Opinions presented in this talk are mine and mine alone, my co-authors may or may not agree, funding agencies likely would disapprove.

Getting Everything Wrong Without Doing Anything Right!

or

The perils of large-scale analysis of GitHub data

Jan Vitek

*with apologies to Mytkowicz, Diwan, Sweeny, and Hauswirth's "Producing Wrong Data Without Doing Anything Obviously Wrong!" ASPLOS'09

Petr

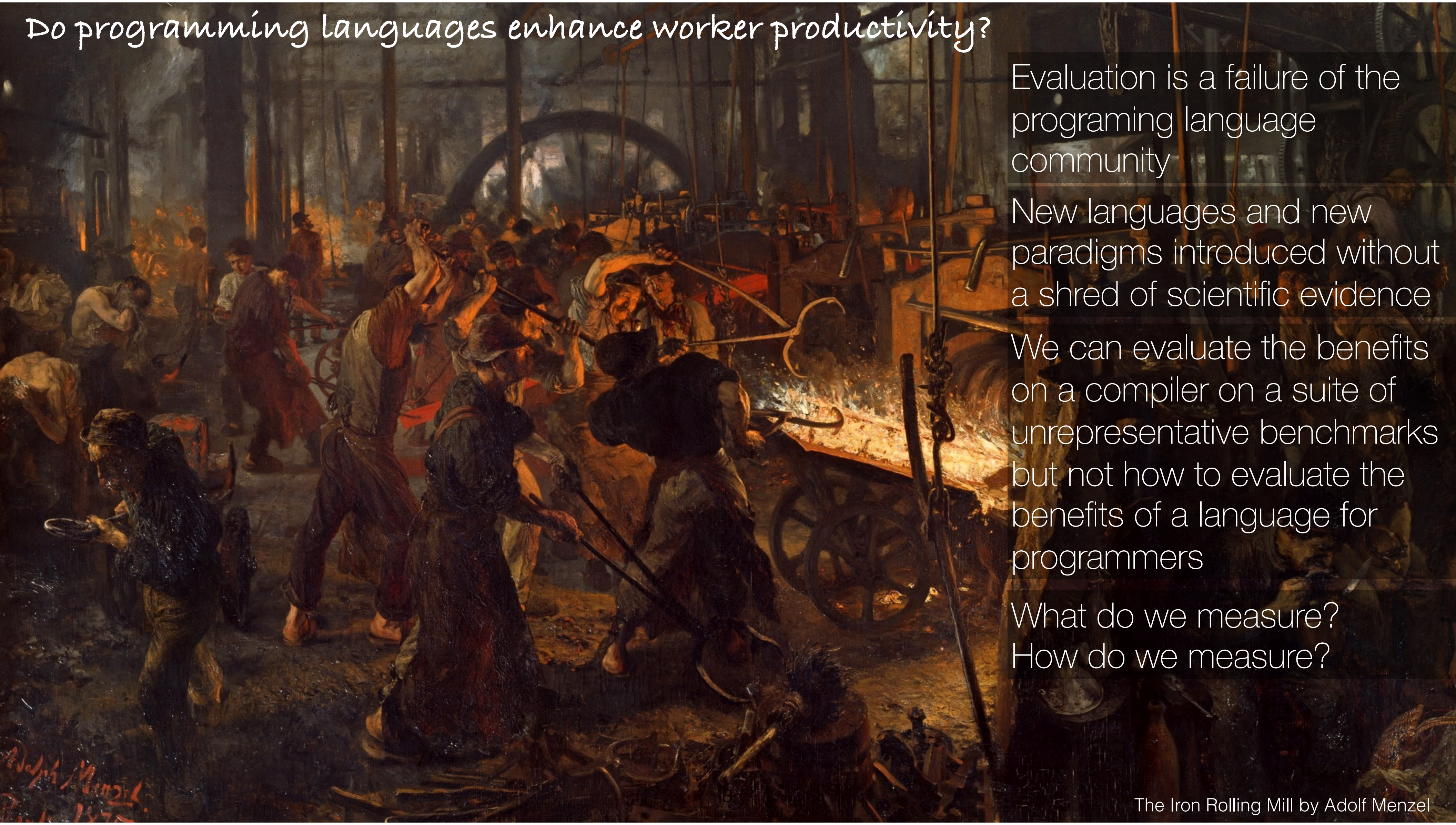
Celeste

Emery

Olga

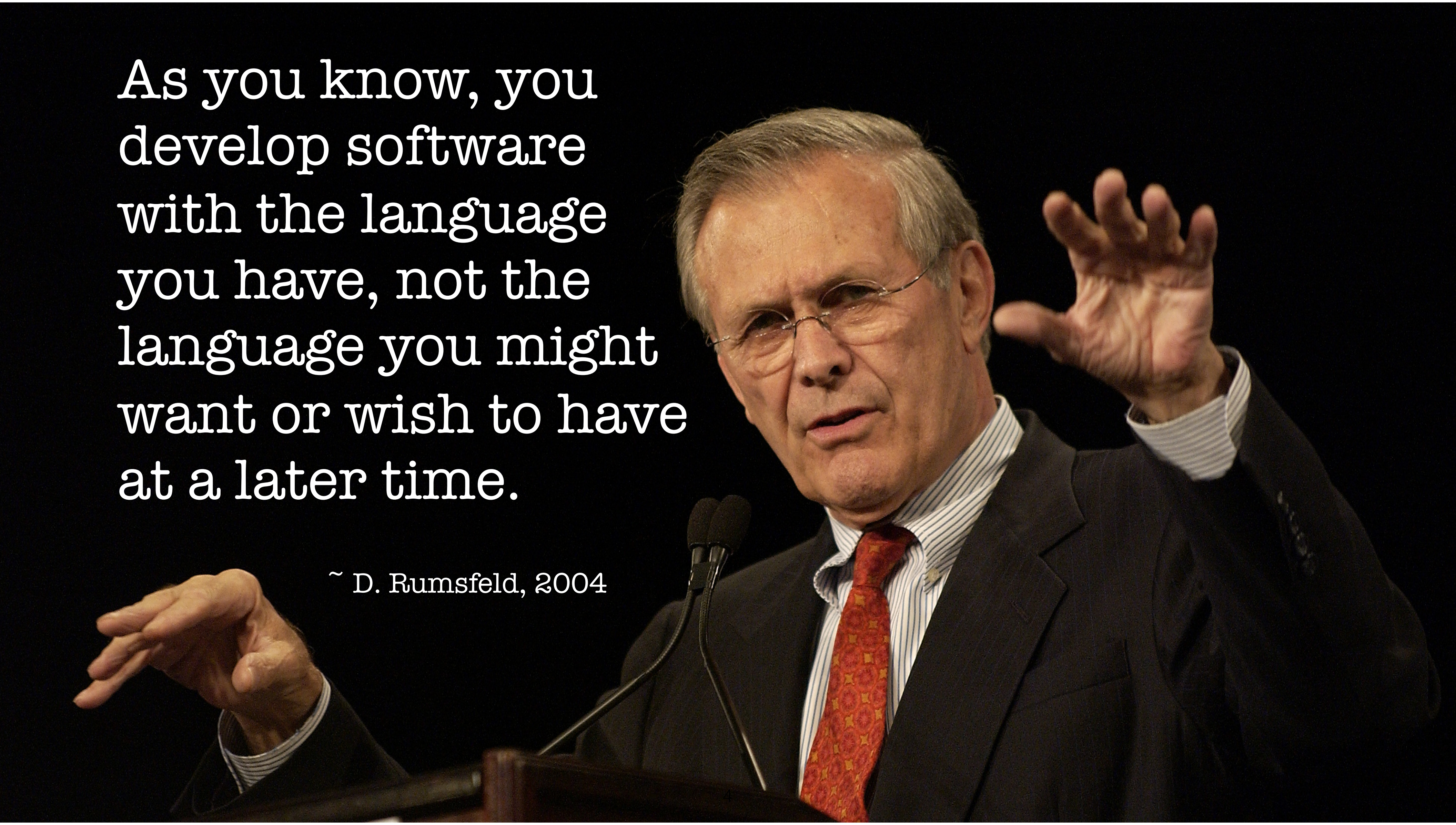
Jan

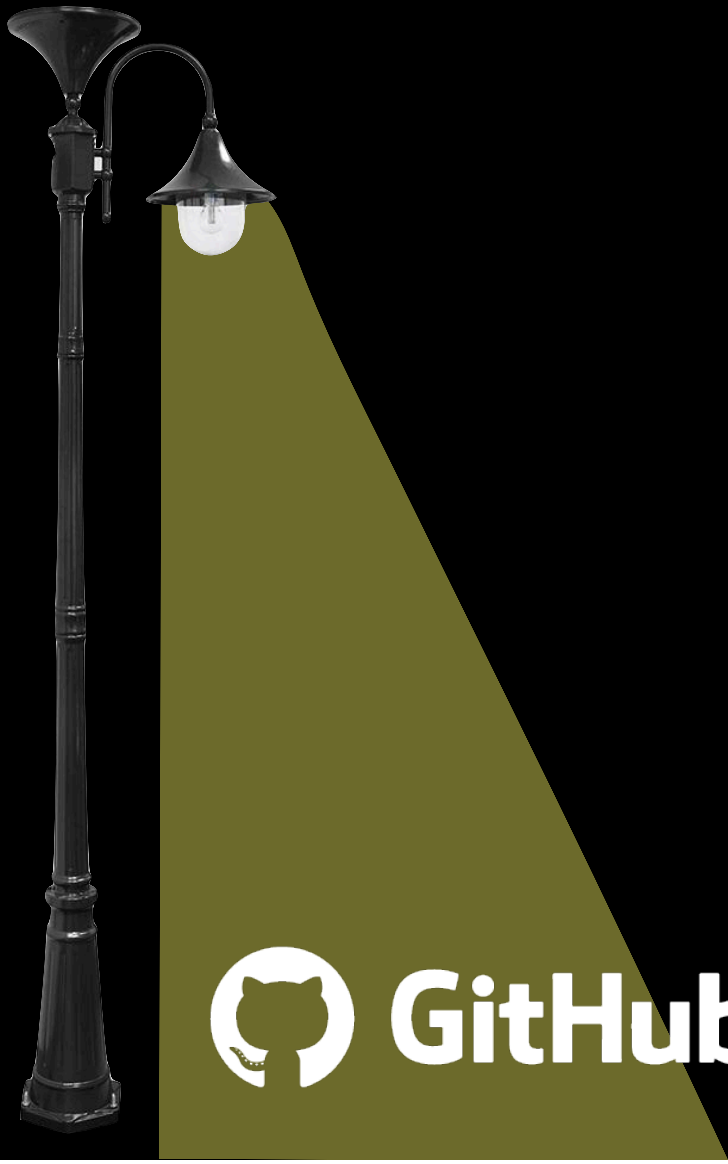
Do programming languages enhance worker productivity?

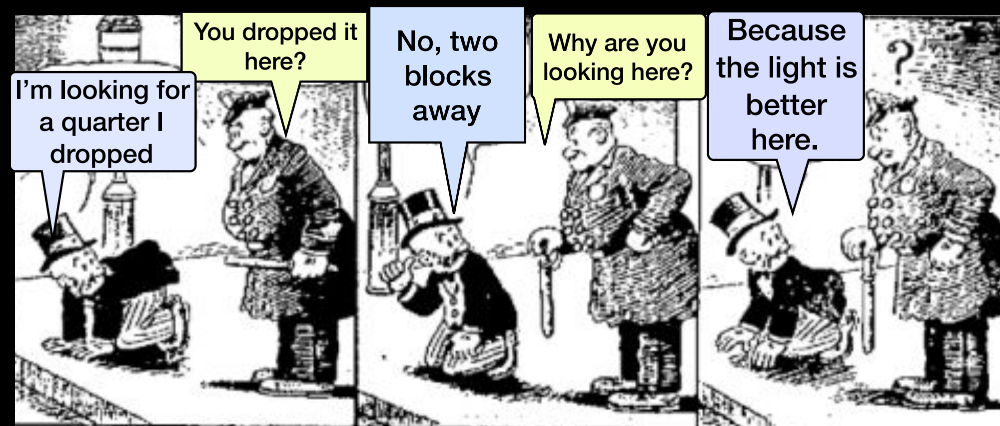


- Evaluation is a failure of the programing language community
- New languages and new paradigms introduced without a shred of scientific evidence
- We can evaluate the benefits on a compiler on a suite of unrepresentative benchmarks but not how to evaluate the benefits of a language for programmers
- What do we measure?
  How do we measure?

The Iron Rolling Mill by Adolf Menzel


