Seminar "Beiträge zum Software Engineering"

On comparing web development platforms

Lutz Prechelt
Freie Universität Berlin, Institut für Informatik
http://www.inf.fu-berlin.de/inst/ag-se/

A class of research questions

An eternal research question in software engineering:

- How do alternative methods, processes, technologies, notations, tools etc. compare?
  - With respect to ease of use, productivity, error-proneness, quality of results, learnability etc.

- Applicable to (for instance):
  - design methods
  - design notations
  - development process models
  - testing methods
  - programming languages
  - middleware technologies
  - etc.

Example: Comparing programming languages

- For instance, assume we would like to compare a number of programming languages
- Questions:
  - How easily are they learned?
  - How productive are they? (time for solving a given problem)
  - How fast do the programs run?
  - How much memory do they need?
  - How many defects do they contain?
  - How reliably do they run?
  - How robust are they?
  - How safe are they? How secure?
  - How long are they?
  - How easy are they to understand, modify, test?
  - and many more

Research method of choice: Controlled experiment

In principle, the perfect research method for addressing these questions is a *controlled experiment*:

- Change one thing (here: The language used)
- Keep everything else the same (this is what is called *control*)
- Observe what happens in each case
  - i.e. write a program and investigate its properties
- Interpret the observations to answer your research questions

Research method problems: Generalizability

- No matter which task we choose for the program
  - it may be more suitable for language A than for B
  - but for a different program it could have been the other way round

- This means, we cannot guarantee that our results generalize to everything that can be done with the languages.
  - There is no solution to this problem
    - The best we could do is use several tasks, not just one.
    - And spread them over the spectrum of possible tasks.

- We should choose a task that can be considered either "typical" or "interesting"
  - Typical: Results describe frequent cases
  - Interesting: Results describe cases with high uncertainty

Research method problems: Criteria needed

- We need operationalizations for the individual attributes targeted by our questions
  - Repeatable procedures for obtaining an answer
- This is straightforward for some attributes, e.g.:
  - How fast do they run?
    → Choose an input, measure execution time in seconds
- It is very difficult for others, e.g.:
  - How many defects do they contain?
  - How safe are they? How secure?
  - How easy are they to understand, modify, test?

Research method problems: Control

- The worst methodological problem in our case, however, is control:

  How to keep "everything else constant"?

- We would need to keep the programmer constant!

- That is obviously impossible:
  - We cannot guarantee that somebody has equivalent knowledge of all languages
  - The programmer learns each time when solving the problem. We cannot erase or reset this experience.

- So we need a replacement for true constancy.

Keeping the programmer "constant"

- Obviously, we need to use a different programmer for each language.
  - How do we make them all the same?

Method 1: Grow them

- Find a large-enough set of monozygotic multiples
- Make sure they have all had the same youth and education
- Make sure they had no programming training yet
- Train each in a different language
  - Make sure each training has the same quality!
- Perform the experiment with them
  - Make sure they are all in identically good condition on that day!

This is practically impossible.

Keeping the programmer "constant" (2)

