uparrow$ with time (users) after the launch, and may be an order of magnitude better one year later[2]

A Game-Theoretic View

- A user i installing at time t_i
- Expected loss $l_i p(t_i)$: decreases
  - where $p(t) = e^{-\alpha n(t)} p(0)$
  - $p(0)$ - the chance of defect at launch
  - $n(t)$ - the number of of users who install by time t
- Value $v_i(T - t_i)$: also decreases



Constraints

- Rate k at which issues are fixed by developers (see C-t III)

Best strategy: $t_i^* = \arg\max_{t_i} v_i(T - t_i) - l_i p(t_i)$

Summary

- Quality of data is key determinant of software success
- The main challenge of EE is OD
  - No two events have the same context
    - Observables represent a mix of platonic concepts
  - Not everything is observed
  - Data are often incorrect
- EE changes the goals of research
  - Model practices of using operational systems
  - Establish data laws
    - Use other sources, experiment, ...
  - Use data laws to
    - Recover the context
    - Correct data
    - Impute missing information

Research Potentially Relevant for ICL

- Improve middleware engineering practices
  - Measure OD from ICL projects
  - Create OD-based tools that improve effectiveness of middleware engineering
- Improve middleware usability/effectiveness
  - Design OD generation for runtime
  - Leverage usage data to
    - Improve design of middleware
    - Make user experience better
    - Improve performance

Bio

Audris Mockus wants to know how and why software development and other complicated systems work. He combines approaches from many disciplines to reconstruct reality from the prolific and varied digital traces these systems leave in the course of operation. Audris 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. In August he will start as Harlan Mills Chair Professor in the Department of Electrical Engineering and Computer Science of the University of Tennessee. Previously he worked at Software Production Research Department of Bell Labs.

Abstract

Structured and unstructured data in operational support tools have long been prevalent in software engineering. Similar data is now becoming widely available in other domains. Software systems that utilize such operational data (OD) to help with software design and maintenance activities are increasingly being built despite the difficulties of drawing valid conclusions from disparate and low-quality data and the continuing evolution of operational support tools. This paper proposes systematizing approaches to the engineering of OD-based systems. To prioritize and structure research areas we consider historic developments, such as big data hype; synthesize defining features of OD, such as confounded measures and unobserved context; and discuss emerging new applications, such as diverse and large OD collections and extremely short development intervals. To sustain the credibility of OD-based systems more research will be needed to investigate effective existing approaches and to synthesize novel, OD-specific engineering principles.

References

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

Audris Mockus, Ping Zhang, and Paul Li.
Drivers for customer perceived software quality.
In *ICSE 2005*, pages 225–233, St Louis, Missouri, May 2005. ACM Press.

Feng Zhang, Audris Mockus, Iman Keivanloo, and Ying Zou.
Towards building a universal defect prediction model.
In *11th IEEE Working Conference on Mining Software Repositories*, May 30–31 2014.

Feng Zhang, Audris Mockus, Ying Zou, Foutse Khomh, and Ahmed E. Hassan.
How does context affect the distribution of software maintainability metrics?
In *Proceedings of the 29th IEEE International Conference on Software Maintenance*, ICSM '13, 2013.

Minghui Zhou and Audris Mockus.
Mining micro-practices from operational data.
In *The 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, Hong Kong, November 16-21 2014.