As you know, you develop software with the language you have, not the language you might want or wish to have at a later time.
~ D. Rumsfeld, 2004


GitHub


I'm looking for a quarter I dropped
You dropped it here?
No, two blocks away
Why are you looking here?
Because the light is better here.
?

A Large Scale Study of Programming Languages and Code Quality on Github

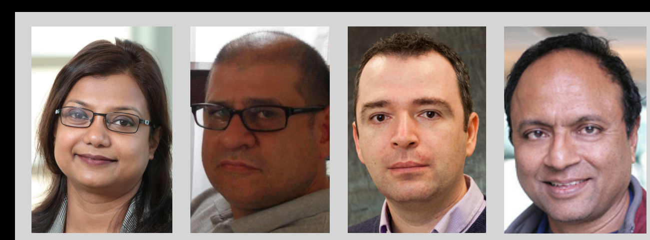
RQ1 *Are some languages more defect prone than others?*

RQ2 *Which language properties relate to defects?*

RQ3 *Does language defect proneness depend on domain?*

RQ4 *What's the relation between language & bug category?*

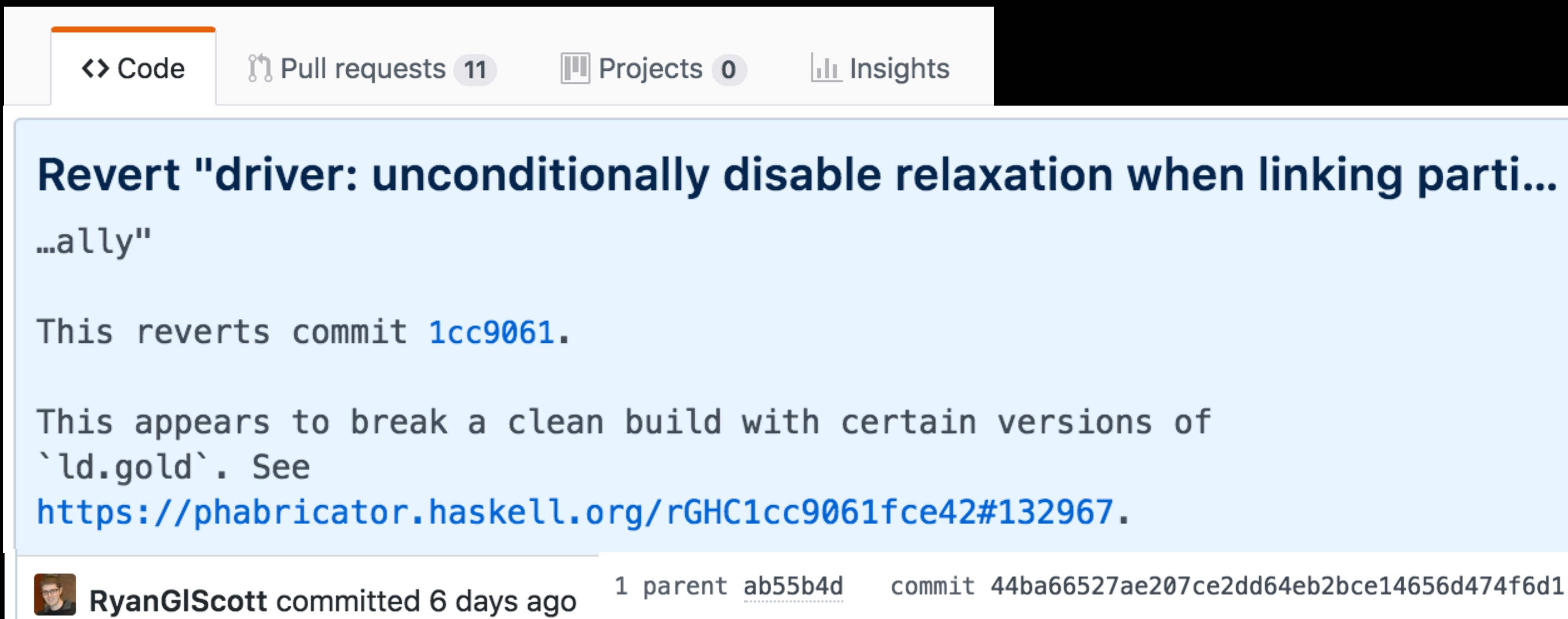
GitHub



Baishaki Ray **Daryl Posnett** **Vladimir Filikov** **Premkumar Devanbu**

UC Davis

A Large Scale Study of Programming Languages and Code Quality on Github



Projects contain a sequence of commits; each commit has a text explanation and affects a number of files in various languages; commits can be labelled as bug-fixing; the prevalence of bug-fixing commits is a proxy for code quality.

