

# Intro to Statistics to Assess Randomized Algorithms

Dr. Andrea Arcuri  
Prof. of Software Engineering  
Kristiania University College, Oslo, Norway  
Oslo Metropolitan University, Oslo, Norway

# In This Seminar

- Why using statistics?
- Basic statistics
  - eg, Wilcoxon-Mann-Whitney U-test
- Examples in R
- If time... automate table generation in Latex
  - ie, avoid filling data by hand
  - add statistics with just couple of lines of code...

Just an *informal* intro for  
beginners...

quite complex topic if you  
look at the details...

Focus on comparisons of testing techniques, tools and algorithms

For empirical studies with human subjects, things are bit different... (eg, constraints on sample size)

Why bother with  
*statistics*?

# Experiments

- Evaluate if new technique **A** is better than **B**
  - eg, compared to the state-of-the-art
  - code coverage, fault detection, etc
- Large experiments can be too time consuming
- Not just software testing
  - most branches of engineering and science (medicine, biology, etc.)

Statistics: *have I run  
enough experiments?*

# Randomised Algorithms

- Run twice, can get different results
- Many different kinds
  - Random Testing (RT)
  - Search-Based Software Testing (SBST)
  - Some kinds of Dynamic Symbolic Execution (DSE)
  - etc
- How many *N runs* with different seeds?



# Case Study Selection

- Space **Z** of possible problem instances
- On some instances, technique **A** works fine (eg better than **B**), but not on others
- How many **N** instances from **Z** to use?
- How can you select?
  - even if unbiased, maybe you just got lucky...
- Still probability involved

# Motivating Example

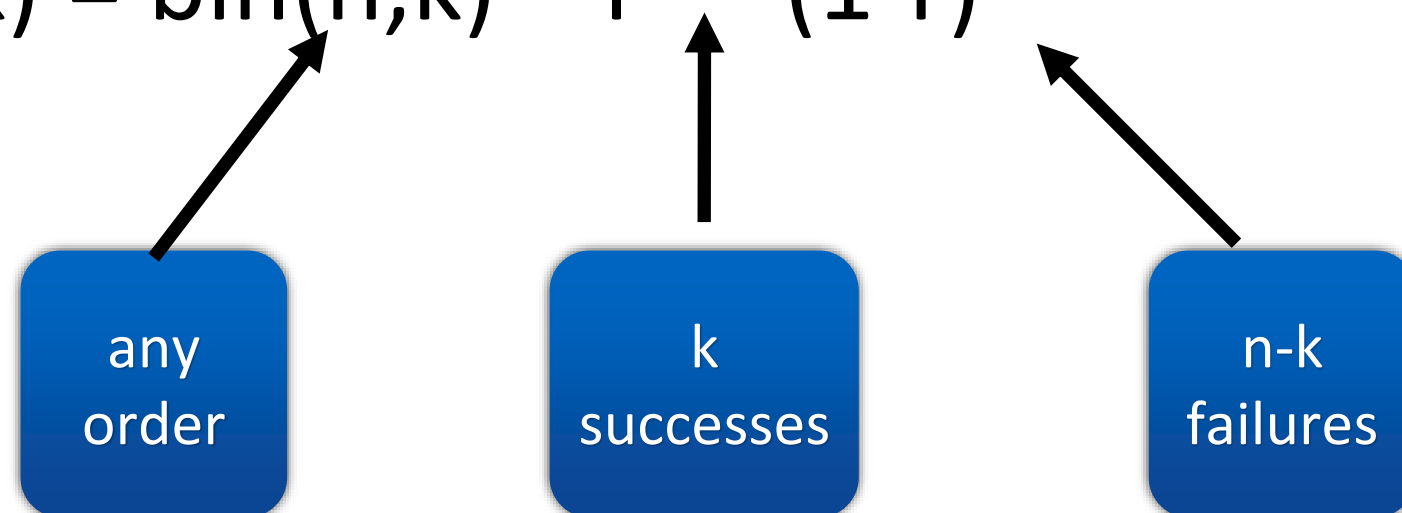
- Two algorithms, **A** and **B**
- Run **n=10** times, or 1 on n=10 different instances
- Binary output: *pass* or *fail*
- **A**: successful **7** times
- **B**: successful **5** times
- **20%** of success rate difference
- *Is A better than B???*

# Probability

- Probability  $p$  in  $[0,1]$  of:
  - a successful run in a randomised algorithm
  - sampling/choosing an instance for which algorithm is successful
- $k$  successes out of  $n$  runs/instances
- $n-k$  failures
- order of  $k$  and  $n-k$  does not matter
- **estimated** success rate of  $p$ :  $r=k/n$ 
  - for  $n$  to infinite,  $r$  would converge to  $p$

# Binomial Distribution $B(n,r)$

- What is the probability  $P$  of having  $k$  successes when I have  $n$  instance/runs with success rate  $r$ ?
- $P(B(n,r)=k) = \text{bin}(n,k) * r^k * (1-r)^{(n-k)}$



# Examples

- In R, “`dbinom(k,n,r)`”
- $P(B(10,0.7)=7) = 0.26$ 
  - If real success rate is 70%, then there is only a 26% probability of obtaining 7 successes out of 10 runs/instances
- But what if actual success rate was 50%?
- $P(B(10,0.5)=7) = 0.11$
- $26\% > 11\%$ , but still 11% not so unlikely...

The most likely estimate  
 $r=k/n$  might not have  
high probability of being  
correct

# Comparing A with B

- **A:**  $P(B(10,0.7)=7) = 0.26$
- **B:**  $P(B(10,0.5)=5) = 0.24$
- $P(B(10,.7)=7) * P(B(10,.5)=5) = 0.06$
- only 6% probability that, if X has success rate 70% and Y has 50%, then X obtains 7 successes and Y 5

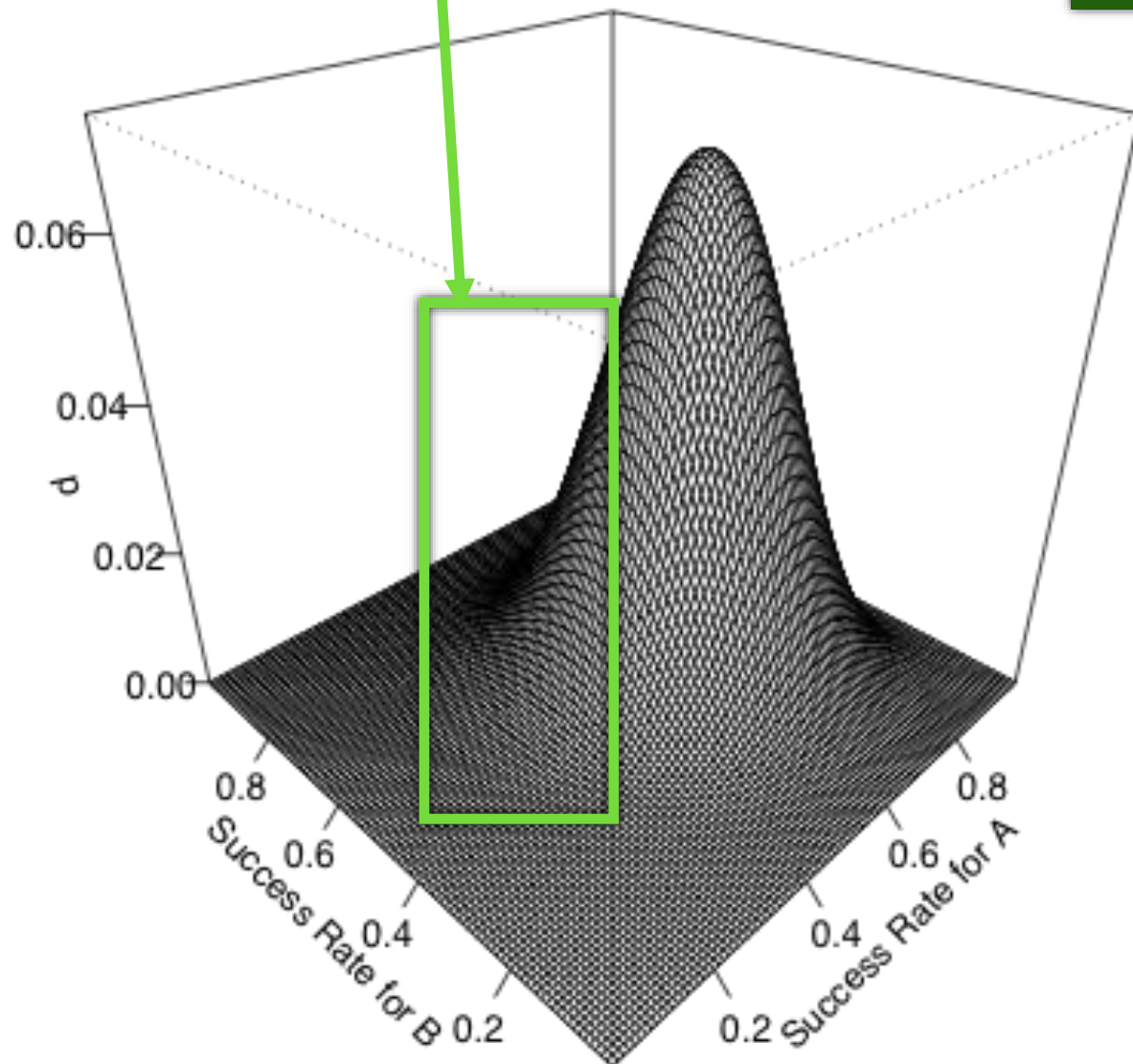
# But what if...