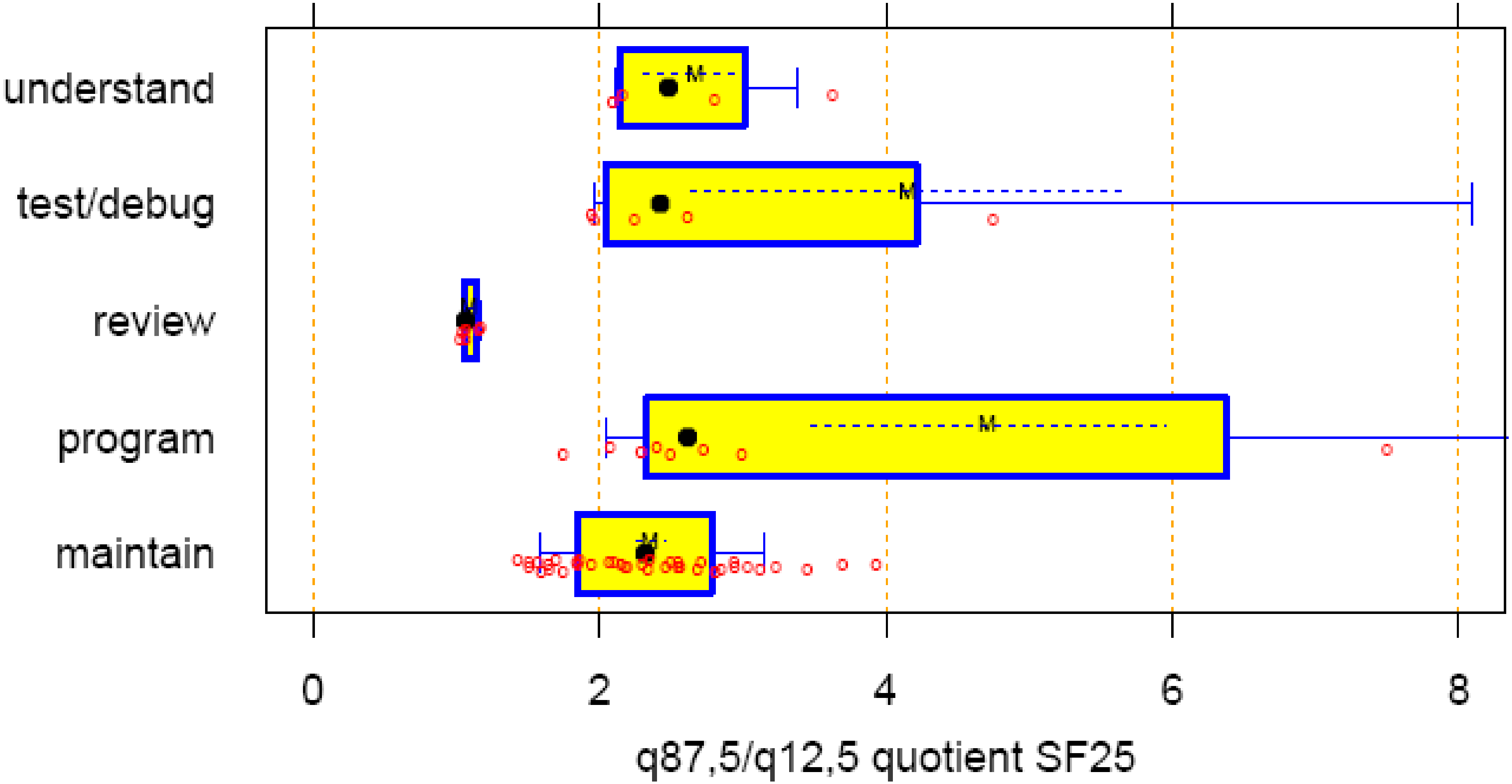
Method 2: Use averaging

- Find a large-enough set of roughly similar programmers for each language
- Make sure none of these groups is inherently inferior to another, programming-wise
- Have each of these persons solve the task
- When evaluating the results, use the average of the results of each language
    - And hope that all differences between the programmers cancel out
        - within each group
        - as well as across groups

This is possible.

