Is Mining Software Repositories Data Science?

Audris Mockus
Avaya Labs Research
audris@avaya.com

[2014-05-30]

Outline

Basics

Definition (Knowledge)

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

Definition (Software Repositories)

Data managed or generated by tools in the regular course of software development and related activities.

Data Science, Operational Data, Software Repositories

Definition (Data Science)

The study of the generalizable extraction of knowledge from data

Goal: Laws of extracting knowledge from data?

Data Science, Operational Data, Software Repositories

Definition (Data Science)

The study of the generalizable extraction of knowledge from data
Goal: Laws of extracting knowledge from data?

- What properties of data make it Data Science?
  `science extracts knowledge from experiment data`

Data Science, Operational Data, Software Repositories

Definition (Data Science)

The study of the generalizable extraction of knowledge from data
Goal: Laws of extracting knowledge from data?

- What properties of data make it Data 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 tools)

- no carefully designed measurement system

Data Science, Operational Data, Software Repositories

Definition (Data Science)

The study of the generalizable extraction of knowledge from data
Goal: Laws of extracting knowledge from data?

- What properties of data make it Data 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 tools)

- no carefully designed measurement system

Definition (Mining Software Repositories)

The (generalizable) analysis of data in software repositories to uncover interesting and actionable information about software systems and projects

Data Science, Operational Data, Software Repositories

Definition (Data Science)

The study of the generalizable extraction of knowledge from data
Goal: Laws of extracting knowledge from data?

- What properties of data make it Data 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 tools)

- no carefully designed measurement system

Definition (Mining Software Repositories)

The (generalizable) analysis of data in software repositories to uncover interesting and actionable information about software systems and projects
Goal: Laws of extracting knowledge from `software` data?

Science: Traditional Data for Surface Temperature

Meteorology

- Weather stations: temperature sensors
  - Locations:
    - Known
    - Distributed over areas both easy and hard to access

Science: Traditional Data for Surface Temperature

Meteorology

- Weather stations: temperature sensors
  - Locations:
    - Known
    - Distributed over areas both easy and hard to access
  - Calibrated sensor, 5 ± 1 ft above the ground, shielded from sun, freely ventilated by air flow ...

Science: Traditional Data for Surface Temperature

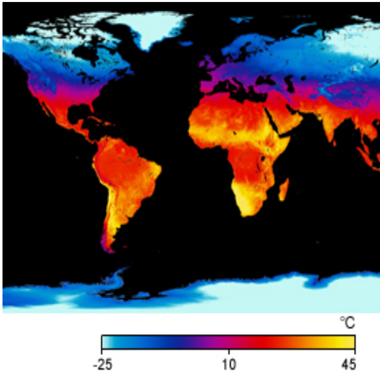
Meteorology

- Weather stations: temperature sensors
  - Locations:
    - Known
    - Distributed over areas both easy and hard to access
  - Calibrated sensor, 5 ± 1 ft above the ground, shielded from sun, freely ventilated by air flow . . .
  - Measures collected at defined times

Science: Traditional Data for Surface Temperature

Meteorology

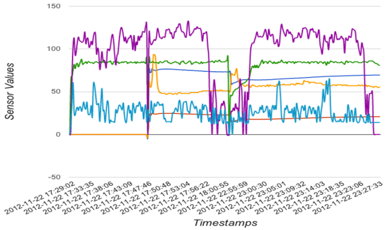
- Weather stations: temperature sensors
  - Locations:
    - Known
    - Distributed over areas both easy and hard to access
  - Calibrated sensor, 5 ± 1 ft above the ground, shielded from sun, freely ventilated by air flow ...
  - Measures collected at defined times
- Can focus on modeling the surface temperature



Data Science: Operational Data for Surface Temperature

Logs from Mobile Phones

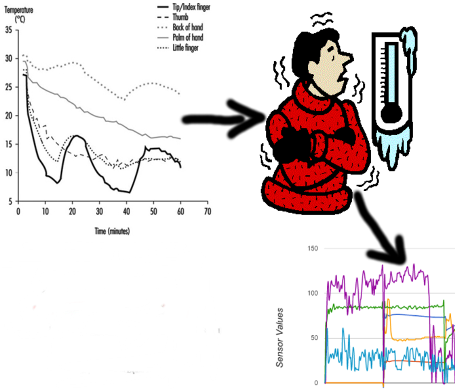
- No temperature measures
  - Geo-location, accelerometer, . . .
  - No context: indoors/in a car/outside
  - Not all locations are covered, not all time