- ... **A** and **B** have exactly same success rate 60%?
  - ie, got bit *lucky* with **A**, and bit *unlucky* with **B**
- $P(B(10,.6)=7) * P(B(10,.6)=5) = 0.04$
- 4% not so huge difference from 6%...
- What if **B** is actually better?
  - less likely, but still possible



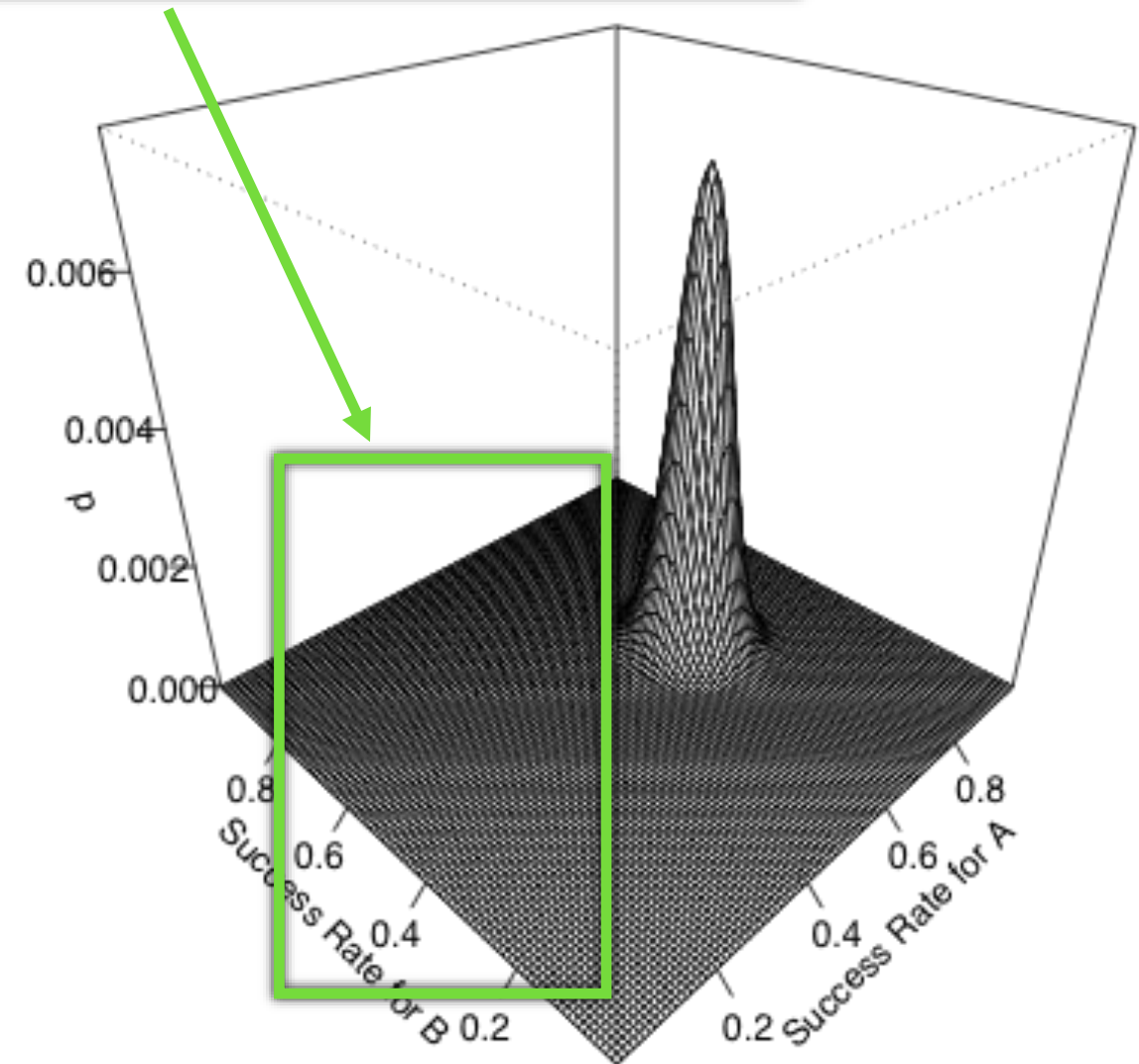
Probability (Z axis) of getting 70% successes with **A** and 50% for **B** for different success rates of **A** and **B** (X and Y axes)

>0% that B is better



$n=10$   
A:  $k=7$   
B:  $k=5$

extremely unlikely that B was better



$n=100$   
A:  $k=70$   
B:  $k=50$

# Don't need math details...

- Statistical test will tell you:
  - 70% and 50% are not really significant if  $n=10$
  - But strongly significant if  $n=100$
- Important to know when and which statistical tests to use
  - eg, in this example, *Fisher Exact Test*

```
compareAB_n10 <- function(){  
  n = 10  
  ka = 7  
  kb = 5  
  m = matrix(c(ka,n-ka,kb,n-kb),2,2)  
  fisher.test(m)  
}
```

```
compareAB_n100 <- function(){  
  n = 100  
  ka = 70  
  kb = 50  
  m = matrix(c(ka,n-ka,kb,n-kb),2,2)  
  fisher.test(m)  
}
```

```
> source("example.R")
> compareAB_n10()
```

### Fisher's Exact Test for Count Data

```
data: m
p-value = 0.6499
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.2725276 21.9391352
sample estimates:
odds ratio
 2.234096
```

```
> compareAB_n100()
```

### Fisher's Exact Test for Count Data

```
data: m
p-value = 0.005937
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 1.256203 4.351520
sample estimates:
odds ratio
 2.323215
```

p-value can give you an hint on whether you can say the two compared A and B are different

eg,  $< 0.05$

# What Statistical Tools to Use?

- Many options, open-source and commercial
- Two most used/popular: R and Python
- **R**: open source, *specific* for statistics, large community
- **Python**: open source, general scripting language, good libraries for statistics

# How to Choose?

- If you already know Python outside of statistics, go for Python
- If not, R might be safest option (de-facto standard)
- Both language have their quirks
- Personally I use R, but it does not mean R is a good language...
- **Most important feature:** need to have scripting, not just a GUI

[\[Home\]](#)

## Download

[CRAN](#)

## R Project

[About R](#)[Logo](#)[Contributors](#)[What's New?](#)[Mailing Lists](#)[Reporting Bugs](#)[Development Site](#)[Conferences](#)[Search](#)

## R Foundation

[Foundation](#)[Board](#)[Members](#)[Donors](#)[Donate](#)

## Documentation

[Manuals](#)[FAQs](#)[The R Journal](#)

# The R Project for Statistical Computing

## Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred [CRAN mirror](#).

If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

## News

- **R version 3.3.1 (Bug in Your Hair)** has been released on Tuesday 2016-06-21.
- **R version 3.2.5 (Very, Very Secure Dishes)** has been released on 2016-04-14. This is a rebadging of the quick-fix release 3.2.4-revised.
- **Notice XQuartz users (Mac OS X)** A security issue has been detected with the Sparkle update mechanism used by XQuartz. Avoid updating over insecure channels.
- The **R Logo** is available for download in high-resolution PNG or SVG formats.
- **useR! 2016**, will take place at Stanford University, CA, USA, June 27 - June 30, 2016.
- **The R Journal Volume 7/2** is available.
- **R version 3.2.3 (Wooden Christmas-Tree)** has been released on 2015-12-10.
- **R version 3.1.3 (Smooth Sidewalk)** has been released on 2015-03-09.

Programming with a *dynamically typed language* (eg, R and Python) can be a *painful experience*...

... but scripts for statistics are usually *short* and *not complex*...

... and *re-used* paper after paper...



# Statistical Tests

# Statistical Difference

- Are **A** and **B** different?
- Type I Error:
  - Claiming difference when no difference
- Statistical tests give *p-value*
- *P-value*: probability of committing Type I error
- eg, *p-value*=0.1 means 10% chance to be wrong if state two algorithms are different
  - more formally, probability of observing the data if the null hypothesis is true

# P-value < 0.05

- “Often”, statistically significant if < 0.05
- Rule of thumb: do not claim difference unless you are very sure...
- *Completely arbitrary threshold*
- Why not 1%? Or 10%?
- In decision problems, you need to make a choice anyway...
  - eg choose technique/tool to test software
- Just report the *p-values*...

# On the Origins of the .05 Level of Statistical Significance

MICHAEL COWLES    *York University, Canada*  
CAROLINE DAVIS    *York University, Canada*

**ABSTRACT:** *Examination of the literature in statistics and probability that predates Fisher's Statistical Methods for Research Workers indicates that although Fisher is responsible for the first formal statement of the .05 criterion for statistical significance, the concept goes back much further. The move toward conventional levels for the rejection of the hypothesis of chance dates from the turn of the century. Early statements about statistical significance were given in terms of the probable error. These earlier conventions were adopted and restated by Fisher.*

Cochran feels that Fisher was fairly casual about the choice, "as the words *convenient* and *prefers* have indicated" (p. 16). However, the statement quoted above leaves no doubt about Fisher's acceptance of the level as the critical cutoff point, once he had decided upon it.

Other writers, well-versed in the history and development of probability, have also fostered the attitude that the level is an arbitrary one. Yule and Kendall (1950), in the 14th edition of a book first published by Yule in 1911, state,

# More than 100 years old

- Student [W. S. Gosset]. *The probable error of a mean*. Biometrika, **1908**
  - about *t*-test: “three times the probable error in the normal curve, for most purposes, would be considered significant”
- Fisher, R. A. *Statistical methods for research workers*. Edinburgh: Oliver & Boyd, **1925**.
  - just rounding 0.0456 to 0.05
  - “rejection level of 3PE is equivalent to two times the SD (in modern terminology, a *z* score of 2)”

# Tests You Need To Know

- If binary (eg success rates): **Fisher** Exact test
- Otherwise: **Wilcoxon-Mann-Whitney** U-test
- Student *t*-test most known, but you shouldn't use it...

# Fisher Exact Test

- In R: `fisher.test(m)`
- `m` is a 2x2 matrix
- Check if can claim different success rates

# Success for A	# Failures for A
# Success for B	# Failures for B

# Example

- Automated Bug Fixing
- You *somehow* sample  $N$  bugs to repair, with as *little bias* as possible
- Want to know if your novel technique **A** is better than state of the art **B**
- Run experiments with  **$N=30$**
- **A** repairs **20**, whereas **B** repairs **15**
- Can claim with *high confidence* that **A** is better?



```
compareAB_bugFixing <- function(){  
  n = 30  
  ka = 20  
  kb = 15  
  m = matrix(c(ka,n-ka,kb,n-kb),2,2)  
  fisher.test(m)  
}
```

```
> source("example.R")  
> compareAB_bugFixing()
```

### Fisher's Exact Test for Count Data

```
data:  m  
p-value = 0.2949  
alternative hypothesis: true odds ratio is not equal to 1  
95 percent confidence interval:  
 0.625873 6.482781  
sample estimates:  
odds ratio  
 1.976627
```

But does it matter that  
not enough evidence to  
claim *A* is better?

# Not really...

- Evidence of 5 bugs fixed by **A** but not **B**
- Can't say **A** is *better*, but can say **A** is *useful*
- Techniques can be combined/integrated
- Given a domain (eg bug fixing), no single paper can solve all instances

# “Similar” Example

- Still Automated Bug Fixing
- Now **A** and **B** are randomised (eg Genetic Programming)
- *Single bug*, run experiments **30** times
- **A** repairs **20** times, whereas **B** repairs **15** times
- Can claim with *high confidence* that *A is better?*
- Can claim that **A** is *useful?*

# Bit different now...

- *A better?* No (recall high *p-value* 0.29)
- **A** useful? Arguably no, i.e. equivalent to **B**
- In this case, warranted to run some more experiments before trying to publish it...