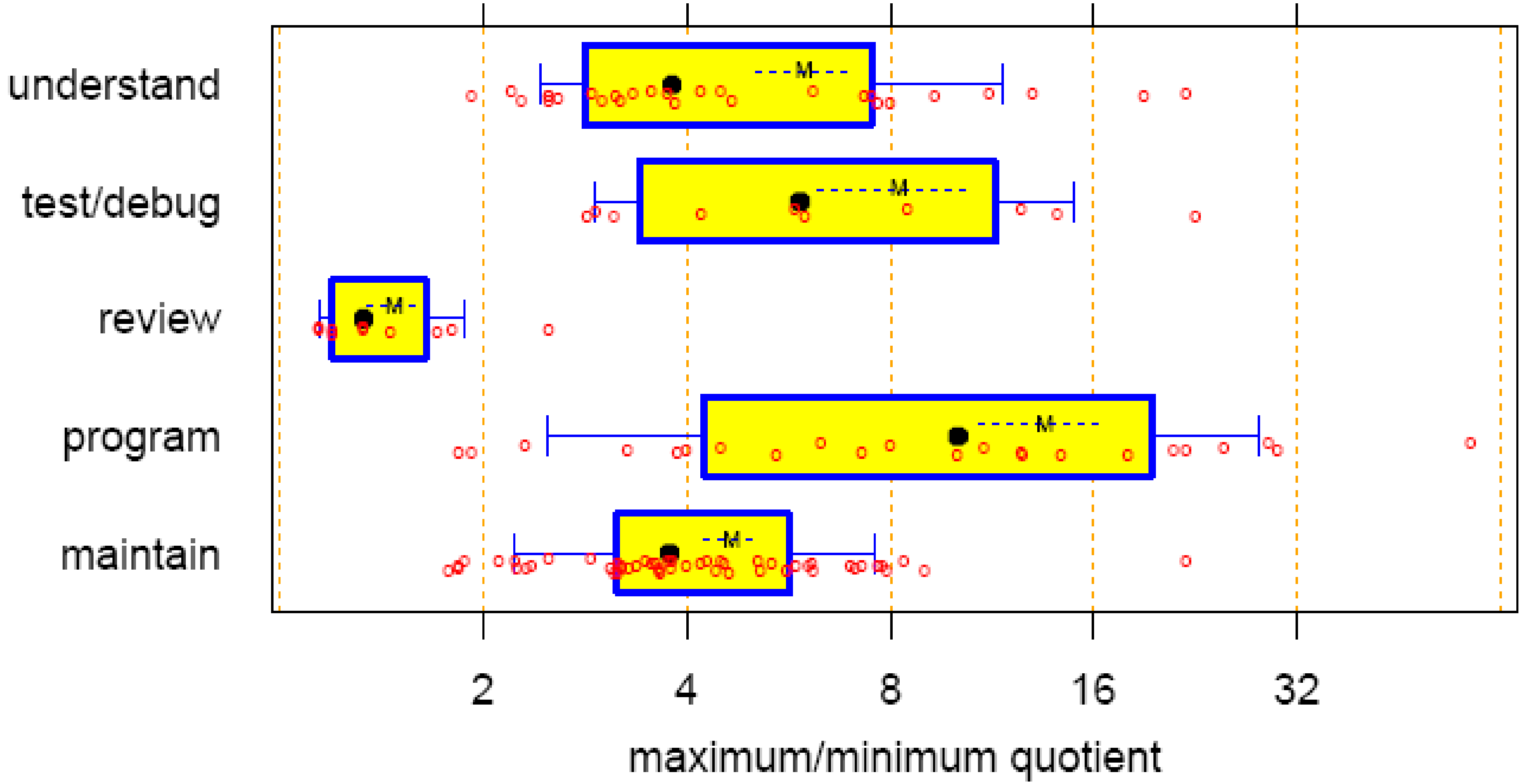
We still need to operationalize "roughly similar" and find such

Variation among "roughly similar" programmers



- Each point represents one group (by task type): The worktime quotient of slowest to fastest quarter of participants

Variation among "roughly similar" programmers (2)



Problem of averaging

- Compare these interpersonal differences to the differences you would expect between languages
- With interpersonal variation this large, will we be able to (clearly) see the language differences at all?

Conclusion:

- It is important to get either
  - fairly similar programmers (not just roughly similar ones)
    - to reduce the interpersonal variation
  - or very many of them
    - The precision of an estimate of the mean is stddev/sqrt(N)

Many or similar: When to use which

We will now shortly review two different studies:

- One used many roughly similar programmers to compare programming languages
- The other (currently in planning) attempts to use fewer, but more similar programmers to compare web development platforms

Example 1: jccpprt

- The study jccpprt compares Java, C, C++, Perl, Python, Rexx, and Tcl.
- It is based on a precise specification of a simple string processing program
- 80 implementations were collected from 74 programmers
  - Some in a controlled experiment on a different topic
  - Others via a call in Usenets newsgroups

language	progs	second	unusable	total
C	8	0	3	5
C++	14	0	3	11
Java	26	2	2	24
Perl	14	2	1	13
Python	13	1	0	13
Rexx	5	1	1	4
Tcl	11	0	1	10
Total	91	6	11	80