A Large Scale Study of Programming Languages and Code Quality on Github

Methodology:

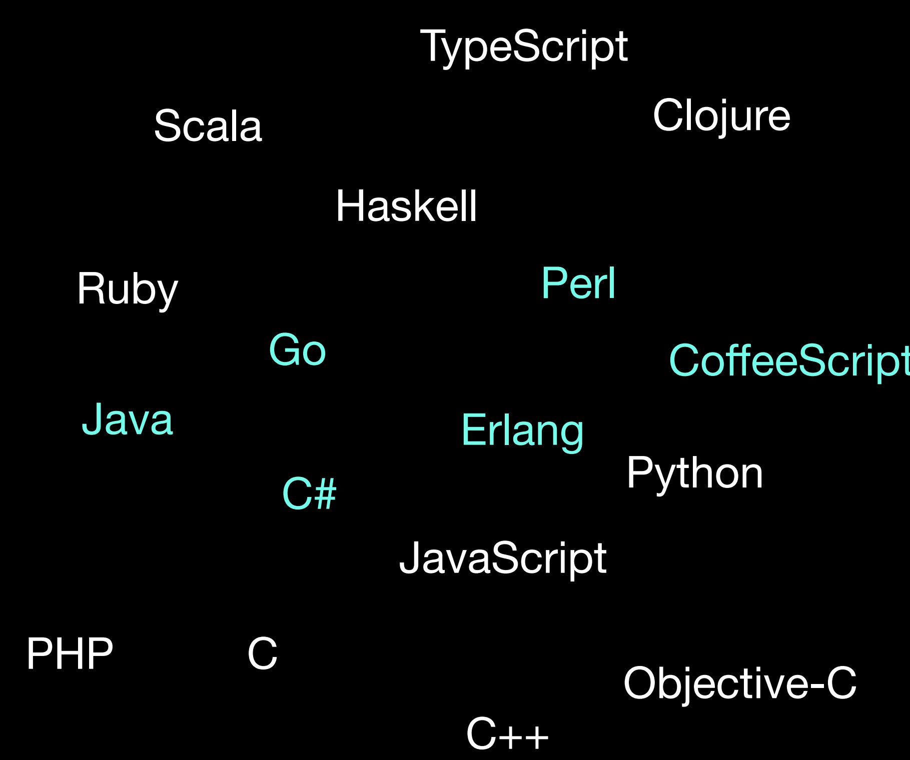
1. Acquire 800 projects written in 17 languages
2. Split by file according to language
3. Filter projects with <20 commits/language
4. Label commits as bug-fixing
5. Negative Binomial Regression predicts bug-fixing commits

GitHub

Projects contain a sequence of commits; each commit has a text explanation and affects a number of files in various languages; commits can be labelled as bug-fixing; the prevalence of bug-fixing commits is a proxy for code quality.

A Large Scale Study of Programming Languages and Code Quality on Github

	Coef	P-val
Intercept	-1.93	<0.001
log commits	2.26	<0.001
log age	0.11	<0.01
log size	0.05	<0.05
log devs	0.16	<0.001
C	0.15	<0.001
C++	0.23	<0.001
C#	0.03	–
Objective-C	0.18	<0.001
Go	-0.08	–
Java	-0.01	–
Coffeescript	-0.07	–
Javascript	0.06	<0.01
Typescript	-0.43	<0.001
Ruby	-0.15	<0.05
Php	0.15	<0.001
Python	0.1	<0.01
Perl	-0.15	–
Clojure	-0.29	<0.001
Erlang	0	–
Haskell	-0.23	<0.001
Scala	-0.28	<0.001



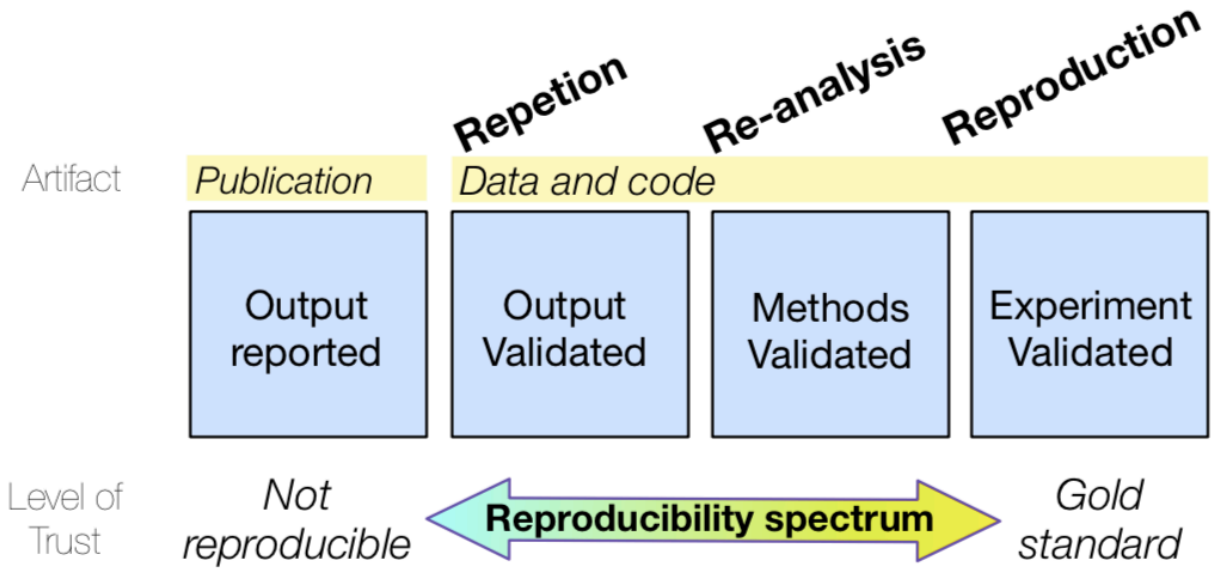
Kutner, et al. 2004. *Applied Linear Statistical Models.* https://books.google.cz/books?id=XAzYCwAAQBAJ

"...a single project, **Google's v8, a JavaScript project,** was responsible for all of the errors in Middleware."

— Ray, Posnett, Filikov, Devambu

"give all of the information to help other judge the value of your contribution; not just the information that leads to a particular judgment"

- R. Feynman, Cargo Cult Science, 1974



*Roger Peng. *Reproducible research in computational science.* Science, 2011

"give all of the information to help other judge the value of your contribution; not just the information that leads to a particular judgment"

- R. Feynman, Cargo Cult Science, 1974

REPETITION

The authors of the original study shared their data (3.4GB) and code (700 loc R)

We thank them

REPETITION

RQ1	RQ2	RQ3	RQ4
✓	✗	✗	✗

Repetition failures caused by:
Nonsensical language classification
Data discrepancies
Missing code

	Original Authors		Repetition	
	(a)		(c)	
	Coef	P-val	Coef	P-val
C	0.15	<0.001	0.16	<0.001
C++	0.23	<0.001	0.22	<0.001
C#	0.03	–	0.03	0.602
Objective-C	0.18	<0.001	0.17	0.001
Go	-0.08	–	-0.11	0.086
Java	-0.01	–	-0.02	0.61
Coffeescript	-0.07	–	0.05	0.325
Javascript	0.06	<0.01	0.07	<0.01
Typescript	-0.43	<0.001	-0.41	<0.001
Ruby	-0.15	<0.05	-0.13	<0.05
Php	0.15	<0.001	0.13	0.009
Python	0.1	<0.01	0.1	<0.01
Perl	-0.15	–	-0.11	0.218
Clojure	-0.29	<0.001	-0.31	<0.001
Erlang	0	–	0	1
Haskell	-0.23	<0.001	-0.24	<0.001
Scala	-0.28	<0.001	-0.22	<0.001

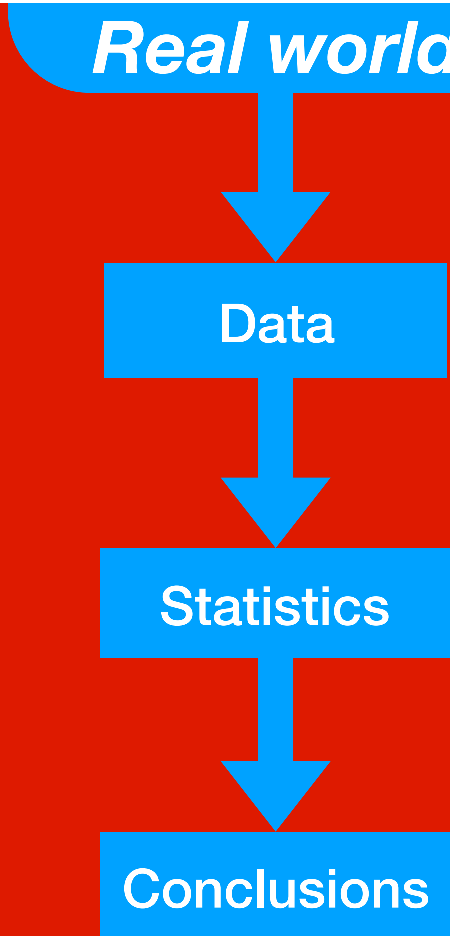
Krishnamurthi, Vitek. *The real software crisis: repeatability as a core value*. CACM'15

REANALYSIS

We focused on RQ1 for a reanalysis as it was mostly repeatable.

The issues we found carry over to the rest of the RQs.

REANALYSIS



Validate data acquisition

Validate data cleaning

Validate data analysis

We focused on RQ1 for a reanalysis as it was mostly repeatable.

The issues we found carry over to the rest of the RQs.