# But what if...

- ... I am comparing randomised algorithms on many problem instances?
- Typical case of SBST and when comparing with RT

# Do Automatically Generated Unit Tests Find Real Faults?

## An Empirical Study of Effectiveness and Challenges

Sina Shamshiri\*, René Just<sup>†</sup>, José Miguel Rojas\*, Gordon Fraser\*, Phil McMinn\* and Andrea Arcuri<sup>‡</sup>

\*Department of Computer Science, University of Sheffield, UK

<sup>†</sup>Department of Computer Science & Engineering, University of Washington, Seattle, WA, USA

<sup>‡</sup>Scienta, Norway, and University of Luxembourg

\*{sina.shamshiri, j.rojas, gordon.fraser, p.mcminn}@sheffield.ac.uk, <sup>†</sup>rjust@cs.washington.edu, <sup>‡</sup>aa@scienta.no

**Abstract**—Rather than tediously writing unit tests manually, tools can be used to generate them automatically — sometimes even resulting in higher code coverage than manual testing. But how good are these tests at actually finding faults? To answer this question, we applied three state-of-the-art unit test generation tools for Java (Randoop, EvoSuite, and Agitar) to the 357 real faults in the Defects4J dataset and investigated how well the generated test suites perform at detecting these faults. Although the automatically generated test suites detected 55.7% of the faults overall, only 19.9% of all the individual test suites detected a fault. By studying the effectiveness and problems of the individual tools and the tests they generate, we derive insights to support the development of automated unit test generators that achieve a higher fault detection rate. These insights include 1) improving the obtained code coverage so that faulty statements are executed in the first instance, 2) improving the propagation of faulty program states to an observable output, coupled with the generation of more sensitive assertions, and 3) improving the simulation of the execution environment to detect faults that are dependent on external factors such as date and time.

faults from open source projects [25]. We applied three state-of-the-art unit test generation tools for Java, RANDOOP [30], EVOSUITE [13], and AGITARONE [1], on the Defects4J dataset, and investigated whether the resulting test suites can find the faults. Specifically, we aimed to answer the following research question:

“How do automated unit test generators perform at finding faults?”

Our experiments indicate that, while it is possible for automated test generation tools to find more than half of the faults, running any individual tool on a given software project is far from providing any confidence about finding faults. In fact, only 19.9% of all the test suites generated as part of our experiments find a fault. This raises a second research question:

“How do automated unit test generators need to be improved to find more faults?”

Our experiments indicate that in many cases, code coverage

# Context

- Generate tests that can trigger bug
- 357 bugs from Defect4J
- Three compared tools
  - *Agitar*: commercial, 1 run per bug
  - *Randoop*: open-source, randomized, 10 runs per bug
  - *EvoSuite*, open-source, randomized, 10 runs per bug



# Results

- Bugs found only by...
  - *Agitar*: 28
  - *Randoop*: 12
  - *EvoSuite*: 35
- No technique subsumes the others
- In many cases, *Randoop/EvoSuite* found bug only in a *subset* of the 10 runs...

# So 2 Steps Analysis

- On single instances to check **A** vs **B** performance
  - need to take into account randomness
- On whole case study
  - based on how was selected, from random sample can infer/estimate performance on whole domain
  - *more on this later...*

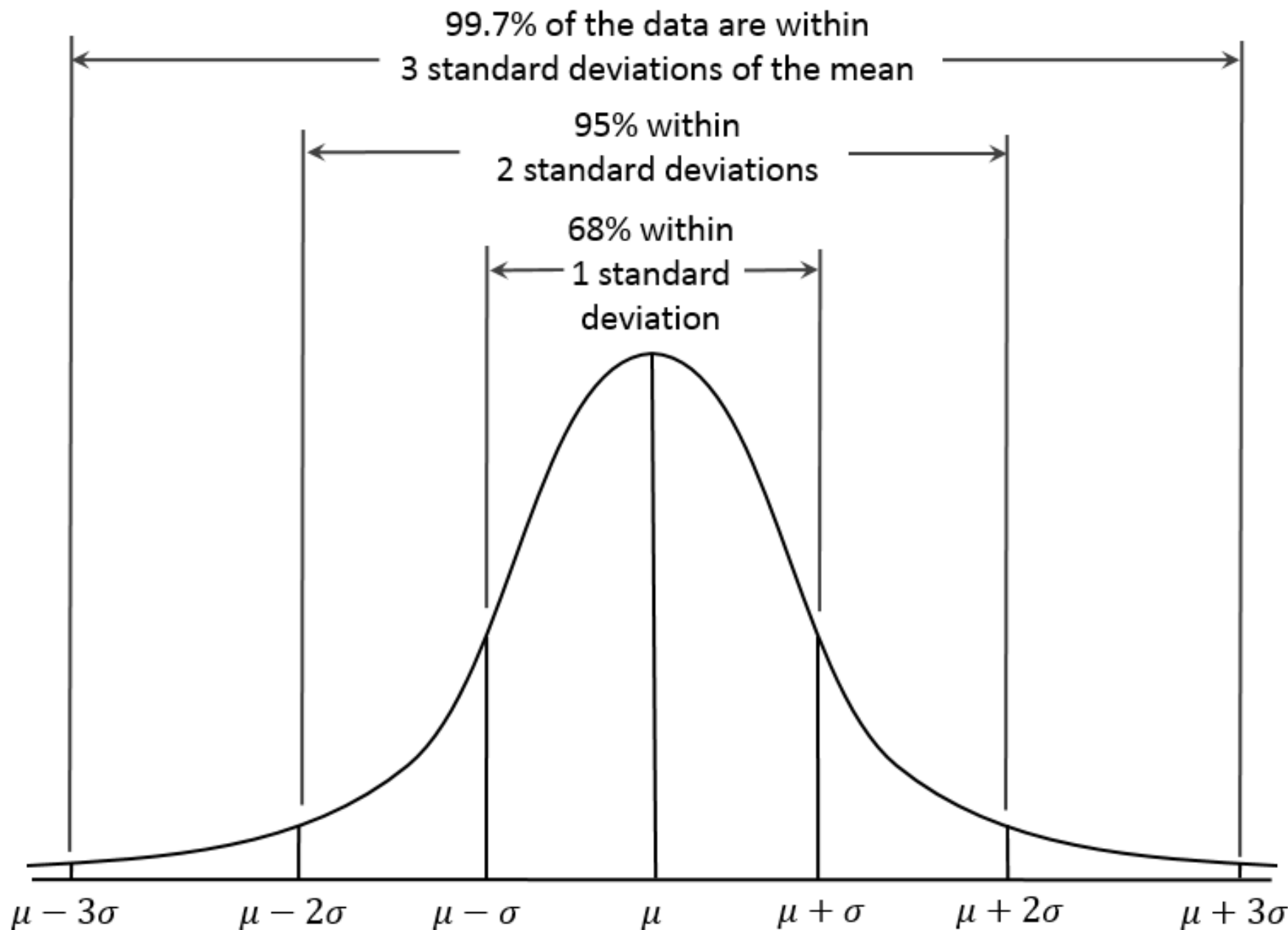
# Non-Binary Performance

- Not just binary *success* and *failure*
- Performance metrics as numerical values
  - branch coverage in test generation, e.g. 75% vs 42%
  - number of test executions in regression testing
  - etc
- Usually looking at aggregated values, like *mean* and *median*

# Student *t*-test

- Used to compare *mean* (average) values of two distributions
- It requires *normality* of data, but...
- “*The assumptions of most mathematical models are always false to a greater or lesser extent*”
  - Glass G, Peckham P, Sanders J. *Consequences of failure to meet assumptions underlying the fixed effects analyses of variance and covariance*. Review of Educational Research 1972; 42(3):237–288.

# Normal Distribution



Fully defined by two properties: mean and standard deviation

Many phenomena in nature fit in a normal distribution

# Example

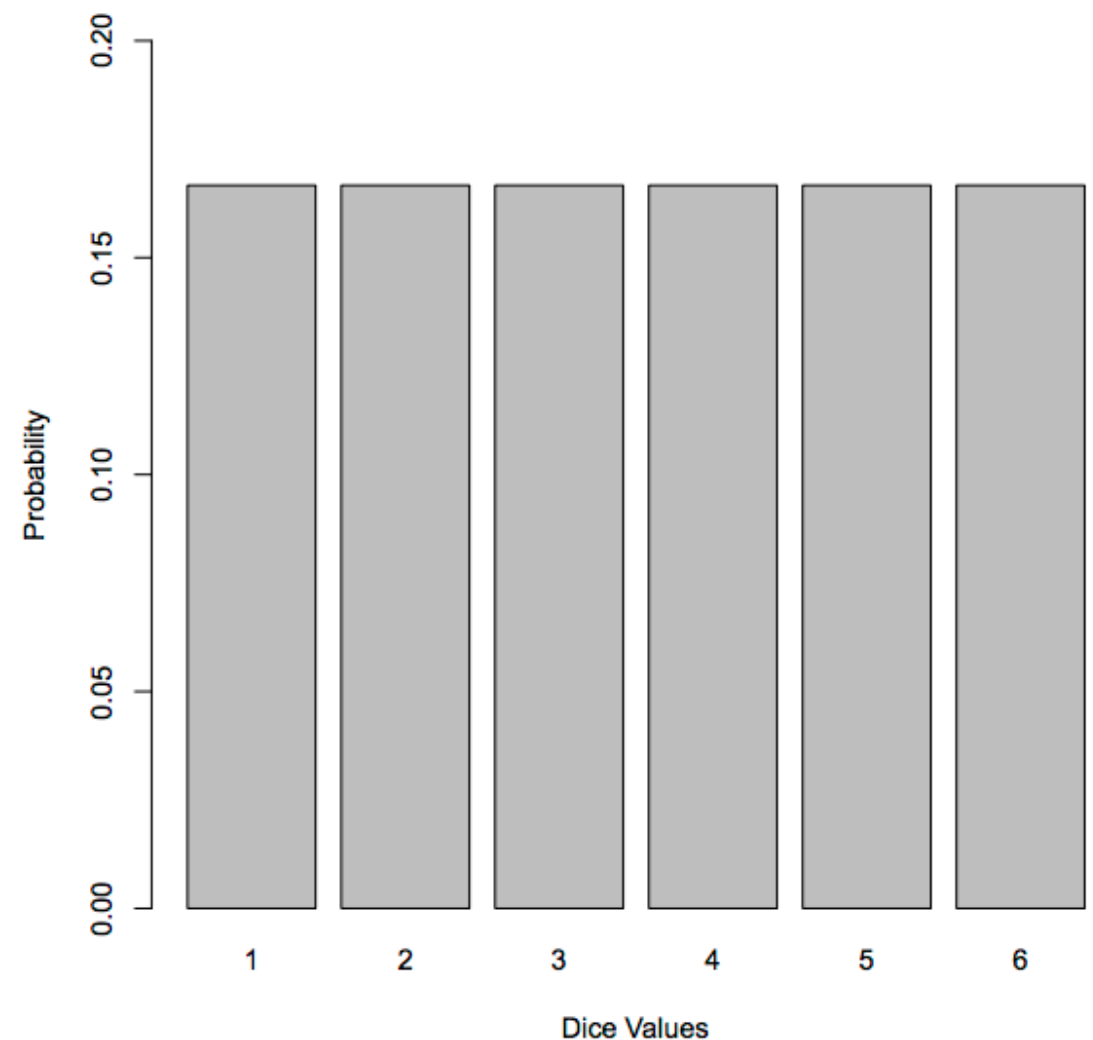
- Generating  $n$  test cases before finding bugs
- Randomised algorithm, so  $n$  can be represented as random variable
- Can  $n$  be normally distributed?
- NEVER:  $n < 0$  is impossible, and discrete value (ie, no fraction of tests, e.g. 1.3)

# $t$ -test (continued.)

- The Central Limit Theorem (CLT) states that the **sum** of  $n$  random variables converges to a *normal* distribution as  $n$  increases
- $t$ -test internally uses sums of random variables
- Often, already with  $n=30$  the  $t$ -test is *robust* to deviation from normality