jccpprt task

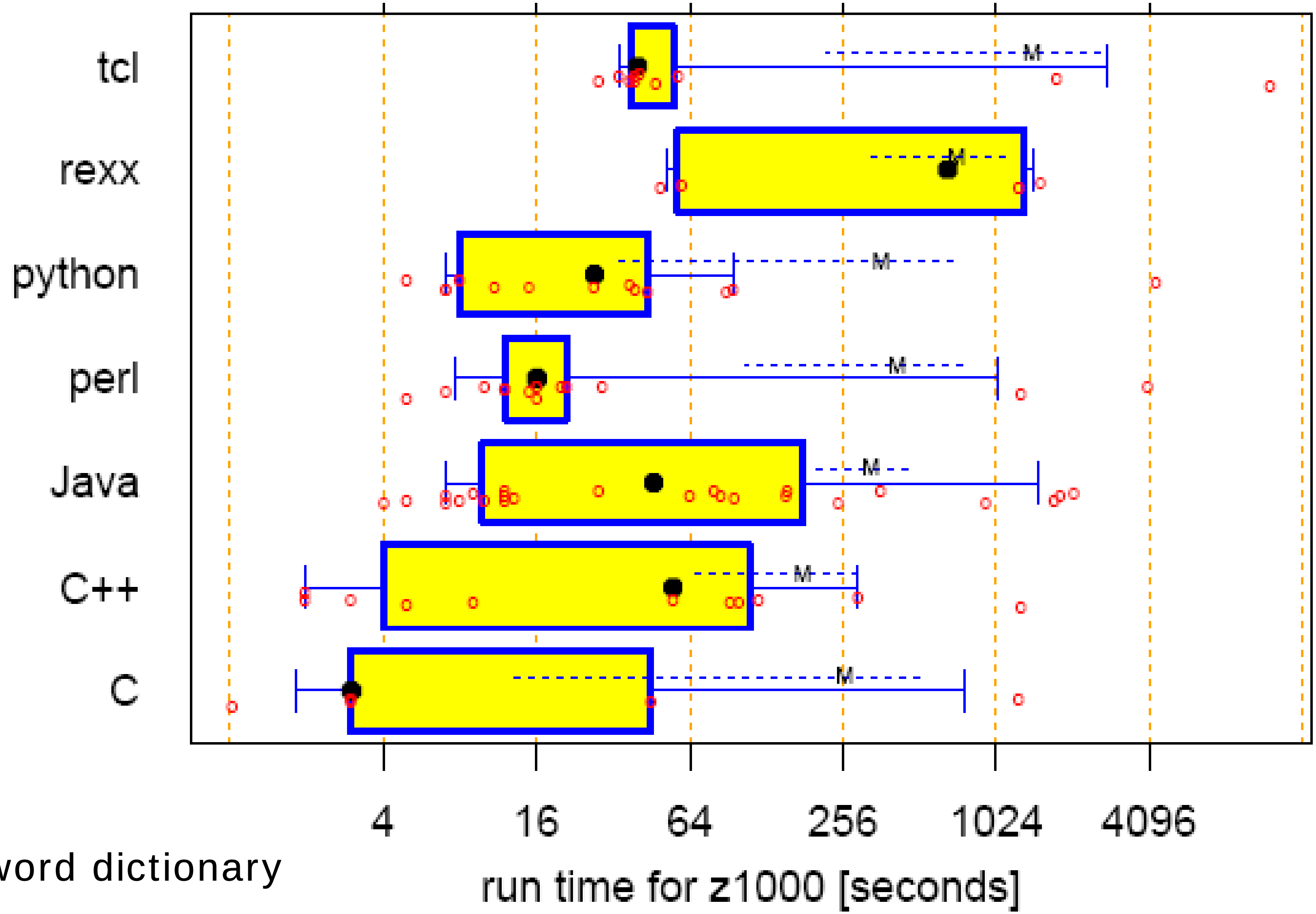
- A fixed mapping from letters to digits is given:

E	J N Q	R W X	D S Y	F T	A M	C I V	B K U	L O P	G H Z
e	j n q	r w x	d s y	f t	a m	c i v	b k u	l o p	g h z
0	1	2	3	4	5	6	7	8	9

- The program receives as input a "dictionary" (list of words) and a list of "phone numbers"
- and must produce as output all possible encodings of the phone numbers by words according to the mapping

```
5624-82: mir Tor
5624-82: Mix Tor
4824: Torf
4824: fort
4824: Tor 4
10/783--5: neu o"d 5
10/783--5: je bo"s 5
10/783--5: je Bo" da
381482: so 1 Tor
```