SIN #1

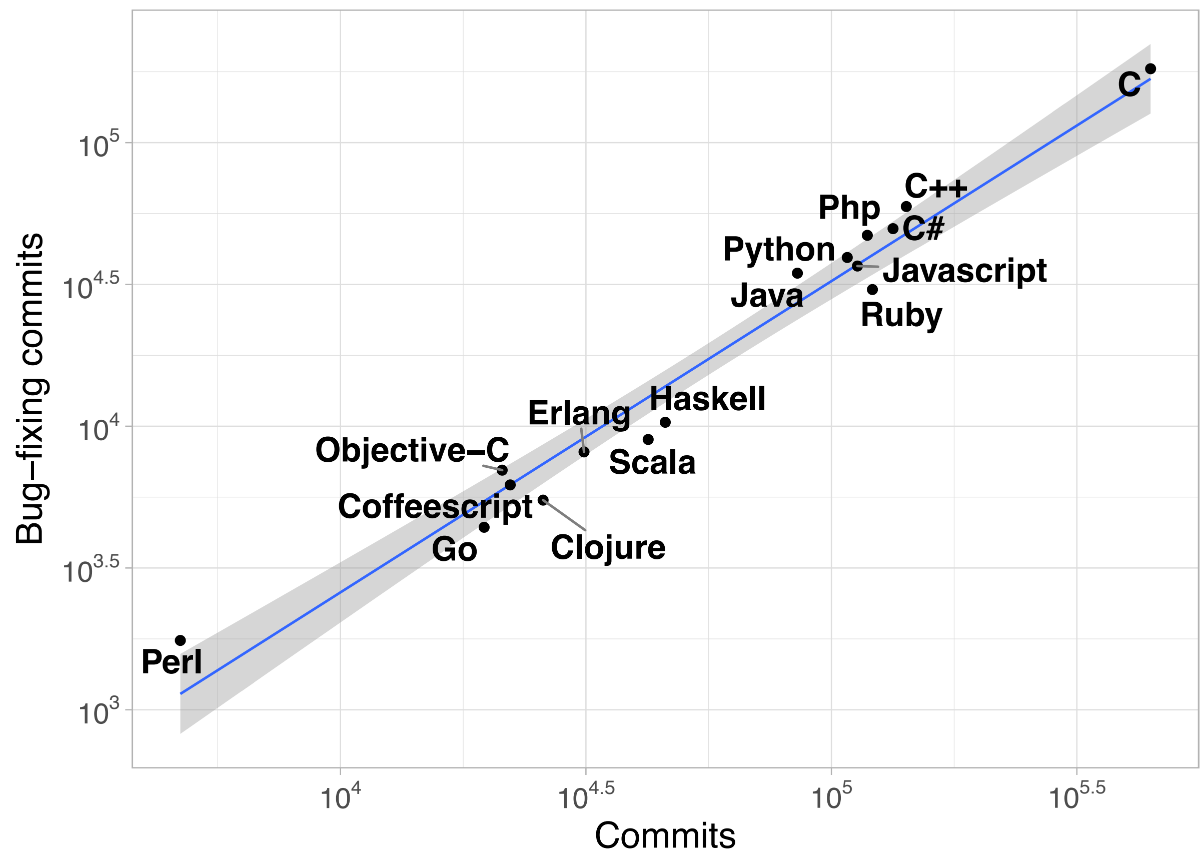
Table 1: Top three projects in each language

	Projects
`C`	linux, git, php-src
`C++`	node-webkit, phantomjs, mongo
`C#`	SignalR, SparkleShare, ServiceStack
`Objective-C`	AFNetworking, GPUImage, RestKit
`Go`	docker, lime, websocketd
`Java`	storm, elasticsearch, ActionBarSherlock
`CoffeeScript`	coffee-script, hubot, brunch
`JavaScript`	bootstrap, jquery, node
`TypeScript`	bitcoin, litecoin, qBittorrent
`Ruby`	rails, gitlabhq, homebrew
`Php`	laravel, CodeIgniter, symfony
`Python`	flask, django, reddit
`Pe`	gitolite, showdown, rails-dev-box
`C`	ghtTable, leiningen, clojurescript
`Erlang`	ChicagoBoss, cowboy, couchdb
`Haskell`	pandoc, yesod, git-annex
`Scala`	Play20, spark, scala

17,388,590 LOC
3,094,437
16
19,129 LOC
61,964

SIN #1

No normalization for lines of code or commits across languages!



729 projects and 1.5 million commits. Data has 148 un-analysed projects.

Found 47K authors vs 29K reported. Explained by paper using committer instead of developer.

80.7 million lines of code. A difference of 17 million SLOC unexplained.

SIN #2

No control for duplication!

Table 1: **Top three projects in each language**

Language	Projects
C	linux, git, php-src
C++	node-webkit, phantomjs, mongo
C#	SignalR, SparkleShare, ServiceStack
Objective-C	AFNetworking, GPUImage, RestKit
Go	[illegible]ker, lime, [illegible]
Java	[illegible]BarSherlock
CoffeeScri[illegible]	[illegible]script, [illegible]ch
JavaScript	bootstrap, jquery, node
TypeScript	bitcoin, litecoin, qBittorrent
Ruby	rails, gitlabhq, homebrew
Php	laravel, CodeIgniter, symfony
Python	flask, django, reddit
Perl	gitolite, showdown, rails-dev-box
Clojure	LightTable, leiningen, clojurescript
Erlang	ChicagoBoss, cowboy, couchdb
Haskell	pandoc, yesod, git-annex
Scala	Play20, spark, scala

Webkit

Webkit

Bitcoin

Bitcoin

SIN #2

No control for duplication!

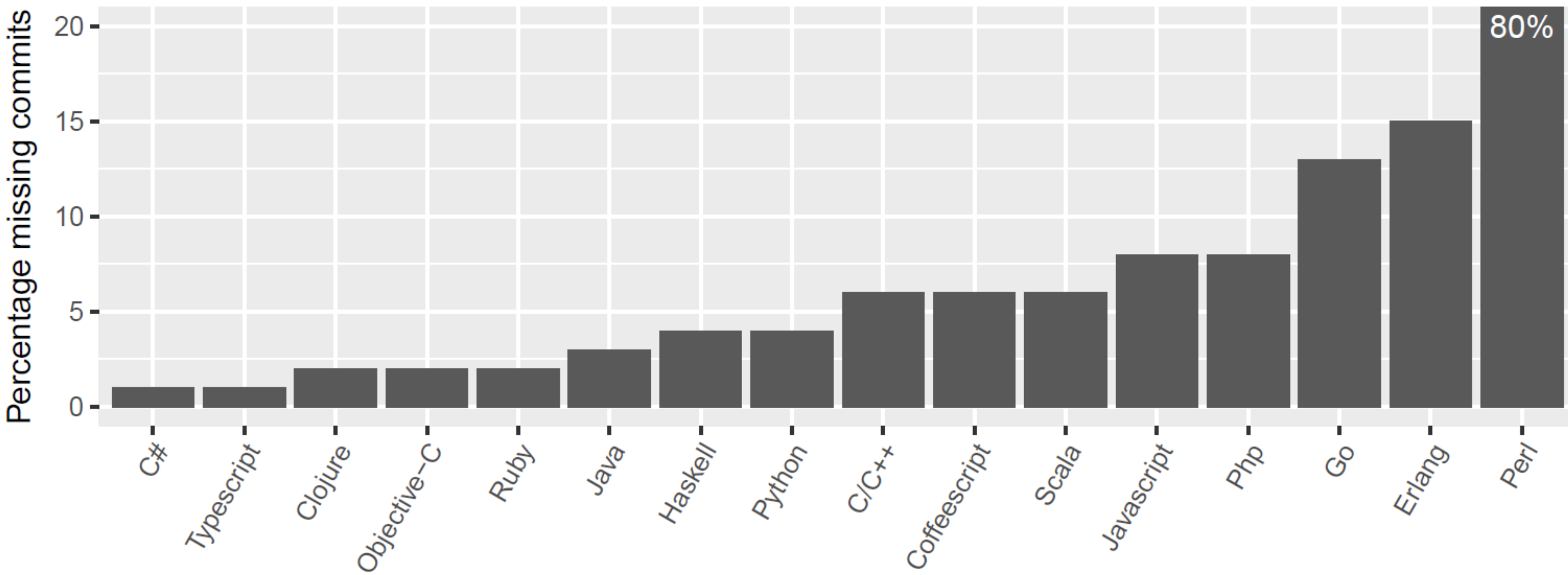
1.86% of data is duplicate commits

litecoin, mega-coin, memorycoin, bitcoin, bitcoin-qt-i2p, anoncoin, smallchange, primecoin, terracoin, zetacoin, datacoin, datacoin-hp, freicoin, ppcoin, namecoin, namecoin-qt, namecoinq, ProtoShares, QGIS, Quantum-GIS, incubator-spark, spark, sbt, xsbt, Play20, playframework, ravendb, SignalR, Newtonsoft.Json, Hystrix, RxJava, clojure-scheme, clojurescript

Lopes, Maj, Martins, Yang, Zitny, Sajnani, Vitek. *Déjà Vu: A Map of Code Duplicates on GitHub.* OOPSLA'17 https://doi.org/10.1145/3133908

SIN #3

Truncated data!



Out of 729 projects, 618 could be downloaded, 423 could be matched (due to owner missing)

Found 106K missing commits (~20% of data)

SIN #4

Erroneous Language Recognition!

First commit for TypeScript @ 2003-03-21


TypeScript

Paradigm	Multi-paradigm: scripting, object-oriented, structured, imperative, functional, generic
Designed by	Microsoft
Developer	Microsoft
First appeared	1 October 2012; 6 years ago[1]

`.ts` are translation files!

41 projects labeled as TypeScript, only 16 have code. Commits 10K=>3K.

Largest projects (`typescript-node-definitions`, `DefinitelyTyped`, `tsd`) are declarations with no code (34.6% of remaining commits).

SIN #4

Erroneous Language Recognition!

V8 is tagged as a JavaScript project

This is correct and it is the largest JavaScript project:

	Commits
C	16
C++	7
Python	488
JavaScript	2,907

Most JavaScript code is test!