Data Science: Operational Data for Surface Temperature

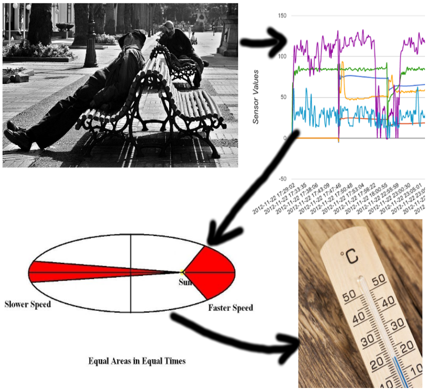
Logs from Mobile Phones

- Discover Data Laws, e.g,
  - How temperature affects sensor data?
  - How to recognize when a subject/sensor is outside?



Data Science: Operational Data for Surface Temperature

Logs from Mobile Phones



- Use Data Laws (Laws of Data Science) to
    - Recover context, correct, impute missing
    - Map sensor output into temperature

Examples of Tools Producing OD

Note:
OD from traces of events or associations are most amenable to reconstructing the past states of the world (work and play).

Examples of Software Tools that Generate OD

- Version control systems (VCS)
  - SCCS, CVS, ClearCase, SVN, Bazaar, Mercurial, Git
- Issue tracking and customer relationship management
  - Bugzilla, JIRA, ClearQuest, Siebel
- Code editing (Eclipse), communication (Twitter), documentation (StackOverflow), . . .

Example OD: Version Control Data

Developers use VCS to make `changes` to code (in parallel)

Traces Left by VCS

Code Before

```
int i = n;
while (i--)
   printf (" %d", i);
```

Code After

```
//print n integers iff n≥ 0
int i = n;
while (--i > 0)
   printf (" %d", i);
```

Example OD: Version Control Data

Developers use VCS to make `changes` to code (in parallel)

Traces Left by VCS

Code Before

```
int i = n;
while (i−−)
   printf (" %d", i);
```

one line deleted

Code After

```
//print n integers iff n≥ 0
int i = n;
while (−−i > 0)
   printf (" %d", i);
```

Example OD: Version Control Data

Developers use VCS to make `changes` to code (in parallel)

Traces Left by VCS

Code Before

```
int i = n;
while (i−−)
   printf (" %d", i);
```

Code After

```
//print n integers iff n≥ 0
int i = n;
while (−−i > 0)
   printf (" %d", i);
```

two lines added

Example OD: Version Control Data

Developers use VCS to make `changes` to code (in parallel)

Traces Left by VCS

Code Before	**Code After**
	//print n integers iff n≥ 0
int i = n;	int i = n;
while (i−−)	while (−−i > 0)
printf (" %d", i);	printf (" %d", i);

two lines unchanged

Example OD: Version Control Data

Developers use VCS to make `changes` to code (in parallel)

Traces Left by VCS

Code Before

```
int i = n;
while (i--)
   printf (" %d", i);
```

Code After

```
//print n integers iff n≥ 0
int i = n;
while (--i > 0)
   printf (" %d", i);
```

one line deleted   two lines added   two lines unchanged

Other attributes: date: 2014-05-29 01:25:30, developer id: audris, branch: master, Comment: "Fix bug 3987 - infinite loop if n $\leq$ 0"

Why Research on Operational Data (OD)?

- Prevalent
  - Massive data from software development
  - OD increasingly used in practice
  - Human activities transition to a digital domain
- Treacherous - unlike experimental data
  - Multiple contexts
  - Missing events
  - Incorrect, filtered, or tampered with
- Continuously changing
  - OS systems and practices are evolving
  - New OS tools are being introduced in SE and beyond
  - Other domains are introducing similar tools

Aim (for MSR and Data Science)

Notes

- OD analysis has to be a software system (ODAS)
- ODAS feeds on (and feeds) operational support (OS) tools

Aim

- Develop approaches and tools for engineering ODAS
    - To ensure the integrity of results
        - To improve effectiveness of operational support tools
    - To simplify building of ODAS
        - To improve quality of OD

Method

- Discover by studying existing ODAS
  - Integrity issues tend to be ignored
  - Cleaning/processing scripts offered
- Borrow suitable techniques from other domains
  - e.g., software engineering, databases, statistics, HCI, . . .
- Synthesize new approaches for unique features of ODAS