Some jccpprt results: Execution time for 1000 numbers



with a
73000-word dictionary

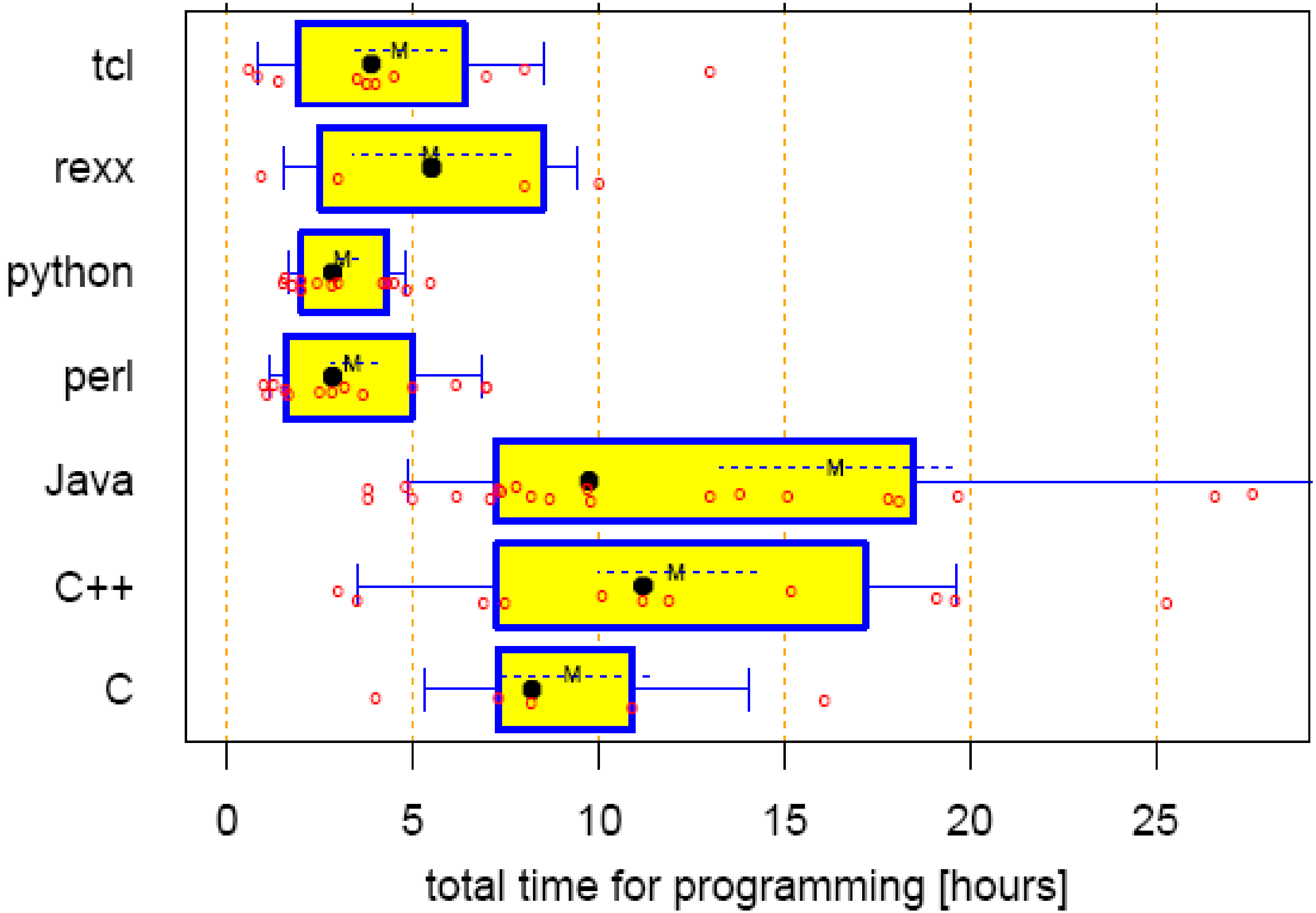
Some jccprt results: Observations

- The individual variation is **huge**
  - Note the scale is logarithmic!
  - Observe how wide the stderr dashes around the means (M) are!