`.C` `.cc` `.CPP` `.c++` `.cp` `.cxx` and `.h` are all ignored, only `.cpp` is used

Checked GitHub Linguist, as of 2014, able to recognize header files and all C++

16.2% of files are tests (801,248 files).

SIN #5

Multiple hypothesis testing

A common mistake in data-driven software engineering

	Original Authors					
	(a) FSE [26]		(b) cleaned data		(c) pV adjusted	
	Coef	P-val	Coef	P-val	FDR	Bonf
Intercept	-1.93	<0.001	-1.93	<0.001	–	–
log commits	2.26	<0.001	0.94	<0.001	–	–
log age	0.11	<0.01	0.05	<0.01	–	–
log size	0.05	<0.05	0.04	<0.05	–	–
log devs	0.16	<0.001	0.09	<0.001	–	–
C	0.15	<0.001	0.11	0.007	0.017	0.118
C++	0.23	<0.001	0.23	<0.001	<0.01	<0.01
C#	0.03	–	-0.01	0.85	0.85	1
Objective-C	0.18	<0.001	0.14	0.005	0.013	0.079
Go	-0.08	–	-0.1	0.098	0.157	1
Java	-0.01	–	-0.06	0.199	0.289	1
Coffeescript	-0.07	–	0.06	0.261	0.322	1
Javascript	0.06	<0.01	0.03	0.219	0.292	1
Typescript	-0.43	<0.001	–	–	–	–
Ruby	-0.15	<0.05	-0.15	<0.05	<0.01	0.017
Php	0.15	<0.001	0.1	0.039	0.075	0.629
Python	0.1	<0.01	0.08	0.042	0.075	0.673
Perl	-0.15	–	-0.08	0.366	0.419	1
Clojure	-0.29	<0.001	-0.31	<0.001	<0.01	<0.01
Erlang	0	–	-0.02	0.687	0.733	1
Haskell	-0.23	<0.001	-0.23	<0.001	<0.01	<0.01
Scala	-0.28	<0.001	-0.25	<0.001	<0.01	<0.01

16 p-Vals =>

family-wise error rate$=1-(1-.05)^{16}= .56$

Bonferroni divides cutoff by the num. of hypotheses

False Discovery Rate (FDR) allows an average pre-specified proportion of false positives in the list of “statistically significant” tests

Reyes, et al. 2018. *Statistical Errors in Software Engineering Experiments* ICSE https://doi.org/10.1145/3180155.3180161

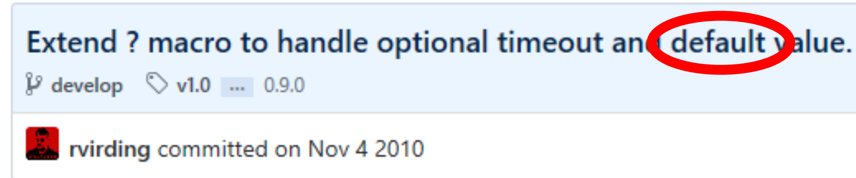
Shaffer. 1995. *Multiple Hypothesis Testing*. Ann.Rev.of Psychology. doi:10.1146/annurev.ps.46.020195.003021

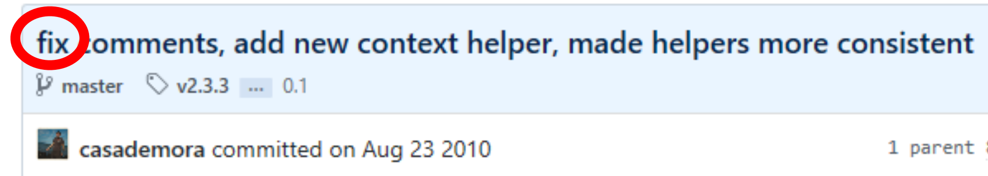
Benjamini, Hochberg. 1995. *Controlling the False Discovery Rate.* J.Royal Statistical Society. https://doi.org/10. 2307/2346101

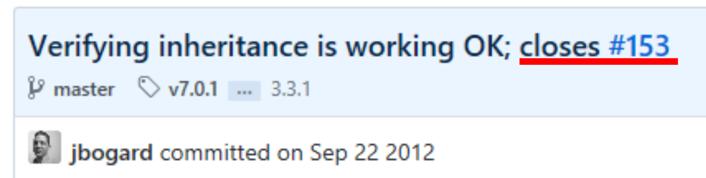
SIN #6

Egregious Labelling Errors!

Which should be labeled bug-fixing?







False positive rate: **36%**

False negative rate: **11%**

Selected randomly 400 commits; 10 independent developers

Each commit labelled by 3 experts. 2+ votes => bug fixes. 54% unanimous.

Meta-analysis of FP: (1) Substrings (2) Non-functional: e.g., changes to variable names (3) Comments (4) Feature enhancements (5) Mismatch: e.g., “this isn’t a bug” (6) Features with unclear messages

Mockus, Votta. 2000. *Identifying Reasons for Software Changes Using Historic Databases.* ICSM. https://doi.org/10.1109/ICSM.2000.883028

…, Filkov, Devanbu. 2009. *Fair and Balanced?: Bias in Bug-fix Datasets.* FSE. https://doi.org/10.1145/1595696.1595716

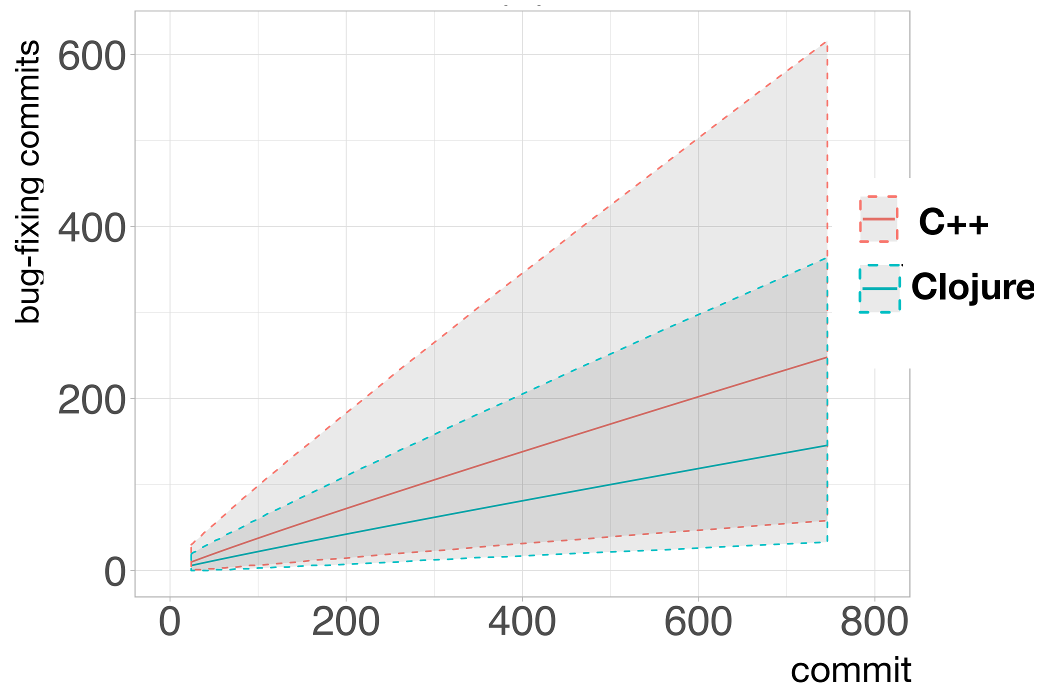
SIN #6

	Original Authors		Reanalysis					
	(a) FSE [26]		(b) cleaned data		(c) pV adjusted		(e) bootstrap	
	Coef	P-val	Coef	P-val	FDR	Bonf	Coef	sig.
~~C~~	~~0.15~~	~~<0.001~~	~~0.11~~	~~0.007~~	~~0.017~~	~~0.118~~	~~0.08~~	
~~C++~~	~~0.23~~	~~<0.001~~	~~0.23~~	~~<0.001~~	~~<0.01~~	~~<0.01~~	~~0.16~~	~~*~~
~~C#~~	~~0.03~~	~~–~~	~~-0.01~~	~~0.85~~	~~0.85~~	~~1~~	~~0~~	
~~Objective-C~~	~~0.18~~	~~<0.001~~	~~0.14~~	~~0.005~~	~~0.013~~	~~0.079~~	~~0.1~~	
~~Go~~	~~-0.08~~	~~–~~	~~-0.1~~	~~0.098~~	~~0.157~~	~~1~~	~~-0.04~~	
~~Java~~	~~-0.01~~	~~–~~	~~-0.06~~	~~0.199~~	~~0.289~~	~~1~~	~~-0.02~~	
~~Coffeescript~~	~~-0.07~~	~~–~~	~~0.06~~	~~0.261~~	~~0.322~~	~~1~~	~~0.04~~	
~~Javascript~~	~~0.06~~	~~<0.01~~	~~0.03~~	~~0.219~~	~~0.292~~	~~1~~	~~0.03~~	
~~Typescript~~	~~-0.43~~	~~<0.001~~						
Ruby	-0.15	<0.05	-0.15	<0.05	<0.01	0.017	-0.08	*
~~Php~~	~~0.15~~	~~<0.001~~	~~0.1~~	~~0.039~~	~~0.075~~	~~0.629~~	~~0.07~~	
~~Python~~	~~0.1~~	~~<0.01~~	~~0.08~~	~~0.042~~	~~0.075~~	~~0.673~~	~~0.06~~	
~~Perl~~	~~-0.15~~	~~–~~	~~-0.08~~	~~0.366~~	~~0.419~~	~~1~~	~~0~~	
Clojure	-0.29	<0.001	-0.31	<0.001	<0.01	<0.01	-0.15	*
~~Erlang~~	~~0~~	~~–~~	~~-0.02~~	~~0.687~~	~~0.733~~	~~1~~	~~-0.01~~	
Haskell	-0.23	<0.001	-0.23	<0.001	<0.01	<0.01	-0.12	*
~~Scala~~	~~-0.28~~	~~<0.001~~	~~-0.25~~	~~<0.001~~	~~<0.01~~	~~<0.01~~	~~-0.13~~	

Bootstrap:
1) sample projects with replacement;
2) #bug-fixing commits generated as B*~Binom(size=B,prob=1–FP)+Binom(size=C–B,prob=FN),
3) analyzed the resampled dataset with NBR. Repeat 100K times.

