Karlsruhe Institute of Technology

Empirical Methods in Software Research:

Which Method Should I Use?

Walter F. Tichy

IPD Tichy, Fakultät für Informatik

Why do we need empirical methods in software research?

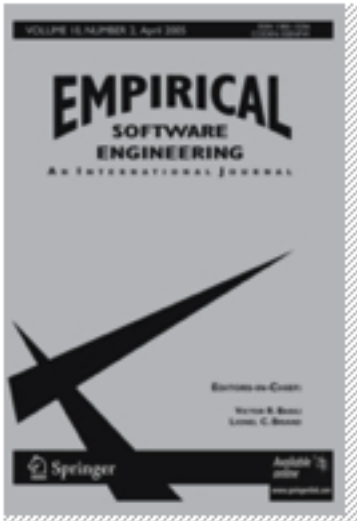
- There are simply too many tools and methods available for an individual or a software organization to try them all out in order to select the best one(s).

- However, the choice is critical for practitioners. Without data, there is no choice than to fall back on trends, fashion, opinions, personal preferences, prejudice, hearsay, salespersons, consultants, gurus.

- Empirical studies investigate, whether differences in software technologies actually exist,
with respect to cost, reliability, maintainability, usability, ease of learning, etc.

Empirical studies have become an active area in software research

Karlsruhe Institute of Technology



Empirical Software Engineering

An International Journal

Editor-in-Chief: Victor R. Basili; Lionel C. Briand

ISSN: 1382-3256 (print version)
ISSN: 1573-7616 (electronic version)

Journal no. 10664

International Symposium on
Empirical Software Engineering and Measurement

esem

16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)

MSR 2012 June 2–3. Zurich, Switzerland

The 9th Working Conference on Mining Software Repositories

Co-located with
ICSE 2012
June 2-9, 2012
Zurich • Switzerland

NEWS: The registration is now open!

Empirical Track

Download CFP REFSQ 2012 Empirical Track

The discussion at recent REFSQs have confirmed the strong need for empirical validation of the effectiveness for our RE methods by case studies and experiments, but the literature to date, including that of the REFSQ series, could show more of this validation. This lack is assumed to be at least partly due to the difficulties of

- bringing academics and practitioners together to pursue empirical studies and
- finding and persuading the participation of a sufficient number of suitable subjects for experiments.

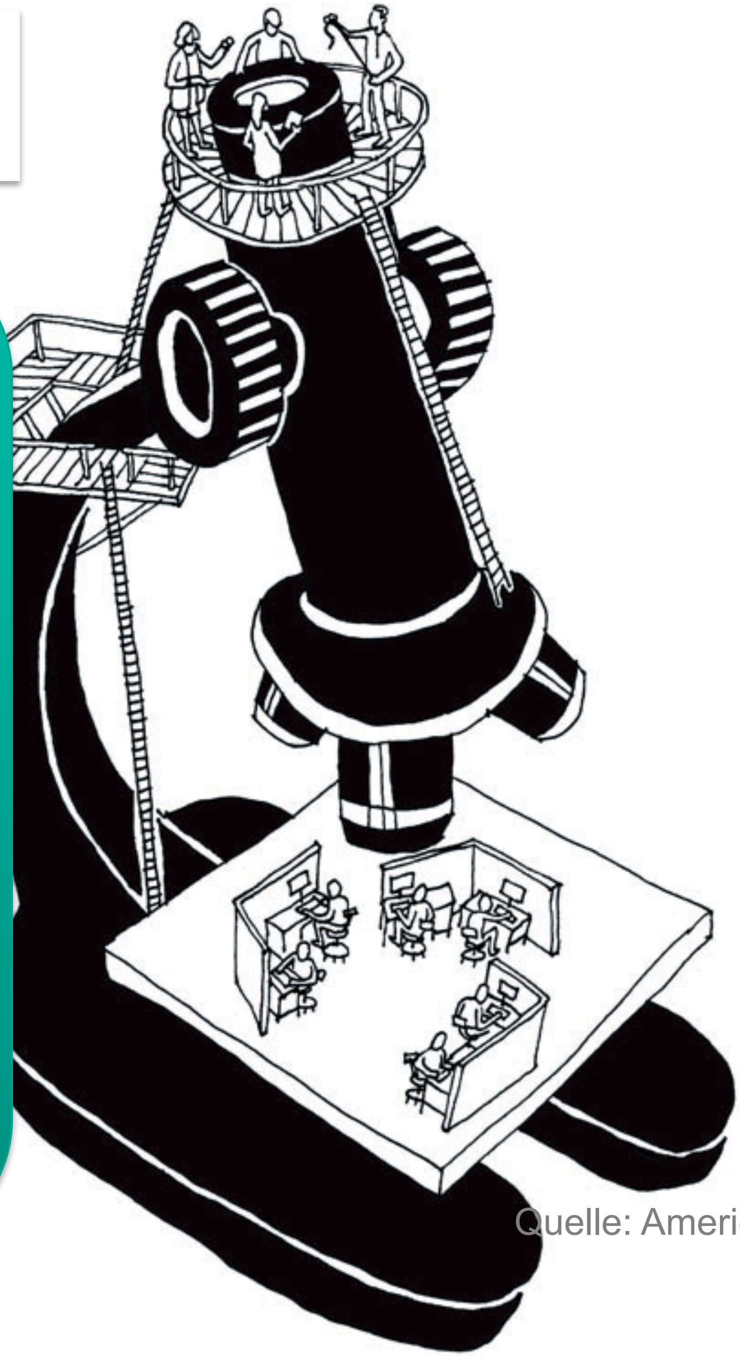
Therefore, REFSQ 2012 will offer two events in its empirical track:

1. Empirical Fair (EF): Practitioners can propose studies that their organizations would like to have conducted, and researchers can propose studies that they would like to conduct in industry. The EF is a meeting point to match the demand and supply of empirical studies among researchers and practitioners.
2. Empirical Studies at REFSQ (ESR): Practitioners and academics will be given the opportunity to conduct a small number of empirical studies during REFSQ 2012 itself. The goals of this opportunity, besides that of permitting the conduct of some studies, are to raise awareness for the necessity and benefits of empirical studies and to show that participating in them is not dangerous to one's health.

Software researchers at work

Invariant questions:

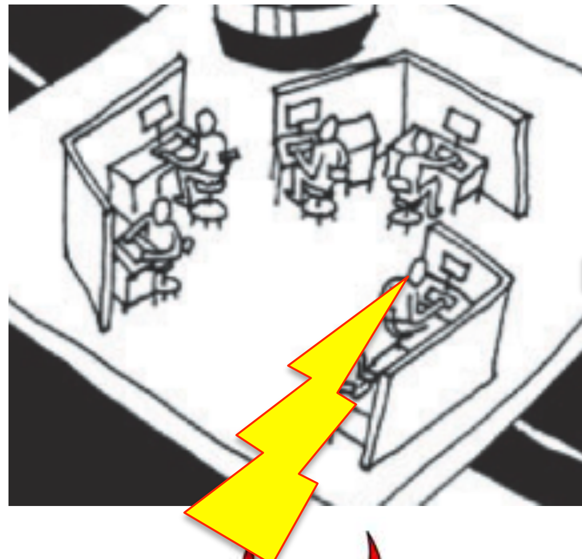
1. How to produce **software better** (faster, cheaper)?
2. How to produce **better software** (more reliable, more usable, more maintainable, etc.)?
3. How to show that 1. or 2. have been achieved?