Unique Features of OD: Multi-context, Missing, and Wrong

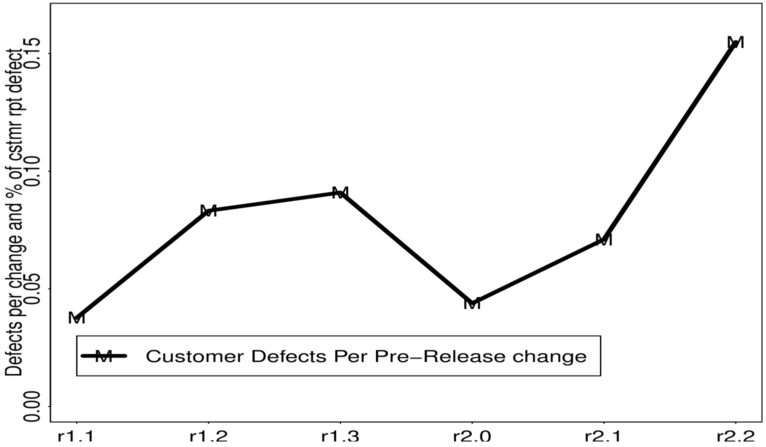
- Example issues with commits in VCS
  - Context:
    - Why: merge/push/branch, fix/enhance/license
    - What: e.g, code, documentation, build, binaries
    - Practice: e.g., centralized vs distributed, frequency of commits
  - Missing: e.g., private VCS, links to defect IDs
  - Incorrect: bug/new, problem description
  - Filtered: small projects, import from CVS
  - Tampered with: `git rebase`

- Establish Data Laws: to segment, impute, and correct
  - Mechanisms
    - Based on the way operational support systems are used
    - Based on the physical and economic constraints
    - Are empirically validated

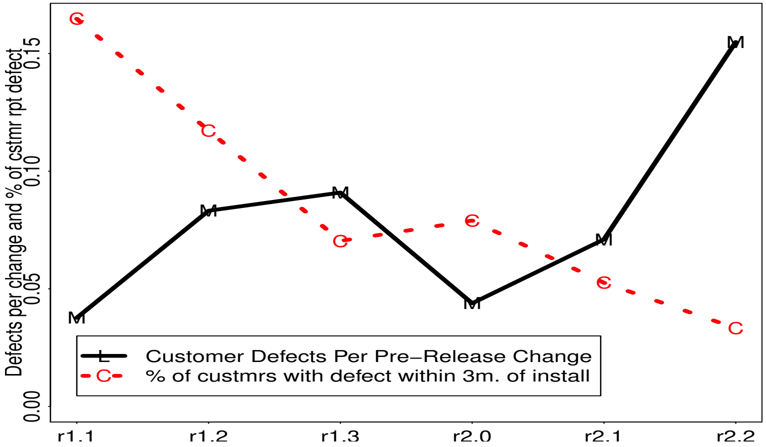
A Puzzle from Software Domain

- Analogy to surface temperature
  - Surface temperature
    - User-Perceived Quality
  - Data
    - Software Repositories
    - Customer Support Tools
- Context
  - Enterprise software products
  - Concern: customer found and reported product defects (CFDs)
- Nature of the puzzle: contradicting measures
  - CFDs per LOC/Change
  - Chances for a user to experience a defect

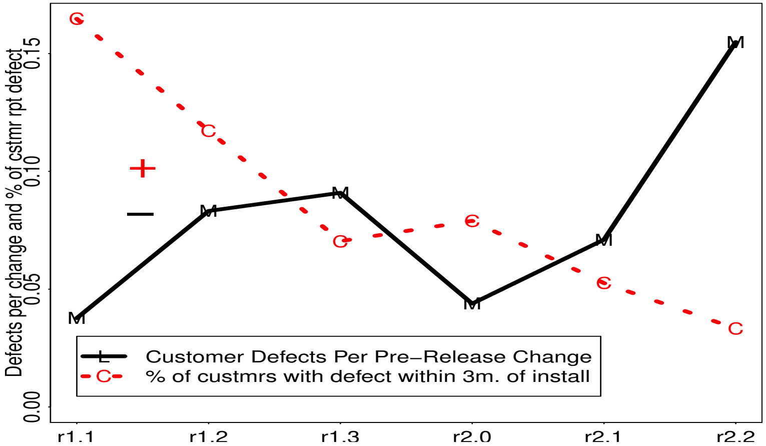
Defects per change — R1.1 - best, R2.2 - Worst?