SIN #6

Egregious Labelling Errors!



Down with p-values

P-values are largely driven by # of observations [1].

Small p-values not necessarily practically important [2].

Practical significance assessed by model-based prediction intervals [3], which predict future commits.

Similar to confidence intervals in reflecting model-based uncertainty.

Differ in that they characterize plausible range of values of future individual data points.

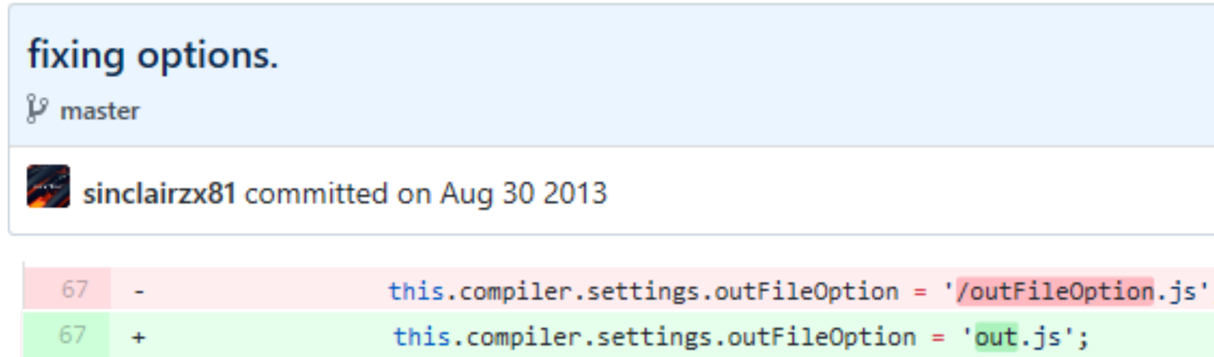
Halsey, et al. 2015. *The fickle P-value generates irreproducible results.* Nature Methods. https://doi.org/10.1038/nmeth.3288

Colquhoun. 2017. *The reproducibility of research and the misinterpretation of p-values.* https://doi.org/10.1098/rsos.171085

Kutner, et al. 2004. Applied Linear Statistical Models. https://books.google.cz/books?id=XAzYCwAAQBAJ

SIN #6

No Relevance to RQ!



How many errors are affected by features of the language?

SIN #7

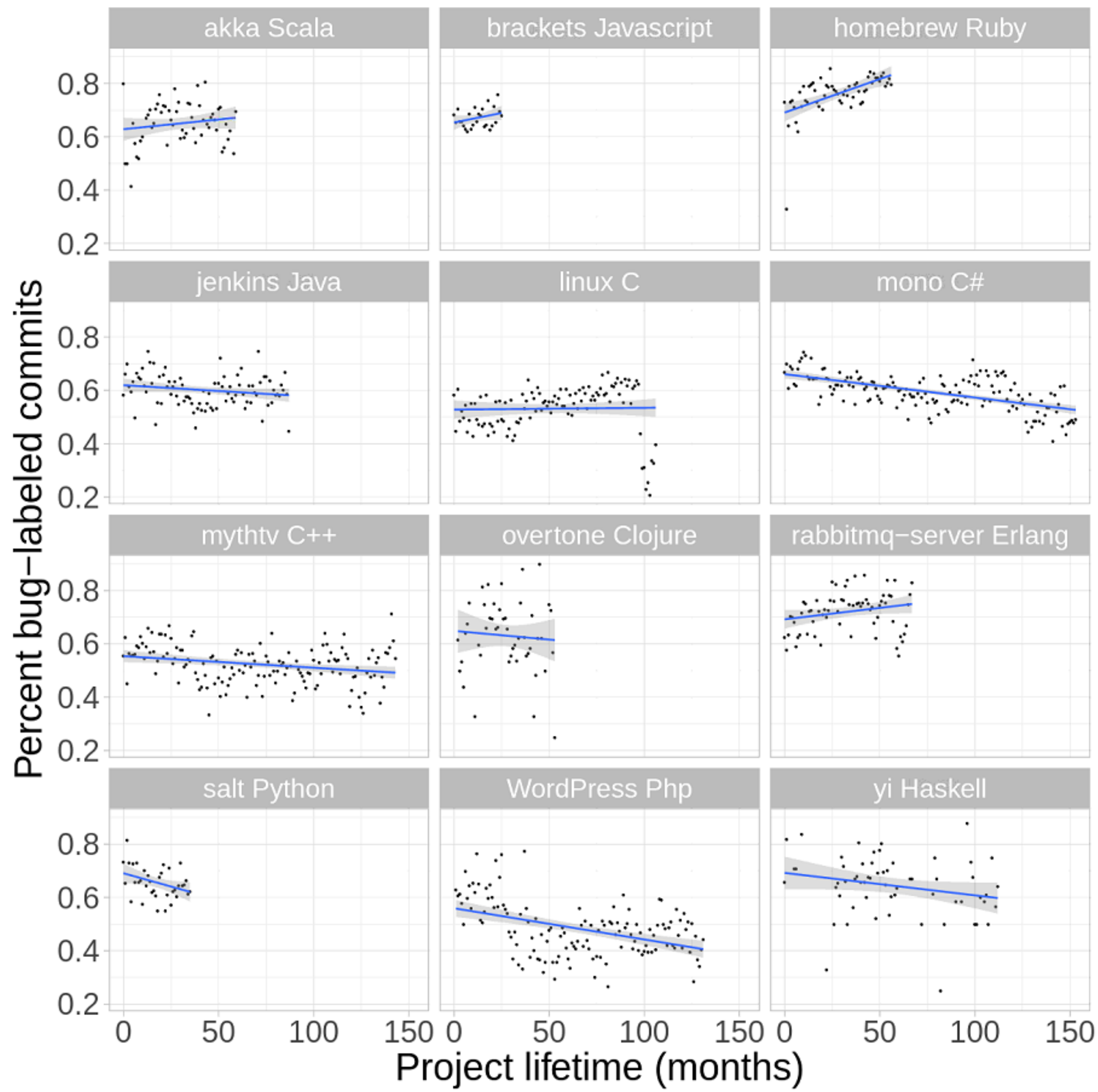
Uncontrolled Effects!

Developers influencing multiple projects (45K developers, 10% of them => 50% of the commits)

Some tasks, such as system programming, may be inherently more error prone than

Commercial vs opens source

Stars as a selection criteria for projects



A Large Scale Study of Programming Languages and Code Quality in Github

Baishakhi Ray, Daryl Posnett, Vladimir Filkov, Premkumar Devanbu
{bairay@, dpposnett@, filkov@cs., devanbu@cs.}ucdavis.edu
Department of Computer Science, University of California, Davis, CA, 95616, USA

ABSTRACT

What is the effect of programming languages on software quality? This question has been a topic of much debate for a very long time. In this study, we gather a very large data set from GitHub (729 projects, 80 Million SLOC, 29,000 authors, 1.5 million commits, in 17 languages) in an attempt to shed some empirical light on this question. This reasonably large sample size allows us to use a mixed-methods approach, combining multiple regression modeling with visualization and text analytics, to study the effect of language features such as static *v.s.* dynamic typing, strong *v.s.* weak typing on software quality. By triangulating findings from different methods, and controlling for confounding effects such as team size, project size, and project history, we report that language design does have a significant, but modest effect on software quality. Most notably, it does appear that strong typing is modestly better than weak typing, and among functional languages, static typing is also somewhat better than dynamic typing. We also find that functional languages are somewhat better than procedural languages. It is worth noting that these modest effects arising from language design are overwhelmingly dominated by the process factors such as project size, team size, and commit size. However, we hasten to caution the reader that even these modest effects might quite possibly be due to other, intangible process factors, *e.g.,* the preference of certain personality types for functional, static and strongly typed languages.