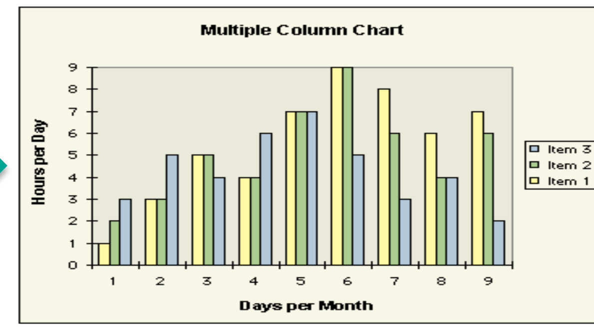
Quelle: American Scientist 6/2006

The controlled, randomized experiment

Vary
Independent
variables



Control
confounding
variables



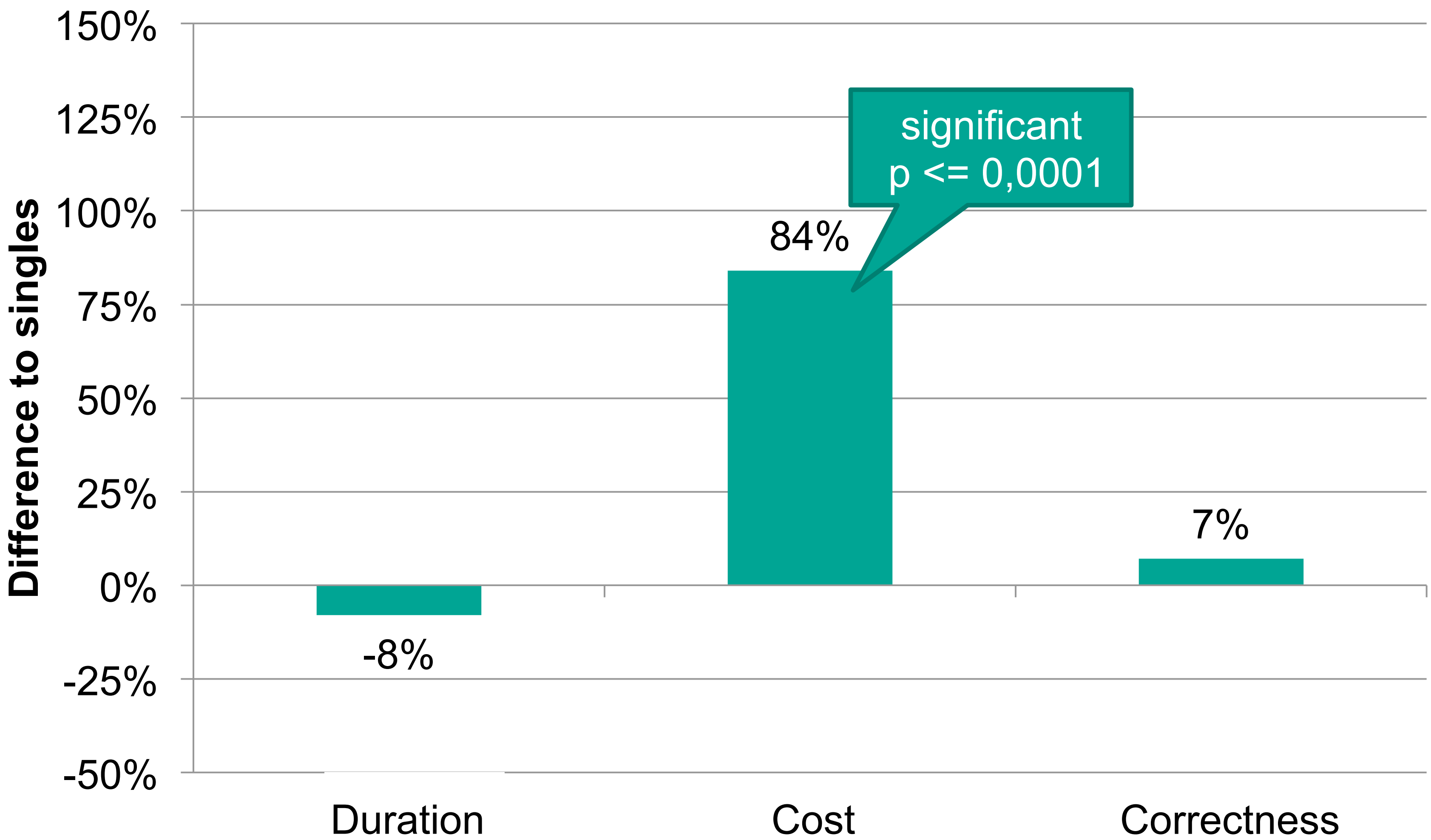
Observe
dependent
variables

Example: Experiment about Pair Programming

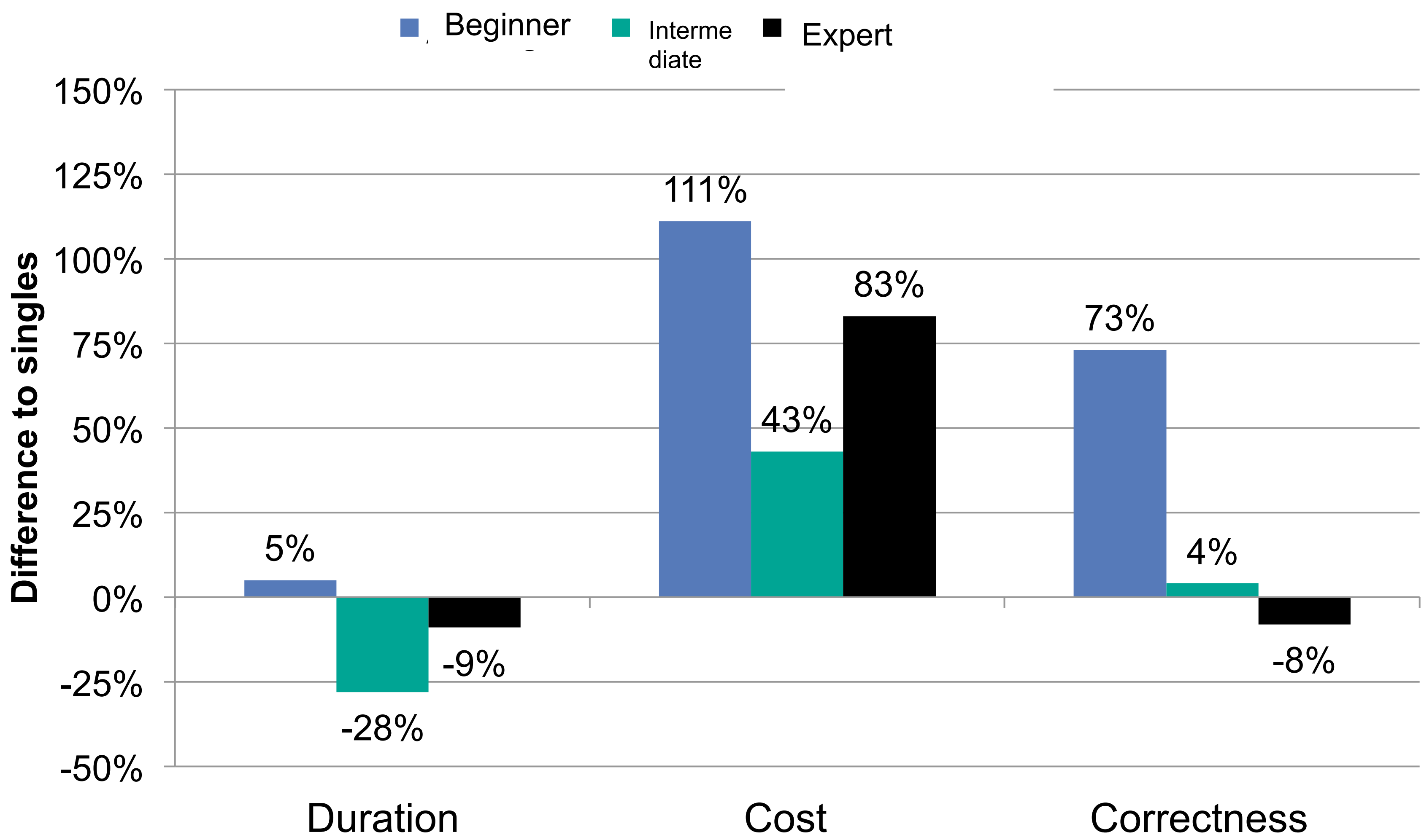
- 295 professional consultants (!)
- split into 99 single programmers and 98 pairs
- coming from 29 consultant companies in Norway, Sweden and GB
  - Accenture
  - Cap Gemini
  - Oracle
  - and others
- Participants were compensated for 5 hours work time.
- Cost for that alone: € 250.000