# CLT Example

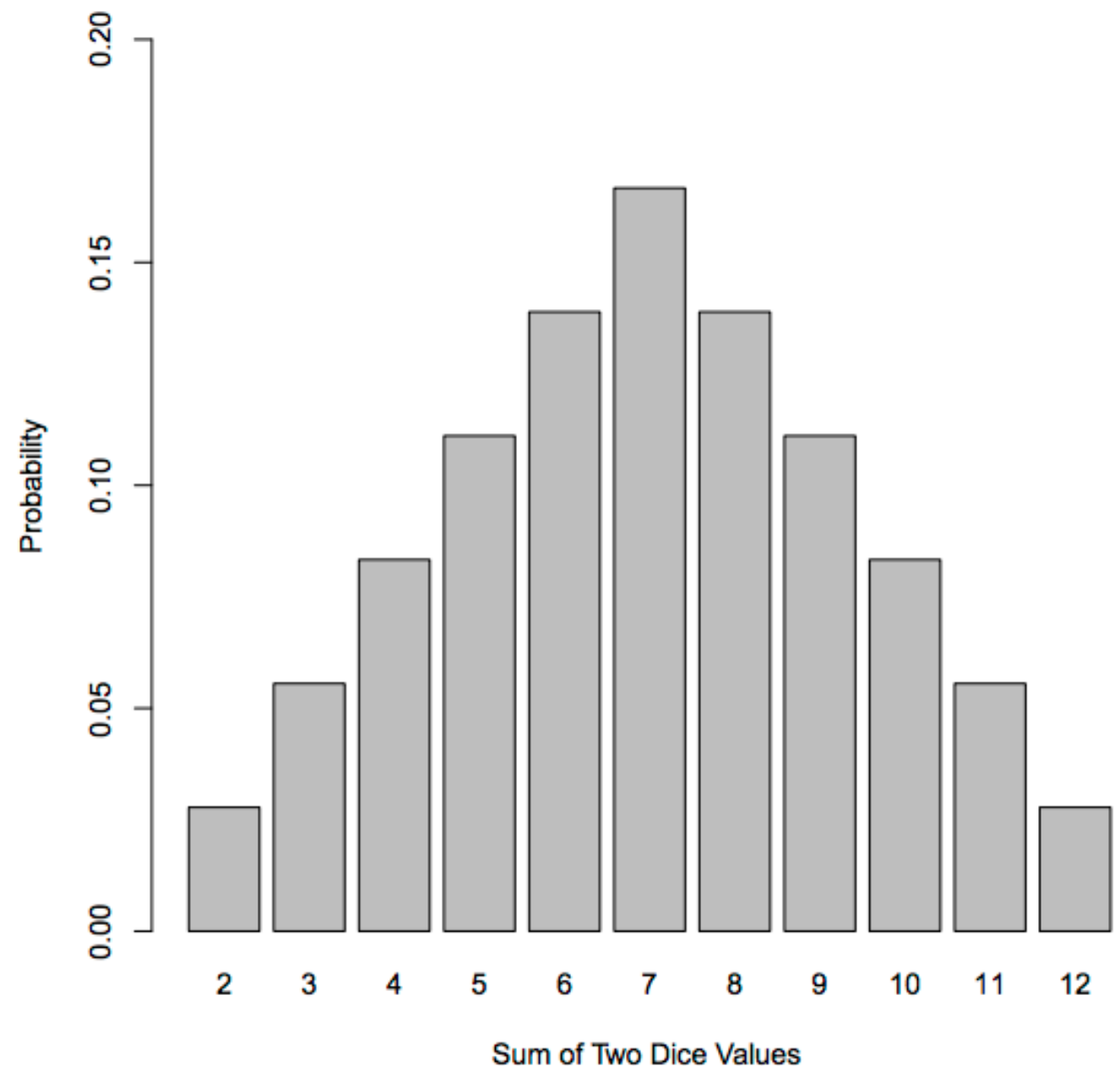
- Rolling a dice
- What is probability distribution of a 6 face dice?
- Each value has 16% probability
- Definitely **not** normal





# CLT (continued.)

- But what is the probability of the **sum** of  $n$  dices?
- Even with as low as  $n=2$ , it starts to look normal...



# A More Relevant Example

- Unit test generation (*EvoSuite*) for code coverage
- Sample 100 projects at *random* from *SourceForge* (the SF100 corpus)
  - yep, that is from before GitHub...
- Coverage [0%,100%] for single classes
- Average/mean coverage on the 100 projects
  - recall **sum** of  $n$  variables divided/scaled by  $n$

# A Large Scale Evaluation of Automated Unit Test Generation Using EvoSuite

Gordon Fraser, Department of Computer Science, University of Sheffield,

Regent Court, 211 Portobello

S1 4DP, Sheffield, UK

Gordon.Fraser@sheffield.ac.uk

Andrea Arcuri, Certus Software V&V Center at Simula Research Laboratory,

P.O. Box 134, Lysaker, Norway

arcuri@simula.no

Research on software testing produces many innovative automated techniques, but because software testing is by necessity incomplete and approximate, any new technique faces the challenge of an empirical assessment. In the past, we have demonstrated scientific advance in automated unit test generation with the EVOSUITE tool by evaluating it on manually selected open source projects or examples that represent a particular problem addressed by the underlying technique. However, demonstrating scientific advance is not necessarily the same as demonstrating practical value: Even if EVOSUITE worked well on the software projects we selected for evaluation, it might not scale up to the complexity of real systems. Ideally, one would use large “real-world” software systems to minimize the threats to external validity when evaluating research tools. However, neither choosing such software systems nor applying research prototypes to them are trivial tasks.

In this paper we present the results of a large experiment in unit test generation using the EVOSUITE tool on 100 randomly chosen open source projects, the 10 most popular open source projects according to the SourceForge website, 7 industrial projects, and 11 automatically generated software projects. The study confirms that EVOSUITE can achieve good levels of branch coverage (on average 71% per class) in practice. However, the study also exemplifies how the choice of software systems for an empirical study can influence the results of the experiments, which can serve to inform researchers to make more conscious choices in the selection of software system subjects. Furthermore, our experiments demonstrate how practical limitations interfere with scientific advances: Branch coverage on an unbiased sample is affected by predominant environmental dependencies. The surprisingly large effect of such practical engineering problems in unit testing will hopefully lead to a larger appreciation of work in this area, thus supporting transfer of knowledge from software testing research to practice.

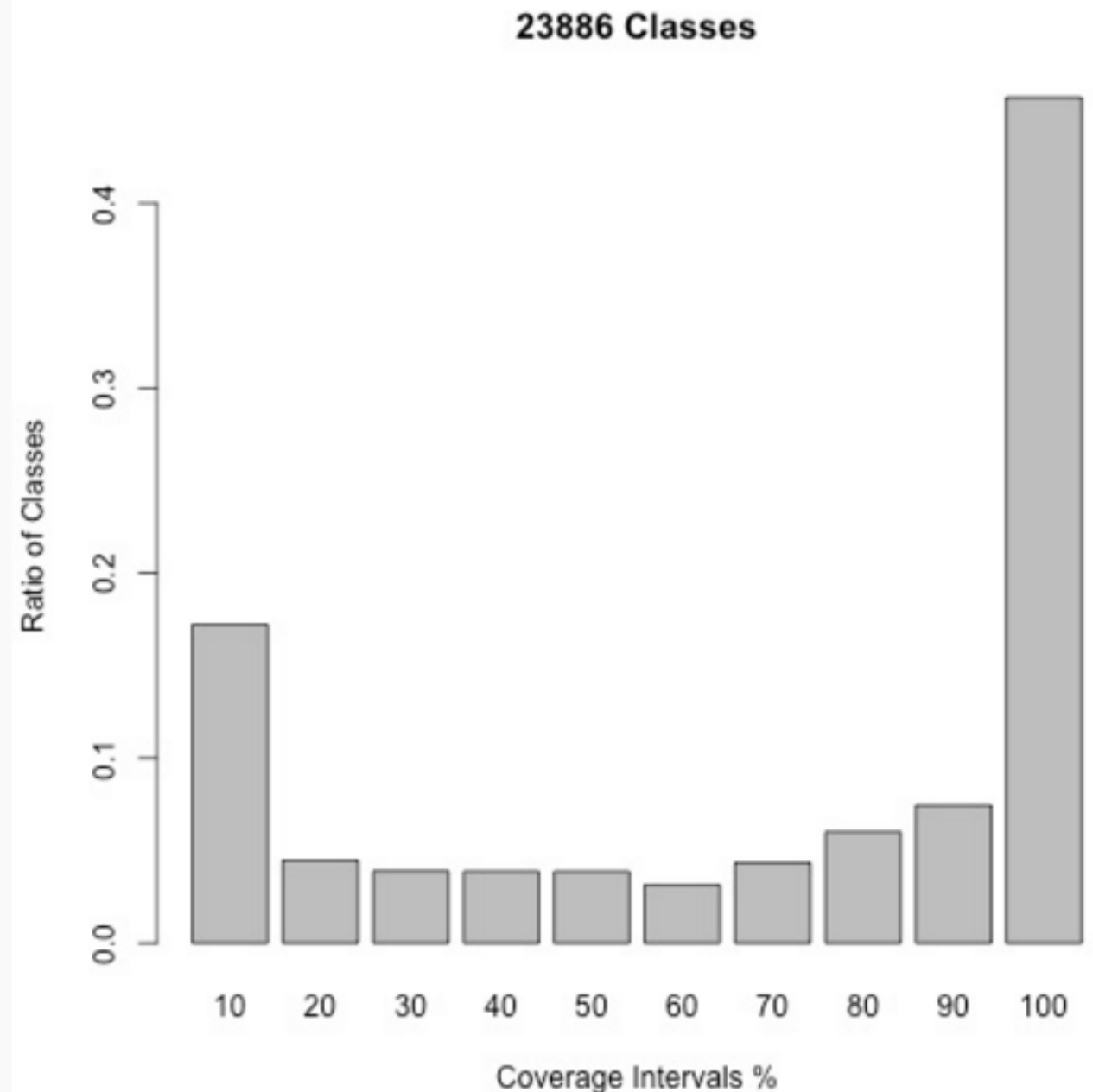
Categories and Subject Descriptors: D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms: Experimentation, Reliability

Additional Key Words and Phrases: Unit testing, automated test generation, branch coverage, empirical software engineering, JUnit, Java, benchmark