Categories and Subject Descriptors

D.3.3 [**PROGRAMMING LANGUAGES**]: [Language Constructs and Features]

General Terms

Measurement, Experimentation, Languages

Keywords

programming language, type system, bug fix, code quality, empirical research, regression analysis, software domain

FSE'14, November 16–21, 2014, Hong Kong, China
Copyright 2014 ACM 978-1-4503-3056-5/14/11...$15.00
http://dx.doi.org/10.1145/2635868.2635922

1. INTRODUCTION

A variety of debates ensue during discussions whether a given programming language is "the right tool for the job". While some of these debates may appear to be tinged with an almost religious fervor, most people would agree that a programming language can impact not only the coding process, but also the properties of the resulting artifact.

Advocates of strong static typing argue that type inference will catch software bugs early. Advocates of dynamic typing may argue that rather than spend a lot of time correcting annoying static type errors arising from sound, conservative static type checking algorithms in compilers, it's better to rely on strong dynamic typing to catch errors as and when they arise. These debates, however, have largely been of the armchair variety; usually the evidence offered in support of one position or the other tends to be anecdotal.

Empirical evidence for the existence of associations between code quality programming language choice, language properties, and usage domains, could help developers make more informed choices.

Given the number of other factors that influence software engineering outcomes, obtaining such evidence, however, is a challenging task. Considering software quality, for example, there are a number of well-known influential factors, including source code size [11], the number of developers [36, 6], and age/maturity [16]. These factors are known to have a strong influence on software quality, and indeed, such process factors can effectively predict defect localities [32].

One approach to teasing out just the effect of language properties, even in the face of such daunting confounds, is to do a *controlled experiment*. Some recent works have conducted experiments in controlled settings with tasks of limited scope, with students, using languages with static or dynamic typing (based on experimental treatment setting) [14, 22, 19]. While type of controlled study is *"El Camino Real"* to solid empirical evidence,another opportunity has recently arisen, thanks to the large number of open source projects collected in software forges such as GitHub.

GitHub contains many projects in multiple languages. These projects vary a great deal across size, age, and number of developers. Each project repository provides a historical record from which we extract project data including the contribution history, project size, authorship, and defect repair. We use this data to determine the effects of language features on defect occurrence using a variety of tools. Our approach is best described as mixed-methods, or triangulation [10] approach. A quantitative (multiple regression) study is further examined using mixed methods: text analysis, clustering, and visualization. The observations from the mixed methods largely confirm the findings of the quantitative study.

FSE 2014

RESEARCH HIGHLIGHTS

A Large-Scale Study of Programming Languages and Code Quality in Github

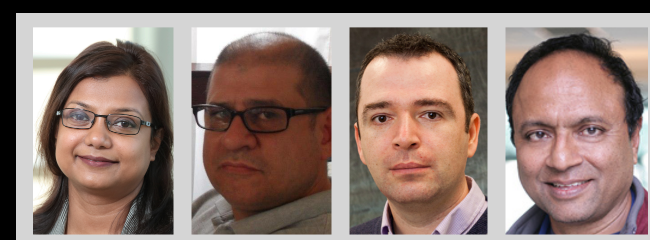
By Baishakhi Ray, Daryl Posnett, Premkumar Devanbu, Vladimir Filkov
Communications of the ACM, October 2017, Vol. 60 No. 10, Pages 91-100
10.1145/3126905
Comments

Credit: Getty Images

What is the effect of programming languages on software quality? This question has been a topic of much debate for a very long time. In this study, we gather a very large data set from GitHub (728 projects, 63 million SLOC, 29,000 authors, 1.5 million commits, in 17 languages) in an attempt to shed some empirical light on this question. This reasonably large sample size allows us to use a mixed-methods approach, combining multiple regression modeling with visualization and text analytics, to study the effect of language features such as static versus dynamic typing and allowing versus disallowing type confusion on software quality. By triangulating findings from different methods, and controlling for confounding effects such as team size, project size, and project history, we report that language design does have a significant, but modest effect on software quality. Most notably, it does appear that disallowing type confusion is modestly better than allowing it, and ... ing is also somewhat better than dynamic typing. We also find that fun... han procedural languages. It is worth noting that these modest ef... verwhelmingly dominated by the process factors such as project size, team size, and commit size. However, we caution the

ARTICLE CONTENTS:

CACM 2017



Baishaki Ray | Daryl Posnett | Vladimir Filikov | Premkumar Devanbu

UC Davis

Correlation is not Causation

Result1: *Some languages have a greater **association** with defects than others, although the effect is small.*

The first principle is that you must not fool yourself—and you are the easiest person to fool. So you have to be very careful about that. After you've not fooled yourself, it's easy not to fool other scientists. You just have to be honest in a conventional way after that.

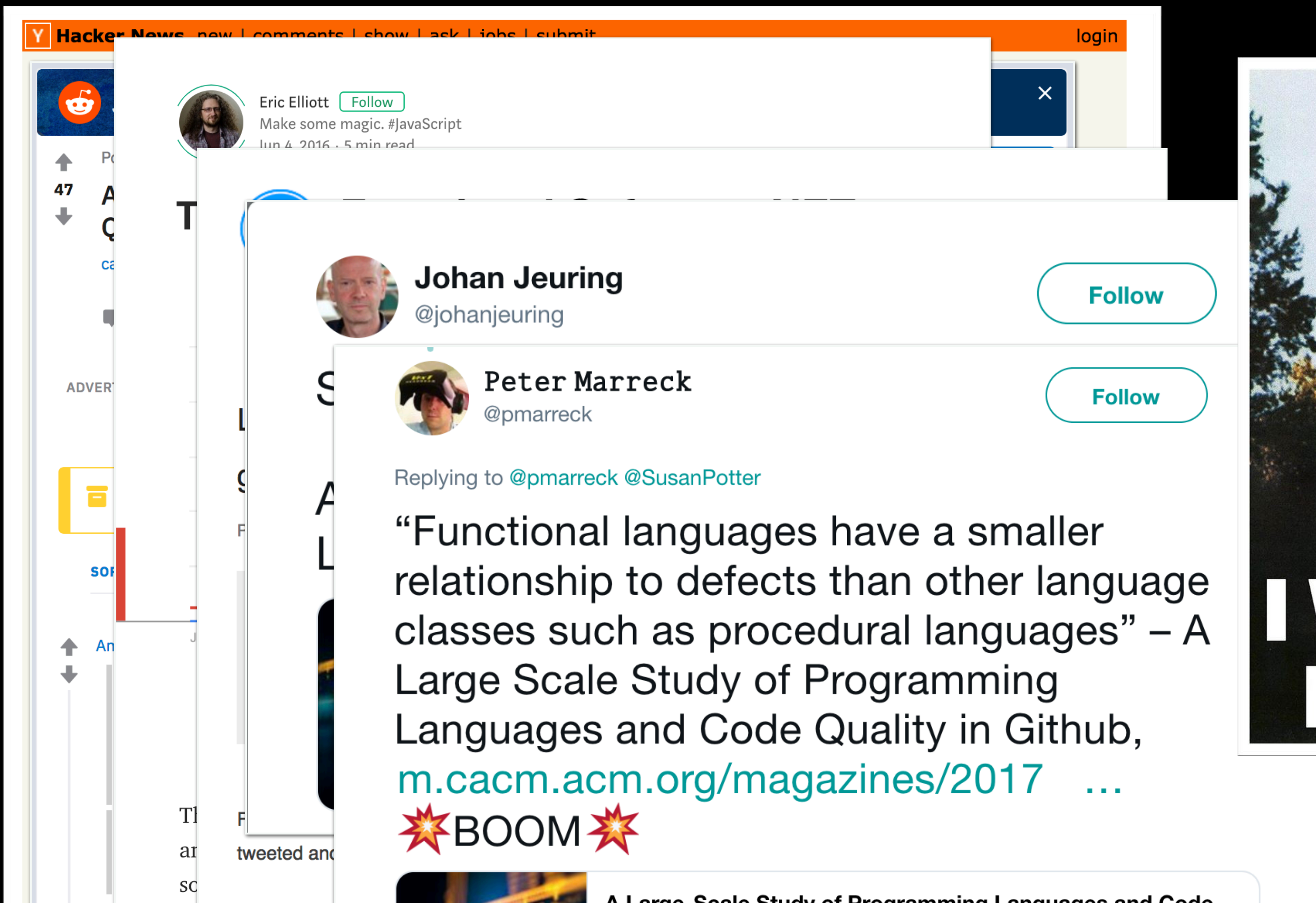
— R. Feynman, Cargo Cult Science, 1974

Correlation is not Causation

Sleeping with one's shoes on is strongly correlated with waking up with a headache.

Therefore, sleeping with one's shoes on causes headache.