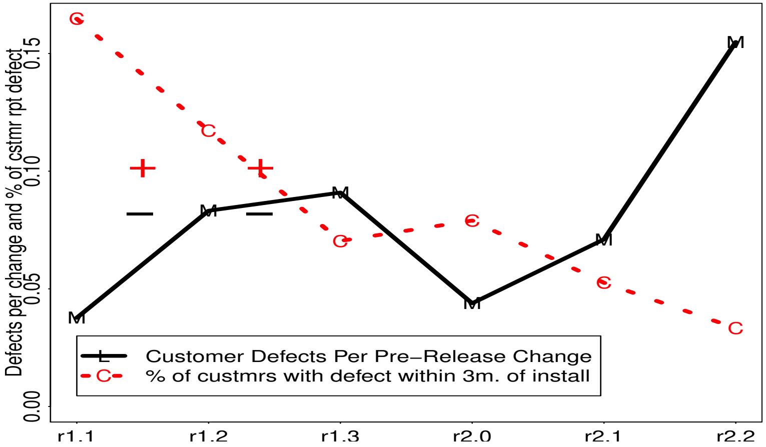
Defects per change — R1.1 - best, R2.2 - Worst?



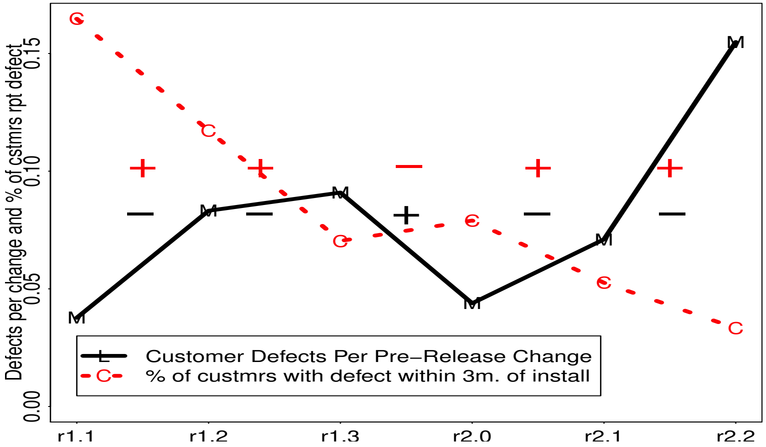
Defects per change — R1.1 - best, R2.2 - Worst?



Defects per change — R1.1 - best, R2.2 - Worst?



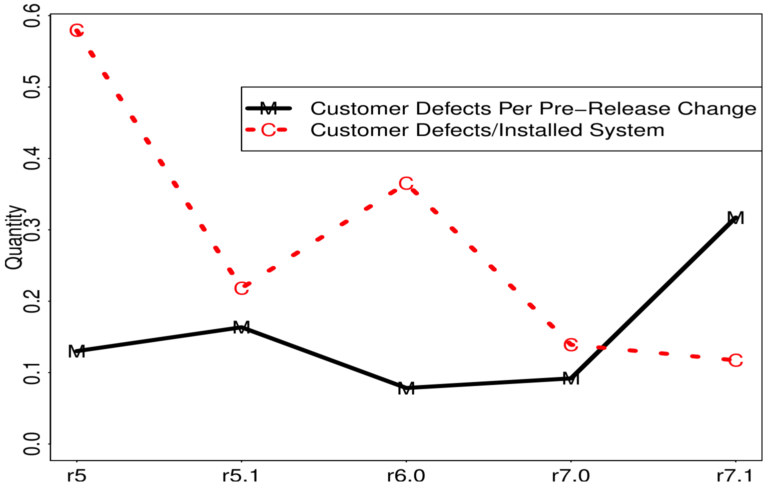
Defects per change — R1.1 - best, R2.2 - Worst?



Reality check: r1.1 - worst, r2.2 - best.

Is this a unique case?

Five releases of another product



Perfect anti-correlation again?!

Are there Systematic Ways to Resolve Similar Paradoxes?

- Derive measures from operational data that
  - Usefully reflect reality
  - Can be used to improve software development

How are Bugs 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.

