Quantitative methods in Software Engineering

Georgios Gousios // @gousiosg
TU Delft

Georgios Gousios

• Assistant professor at TU Delft, WIS group
• Doing research on
  • Software analytics
  • Distributed software development
  • Software testing
• Hiring!

Tutorial goals

• Understand some types of research in SE
• By the end, you should be able to
  • Understand the stats methods used in most papers
  • Apply quantitative methods on SE data
  • Mine GitHub

Why quantitative software engineering?

• Transform traces into working knowledge
• Identify and optimise patterns and trends
• Quantify and improve processes and products

Applications of QSE

• Pattern mining and recommendation
• “You changed this file, perhaps you might also want to change that one as well”
• “This bug report is duplicate from…”
• Autocompletion in Eclipse
• Process improvement
• Test case runtime optimisation
• Work prioritization
• Evolution analysis
• Software quality analysis

Empirical Methods in SE
Empirical methods 101

- Experiment
- Case study — Most SE studies
- Field study
- Ethnography

Can be both Quantitative or Qualitative

Quantitative research

Extract answers to RQs using mathematical, statistical or numerical techniques

- Generation of theories and hypotheses
- Development of instruments for measurement
- Collection of empirical data
- Modelling and analysis of data ← “Data Science”

Hypotheses

Propose an explanation for a phenomenon

Defined in pairs

- $H_1$: Default hypothesis
- $H_0$: Null hypotheses

A good hypothesis is readily falsifiable

Hypotheses — p values

- Statistical tests (usually) return the probability ($p$) that $H_0$ is true
- To accept $H_1$, we set a threshold (usually, $p < 0.05$)
- “A p-value, or statistical significance, does not measure the size of an effect or the importance of a result.”

Measurement

Extract samples of data from a running process. Data types:

- **Continuous**: Can calculate a degree of difference between measurements
- **Categorical**: One of a predefined set (sorting not important)
- **Ordinal**: One of a predefined set (sorting is important)

R in one slide!

- An interactive stats and ML environment
- `data.frame`: R’s 2-dim table
  - Rows are measurements
  - Columns are variables (with attached data type)
Basic measurements
Mean, median, quantiles, max and min

<table>
<thead>
<tr>
<th></th>
<th>mpg</th>
<th>cyl</th>
<th>disp</th>
<th>hp</th>
<th>drat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>10.40</td>
<td>4.000</td>
<td>71.1</td>
<td>52.0</td>
<td>2.760</td>
</tr>
<tr>
<td>1st Qu.</td>
<td>15.43</td>
<td>4.000</td>
<td>120.8</td>
<td>96.5</td>
<td>3.080</td>
</tr>
<tr>
<td>Median</td>
<td>19.20</td>
<td>6.000</td>
<td>230.7</td>
<td>123.0</td>
<td>3.695</td>
</tr>
<tr>
<td>Mean</td>
<td>20.09</td>
<td>6.188</td>
<td>230.7</td>
<td>146.7</td>
<td>3.597</td>
</tr>
<tr>
<td>3rd Qu.</td>
<td>22.80</td>
<td>8.000</td>
<td>326.0</td>
<td>180.0</td>
<td>3.920</td>
</tr>
<tr>
<td>Max.</td>
<td>33.90</td>
<td>8.000</td>
<td>472.0</td>
<td>335.0</td>
<td>4.930</td>
</tr>
</tbody>
</table>

Visualisation - Histogram
Probability distribution of 1 variable

> hist(mtcars$mpg)

Visualisation - Scatter plot
Values from 2 variables on a 2D plot

> plot(mtcars$mpg, mtcars$qsec)

Visualisation - Boxplot
Summary of 1 variable, possibly grouped by another

> boxplot(mpg ~ cyl, mtcars)

Visualisation - Barplot
Frequencies of 1 group of data

> counts <- table(mtcars$gear)
> barplot(counts, main="Car Distribution", xlab="Number of Gears", ylab="Number of cars")

Visualisation - Barplot
Frequencies of >1 group of data

> counts <- table(mtcars$vs, mtcars$gear)
> barplot(counts, xlab="Number of Gears", col=c("darkblue","red"), legend = rownames(counts), beside=TRUE)
Distributions — Normal
Identified by the characteristic “bell curve” histogram

Distributions — Others
Histograms are left or right skewed

Basic statistical tests
- Normality: Shapiro-Wilks
- Correlation: Pearson for normally distributed data, Spearman/Kendal for non-normally distributed
  - When data is categorical: $\chi^2$
  - Differences between groups: Wilcoxon
  - Always to be reported with an effect size metric, e.g. Cliff’s $\delta$

ML: Two slide definition
Given a set of observations (tuples) $A = \{a_1…a_n\}$ and a set of outcomes $B = \{b_1…b_n\}$

Ways of learning:
- Supervised: Approximate a function $F: A \rightarrow B$
- Unsupervised: Find patterns in $A$
- Reinforced: $A$ and $B$ get updated as we learn $F$, so we need to re-learn $F$

Types of ML
- Classification: When $B$ is a categorical variable (binary or other)
- Regression: When $B$ is a continuous variable
- Clustering: Find groups in $A$ according to some measure of similarity
- Association rules: Find items in $A$ that change together frequently

Notable developments in QSE
A brief history of QSE

- late 70s — Metrics: McCabe complexity, Halstead software science, LoCs
- mid 80s — Measuring projects: Function Points, COCOMO
- mid 00s — OSS repositories: Mining VCS, BTS, MLs
- mid 10s — software analytics and big data

McCabe complexity

- First work to define complexity by counting number of branches.
- Triggered the metrics craziness
- Still used today in most models for imperative languages.