Erik Arisholm, Hans Gallis, Tore Dyba, Dag Sjoberg,
„Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise“,
IEEE Trans. On Software Engineering, Vol 33, no 2, Feb. 2007, 65-85.

Results for Pair Programming



Difference according to programmer competence



Results

- Large study, with almost 300 professional subjects
  - Generalizability is excellent.
- Distinguishes competence and sw complexity
  - PP is effective for beginners, especially when the sw is complex.
  - PP is ineffective for experts (without PP experience).
  - Recommendation: use pair programming for beginners

- Many studies use students as subjects.
  Have results with student subjects any relevance for professionals?

Some results from Experiments

- Inspections help find software defects early.
- Design patterns work as advertised.
- Inheritance depth is a poor predictor for maintenance effort.
- Pair programming only works for beginners.
- Pair programming can be replaced with single programmers and inspections (for beginners)
- Test-first is not better than test-last.
- UML does not help in maintenance tasks.

- Note: these are all experiments about software processes, not about tools (other than the last).

Pros and Cons of Experiments?

Karlsruhe Institute of Technology

- Advantages:
  - Establishes cause-effect relationship
  - Experimental method is well developed (methods, statistics)
- Disadvantages:
  - Exp
  - Prof … for their time
  - Exp … PhD student)
  - Negative results are the rule
  - Only feasible, if tools/methods are easy to learn

Suppose you are developing a new sw technique.
Your are busy improving it.
Experiment is much too expensive and time-consuming for each improvement step.
How can we make progress more quickly?

Alternative: Ex post facto Studies: Analyse Software Repositories

- Look for correlations in software repositories including bug histories
- Example: Can software metrics predict fault-prone components?

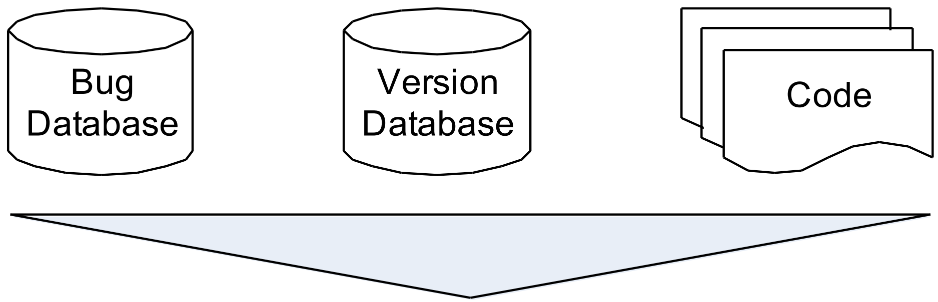
Nagappan, Ball, Zeller: Mining Metrics to Predict Component Failures, ICSE 2006

Zimmermann et al: Cross-project Defect Prediction, ESEC/FSE 2009.