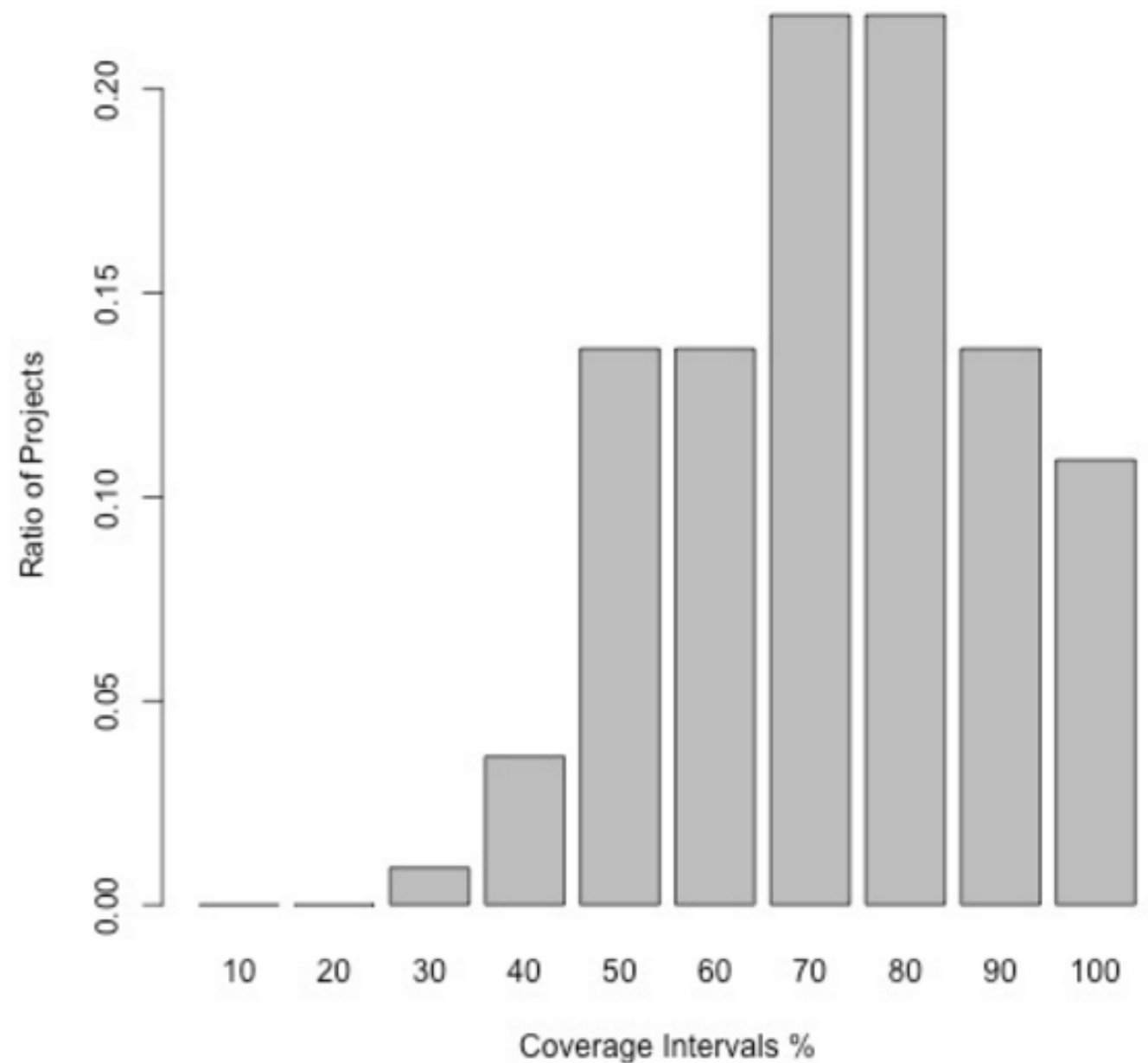
# Probability Distribution

- Single classes
- Many (40%) are trivial,  $\text{cov} > 90\%$
- Some (20%) are too difficult,  $\text{cov} < 10\%$
- Does not look normally distributed...



# (continued.)

- Average on each of the projects
- Start to *resemble* a normal distribution
- Interestingly, same distribution when looking at the used industrial systems



# What about checking normality?

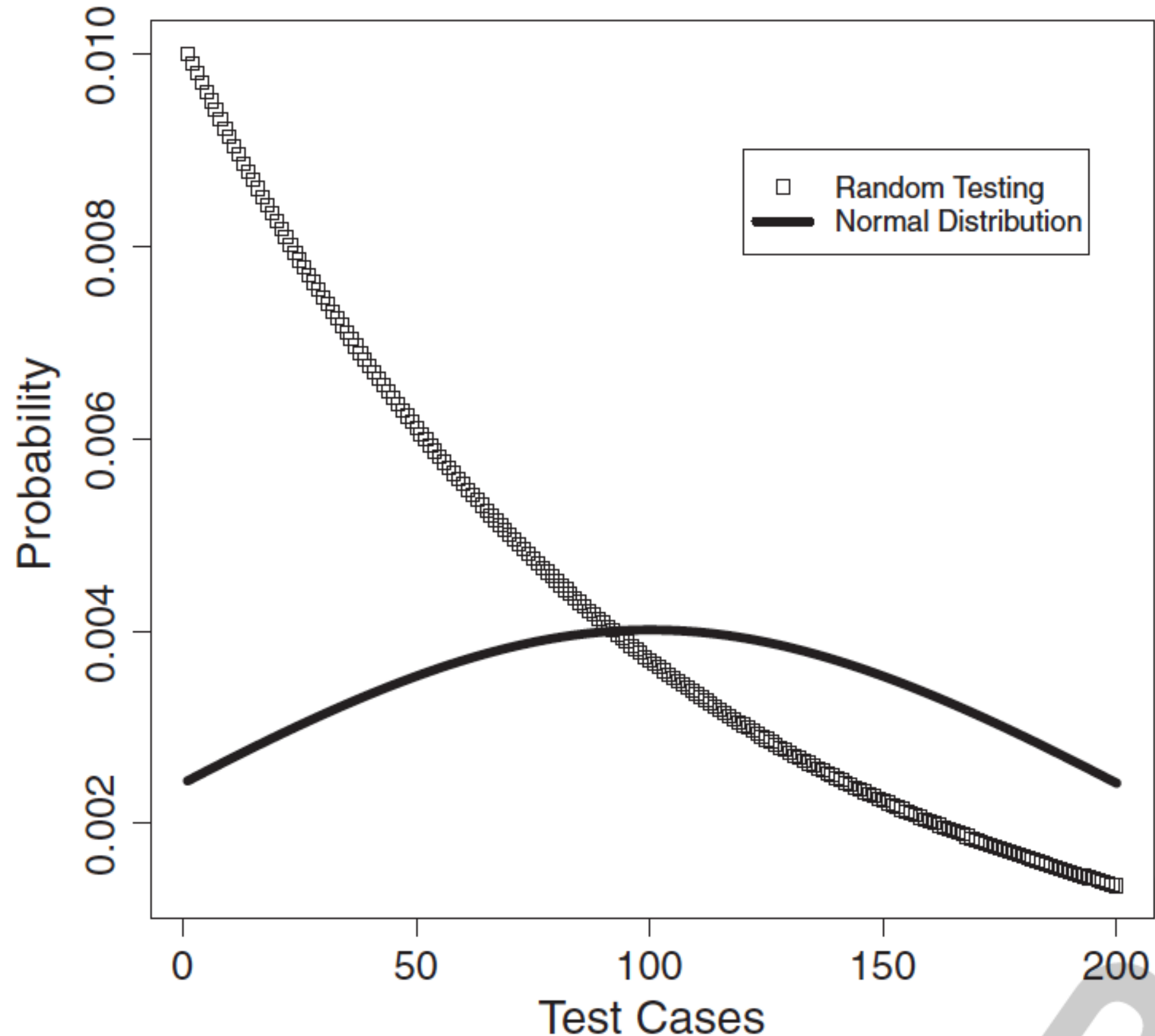
- Check normality before using  $t$ -test
- Shapiro-Wilk test
  - hypothesis normality,  $p$ -value to check if can reject it
- Few problems with this approach:
  - increase error, as doing 2 tests now
  - small  $n$ : **everything** looks normal
  - some algorithms have known non-normal distributions

# Random Testing

- Often used as a baseline when evaluating new techniques
- Easy to implement, in some cases can give good results
- Eg, sample  $n$  test cases until trigger a bug
- $n$  follows a *geometric* distribution, not a *normal* one



Example of geometric (RT) and normal distribution with same mean and standard deviation (failure rate 0.01)





# Non-Parametric Tests

- Do not make assumptions on distributions
- “Safer” to use
- Downsize: usually (not always) less powerful, i.e. more difficult to detect statistical difference

# Wilcoxon-Mann-Whitney U-test

- Check if two distributions **A** and **B** have same *stochastic order*
- I.e., when sampling:  $P(A > B) = P(B > A)$
- Look at relative order of sampled elements, and not their absolute values
- Robust to outliers

# Example

A	1	2	3	4	5	6	49
B	7	8	9	10	11	12	13

- All values in **B** are greater than **A**, but for 49
- Assuming larger values are better, are **A** and **B** equivalent? Or is one better?
- What would statistical tests say here?

```
comparisonTandU <- function(){
```

```
  a = c(1,2,3,4,5,6,49)
```

```
  b = c(7,8,9,10,11,12,13)
```

```
  cat("Mean a: ", mean(a), "\n")
```

```
  cat("Mean b: ", mean(b), "\n")
```

```
  print(wilcox.test(a,b))
```

```
  print(t.test(a,b))
```

```
}
```

Strong difference  
for U-test

```
> comparisonTandU()
```

```
Mean a: 10
```

```
Mean b: 10
```

Wilcoxon rank sum test

```
data: a and b
```

```
W = 7, p-value = 0.02622
```

```
alternative hypothesis: true location shift is not equal to 0
```

Welch Two Sample t-test

```
data: a and b
```

```
t = 0, df = 6.1875, p-value = 1
```

```
alternative hypothesis: true difference in means is not equal to 0
```

```
95 percent confidence interval:
```

```
-15.98999 15.98999
```

```
sample estimates:
```

```
mean of x mean of y
```

```
10
```

```
10
```

Same mean 10,  
so no difference  
for t-test

# Which One To Use?

- If few data (ie low  $n$ ):
  - $t$ -test might not be robust to deviation from normality
  - U-test might be not enough powerful
- If lot of data
  - $t$ -test fine even if data is not normal (Central Limit Theorem)
  - U-test becomes enough powerful

# (continued.)

- If not enough data
  - need to study details of statistics, and choose the right test
  - eg, empirical study with 20-30ish students
- If lot of data
  - does not matter so much what you use (eg  $t$ -test vs. U-test)
  - eg, test data generation on cluster of computers

# So If In Doubts...

- ... just try to increase the amount of data!
  - run experiments for longer
  - use cluster of computers
- Not always possible:
  - human subjects
  - industrial case studies

# In a Nutshell...

- Main purpose of statistics is to handle limited amount of data, when is expensive to acquire
  - eg, medical trials for new drugs
- When can run large experiments, statistics becomes less important



# Side-effects

With a lot of data...

... every difference, even if tiny, becomes *statistically significant*

eg, very easy to get *p-value*  $< 0.05$

# Wrong Approach

- Run experiments
- Compare **A** with **B**
- Use statistics
- $p\text{-value} < 0.05$  (or any other threshold)
- Claim contribution because results are *statistically significant*

# A Better Approach

- Run experiments on **A** and **B**
- Is the difference of *practical significance*?
  - this is problem dependent, e.g. enough branch coverage improvement
- If **yes**: do statistical tests to check if you had enough data
- If **not**: who cares of *p-values*...