- General trends are hence unclear

- But (looking at the medians), we find that
  scripting language programs are <u>not</u> generally much slower than programs in compiled languages

Some jccpprt results: Work time required



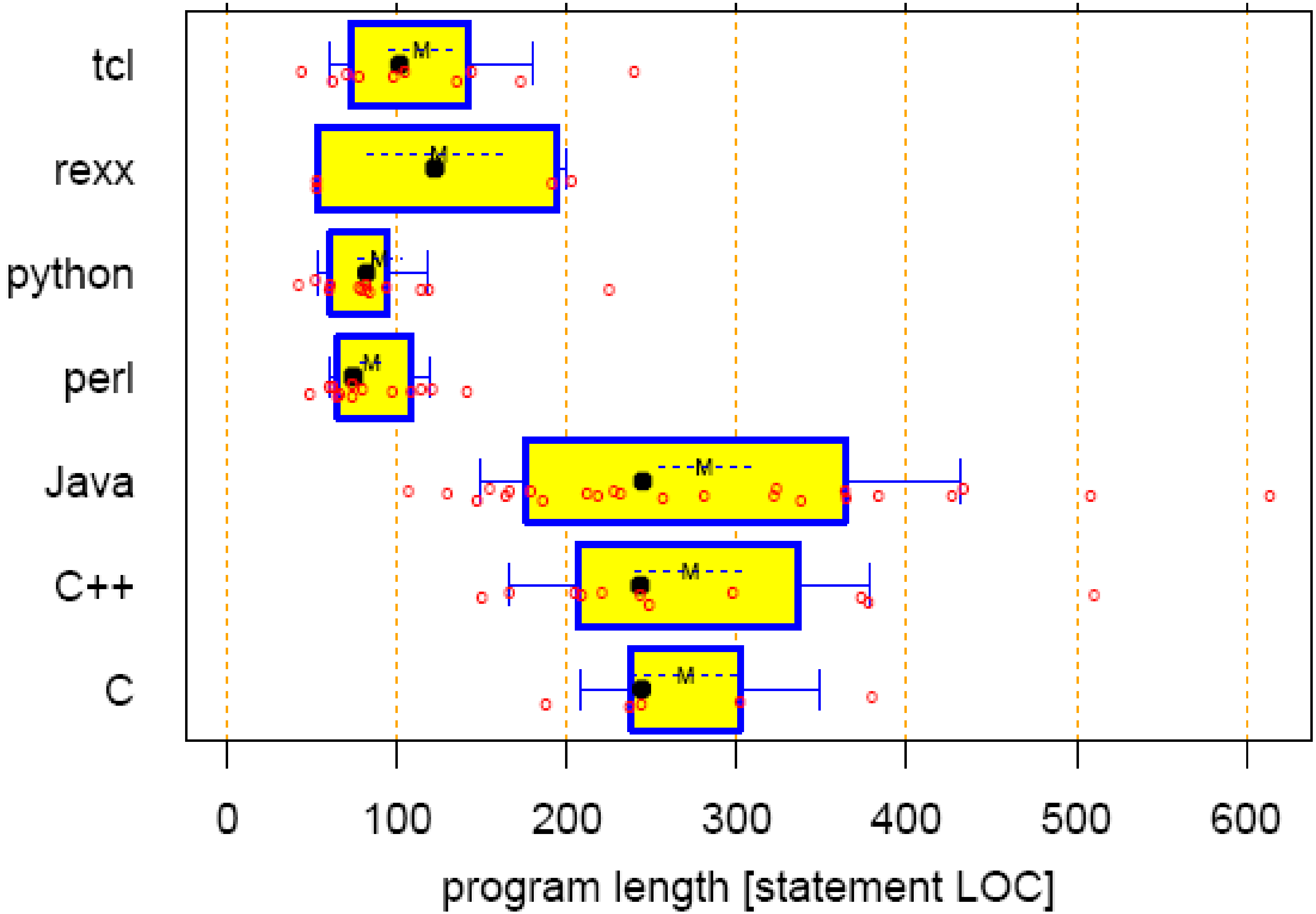
Some jccpprt results: Observations

- It looks like the compiled languages require 2-3 times as long for solving the problem with them
  - and interpersonal variation is also larger there

But beware:

- The Java, C, C++ times were measured in a controlled experiment → they are reliable
- The scripting language times are self-reported by participants found via Usenet postings
- Can we trust the latter?