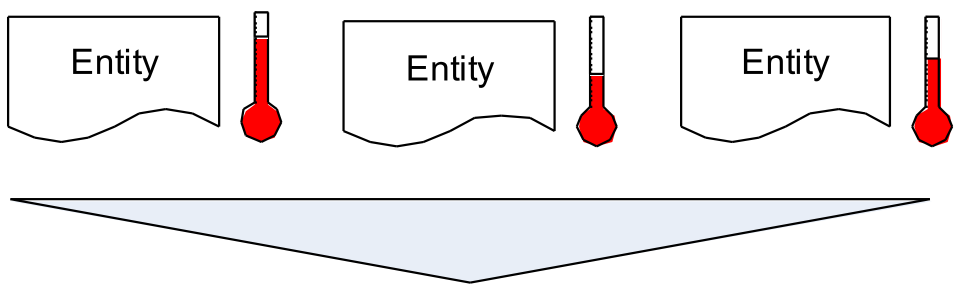
High level description

Karlsruhe Institute of Technology

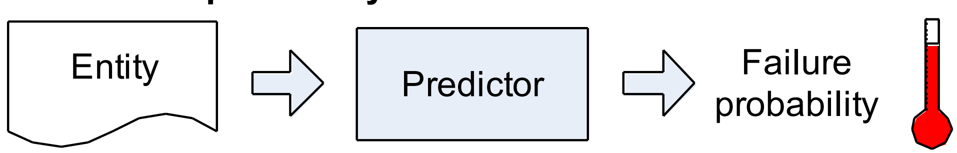
1. Collect input data



2. Map post-release failures to defects in entities



3. Predict failure probability for new entities



Source: Nagappan

Projects researched

- Internet Explorer 6
- IIS Server
- Windows Process Messaging
- DirectX
- NetMeeting

>1,000,000 Lines of Code

Quelle: Nagappan

Per-function metrics — correlation with maximum and sum of metric across all functions *f*() in a module *M*							
Lines	# executable lines in *f*()	Max	-0.236	**0.514**	**0.585**	**0.496**	**0.509**
		Total	0.131	**0.709**	**0.797**	0.187	**0.506**
Parameters	# parameters in *f*()	Max	-0.344	0.372	**0.547**	0.015	0.346
		Total	0.116	**0.689**	**0.790**	0.152	0.478
Arcs	# arcs in *f*()'s control flow graph	Max	-0.209	0.376	**0.587**	0.527	0.444
		Total	0.127	**0.679**	**0.803**	0.158	**0.484**
Blocks	# basic blocks in *f*()'s control flow graph	Max	-0.245	0.347	**0.585**	0.546	0.462
		Total	0.128	**0.707**	**0.787**	0.158	0.472
ReadCoupling	# global variables read in *f*()	Max	-0.005	**0.582**	**0.633**	0.362	0.229
		Total	-0.172	**0.676**	**0.756**	0.277	0.445
WriteCoupling	# global variables written in *f*()	Max	0.043	**0.618**	**0.392**	0.011	0.450
		Total	-0.128	**0.629**	**0.629**	0.230	0.406
AddrTakenCoupling	# global variables whose address is taken in *f*()	Max	0.237	**0.491**	**0.412**	0.016	0.263
		Total	0.182	**0.593**	**0.667**	0.175	0.145
ProcCoupling	# functions that access a global variable written in *f*()	Max	-0.063	**0.614**	**0.496**	0.024	0.357
		Total	0.043	**0.562**	**0.579**	0.000	0.443
FanIn	# functions calling *f*()	Max	0.034	**0.578**	**0.846**	0.037	**0.530**
		Total	0.066	**0.676**	**0.814**	0.074	**0.537**
FanOut	# functions called by *f*()	Max	-0.197	0.360	**0.613**	0.345	0.465
		Total	0.056	**0.651**	**0.776**	0.046	**0.506**
Complexity	McCabe's cyclomatic complexity of *f*()	Max	-0.200	0.363	**0.594**	0.451	**0.543**
		Total	0.112	**0.680**	**0.801**	0.165	**0.529**

Metrics and their Correlation with Post-Release Defects

Karlsruhe Institute of Technology

Per-class metrics — correlation with maximum and sum of metric across all classes *C* in a module *M*							
ClassMethods	# methods in *C* (private / public / protected)	Max	0.244	**0.589**	**0.534**	0.100	0.283
		Total	**0.520**	**0.630**	**0.581**	0.094	0.469
InheritanceDepth	# of superclasses of *C*	Max	**0.428**	**0.546**	0.303	0.131	0.323
		Total	**0.432**	**0.606**	**0.496**	0.111	0.425
ClassCoupling	# of classes coupled with *C* (e.g. as attribute / parameter / return types)	Max	**0.501**	**0.634**	**0.466**	-0.303	0.264
		Total	**0.547**	**0.598**	**0.592**	-0.158	0.383
SubClasses	# of direct subclasses of *C*	Max	0.196	**0.502**	**0.582**	-0.207	0.387
		Total	0.265	**0.560**	**0.566**	-0.170	0.387

Quelle: Nagappan

Do metrics correlate with failures?

Project	Metrics correlated w/ failure
A	*#Classes* and 5 derived
B	almost all
C	all except *MaxInheritanceDepth*
D	only *#Lines (software was refactored if metrics indicated a problem)*
E	*#Functions, #Arcs, Complexity*

Do metrics correlate with failures?

KIT Karlsruhe Institute of Technology

Project	Metrics correlated w/ failure
A	*#Classes* and 5 derived
B	alm… all
C	…xcept *…nheritanceD…*
D	…ly *#Lines…*
E	*…Functions…* …*mpl…*

YES

Given enough data for a project, a predictor for this project can be built.

Quelle: Nagappan

Is there a set of metrics that fits all projects?

Project	Metrics correlated w/ failure
A	*#Classes* and 5 derived
B	almost all
C	all except *MaxInheritanceDepth*
D	only *#Lines*
E	*#Functions, #Arcs, Complexity*

Is there a set of metrics that fits all projects?