# How to Quantify Practical Relevance?

- Problem dependent
  - improvement in branch coverage
- Often, given a measure  $K$ , compare *average* of  $K(A)$  with  $K(B)$  over all instances
  - eg, 67% vs 72% coverage

# Issues With Averages

- $diff = mean(K(A)) - mean(K(B))$
- Would ignore variability in the data
  - eg if high standard deviation
- Improvement from 1% to 3% is not the same as from 91% to 93%, although still  $diff=2\%$
- Issue if different order of measures, e.g.
  - *easy* problems: e.g. A=10 vs B=20 test cases
  - *difficult* problems: e.g. A=200,000 vs B=100,000
- Skewed results if outliers

# Standardised Effect Sizes

- Independent from unit of measure
  - robust if high variability in the instances of the case study
- Single measure, which takes into account also variability
- Easier to compare among studies

# Main Effect Sizes

- Odds Ratio
  - for dichotomous results (eg success/failure)
- Cohen  $d$ 
  - most known, but “makes sense” only if data is normal, so quite seldom... plus, difficult to interpret
- Vargha and Delaney  $A_{12}$  measure
  - non-parametric, very easy to interpret
  - personally, the one I use 99% of the times...

# Odds Ratio

*“measure of how many times greater the odds are that a member of a certain population will fall into a certain category than the odds are that a member of another population will fall into that category”*

$$\psi = \frac{a + \rho}{n + \rho - a} / \frac{b + \rho}{n + \rho - b}$$




# (continued)

- If same success rate, odds ratio = 1
- Automatically computed with Fisher test

```
> compareAB_n100()
```

Fisher's Exact Test for Count Data

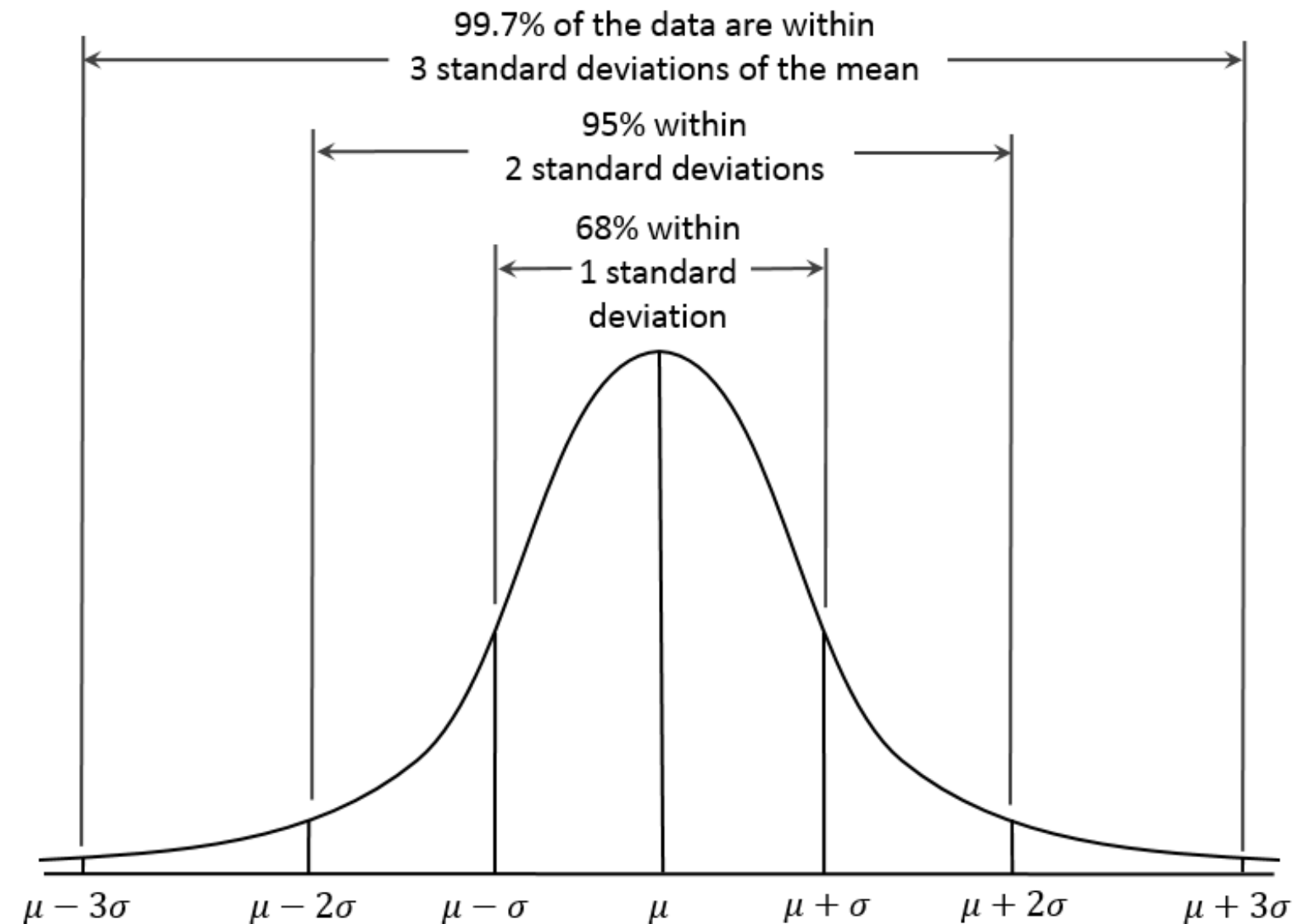
```
data:  m
p-value = 0.005937
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 1.256203 4.351520
sample estimates:
odds ratio
 2.323215
```



# Cohen $d$

- Difference in mean divided by pooled standard deviation  $(\mu^A - \mu^B)/\sigma$
- Larger mean difference: greater  $d$
- Smaller standard deviation: greater  $d$
- If no difference,  $d=0$

# (continued.)



standard deviation  
(SD) has special  
characteristics in  
normal distribution

but in other  
distributions, dividing  
by SD can be  
meaningless

# A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong



András Vargha

Department of Experimental Psychology, ELTE, Hungary

Harold D. Delaney

Department of Psychology, UNM, U.S.A

## Abstract

---

*McGraw and Wong (1992) described an appealing index of effect size, called CL, which measures the difference between two populations in terms of the probability that a score sampled at random from the first population will be greater than a score sampled at random from the second. McGraw and Wong introduced this "common language effect size statistic" for normal distributions and then proposed an approximate estimation for any continuous distribution. In addition, they generalized CL to the n-group case, the correlated samples case, and the discrete values case.*

*In the current paper a different generalization of CL, called the A measure of stochastic superiority, is proposed, which may be directly applied for any discrete or continuous variable that is at least ordinally scaled. Exact methods for point and interval estimation as well as the significance tests of the  $A = .5$  hypothesis are provided. New generalizations of CL are provided for the multi-group and correlated samples cases.*

# Vargha-Delaney A Measure

- Non-parametric
- $A_{12}$  defines probability  $[0,1]$  that running algorithm (1) yields higher values than running another algorithm (2).
- If the two algorithms are equivalent, then  $A_{12}=0.5$
- Eg,  $A_{12}=0.7$  means 70% probability that (1) gives better results

# (continued.)

$$\hat{A}_{12} = (R_1 / m - (m + 1) / 2) / n$$

- $R_1$  is the rank sum of the values of (1)
  - eg,  $A=\{42,11,7\}$  and  $B=\{1,20,5\}$ , then  $R_1=\text{sum}(6,4,3)=13$
- $m=|A|$ , and  $n=|B|$ , usually  $m=n$
- Limitation:  $A_{12}$  tells you how often better results, but not by how much

# R Code For $A_{12}$ Measure

```
measureA <- function(a,b){  
  
  r = rank(c(a,b))  
  r1 = sum(r[seq_along(a)])  
  
  m = length(a)  
  n = length(b)  
  A = (r1/m - (m+1)/2)/n  
  
  return(A)  
}
```

# On $t$ -test vs. U-test

- Both are fine for statistical tests
- Can interpret U-test with  $A_{12}$
- If data not normal,  $t$ -test is still fine (Central Limit Theorem), but NOT the Cohen  $d$
- Having  $A_{12}$  is a good reason to choose U-test over  $t$ -test
- Still can use both (they measure different things)



# For Large Experiments...

- *p-values* get smaller and smaller
- NOT the standardised effect sizes
  - they just get more precise
  - you'll not see much difference between  $n=100$  and  $n=1,000,000$

# The Wrath of Bonferroni

- When running several tests, increase Type I error
  - ie high probability of wrongly rejecting at least one null hypothesis
- Bonferroni adjustment: reduce threshold for statistical significance based on number  $k$  of tests
  - eg  $0.05/k$
- I NEVER use it (long story)... but, if reviewers ask for it...

# Forum

## A farewell to Bonferroni: the problems of low statistical power and publication bias

Shinichi Nakagawa

Department of Animal and Plant Sciences, University of Sheffield,  
Sheffield S10 2TN, United Kingdom

Recently, Jennions and Møller (2003) carried out a meta-analysis on statistical power in the field of behavioral ecology and animal behavior, reviewing 10 leading journals including *Behavioral Ecology*. Their results showed dismayingly low average statistical power (note that a meta-analytic review of statistical power is different from post hoc power analysis as criticized in Hoenig and Heisey 2001). The statistical power

associated with the standard Bonferroni procedure is a substantial reduction in the statistical power of rejecting an incorrect  $H_0$  in each test (e.g., Holm, 1979; Perneger, 1998; Rice, 1989). The sequential Bonferroni procedure also incurs reduction in power, but to a lesser extent (which is the reason that the sequential procedure is used in preference by some researchers; Moran, 2003). Thus, both procedures exacerbate the existing problem of low power, identified by Jennions and Møller (2003).

For example, suppose an experiment where both an experimental group and a control group consist of 30 subjects. After an experimental period, we measure five different variables and conduct a series of  $t$  tests on each variable. Even prior to applying Bonferroni corrections, the statistical power of each test to detect a medium effect is 61% ( $\alpha = .05$ ), which is less than a recommended acceptable 80%

# What's wrong with Bonferroni adjustments

*BMJ* 1998 ; 316 doi: <http://dx.doi.org/10.1136/bmj.316.7139.1236> (Published 18 April 1998)

Cite this as: *BMJ* 1998;316:1236

[Article](#)

[Related content](#)

[Metrics](#)

[Responses](#)

Thomas V Perneger ([perneger@cmu.unige.ch](mailto:perneger@cmu.unige.ch)), medical epidemiologist

