

# Engineering “Big Data” Solutions

Audris Mockus  
Avaya Labs Research  
audris@avaya.com

*[2014-06-04]*

# Outline

Preliminaries

Illustration: Traditional vs Data Science

Why OD is a Promising Area?

Engineering OD Solutions: Goals and Methods

Missing Data: Defects

Summary

# Premises

## Definition (Knowledge)

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

## Definition (Big Data)

Data sets with sizes beyond the ability of commonly used software tools to capture, curate, manage, and process the data within a reasonable time

## Definition (Data Science)

The study of the generalizable extraction of knowledge from data

# Why not Science?

Science extracts knowledge from  
experiment 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,  $5 \pm 1$  ft above the ground, shielded from sun, freely ventilated by air flow . . .



# Science: Temperature Experiment Data

## Meteorology

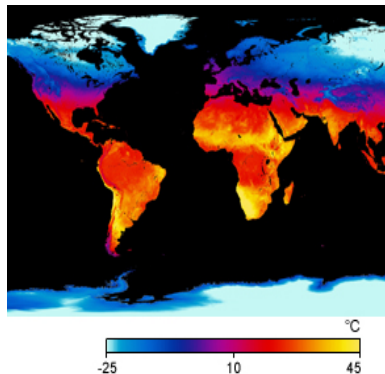
- ▶ Weather stations
  - ▶ Known locations everywhere
  - ▶ Calibrated sensor,  $5 \pm 1$  ft above the ground, shielded from sun, freely ventilated by air flow . . .
  - ▶ Measures collected at defined times



# Science: Temperature Experiment Data

## Meteorology

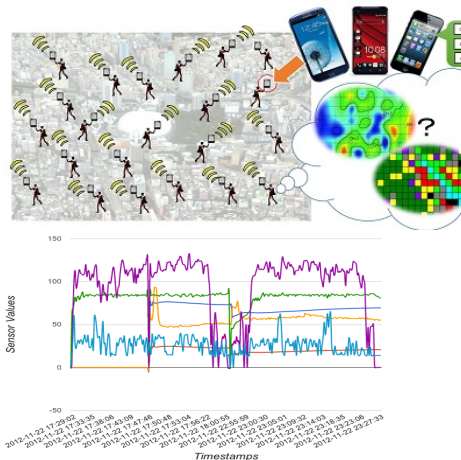
- ▶ Weather stations
  - ▶ Known locations everywhere
  - ▶ Calibrated sensor,  $5 \pm 1$  ft above the ground, shielded from sun, freely ventilated by air flow . . .
  - ▶ Measures collected at defined times
- ▶ Use measures directly in models



# Data Science: Operational Data

## Mobile Phones

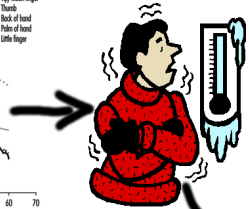
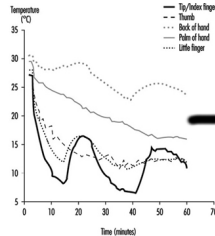
- ▶ Location, accelerometer, **no temperature**
  - ▶ No context: indoors/outside
  - ▶ Locations/times missing
  - ▶ Incorrect values



# Data Science: Operational Data

## Mobile Phones

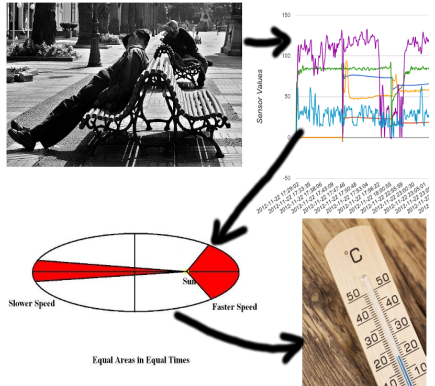
- ▶ **Data Laws**, e.g.,
  - ▶ Temperature → sensor?
  - ▶ When outside?