Karlsruhe Institute of Technology

Project	Metrics correlated w/ failure
A	*#Classes* and 5 derived
B	[illegible]lmost a[illegible]
C	[illegible]cep[illegible] *[illegible]xInh[illegible]nceDep[illegible]*
D	[illegible]nly[illegible]
E	*#Functio[illegible]Arcs, Co[illegible]y*

NO

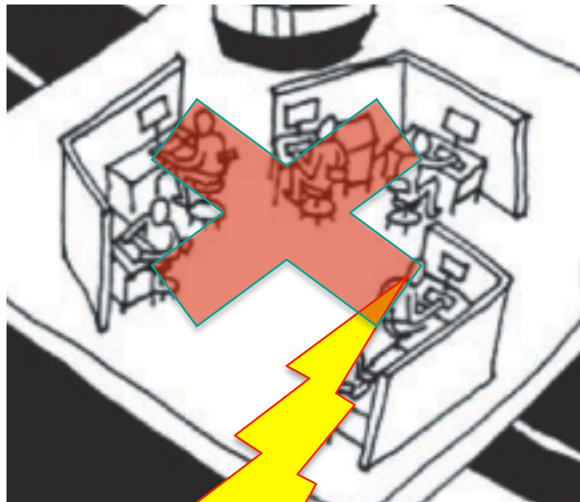
Quelle: Nagappan

Pros and Cons of SW Repositories

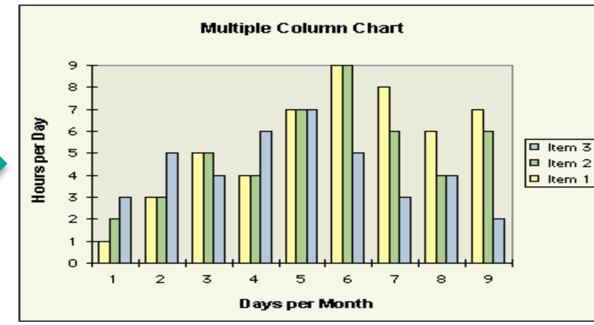
- Advantages
  - Large data sets available, even open source
  - Automate analysis
  - Quantitative results
  - Don’t need to deal with, or search for, human subjects. ☺
- Disadvantages
  - You only get correlations, no cause-effect relationship
  - Can only analyze what is there. If a new technique has not been used, then there is no data to analyze.
  - So it is useless for untried tools and methods

Analysis of software repositories

~~Vary independent variables~~



~~Control confounding variables~~



Observe dependent variables

Forschungszentrum Karlsruhe
in der Helmholtz-Gemeinschaft
Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

What to Do?

- How can the empirical community contribute useful insights that demonstrably improve software engineering?
- And do so faster than it has in the past?
- Note: “More money” is the wrong answer.

Recommendation: Use Benchmarks!

- Benchmarks are sets of problems with a quality metric for solutions (or gold standard solutions)
  - Independent teams apply their automated “solvers” to the problem and the quality of the solutions can be compared.
  - Benchmarks have a tremendous advantage over experiments with human subjects: they can be repeated as often as necessary, usually at moderate cost.
  - Setting up a benchmark is usually not for free: data has to be collected, benchmark programs have to be prepared.
  - However, this cost can be amortized over many trials and provides a basis for comparison.
  - Over time, the benchmark must evolve (become harder, more general, avoid overfitting.)

Benchmarks have been extremely successful in driving research

- **Computer architecture:** Various benchmarks have been used for decades in order to compare processor performance.
  - The Standard Performance Evaluation Corporation (SPEC) publishes benchmarks to evaluate a range of performance criteria (CPU, Web server, Mail Server, AppServer, power consumption, etc.)
  - Benchmarks combined with simulation have made computer architecture research quantitative.
  - Every performance feature must be substantiated on relevant benchmarks.

Autonomic vehicles: DARPA Grand Challenge

Google's autonomic vehicle

2007 DARPA
Urban Challenge

2004, 2005 DARPA
Grand Challenge

Where Benchmarks Rule:

- **Databases:** Transaction Processing Performance Council (TPC)
- **Speech recognition:** large databases of speech samples are used in competitions to determine the best speech recognizer
  - Here, the issue is not speed, but error rate.
- **Speech translation:** same idea.

In all of these cases, benchmarks resulted in swift and substantial progress.
The winning techniques were quickly adopted by other teams and improved upon.
How could we achieve comparable progress in software research?

Software research could use more benchmarks

Karlsruhe Institute of Technology

- Benchmarks apply to any tool that automates an aspect of software engineering.
- Share the work on developing a wider range of meaningful and challenging benchmarks, so
  - The work is spread over several teams
  - better tools can be built,
  - we know which techniques work best,
  - progress accelerates.

- Some examples of SE benchmarks follow.