[Author affiliations](#) ▼

Correspondence to: Dr Perneger

**Accepted** 16 January 1998

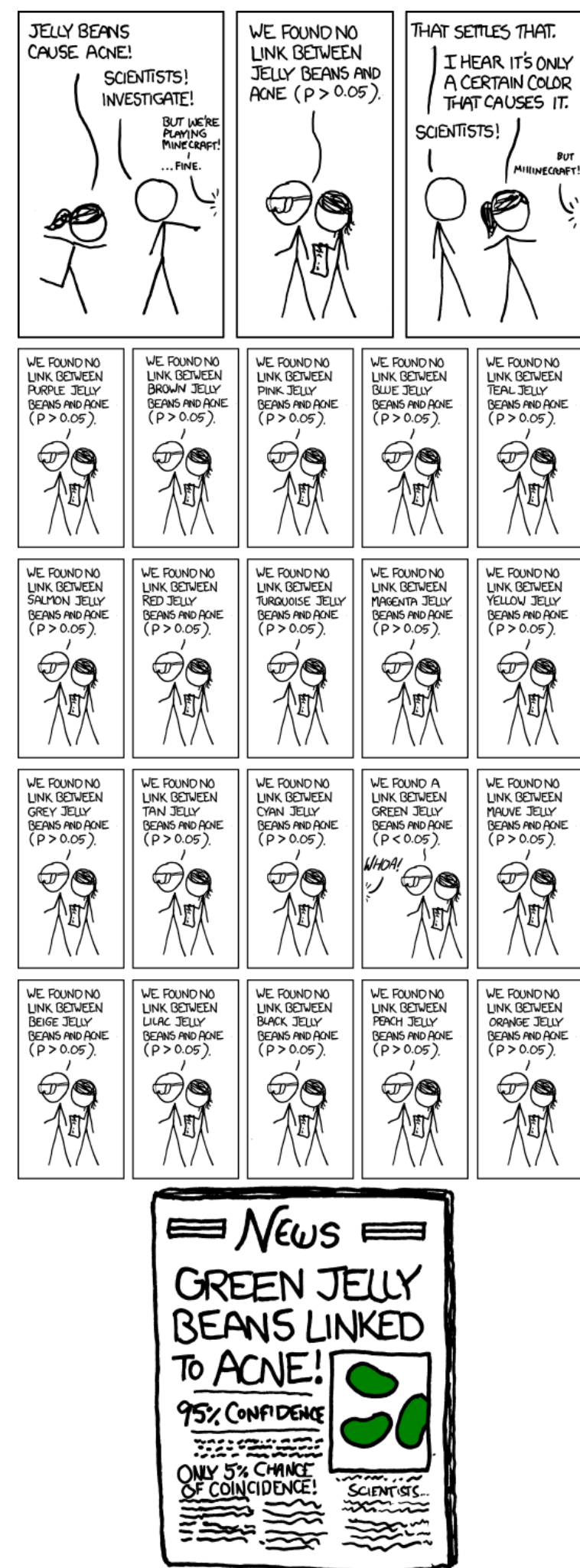
When more than one statistical test is performed in analysing the data from a clinical study, some statisticians and journal editors demand that a more stringent criterion be used for “statistical significance” than the conventional  $P < 0.05$ .<sup>1</sup> Many well meaning researchers, eager for methodological rigour, comply without fully grasping what is at stake. Recently, adjustments for multiple tests (or Bonferroni adjustments) have found their way into introductory texts on medical statistics, which has increased their apparent legitimacy. This paper advances the view, widely held by epidemiologists, that Bonferroni adjustments are, at best, unnecessary and, at worst, deleterious to sound statistical inference.

## Summary points

Adjusting statistical significance for the number of tests that have been performed on study data—the Bonferroni method—creates more problems than it solves

# Beware...

- When running many separated experiments, can get *low p-value by chance*...
- In these cases, single instances should not matter, but rather how often they happen
  - need to look at whole picture
- <https://xkcd.com/882/>



# Multiple Experiments

- Collection of **N** instances
  - eg, 20 open-source projects
- If randomised algorithm, repeat each experiment **K** times on each instance
- *Very typical scenario*
- How to choose **N** vs **K**?
- What statistics to use?

# Controversial Topic...

- You might read a lot of different opinions...
  - eg, on the use of Bonferroni
- Try to have big **N**, but at least **K=10** (better 30)
- On each of **N** instance, apply basic test (eg U-test)
- Count and report number of instances with statistical difference (eg, at arbitrary 0.05)
  - both positive and negative (ie improvements vs worse results)

# More Positive or Negative Cases?

- Calculate effect size  $A_{12}$  on each of the  $N$  instances
- Report average/median effect sizes
- Statistics on whether the  $N$  values of  $A_{12}$  are symmetric around 0.5
- `wilcox.test(c(a1,a2,a3,...,an), mu=0.5)`



# Bayesian Data Analysis

- **I am no expert on this...**
- Every time I speak about statistics, I often get asked about it
- *Bayesian* statistics currently promoted as a “better alternative” to *frequentist* statistics (eg *p-values*)

# Bayesian Data Analysis in Empirical Software Engineering Research

Carlo A. Furia<sup></sup>, Robert Feldt<sup></sup>, and Richard Torkar

**Abstract**—Statistics comes in two main flavors: frequentist and Bayesian. For historical and technical reasons, frequentist statistics have traditionally dominated empirical data analysis, and certainly remain prevalent in empirical software engineering. This situation is unfortunate because frequentist statistics suffer from a number of shortcomings—such as lack of flexibility and results that are unintuitive and hard to interpret—that curtail their effectiveness when dealing with the heterogeneous data that is increasingly available for empirical analysis of software engineering practice. In this paper, we pinpoint these shortcomings, and present Bayesian data analysis techniques that provide tangible benefits—as they can provide clearer results that are simultaneously robust and nuanced. After a short, high-level introduction to the basic tools of Bayesian statistics, we present the reanalysis of two empirical studies on the effectiveness of automatically generated tests and the performance of programming languages. By contrasting the original frequentist analyses with our new Bayesian analyses, we demonstrate the concrete advantages of the latter. To conclude we advocate a more prominent role for Bayesian statistical techniques in empirical software engineering research and practice.

**Index Terms**—Bayesian data analysis, statistical analysis, statistical hypothesis testing, empirical software engineering



# My Personal Opinion...

- I do not particularly like it...
- Trading a set of problems (eg  $P(D|H_0) \neq P(H_0|D)$ ) with another set of problems (choice of *prior distributions*)
- Harder to use / setup
- Might have good applications, but not so sure about randomized algorithms
  - I would need to see better evidence on this

# How To Visualise The Data and Results of the Statistical Tests?

Table 1: Branch coverage comparison of EvoSuite with (JEE) and without (Base) support for JEE, on the 25 classes with the largest increase. Note, some classes have the same name, but they are from different packages.

Class	Base	JEE	$\hat{A}_{12}$	$p$ -value
ManagedComponent	14.3%	41.2%	<b>0.96</b>	$\leq 0.001$
UnManagedComponent	47.0%	51.6%	<b>0.80</b>	$\leq 0.001$
ItemBean	89.7%	100.0%	<b>1.00</b>	$\leq 0.001$
HATimerService	57.1%	93.3%	<b>1.00</b>	$\leq 0.001$
SchedulerBean	60.0%	97.3%	<b>0.98</b>	$\leq 0.001$
IntermediateEJB	33.3%	66.7%	<b>1.00</b>	$\leq 0.001$
SecuredEJB	80.0%	98.7%	<b>0.97</b>	$\leq 0.001$
AsynchronousClient	20.0%	29.3%	<b>0.97</b>	$\leq 0.001$
RemoteEJBClient	25.0%	58.3%	<b>1.00</b>	$\leq 0.001$
TimeoutExample	60.0%	99.3%	<b>1.00</b>	$\leq 0.001$
GreetController	66.7%	100.0%	<b>1.00</b>	$\leq 0.001$
HelloWorldJMSClient	4.9%	23.1%	<b>1.00</b>	$\leq 0.001$
MemberResourceRESTService	19.1%	69.2%	<b>1.00</b>	$\leq 0.001$
MemberResourceRESTService	19.3%	67.7%	<b>0.96</b>	$\leq 0.001$
MemberRegistrationServlet	18.2%	87.1%	<b>1.00</b>	$\leq 0.001$
HelloWorldMDBServletClient	26.0%	61.3%	<b>0.99</b>	$\leq 0.001$
TaskDaoImpl	55.6%	77.8%	<b>1.00</b>	$\leq 0.001$
AuthController	30.0%	100.0%	<b>1.00</b>	$\leq 0.001$
TaskController	42.9%	100.0%	<b>1.00</b>	$\leq 0.001$
TaskDaoImpl	55.6%	75.0%	<b>0.96</b>	$\leq 0.001$
TaskListBean	75.0%	96.7%	<b>0.93</b>	$\leq 0.001$
TaskDaoImpl	55.6%	77.8%	<b>1.00</b>	$\leq 0.001$
TaskResource	55.4%	84.3%	<b>1.00</b>	$\leq 0.001$
Servlet	40.0%	60.9%	<b>0.92</b>	$\leq 0.001$
XAService	44.7%	49.3%	<b>0.96</b>	$\leq 0.001$
Average	43.8%	74.6%	0.98	

# Tables

- Good way to show statistical tests
  - a row for each problem instance (if not too many)
  - a column for effect size (in bold if  $p\text{-value} < 0.05$ )
  - a column for *p-value*
  - a final row with average values
- Of course, many more options (eg, graphs, boxplots, etc.)

# Raise Your Hand If...

- you have ever filled a table by hand, e.g. in Microsoft Word, and ...
- data is invalid, get new data, refill table by hand...
- ... again and again...
- (extra points if at 4am of a conference deadline night)

# DO NOT DO IT

- Unless it is trivial small (eg 2x2 table)
- Very error prone (especially at 4am)
- If you see colleague/student doing it, *hit* them
  - or use any other form of punishment legal in your country
  - they will be grateful, one day...
- Invest time in learning **Latex** and **R**
  - or any other equivalent combination



# From Scripts to PDF

- Given some data  $X$  on file (eg CSV)
- Use scripts (R/Python) to load it
- Create “textual” representation of table, and save it to a file  $K$
- Automatically import  $K$  in your paper document (eg if written in Latex)
- Generate PDF from your paper document

```
\begin{table}[t!]
\centering
\caption{\label{table:selection}
Branch coverage comparison of \code{evo} with (JEE) and without (Base)
support for JEE, on the 25 classes with the largest increase.
Note, some classes have the same name, but they are from different packages.
}
\scalebox{0.85}{
\input{generated_files/tableSelection.tex}
}
\end{table}
```

To get a better picture of the importance of handling JEE features, Table~\ref{table:selection} shows detailed data on the 25 challenging classes where JEE handling had most effect: On these classes, the

$\begin{tabular}{|l@{\hspace{0.5cm}}r@{\hspace{0.5cm}}r@{\hspace{0.5cm}}r@{\hspace{0.5cm}}r}\toprule$

Class & Base & JEE &  $\hat{A}_{12}$  &  $\$p\$-value$   $\backslash\backslash$

$\midrule$

ManagedComponent & 14.3\% & 41.2\% &  $\{bf\ 0.96\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

UnManagedComponent & 47.0\% & 51.6\% &  $\{bf\ 0.80\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

ItemBean & 89.7\% & 100.0\% &  $\{bf\ 1.00\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

HATimerService & 57.1\% & 93.3\% &  $\{bf\ 1.00\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

SchedulerBean & 60.0\% & 97.3\% &  $\{bf\ 0.98\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

IntermediateEJB & 33.3\% & 66.7\% &  $\{bf\ 1.00\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

SecuredEJB & 80.0\% & 98.7\% &  $\{bf\ 0.97\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

AsynchronousClient & 20.0\% & 29.3\% &  $\{bf\ 0.97\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

RemoteEJBClient & 25.0\% & 58.3\% &  $\{bf\ 1.00\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

TimeoutExample & 60.0\% & 99.3\% &  $\{bf\ 1.00\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