Some jccpprt results: Length of resulting program



Some jccpprt results: Observations

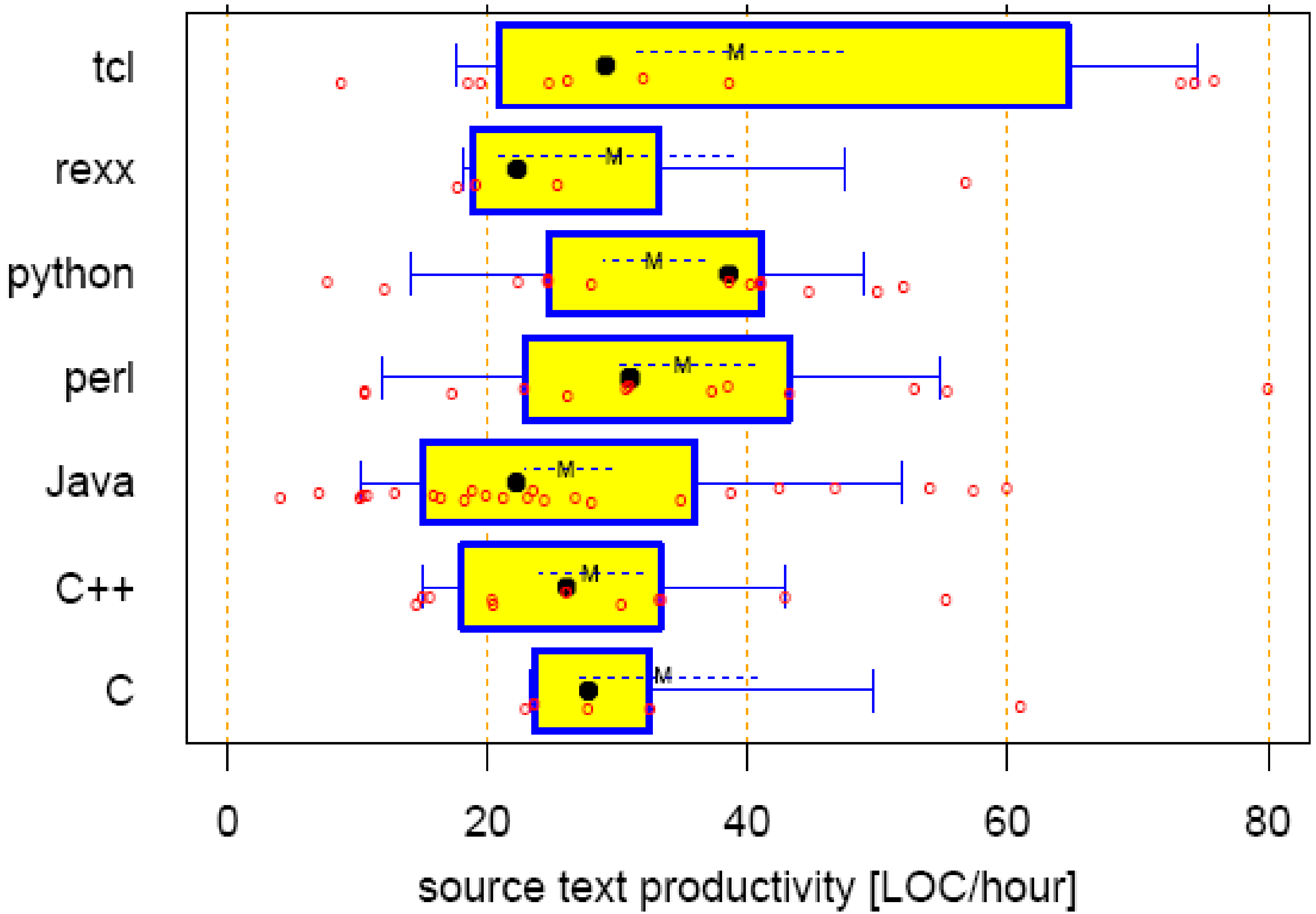
Freie Universität Berlin

- The length data are undisputable and confirm the work time data
  - if we assume that lines written per hour is language-independent

That leaves one problem:

- Are the length differences due to language differences or due to better qualifications of the script programmers?
  - Validation idea:
    Better-qualified programmers would write more lines per hour

Validation of comparability of programmer qualifications



jccpprt summary

- Overall, the sheer mass of data points allows to overcome the significant problems of interpersonal variation
- and even (to a reasonable degree) the validity problems regarding the trustworthyness of some of the data

- So how about comparing web development platforms in the same manner?