# 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

# Why OD is a Promising Area?

- ▶ Prevalent
  - ▶ Massive data from software development
  - ▶ Increasingly used in practice
  - ▶ Many activities transitioning 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

# Engineering OD Solutions: Goals

## Premise

- ▶ OD Solutions (ODS) are software systems
  - ▶ Complex/large data, imputation/cleaning/correction
- ▶ ODS feeds on (and feeds) OS tools

## Goal

- ▶ Approaches and tools for engineering ODS
  - ▶ To ensure the integrity of ODS
- ▶ To simplify building and maintenance of ODS

# Method

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

# 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, links to defect IDs
  - ▶ Incorrect: bug/new, problem description
  - ▶ Filtered: small projects, import from CVS
  - ▶ Tampered with: `git rebase`
- ▶ Data Laws: to segment, impute, and correct
  - ▶ 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

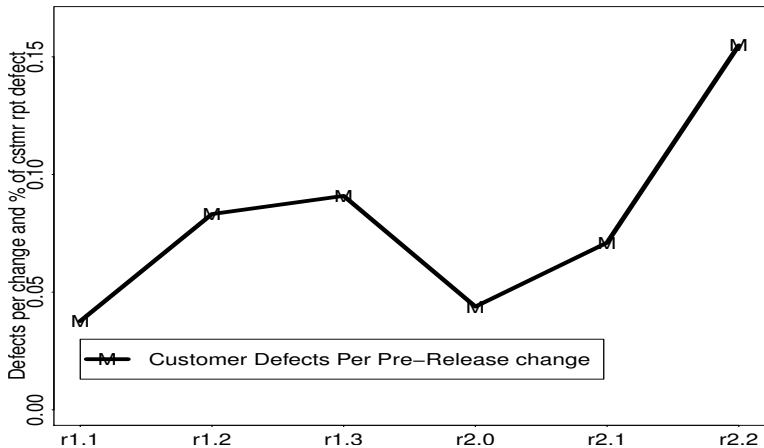
## Definition (Customer Found Defect (CFD))

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

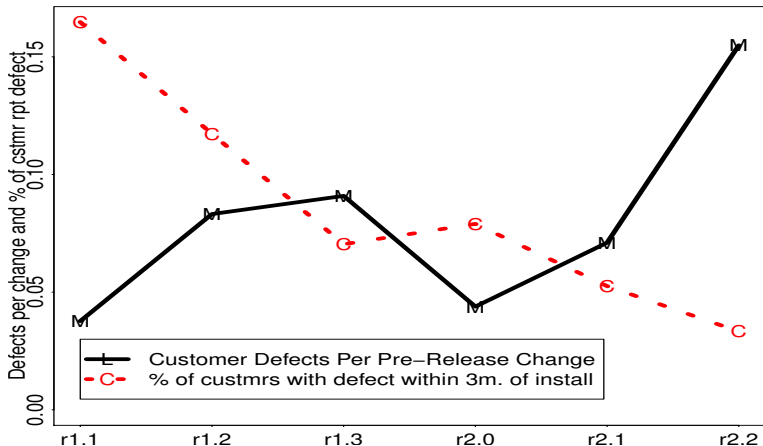
# Using OD to Count CFDs

- ▶ **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** → more use → a better product
  - ▶ Smaller chances of discovering a **CFD** by later users

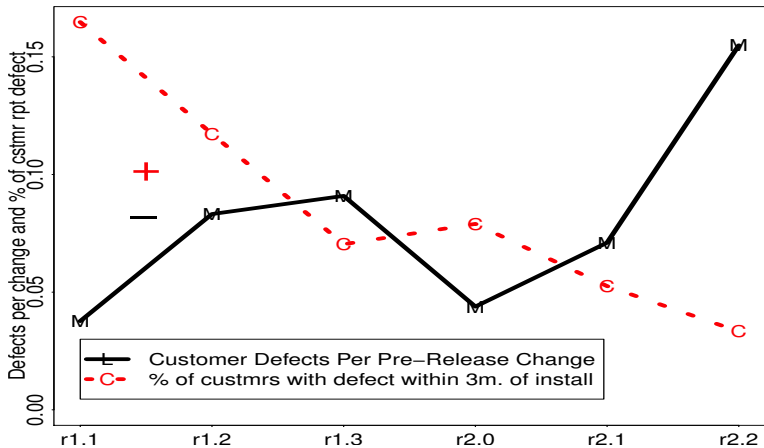
# Example: CFDs per change and % of users with CFD