Applying metrics to quantify a software property

“All three metrics (Halstead volume, McCabe complexity, LoCs) correlated with both the accuracy of the modification and the time to completion.”

Applying metrics to solve the “software crisis”

Defined the COCOMO model

- Linked effort to team performance ($e = a \cdot KLOC^b$)
- Basic estimator is size
- Book full of case studies (!)

Measuring software quality

- Defect density
- Customer satisfaction
- Backlog management
- Bug removal effectiveness
- Availability
- $\sigma$
- ...(more metrics)

ISO 9126

- Reliability
- Usability
- Maintainability
- Portability
- Functionality
- Interoperability
- Accuracy
- Integrity
- Security
- Compliance
- Interoperability
- Quality
- ...
Empirical Software Engineering

First work that treated software engineering as a formal empirical field

OO hits the shelves, new metrics wanted!

6 metrics tuned to OO design

Measure unique aspects of encapsulation and nestedness

Found to be able to predict defects in OO projects, at the design phase

Used to guide refactoring efforts

Too many metrics!

Goal Question Metric to the rescue

- Define measurement Goals
- Ask improvement Questions that can be quantified
- Use or invent Metrics to answer those

An undergoing revolution

The GNU project (1983), the Linux kernel (1991) etc changed the way software is written

- In the open
- Massively distributed
- All information accessible online

OSS made QSE bloom

By the late 90s, researchers had access to

- Source code
- Version control system traces
- Bug tracking databases
- Mailing list communications
- Forums, IRC chats
  en masse and for free!

Both the software itself and the OSS phenomenon needed investigation

The primordial OSS study

Not the first to use OSS data, but

- Pioneered the case study as a research instrument in QSE
- Used robust statistical methods
- Combined traces from multiple OSS data sources (VCS/BTS)
How does OSS work?

Defined the, now obvious, terms
- joining script
- contribution barrier
- coordination and awareness


How does global software development work?

“One key finding is that distributed work items appear to take about two and one-half times as long to complete as similar items where all the work is colocated”

Also, established the basis of the socio-technical congruence team model


Data science goes mainstream in SE

Used unsupervised learning (association rules) to answer the question:

“Programmers who changed these functions also changed….”

Kickstarted MSR as a research area


Metrics put to real use

“Using principal component analysis on the code metrics, we built regression models that accurately predict the likelihood of post-release defects for new entities.”

Nachiappan Nagappan, Thomas Ball, and Andreas Zeller. Mining metrics to predict component failures. ICSE ’06.

Metrics put to real use

“In this paper […] we identify a number of requirements to be fulfilled by a maintainability model to be usable in practice. We sketch a new maintainability model that alleviates most of these problems, and we discuss our experiences with using such as system for IT management consultancy activities.”

Social network analysis

All data analysis methods are possible. Participation to mailing lists:

- strongly correlates with code change activity
- moderately correlates with document change activity

Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. Mining email social networks. MSR ’06

Process metrics can predict failures

One of the first works to prove the connection of process metrics to faults, thereby providing evidence towards validating the old conjecture that process quality leads to product quality.

Ahmed E. Hassan. Predicting faults using the complexity of code changes. ICSE ’10

Empirical SE @ CACM!

“We studied the post-release failures for the Windows Vista code base and concluded that distributed development has little to no effect.”

Christian Bird, Nachiappan Nagappan, Premkumar Devanbu, Harald Gall, and Brendan Murphy. Does distributed development affect software quality?: an empirical case study of Windows Vista. CACM 52, 8 (August 2009), 85-93

No single model will rule them all

Established that software projects are different and therefore models need to be localised and specialised.

Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. 2009. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. FSE 09

Naturalness

“We show that code is very repetitive, and in fact even more so than natural languages. As an example use of the model, we have developed a simple code completion engine for Java that, despite its simplicity, already improves Eclipse’s completion capability.”


OSS v2.0

GitHub started in 2008 and by 2009 most of Ruby development was happening on it. By 2010, it hosted >1M repos. By 2011, more than >1M users. It revolutionised OSS by:

- centralising it, making it uniform
- enabling anyone to contribute

The AppStore

Centralised application distribution allows everybody to examine applications en masse and apply static analysis, dynamic analysis and ML techniques to analyse millions of shipped apps.

Big data

TBs of QSE data at the hands of any researcher.

Big security

“After clustering Android apps by their description topics, we identify outliers in each cluster with respect to their API usage.”

Big testing

“THEO is based on a cost model, which dynamically skips tests when the expected cost of running the test exceeds the expected cost of removing it. We replayed past development periods of three major Microsoft products resulting in a reduction of 50% of test executions, saving millions of dollars per year, while maintaining product quality.”

Deep learning

Use CNNs to automatically give names to methods based on their contents

Deep learning

Derive a natural language summary to a piece of code
What’s next?

• Treat code as data: deep learn all the things
• Make code composition intelligent
• Ecosystems
• DevOps — Fast (software) analytics
• Study new programming languages + paradigms
• Bring science into production, exploit uniformity, exploit intelligence in static analysis tools

My POV

• All the easy problems have been solved
• Significant advances, but not much was put in practice
• Lots of overlapping (good!) and opportunistic (hmmm…) research
• Most of the code needed has been written
• The future:
  • Deep learning FTW!
  • Feedback-driven SE