Example 1: Data Race Detection

- Data races (unsynchronized accesses to shared variables) are a common defect in parallel programs.
- They are difficult to find.
- Current race detectors are impractical
  - They produce thousands to millions of false alarms.
  - Programmers are overwhelmed.
- Why false positives?
  - Ad-hoc, programmer-defined synchronizations
  - Unknown synchronization libraries
  - Detectors cannot reason about these, causing many false positives
- Contribution: how to handle user-defined synchronization and unknown synchronization libraries, reducing false positives.

What is a Data Race?

- Two or more concurrent accesses to a shared location, at least one of them a write.

Thread 1	Thread 2
X = 0	
X++	T = X

T=0 or T=1?

Ad-hoc (User-defined) Synchronization

Karlsruhe Institute of Technology

- Synchronization constructs implemented for performance reasons

/*Initially **FLAG** is zero */

Thread 1

```
DATA++

FLAG = 1
...
```

Thread 2

```
...

while(FLAG == 0)
 --do nothing

DATA--
```

- Ad-hoc synchronizations are widely used
  - 12 - 31 in SPLASH-2 and 32 - 329 in PARSEC 2.0

Test Suite – data-race-test

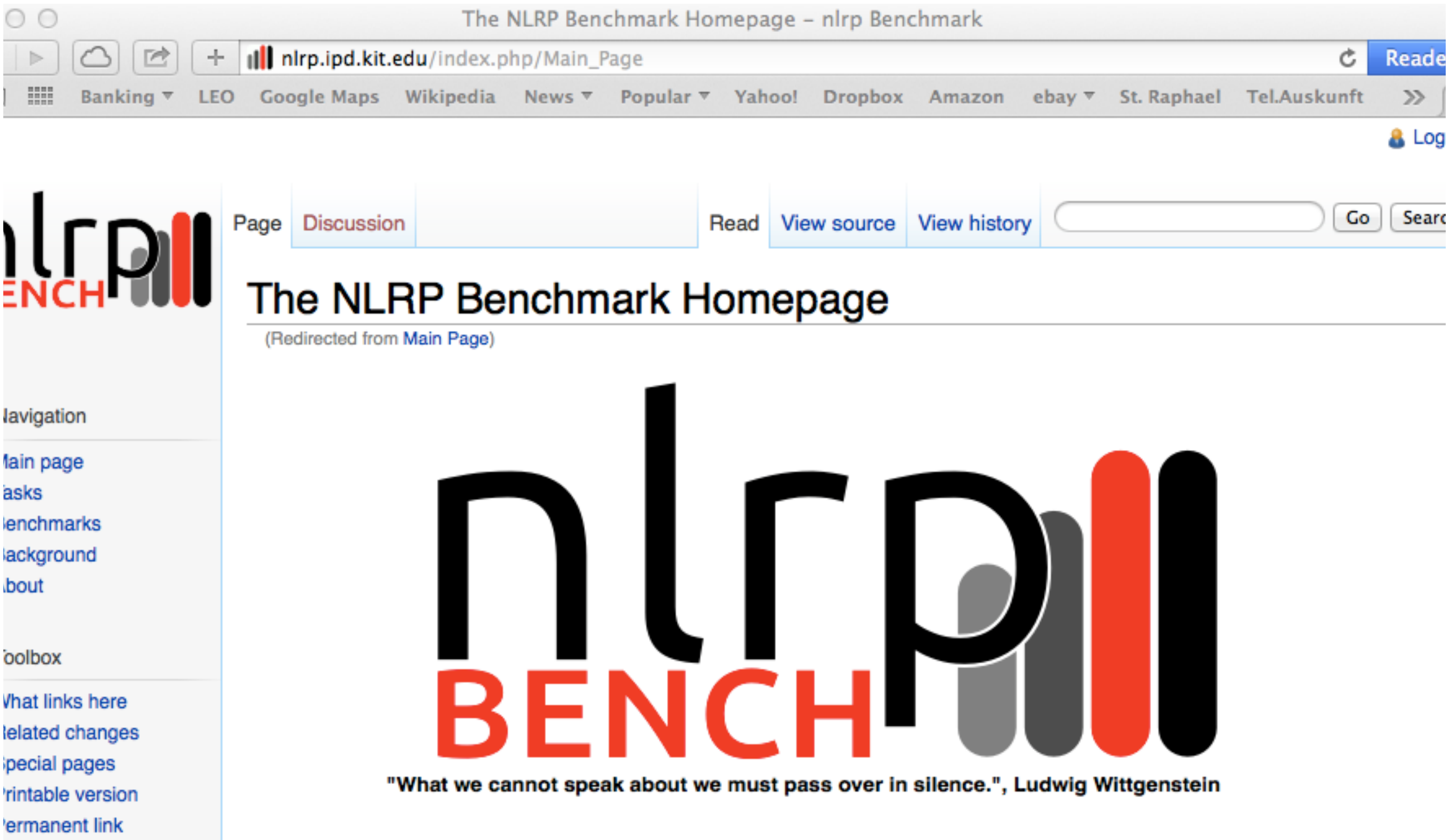
- 120 different test cases (2-16 Threads)
  - Test cases are racy or race-free programs (using Pthread)
    - Includes difficult cases
  - Spinning read loop detection of up to 7 basic blocks
    - 24 false positives and one false negative are removed
  - Removing information about Pthread library (unknown library)
    - Only one false positive more