Correlation is not Causation

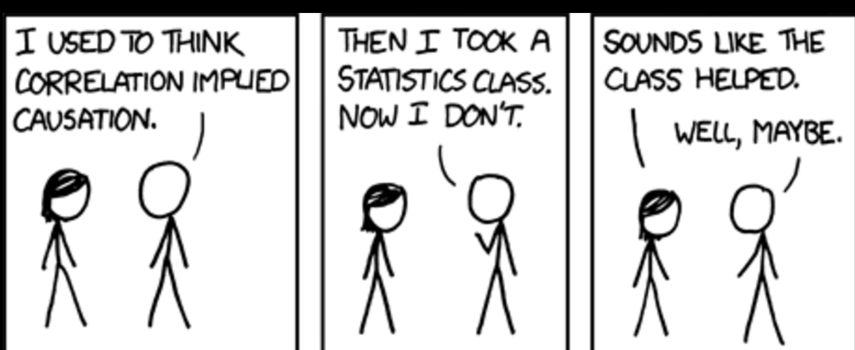


Correlation is not Causation

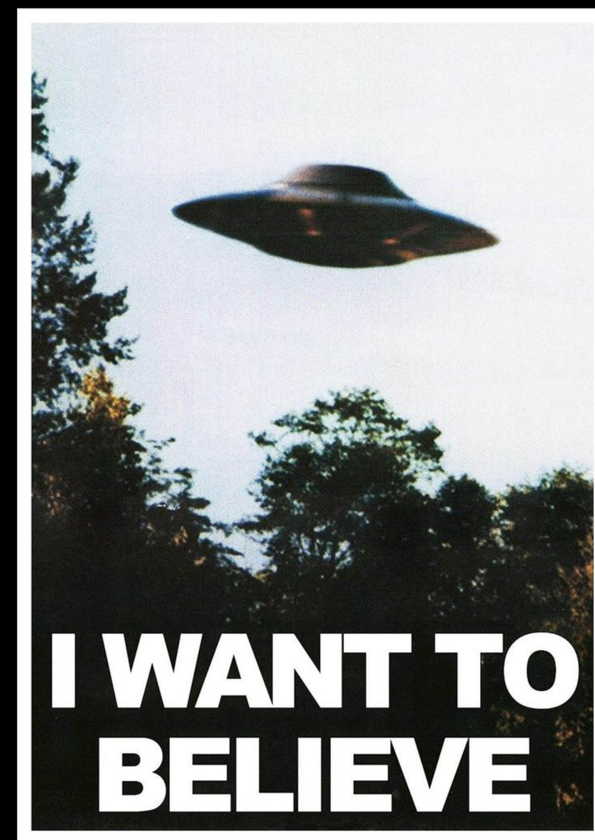
"…They found language design did have a signicant, but modest effect on software quality."

"…The results indicate that strong languages have better code quality than weak languages."

"…functional languages have an advantage over procedural languages."



	Cites	Self
Cursory	77	1
Methods	12	0
Correlation	2	2
Causation	24	3



On the Impact of Programming Languages on Code Quality

Emery D. Berger, Celeste Hollenbeck, Petr Maj, Olga Vitek, Jan Vitek

(Submitted on 29 Jan 2019)

This paper is a reproduction of work by Ray et al. which claimed to have uncovered a statistically significant association between eleven programming languages and software defects in projects hosted on GitHub. First we conduct an experimental repetition, repetition is only partially successful, but it does validate one of the key claims of the original work about the association of ten programming languages with defects. Next, we conduct a complete, independent reanalysis of the data and statistical modeling steps of the original study. We uncover a number of flaws that undermine the conclusions of the original study as only four languages are found to have a statistically significant association with defects, and even for those the effect size is exceedingly small. We conclude with some additional sources of bias that should be investigated in follow up work and a few best practice recommendations for similar efforts.

Comments:	21 pages
Subjects:	**Software Engineering (cs.SE)**
Cite as:	arXiv:1901.10220 **[cs.SE]**
	(or **arXiv:1901.10220v1** **[cs.SE]** for this version)

ShriramKrishnamurthi
@ShriramKMurthi

Following

The "debunking" paper by @emeryberger, @j_v_66, @olgavitek, and others, of that "programming languages and code quality" study, hits arXiv. Expect fireworks.

Boffins debunk study claiming certain languages (cough, C, PH
Hard evidence that some coding lingo encourage flaws remains elu
theregister.co.uk

The Register®
Biting the hand that feeds IT

NTRE | **SOFTWARE** | SECURITY | DEVOPS | BUSINESS | PERSONAL TECH | SCIENCE

Software

Boffins debunk study claiming certain languages (cough, C, PHP, JS...) lead to more buggy code than others

Hard evidence that some coding lingo encourage flaws remains elusive

By Thomas Claburn in San Francisco 30 Jan 2019 at 21:45 154 SHARE ▼

FSE 2017

…I don’t understand why …use a Bonferroni correction, which is generally overly conservative. Why not use a Benajamini-Hotchberg?…

…missing code and data…

…largest source of contrasting results…comes from the bootstrapping method. This was clever. However, it relies on the really low bug-labeling accuracy data…a larger sample of rated messages, with multiple raters, would be worthwhile…

ICSE 2018

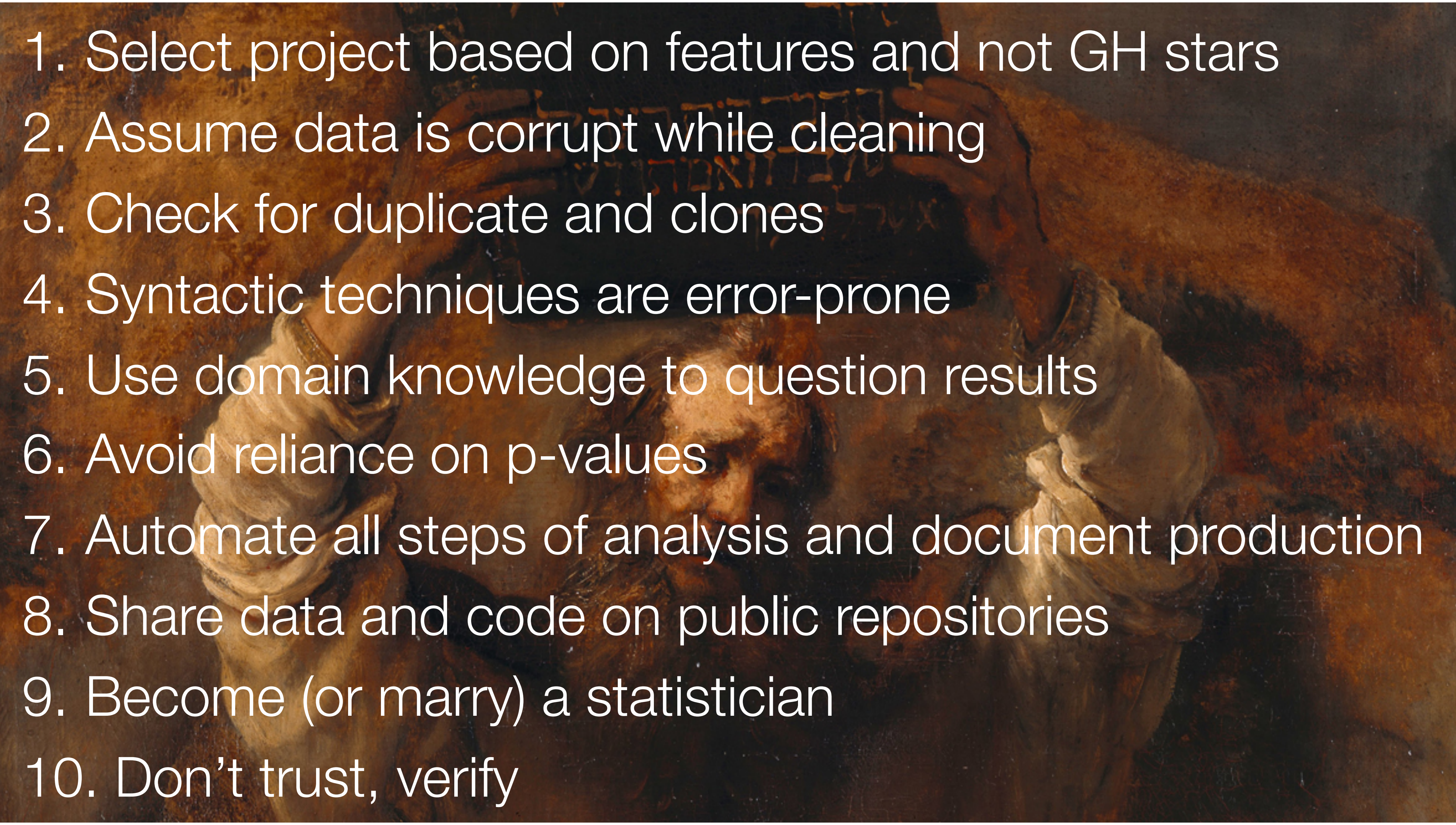
*….Hence, the **reanalysis actually confirmed** the original conclusion…*

*…The current study produces essentially the same result … that some of the language coefficients reported to be statistically significant in the original paper, lose statistical significance now, given **some differences in operationalization or analysis**…*

*…The paper appears **politically motivated**…*

The first principle is that you must not fool yourself—and you are the easiest person to fool. So you have to be very careful about that. After you've not fooled yourself, it's easy not to fool other scientists. You just have to be honest in a conventional way after that.

— R. Feynman, Cargo Cult Science, 1974


1. Select project based on features and not GH stars
2. Assume data is corrupt while cleaning
3. Check for duplicate and clones
4. Syntactic techniques are error-prone
5. Use domain knowledge to question results
6. Avoid reliance on p-values
7. Automate all steps of analysis and document production
8. Share data and code on public repositories
9. Become (or marry) a statistician
10. Don’t trust, verify

GETTING EVERYTHING WRONG WITHOUT DOING ANYTHING RIGHT!

or

The perils of large-scale analysis of GitHub data


Petr


Celeste


Emery


Olga


Jan