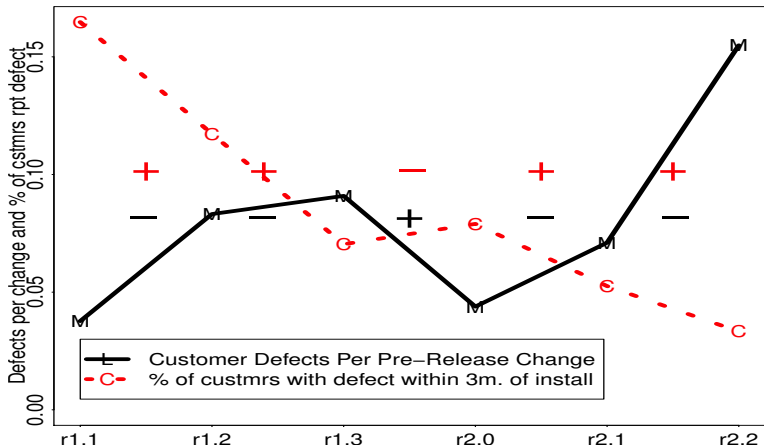
# Example: CFDs per change and % of users with CFD



# Example: CFDs per change and % of users with CFD



# 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

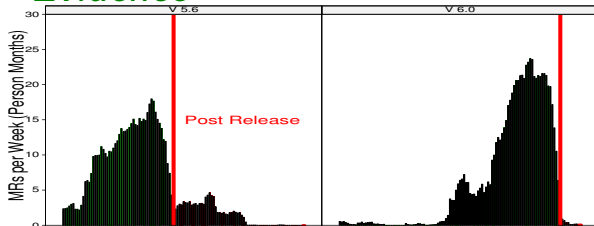
# Law II: Deploying to More Users will Increase Odds of CFDs

## Mechanism

- ▶ New use profiles
- ▶ Different environments



## Evidence



Release with no  
users

has no CFDs

# Commandment I: Don't Be the First User

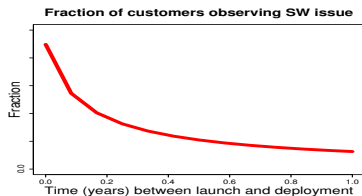
## Formulation

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

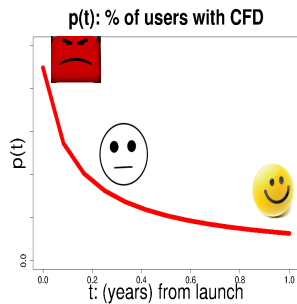
## Evidence



- ▶ Quality  $\uparrow$  with time (users) after the launch, and may be an order of magnitude better one year later[1]

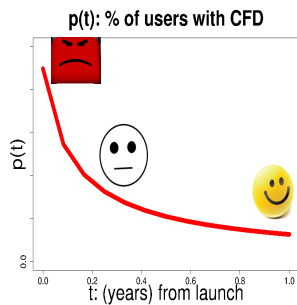
# A Game-Theoretic View

- ▶ 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 users who install by time  $t$
- ▶ Value  $v_i(T - t_i)$ : also decreases



# 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 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

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

# Summary

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

# Summary

- ▶ Research for OD-based engineering
  - ▶ Is badly needed and challenging
  - ▶ Should be fruitful
- ▶ Defining features of OD
  - ▶ No two events have the same context
    - ▶ Observables represent a mix of platonic concepts
  - ▶ Not everything is observed
  - ▶ Data may be incorrect
- ▶ How to engineer ODS?
  - ▶ Understand practices of using operational systems
  - ▶ Establish Data Laws
    - ▶ Use other sources, experiment, ...
  - ▶ Use Data Laws to
    - ▶ Recover the context
    - ▶ Correct data
    - ▶ Impute missing information
  - ▶ Bundle with existing operational support systems


# 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

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