GreetController & 66.7\% & 100.0\% &  $\{bf\ 1.00\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

HelloWorldJMSClient & 4.9\% & 23.1\% &  $\{bf\ 1.00\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

MemberResourceRESTService & 19.1\% & 69.2\% &  $\{bf\ 1.00\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

MemberResourceRESTService & 19.3\% & 67.7\% &  $\{bf\ 0.96\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

MemberRegistrationServlet & 18.2\% & 87.1\% &  $\{bf\ 1.00\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

HelloWorldMDBServletClient & 26.0\% & 61.3\% &  $\{bf\ 0.99\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

TaskDaoImpl & 55.6\% & 77.8\% &  $\{bf\ 1.00\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

AuthController & 30.0\% & 100.0\% &  $\{bf\ 1.00\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

TaskController & 42.9\% & 100.0\% &  $\{bf\ 1.00\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

TaskDaoImpl & 55.6\% & 75.0\% &  $\{bf\ 0.96\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

TaskListBean & 75.0\% & 96.7\% &  $\{bf\ 0.93\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

TaskDaoImpl & 55.6\% & 77.8\% &  $\{bf\ 1.00\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

TaskResource & 55.4\% & 84.3\% &  $\{bf\ 1.00\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

Servlet & 40.0\% & 60.9\% &  $\{bf\ 0.92\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

XAService & 44.7\% & 49.3\% &  $\{bf\ 0.96\}$  &  $\{bf\ \$\le\ 0.001\}$   $\backslash\backslash$

$\midrule$

Average & 43.8\% & 74.6\% & 0.98 &  $\backslash\backslash$

$\bottomrule$

$\end{tabular}$

```

bestSelectionTable <- function(){
  dt <- read.table(gzfile(SELECTION_ZIP_FILE),header=T)
  TABLE = paste(GENERATED_FILES,"/tableSelection.tex",sep="")
  unlink(TABLE)
  sink(TABLE, append=TRUE, split=TRUE)
  cat("\\begin{tabular}{l @{}\\hspace{0.5cm}\\r@{}\\hspace{0.5cm}\\r@{}\\hspace{0.5cm}\\r @{}\\hspace{0.5cm}\\r }\\toprule","\\n")
  cat("Class & Base & JEE & $\\hat{A}_{12}$ & $p$-value \\\\", "n")
  cat("\\midrule","\\n")
  Bv = c(); Fv = c(); Av = c()

  selection = getBestClassesSelection(dt,25)
  projects = sort(unique(dt$group_id))
  for(proj in projects){
    classes = unique(dt$TARGET_CLASS[dt$group_id==proj])
    classes = sort(classes[areInTheSubset(classes,selection)])
    for(cl in classes){
      mask = dt$group_id==proj & dt$TARGET_CLASS==cl
      base = dt$BranchCoverage[mask & dt$configuration_id=="Base"]
      pafm = dt$BranchCoverage[mask & dt$configuration_id=="JEE"]
      a12 = measureA(pafm,base)
      w = wilcox.test(base,pafm,exact=FALSE,paired=FALSE)
      pv = w$p.value
      if(is.nan(pv)) {pv = 1}
      stat = pv < 0.05
      if(pv < 0.001){ pv = "\\le 0.001"
      } else { pv = formatC(pv,digits=3,format="f")}
      cat(getClassName(cl))
      cat(" & "); cat(formatC(100*mean(base),digits=1,format="f"),"\\%",sep="")
      cat(" & "); cat(formatC(100*mean(pafm),digits=1,format="f"),"\\%",sep="")
      a12Formatted = formatC(a12,digits=2,format="f")
      cat(" & ")
      if(!stat) {
        cat(a12Formatted, " & ")
        cat("$",pv,"$",sep="")
      } else {
        cat("\\bf ",a12Formatted," & ",sep="")
        cat("\\bf $",pv,"$",sep="")
      }
      cat("\\\\ \\\n")
      Bv = c(mean(base),Bv)
      Fv = c(mean(pafm),Fv)
      Av = c(a12,Av)
    }
  }
  cat("\\midrule","\\n")
  avgB = paste(formatC(100*mean(Bv),digits=1,format="f"),"\\%",sep="")
  avgF = paste(formatC(100*mean(Fv),digits=1,format="f"),"\\%",sep="")
  avgA = formatC(mean(Av),digits=2,format="f")
  cat("Average & ", avgB, " & ", avgF, " & ", avgA, " & \\\n")
  cat("\\bottomrule","\\n")
  cat("\\end{tabular}","\\n")
  sink()
}

```

# analyze.R

actual statistics is  
just couple of lines

When generating  
tables/graphs in R/Latex,  
adding statistics is trivial!

Data change? No problem:

```
$> bestSelectionTable()
```

```
$> pdflatex paper.tex
```

Table 1: Branch coverage comparison of EvoSuite with (JEE) and without (Base) support for JEE, on the 25 classes with the largest increase. Note, some classes have the same name, but they are from different packages.

Class	Base	JEE	$\hat{A}_{12}$	$p$ -value
ManagedComponent	14.3%	41.2%	<b>0.96</b>	$\leq 0.001$
UnManagedComponent	47.0%	51.6%	<b>0.80</b>	$\leq 0.001$
ItemBean	89.7%	100.0%	<b>1.00</b>	$\leq 0.001$
HATimerService	57.1%	93.3%	<b>1.00</b>	$\leq 0.001$
SchedulerBean	60.0%	97.3%	<b>0.98</b>	$\leq 0.001$
IntermediateEJB	33.3%	66.7%	<b>1.00</b>	$\leq 0.001$
SecuredEJB	80.0%	98.7%	<b>0.97</b>	$\leq 0.001$
AsynchronousClient	20.0%	29.3%	<b>0.97</b>	$\leq 0.001$
RemoteEJBClient	25.0%	58.3%	<b>1.00</b>	$\leq 0.001$
TimeoutExample	60.0%	99.3%	<b>1.00</b>	$\leq 0.001$
GreetController	66.7%	100.0%	<b>1.00</b>	$\leq 0.001$
HelloWorldJMSClient	4.9%	23.1%	<b>1.00</b>	$\leq 0.001$
MemberResourceRESTService	19.1%	69.2%	<b>1.00</b>	$\leq 0.001$
MemberResourceRESTService	19.3%	67.7%	<b>0.96</b>	$\leq 0.001$
MemberRegistrationServlet	18.2%	87.1%	<b>1.00</b>	$\leq 0.001$
HelloWorldMDBServletClient	26.0%	61.3%	<b>0.99</b>	$\leq 0.001$
TaskDaoImpl	55.6%	77.8%	<b>1.00</b>	$\leq 0.001$
AuthController	30.0%	100.0%	<b>1.00</b>	$\leq 0.001$
TaskController	42.9%	100.0%	<b>1.00</b>	$\leq 0.001$
TaskDaoImpl	55.6%	75.0%	<b>0.96</b>	$\leq 0.001$
TaskListBean	75.0%	96.7%	<b>0.93</b>	$\leq 0.001$
TaskDaoImpl	55.6%	77.8%	<b>1.00</b>	$\leq 0.001$
TaskResource	55.4%	84.3%	<b>1.00</b>	$\leq 0.001$
Servlet	40.0%	60.9%	<b>0.92</b>	$\leq 0.001$
XAService	44.7%	49.3%	<b>0.96</b>	$\leq 0.001$
Average	43.8%	74.6%	0.98	

implementation of all the techniques presented in this paper as open-source (GPL license), and we made it available on a public repository<sup>11</sup>.

Threats to *construct* validity come from what measure we chose to evaluate the success of our techniques. We used branch coverage, which is a common coverage criterion in the software testing literature. However, it is hard to automatically quantify the negative effects of tests that do not handle dependency injection, as the presence of false positive tests on software maintenance is a little investigated topic in the literature.

Threats to *external* validity come from how well the results generalize to other case studies. To have a variegated set of classes showing different features of JEE, we chose the JEE examples used to demonstrate the JBoss EAP / WildFly application servers, which consist of 247 Java classes. Larger case studies on industrial systems will be needed to further generalize our results.

<sup>11</sup>[www.github.com/EvoSuite/evosuite](http://www.github.com/EvoSuite/evosuite)

# ROI of learning Latex/R

- Steep learning curve
- Will save you lot of time, year after year
- Most analyses in the papers are similar
  - i.e. copy&paste R scripts from paper to paper
- If you are doing a PhD in Software Engineering, writing 50-100 lines of R should not be a big problem...

# Conclusion

- Just a brief, high level introduction
- Many more things to cover
- Few take away lessons...



# (1) Statistics Is Not Mumbo Jumbo

- If used in all fields of science and engineering for more than 100 years, there must be a reason...
- ISSTA'13 review: "...1217 is simply a statistically estimated (not real) lower bound based on inspection of only 20 classes! The authors should just report exactly what they confirmed on the sampled 20 cases *rather than using statistically estimated numbers.*"
- ... not a big deal, you just submit to EMSE...

# (2) The more data, the better

- If in doubt...
  - $t$ -test or U-test?
  - Is it wrong to skip Bonferroni?
  - Do I really have to learn all that math stuff?
- Well, the more data, the less important the statistical tests become
- If can't have experiments with 1000 students (or industrial systems), maybe can on 1000 open-source projects

# (2.1) Corollarium

- Learn how to use clusters of computers for experiments
- Many universities do have them
- Usually just need a 10-30 lines of Bash script...

# (3) Main Statistics To Use

- Fisher Exact test with odds ratio
- U-test with  $A_{12}$  effect sizes
- There are more, but those are good starting points
- *Remember 0.05 threshold is arbitrary...*

# (4) Automation

- Automate the generation of tables/graphs
- Learn R/Python and Latex
- When generation is automated, adding statistics takes just a couple of minutes

---

Research Article

## **A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering<sup>†</sup>**

Andrea Arcuri<sup>1,\*</sup> and Lionel Briand<sup>2</sup>

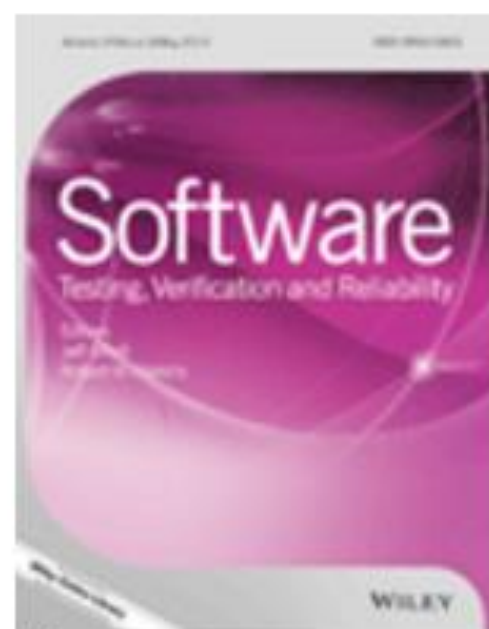
Version of Record online: 6 NOV 2012

DOI: 10.1002/stvr.1486

Copyright © 2012 John Wiley & Sons, Ltd.

---

Issue



**Software Testing, Verification  
and Reliability**

**Volume 24, Issue 3, pages  
219–250, May 2014**