Tools	False alarms	Missed races	Failed cases	Correctly analyzed cases
Helgrind$^+$ lib	32	8	40	80
Helgrind$^+$ lib+spin(7)	**8**	**7**	**15**	**105**
Helgrind$^+$ nolib+spin(7)	9	7	16	104
DRD	13	20	33	87

Exmple 2: Auto-Parallelization Benchmark

- To test automatic parallelizers, we construct a benchmark
  - sequential implementations
  - hand-parallelized implementation
- We test auto-future detection, pipelines, master/worker and other patterns
  - Is all parallelization potential found?
  - Were correct transformations steps performed?
  - Were concurrency bugs introduced?
  - What speed-up was achieved?

Example 3: NLRP-Bench
A Benchmark for Requirements Processing

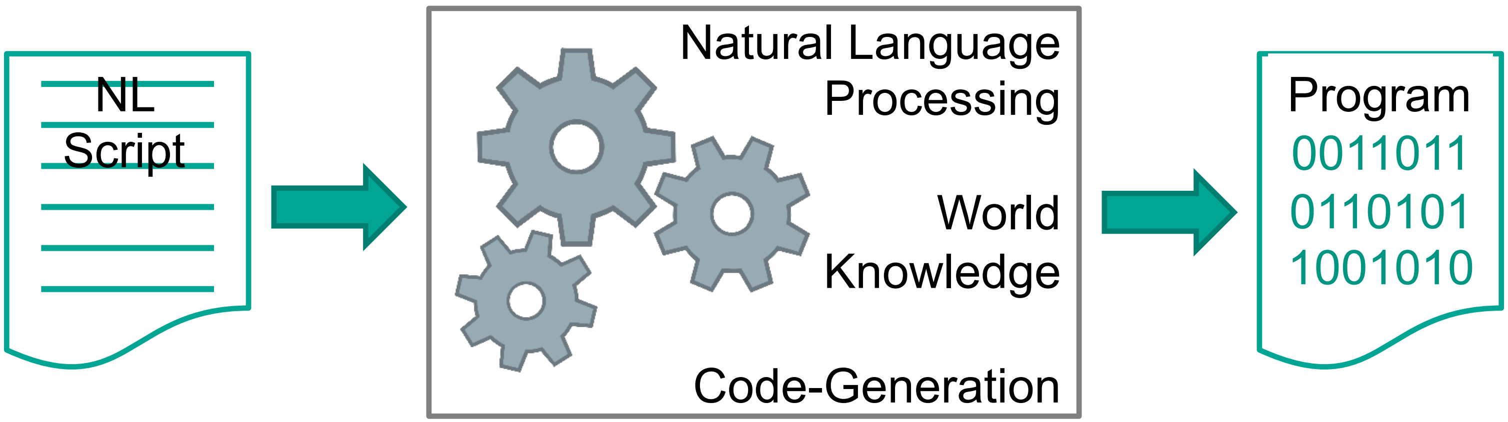


Online at http://nlrp.ipd.kit.edu

- Sample Requirements Specs:
- ITrust Medical Care
- Pacemaker
- Elevator
- Steam Boiler
- Ambulance Dispatching System
- Movie Theatre
- Kuchenrezept
- Ludo
- Problemmelder
- Pflichtenheft Handyverträge

- RE UTS Coincidence Matrix in the ATLAS Muon Spectrometer
- Quasar Fraunhofer Türsteuergerät
- German Health Professional Card and Security Module Card
- ERS ACME - University Library Information System
- Racing
- Timbered House
- Whois Protocol
- Display Management System
- Cable TV Package Purchase
- DaimlerChrysler Demonstrator: Instrument Cluster

A Grand Challenge:
Programming in ordinary language



Benchmarks would be good for evaluation.
But where to get them?

Answer: Create Animations and let subjects describe them in their own words. Then use the stories as input to the generator.

Subjects are shown the video and tell the story

- 10 different animations so far,
- 90 stories, which are the benchmark for AliceNLP.

The astronaut says, "That's one small step for a man...". As he says this, the alien is moving on his wheels toward him. The astronaut continues, "...one...giant leap for...". He stops as he sees the alien moving towards […]

The spaceman makes a step forward. While he makes the step, he says, "That's one small step for a man!". Then, the alien moves a few meters forward and turns a bit to its left. […]

Conclusions

- I think the use of benchmarks in software research is not as high as it could be.
- All areas of SE could benefit: requirements, design, implementation, testing, maintenance.
- With realistic benchmarks, one gets reliable and testable results.
- Benchmarks accelerate progress: they eliminate inferior choices quickly, help concentrate on the challenges.
- Share the work of preparing benchmarks.
- With a concentrated effort in benchmarking, we might speed up tool research dramatically.
- When tool progress has been made, check usability with human subjects (the expensive experiment).

“If you are not keeping score, you’re just practicing.”

Vince Lombardi
Berühmter US Football Trainer

Barcelona gegen Manchester United: Wer spielt besser?