Example 2: A contact

- Late in 2005, Gaylord Aulke, CEO of the web development service company 100days, contacted me roughly like this:

- "We are using PHP in our projects.
  I use jccpprt to argue that scripting languages are to be taken seriously.
  Couldn't we do a similar study for PHP?"

Comparing web development platforms: The main problem

- The main success factor in jccpprt was the large number of implementations per language
- The large number was possible, because the task was small
  - Well under 10 person-hours for most participants, despite high reliability requirements and modest skills
- The task could be small, because the domain "language comparison" could be meaningfully addressed with an algorithmic problem.
  - The task was not more or less representative than any other.

- However web applications cannot reasonably be as small
  - The whole point of web development platforms is giving good support and reuse for many complex development tasks.
    - So those need to be present in the task

Which platforms?

- So we obviously have a scaling problem.
- Let's see how big that is.

First, which platforms would we need to look at?

- The "Big Money" platforms: Java Enterprise Edition, .NET
- The ubiquitous one: PHP
- The grand old lady: Perl
- Possibly younger contesters: Python, Ruby-on-Rails

- So we need to obtain 4 to 6 groups of implementations

Which aspects in the task?

Second, certain aspects should be present in the task to make the comparison meaningful:

- User registration, authentication, session tracking, forms and validation, dynamic results, reports, persistence, web services, ...

- So we need to have a task that is many times larger than that of jccpprt

How to make clear-enough differences likely?

Again, we have two possible approaches:

- Obtain a large number of implementations
- or obtain only a few, but from fairly similar programmers

Discussion:

- Which way to go?
- Why? How?
- How to ensure validity of the study?

One possible design: Plat_Forms

Plat_Forms The web development platform comparison

- Ensure validity by creating all implementations under supervision
  - same time, same place
- Make sufficient task size possible by inviting teams of 3 for 30 hours
  - A (hopefully sensible) balance of size and cost
- Reduce interpersonal variation by soliciting participation from top-class programmers only
- Make sufficient participation likely by advertising the study as a contest
  - an opportunity for software development service organizations to show off their capabilities
  - at most 3 teams will be admitted per platform

Lutz Prechelt, prechelt@inf.fu-berlin.de

Current status

- The Plat_Forms contest and study has been announced by 'iX in their November 2006 issue
  - and on heise online Newsticker on October 11
- The contest is planned for January 25-26, 2007, in Nürnberg
- We are now looking for
  - sponsors
  - platform representatives who will help select teams from among the applicants
  - teams who want to participate
  - students who would like to participate in the scientific evaluation afterwards

Tasks during the evaluation

- Develop tests and criteria for correctness and usability
  - May require comparing apples and oranges meaningfully
- Obtain load testing software, design and set up load tests
  - May be difficult if a lot of Javascript is used
- Develop categories and criteria for describing the structure, size, and modularity of the solutions
  - Must accomodate the very different approaches of the platforms
- Develop scenarios for characterizing the flexibility and modifiability of the solutions
  - Requires sufficient understanding of each platform
- Perform all these evaluations and record results
  - during February, March, April
- Write result report

Pointers

- Lutz Prechelt: *"An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program"*, Technical Report 2000-5, Fakultät für Informatik, Univ. Karlsruhe
  - The language comparison
  - http://page.mi.fu-berlin.de/~prechelt/Biblio/
  - Short version in IEEE Computer 33(10):23-29, October 2000
- http://www.plat-forms.org/
  - The web platform comparison
- Lutz Prechelt: *"The 28:1 Grant/Sackman legend is misleading, or: How large is interpersonal variation really?"*, Technical Report 1999-18, Fakultät für Informatik, Univ. Karlsruhe
  - Programmer variation study
  - Also in *"Kontrollierte Experimente in der Softwaretechnik"*, Springer Verlag, 2001
- Course *"Empirische Bewertung in Informatik"* (Empirical evaluation in informatics)
  - SoSe, 2V+2Ü

Thank you!