Implications of Using Software Repository OD to Count CFDs

- CFDs are observed/measured, not defects
  - By definition, CFDs are introduced by users, not developers
- 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
- Can't introduce flaws intentionally to increase release "quality"
  - Because users will not install bad release

Laws and Commandments
Mechanisms and Good Practices for CFDs

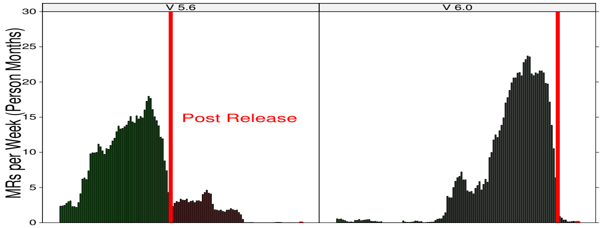
- Law I: Code Change Increase Odds of CFDs
- Law II: Deploying to More Users will Increase Odds of CFDs
- Law III: Longer (and Heavier) Use will Increase Odds of CFDs
- Commandment I: Don't Be the First User of a Release
- Commandment II: Don't Panic After Install/Upgrade
- Commandment III: You Should 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 of a Release

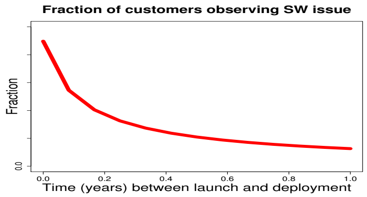
Formulation

`First users are more likely to experience a CFD`

Mechanism

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

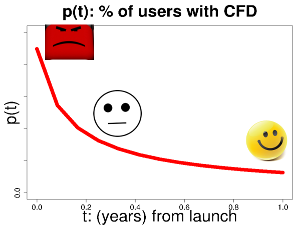
Evidence



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

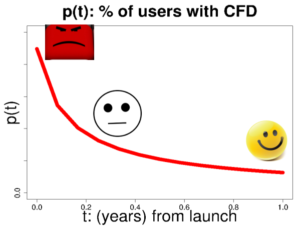
Why Users Like More CFDs? A Game-Theoretic View

- Utility/Loss for a customer i installing at time t_i
  - Loss $l_i p(t_i)$: (see Law II, C-t I)
  - $p(t) = e^{-\alpha n(t)} p(0)$
    - $p(0)$ - the chance of defect at launch
    - $n(t)$ - the number of of users who install by time t



Why Users Like More CFDs? A Game-Theoretic View

- Utility/Loss for a customer i installing at time t_i
  - Loss $l_i p(t_i)$: (see Law II, C-t I)
  - $p(t) = e^{-\alpha n(t)} p(0)$
    - $p(0)$ - the chance of defect at launch
    - $n(t)$ - the number of of users who install by time t



Constraints

- Rate k at which issues are fixed by developers (see C-t III)
- Value of new features: $v_i(T - t_i)$: decreases with install delays

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

Implications

- How quality translates into revenue [1]?
  - Can bound the deplyment/sales based on
    - Launch quality $p(0)$
    - Rate at which CFDs are fixed k

$$n(T) < \frac{1}{\alpha p(0)} \left(e^{\alpha k T} - 1 \right)$$

Summary: Yes, OD are common to DS and MSR

Summary: Yes, OD are common to DS and MSR

- Research for OD-based engineering
  - Is badly needed and challenging
  - Should be fruitful

Summary: Yes, OD are common to DS and MSR

- Research for OD-based engineering
  - Is badly needed and challenging
  - Should be fruitful
- Defining features of OD
  - No two events have the same context
    - Anything observable represents a mix of platonic concepts
  - Not everything is observed
  - Data may be incorrect

Summary: Yes, OD are common to DS and MSR

- Research for OD-based engineering
  - Is badly needed and challenging
  - Should be fruitful
- Defining features of OD
  - No two events have the same context
    - Anything observable represents a mix of platonic concepts
  - Not everything is observed
  - Data may be incorrect
- How to engineer ODAS?
  - Understand practices of using operational systems
  - Use other sources (e.g., traditional observation) to establish relationships among observed and unobserved factors (Data Laws) to
    - Recover the context
    - Correct data
    - Impute missing information

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. He works at Avaya Labs Research. Previously he worked at Software Production Research Department of Bell Labs.

Abstract

Trick question: what is Data Science? The collection and use of low-veracity data in software repositories and other operational support systems is exploding. It is, therefore, imperative to elucidate basic principles of how such data comes into being and what it means. Are there practices of constructing software data analysis tools that could raise the integrity of their results despite the problematic nature of the underlying data? The talk explores the basic nature of data in operational support systems and considers approaches to develop engineering practices for software mining tools.

References

Audris Mockus.
Engineering big data solutions.
In *FOSE, ICSE 2014*, Hyderabad, India, June 1–6 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.

:noexport: