Empirie in der Software-Forschung:

Welche Methode wann?

Walter F. Tichy

IPD Tichy, Fakultät für Informatik

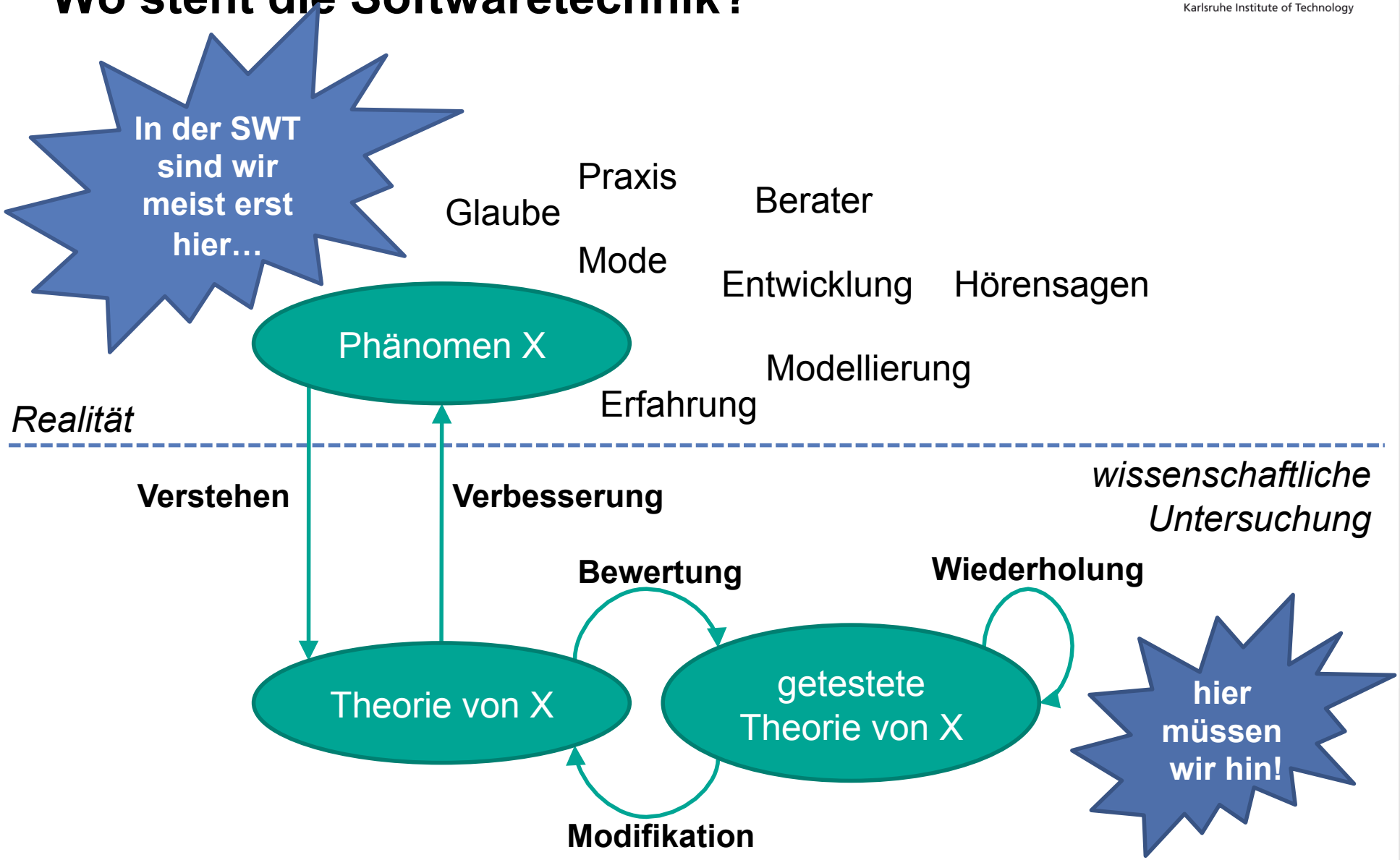
Wozu Empirie in der SWT?

- Es existieren zu viele SW-Methoden und Werkzeuge, als dass der einzelne Programmierer oder auch ein Softwarehaus die beste Wahl durch Ausprobieren ermitteln könnte.

- Diese Wahl ist aber wichtig für den Praktiker. Ohne fundierte Erkenntnisse ist er Moden, Meinungen, Vorlieben, Hörensagen, Verkäufern, Beratern und Gurus ausgesetzt.

- Empirische Studien untersuchen, ob Unterschiede zwischen verschiedenen Softwaretechniken tatsächlich beobachtet werden können,
  z.B. in Bezug auf Zuverlässigkeit, Kosten, Wartbarkeit, Erlernbarkeit, Gebrauchstauglichkeit, etc.

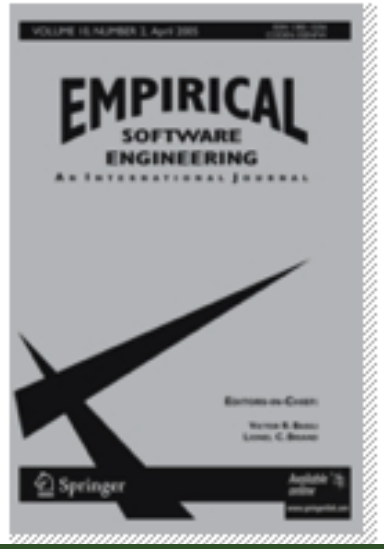
Wo steht die Softwaretechnik?



Rolle der Empirie

- Empirie beobachtet, misst und bewertet Phänomene
- Empirie bestätigt oder verwirft Theorien
- Empirie regt neue/verbesserte Theorien an

Empirie ist in der SW-Forschung angekommen



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.

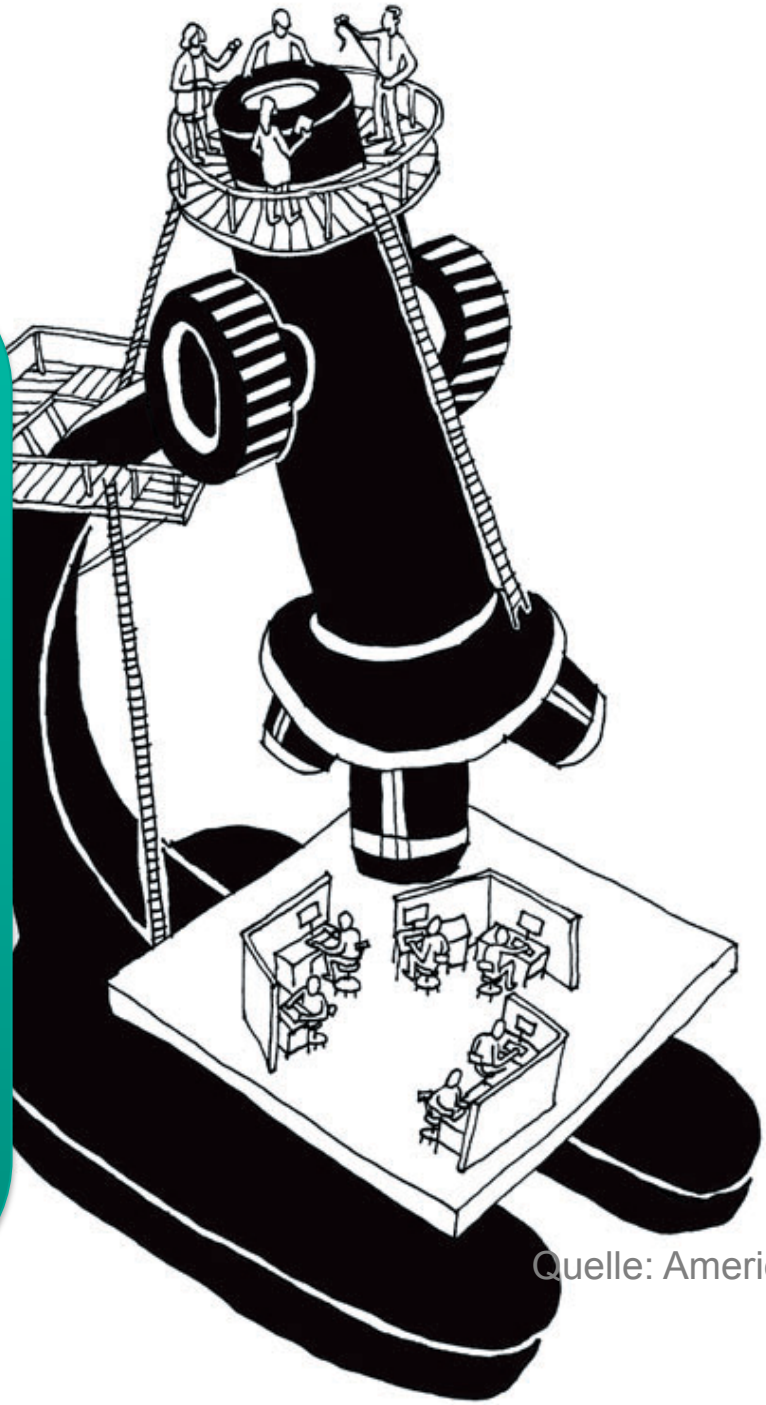
Therefore, REFSQ 2012 will offer two events in its empirical track:

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.

Softwareforscher bei der Arbeit

Ewig gleiche Grundfragen:

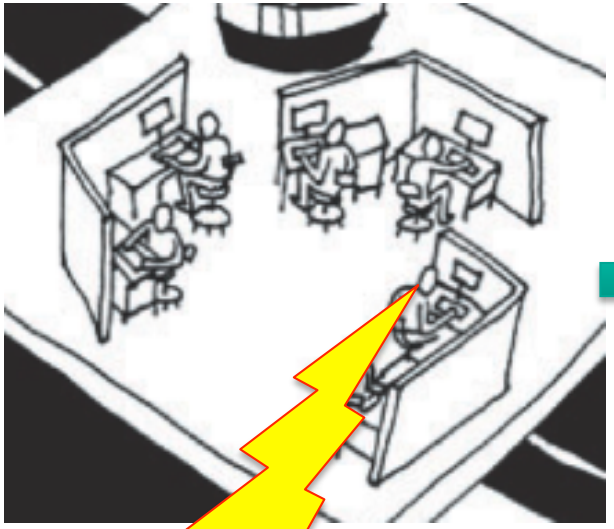
1. Wie kann man SW besser konstruieren (schneller, günstiger)?
2. Wie kann man bessere Software konstruieren (zuverlässiger, sicherer, brauchbarer, etc.)?
3. Und wie zeigt man, dass man 1. oder 2. erreicht hat?



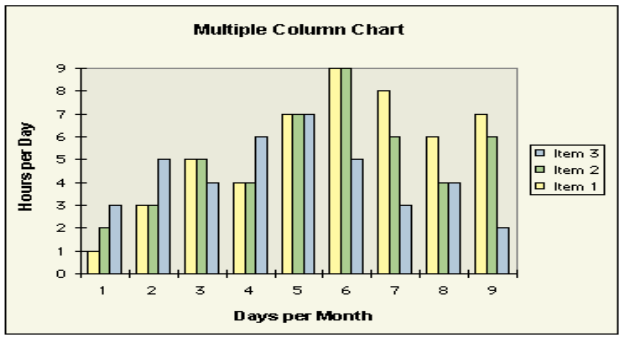
Quelle: American Scientist 6/2006

Das kontrollierte, randomisierte Experiment

variiere
unabhängige
Variablen



kontrolliere
Störvariablen



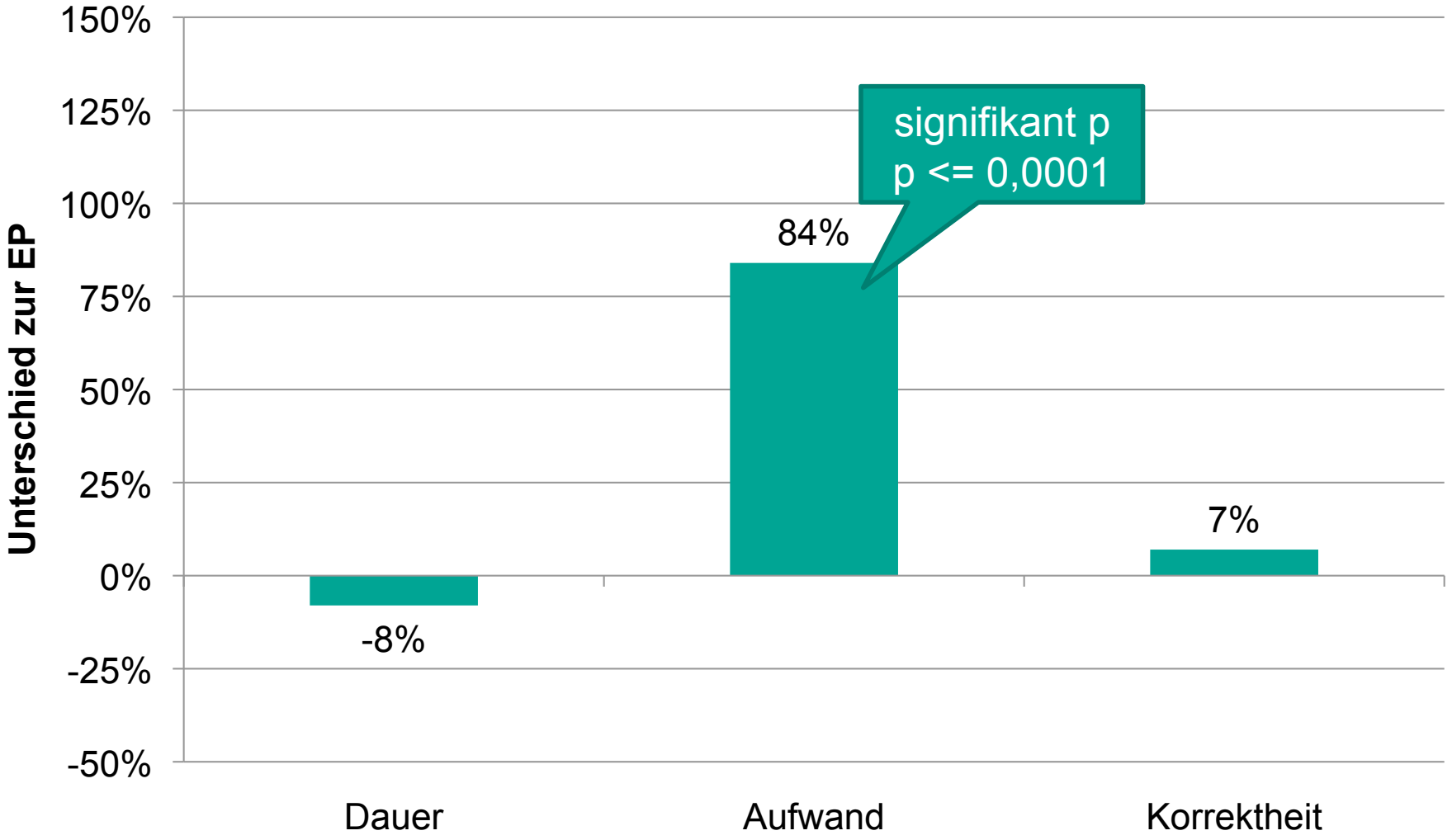
Beobachte
abhängige
Variablen

Beispiel: Experiment zum Paarprogrammieren

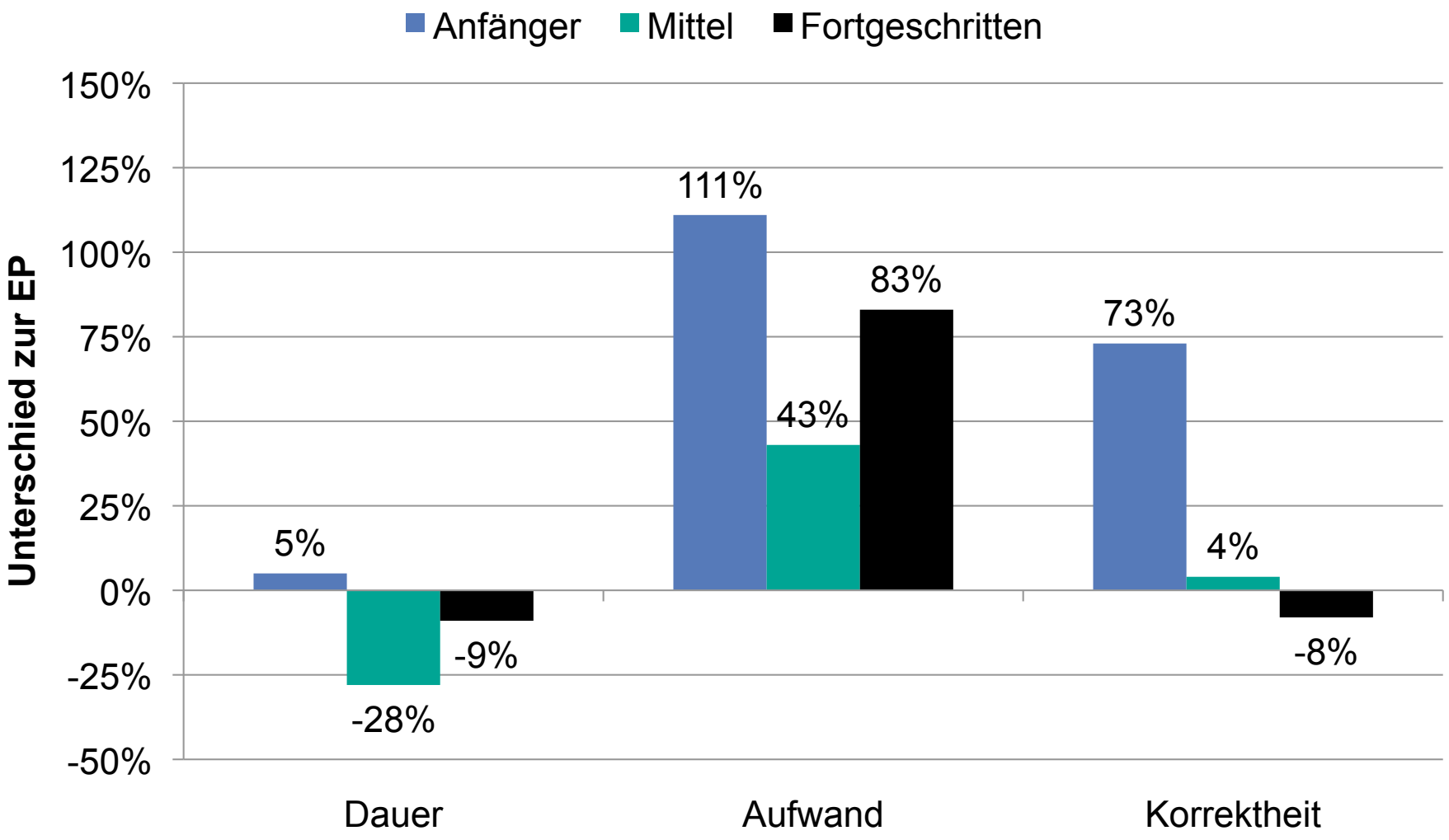
- 295 professionelle Berater (!)
- Aufgeteilt in 99 Einzelprogrammierer und 98 Paare
- aus 29 Beraterfirmen in Norwegen, Schweden und GB:
  - Accenture
  - Cap Gemini
  - Oracle
  - u. a.
- Teilnehmer wurden für fünf Stunden bezahlt.
- Kosten dafür: ca. € 250.000

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.

Auswirkung der PP



Unterscheidung nach Kompetenz der Programmierer



Fazit

- Große Studie, mit fast 300 professionellen Teilnehmern.
- Berücksichtigte Systemkomplexität und Kompetenz.
- Ergebnisse:
  - PP ist effektiv für Anfänger, besonders wenn das zu ändernde System komplex ist.
  - PP ist ineffektiv für fortgeschrittene Programmierer (ohne Erfahrung mit PP).
  - Empfehlung: benutze Paare bei Wartung durch Anfänger.

- Sind Studenten als Teilnehmer brauchbar, wenn man auf Profis generalisieren möchte?

Einige Ergebnisse aus Experimenten

- Inspektion sind effektiv bei der Defekt-Eliminierung.
- Entwurfsmuster funktionieren wie beworben.
- Vererbungstiefe ist schlechter Prädiktor für Wartungsaufwand.
- Paarprogrammierung hilft nur Anfängern.
- Paarprogrammierung kann durch Einzelprogrammierung mit Reviews ersetzt werden (bei Anfängern).
- Test-getriebene Entwicklung bringt nichts.
- UML bringt für die Wartung nichts.

Beachte: alles Prozessexperimente, Erlernen geht rasch. Kaum Werkzeugentwicklung nötig.

Wann benutzt man das Experiment?

- Vorteile:
  - Kann Ursache-Wirkungs-Zusammenhang identifizieren
  - Experimentelle Methodik exzellent entwickelt (Statistik, Kontrolle)
- Nachteile:
  - Aufwändig, teuer
  - Professionelle Teilnehmer sind schwierig zu kriegen
  - Experimente dauern lange (min. 1 Jahr pro Experiment)
  - Negative Ergebnisse sind die Regel
  - Nur brauchbar, wenn Methodik/Werkzeug ausgereift und Erlernbarkeit kurzfristig möglich.
- Welche empirische Methode soll man nehmen, wenn eine Technik erst noch entwickelt und verbessert werden soll? Denn da ist ein Experiment meist zu teuer.

Ex post facto Studien: Analysieren von Software-Depots

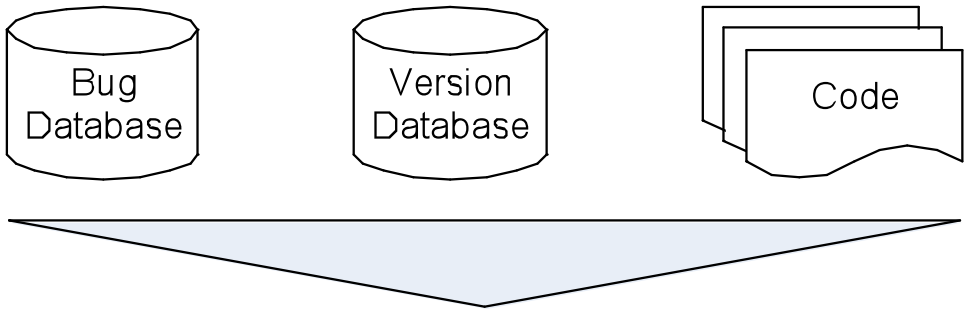
- Untersuche die Versionshistorie von Software in Verbindung mit Fehlermeldung
- Beispiel: Können Softwaremetriken fehleranfällige Komponenten vorhersagen?

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

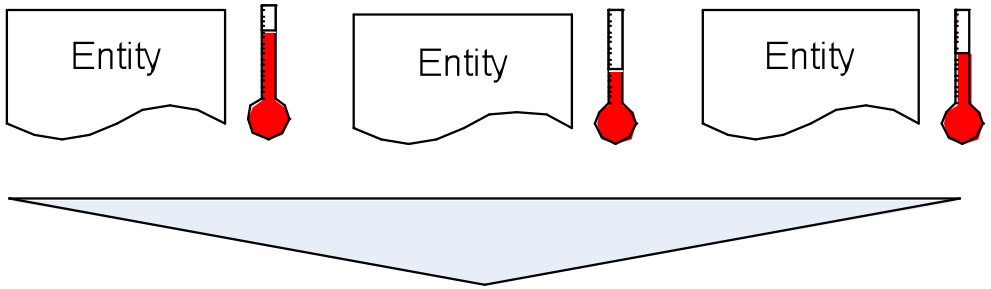
Zimmermann et al: Cross-project Defect Prediction, ESEC/FSE 2009.

High level description

1. Collect input data



2. Map post-release failures to defects in entities



3. Predict failure probability for new entities

Quelle: 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

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?

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

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?

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

NO

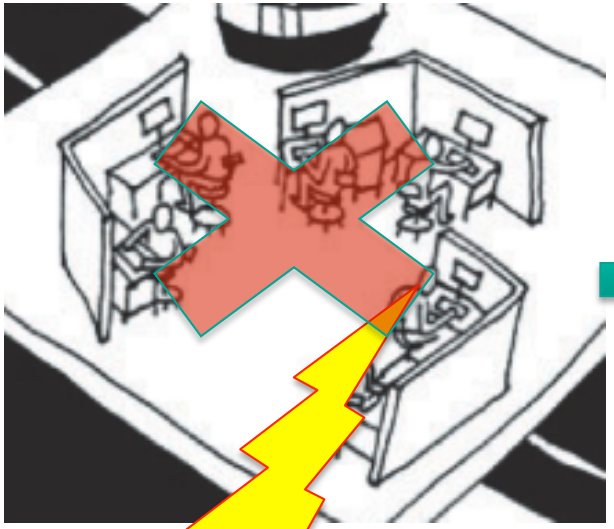
Quelle: Nagappan

Wann eignet sich die Analyse von Software?

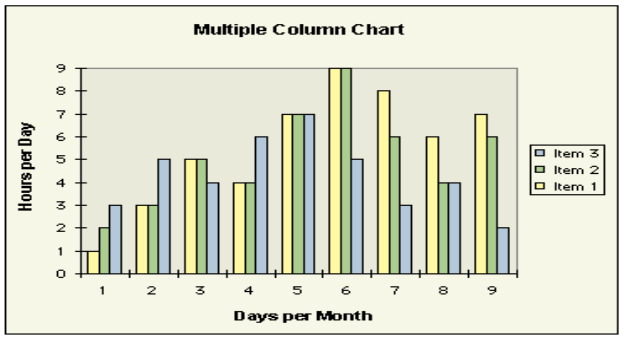
- Vorteile
  - Große Datenmengen verfügbar
  - Analyse automatisierbar
  - Kein Suche nach, kein Kontakt mit, menschlichen Subjekten. ☺
- Nachteile
  - Man kann nur analysieren, was gegeben ist—wenn z.B. kein Paarprogrammieren durchgeführt wurde, kann man es auch nicht analysieren.
  - D.h., neue Techniken sind frühzeitig kaum analysierbar.
  - Man erhält bestenfalls Korrelationen, aber keinen sicheren Ursache-Wirkungs-Zusammenhang.

Analyse von Software-Deponien

variiere unabhängige Variablen



kontrolliere Störvariablen



Beobachte abhängige Variablen

Was gibt´s denn noch?

Empirische Forschungsmethoden

Deskriptive Forschung
beschreibt Phänomene,
Ereignisse, Situationen

Experimentelle Forschung
Beschreibt Ursache-Wirkung,
ist quantitativ

Quantitative Studie

Umfrage
Korrelations-Studie
Ex post facto Studie
Langzeit- und Querschnittsstudie
Naturalistische Beobachtung
Meta-Studie

Qualitative Studie

Fallstudie
Ethnographie
Phänomenologie

Laborexperiment
Feldexperiment
Simulation
Benchmark

Was sind die Probleme der heutigen SW-Empirie?

- Fortschritt ist zu langsam, Kosten zu hoch.
- Ergebnisse sind nicht konstruktiv.
- Der typische Informatiker will etwas konstruieren, nicht nur analysieren (Nachwuchsproblem bei „reiner“ empirischen Forschung).

- Wie kann die Empirie Beiträge liefern, die die Softwarewerkzeuge nachweislich verbessern,

- und das schneller als bisher?

Vorschlag: Benutze Benchmarks!

- Benchmarks sind Mengen von Problemen mit einer Qualitätsmetrik für Lösungen.
  - Unabhängige Forschungsgruppen wenden ihre „Löser“ an und vergleichen ihre Ergebnisse mit denen von anderen.
  - Verbesserungen können ohne große Verzögerung implementiert werden, gehen in richtige Richtung.
  - Benchmarks haben einen riesigen Vorteil gegenüber Versuchen mit menschl. Teilnehmern: Sie können beliebig oft wiederholt werden.
  - Benchmarks müssen weiterentwickelt werden, um Überanpassung zu vermeiden.

Benchmarks sind effektiv, Forschung vorwärts zu treiben.

- **Rechnerarchitektur:** Benchmarks werden seit Dekaden zum Leistungsvergleich benutzt.
  - Standard Performance Evaluation Corporation (SPEC) publiziert eine Reihe von Benchmarks (CPU, Web server, Mail Server, AppServer, power consumption, etc.)
  - Benchmarks und Simulation haben Rechnerarchitektur quantitativ gemacht.
  - Jede Prozessoreigenschaft muss sich an Benchmarks bewähren.
- Marketing Benchmarks: um die zu vermeiden, wurden unabhängige Benchmark-Organisationen gegründet.

Wo Benchmarks regieren:

- **Databanken:** Transaction Processing Performance Council (TPC)
- **Spracherkennung:** große Mengen von Sprachproben werden benutzt, um Spracherkenner zu trainieren und zu vergleichen (Fehlerraten) (DARPA)
- **Sprachübersetzung:** genauso.

Autonome Fahrzeuge: DARPA Herausforderungen

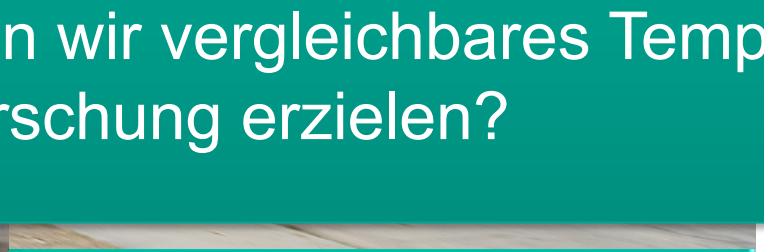
Google automes Fahrzeug

2007 DARPA
Urban Challenge

2004, 2005 DARPA
Grand Challenge

Autonome Fahrzeuge: DARPA Herausforderungen

2004, 2005 DARPA
Grand Challenge



2007 DARPA
Urban Challenge

In all diesen Fällen haben Benchmarks zu raschen und substanziellen Fortschritten geführt.

Erfolgreiche Techniken wurden von anderen Teams rasch aufgegriffen und verbessert.

Wie könnten wir vergleichbares Tempo in der Softwareforschung erzielen?

Software-Forschung könnte mehr Benchmarks benutzen

- Benchmarks können überall da angewandt werden, wo Softwareprozesse automatisiert werden.
- Beteilige dich an der Entwicklung brauchbarer Benchmarks, damit
  - die Arbeit auf mehrere Schultern verteilt wird,
  - bessere Werkzeuge gebaut werden können,
  - die besten Techniken herausgefiltert werden,
  - der Fortschritt beschleunigt wird.

- Beispiele ungewöhnlicher Benchmarks in der SWT folgen.

Beispiel: Verarbeitung natürlichsprachlicher Anforderungen

- Problem: übersetze sprachliche Anforderungen in UML Modelle (und zurück).
- Über 70% der Anforderungen sind in uneingeschränkter, natürlicher Sprache geschrieben.
- Benchmarks: Echte Anforderungen sind erstaunlich schwierig zu finden:
  - Lehrbücher enthalten wenige Beispiele, die die Autoren selbst erfunden haben, oder von anderen kopiert haben, die sie auch erfunden haben.
  - Firmen wollen keine freigeben (Qualität oft peinlich).
- Mit Beispielen könnten wir ein Problem nach dem anderen anpacken.
- Unsere Sammlung: http://nlre.wikkii.com/wiki/The_NLRP_Benchmark_Homepage

Ansatz: Thematische Rollen

Einige für SW-Ingenieure interessante Rollen:

- agens (AG) – der Handelnde.
  patiens (PAT) – der Behandelte.
  actus (ACT) – die Handlung (für Tätigkeitsverben und deren Nominalisierungen)

  Peter_{AG} wirft_{ACT} einen Ball_{PAT}.
  Peter_{AG} tätigt einen Wurf_{ACT} des Balls_{PAT}.

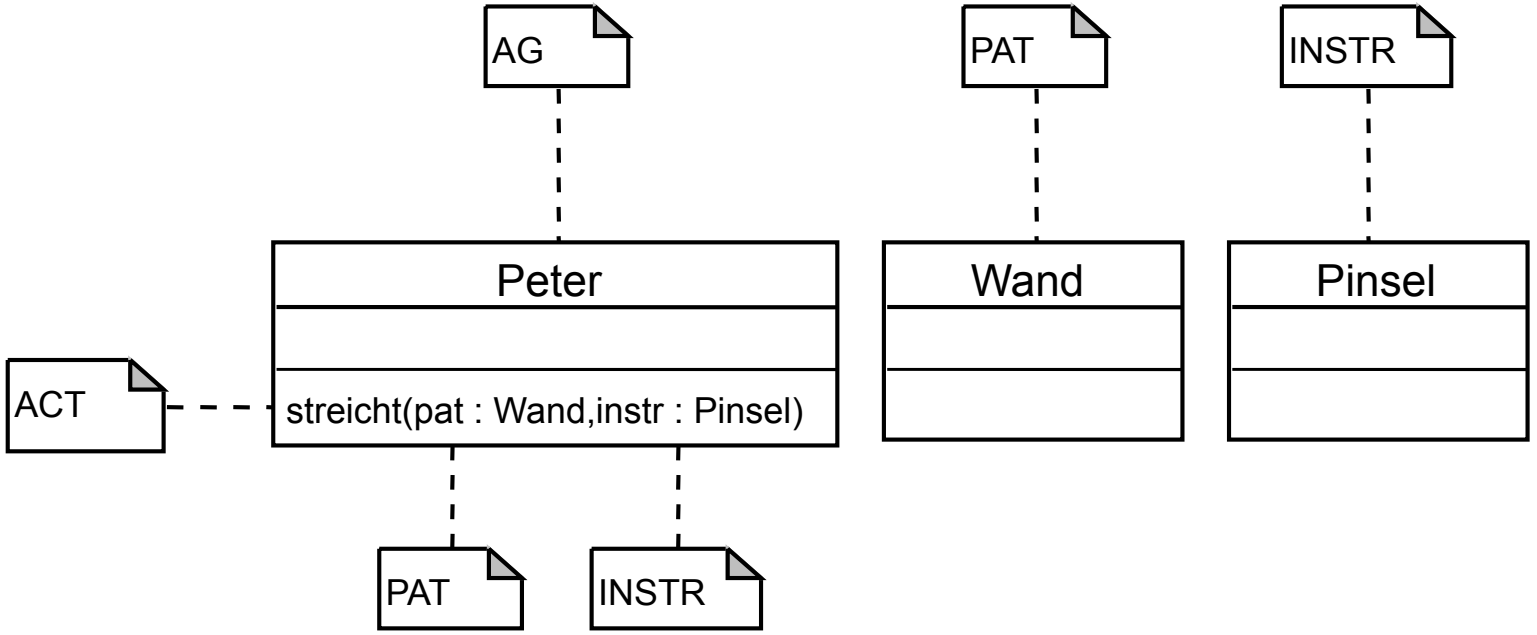
- instrumentum (INSTR) – das Hilfsmittel
  Peter streicht die Wand mit einem Pinsel_{INSTR}.

- status (STAT) – der Zustand (für Zustandsverben und deren Nominalisierungen)
  Peter wohnt_{STAT} in Bonn.

(Gelhausen2007)

Abbildung thematischer Rollen

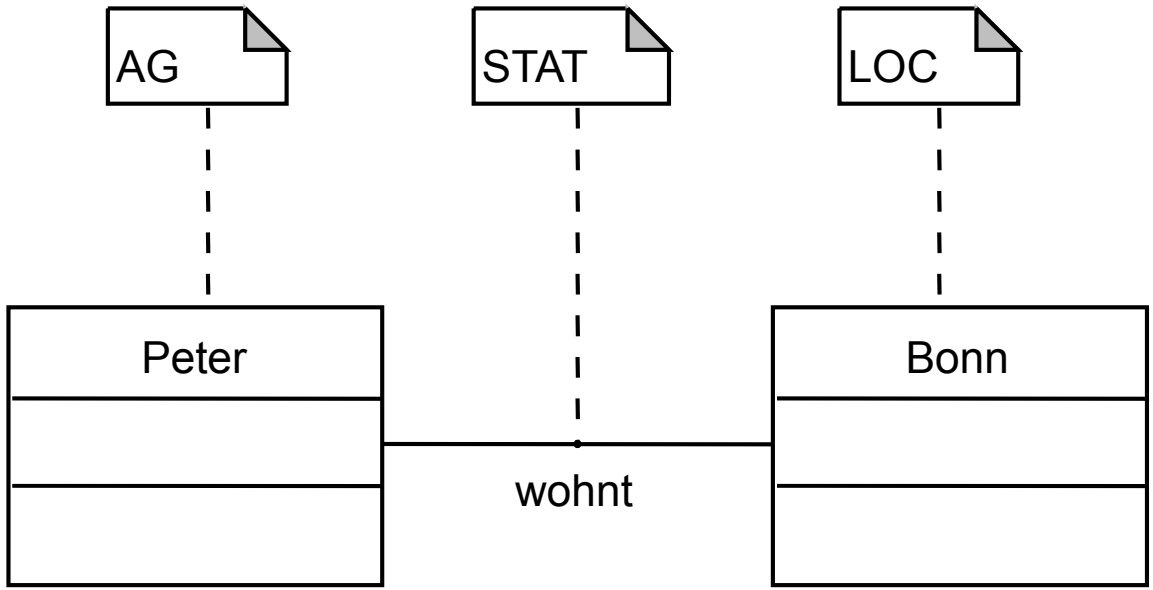
- „Peter_{AG} streicht_{ACT} die Wand_{PAT} mit einem Pinsel_{INSTR}.“
- Wir hätten auch schreiben können: „Die Wand_{PAT} wird von Peter_{AG} gestrichen_{ACT}, und zwar mittels eines Pinsels_{INSTR}.“



(Gelhausen2007)

Abbildung von Zustandsverben

- „Peter_{AG} wohnt_{STAT} in Bonn_{LOC}.“



Beispiel für eine einfache Relation.

(Gelhausen2007)

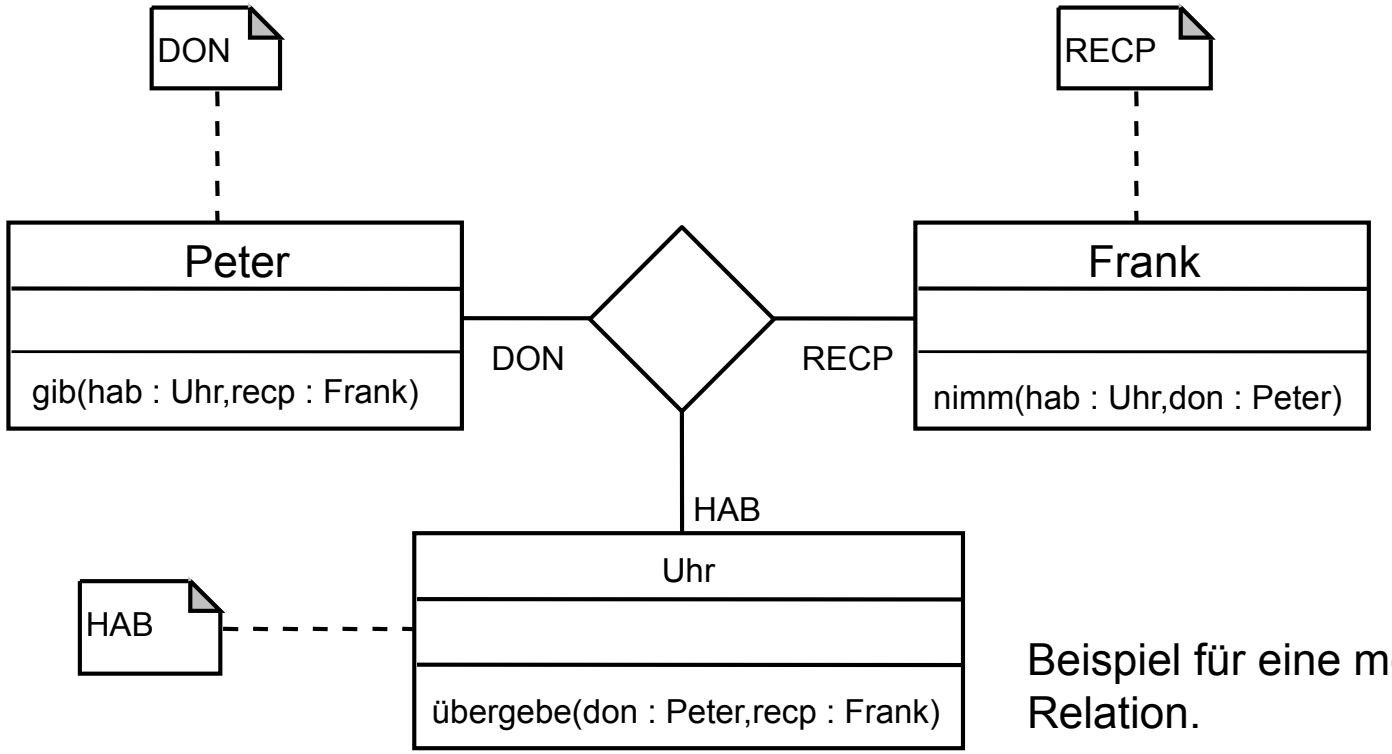
Thematische Rollen

- Einige für SW-Ingenieure interessante Rollen:
  - Donor (DON) – der, der etwas gibt
  - Recipient (RECP) – der Empfänger einer Sache.
  - Habitum (HAB) – etwas, das gegeben oder „gehabt“ wird.
    $\text{Peter}_{\text{DON}}$ schenkt $\text{Frank}_{\text{RECP}}$ eine Uhr_{HAB}.

(Gelhausen2007)

Beispiele für Abbildungen: Beziehungen

- „Peter$_{DON}$ schenkt Frank$_{RECP}$ eine Uhr$_{HAB}$.“



Beispiel für eine mehrstellige Relation.

(Gelhausen2007)

Etwas komplexeres Beispiel
Whois Protokoll

- Eine Spezifikation

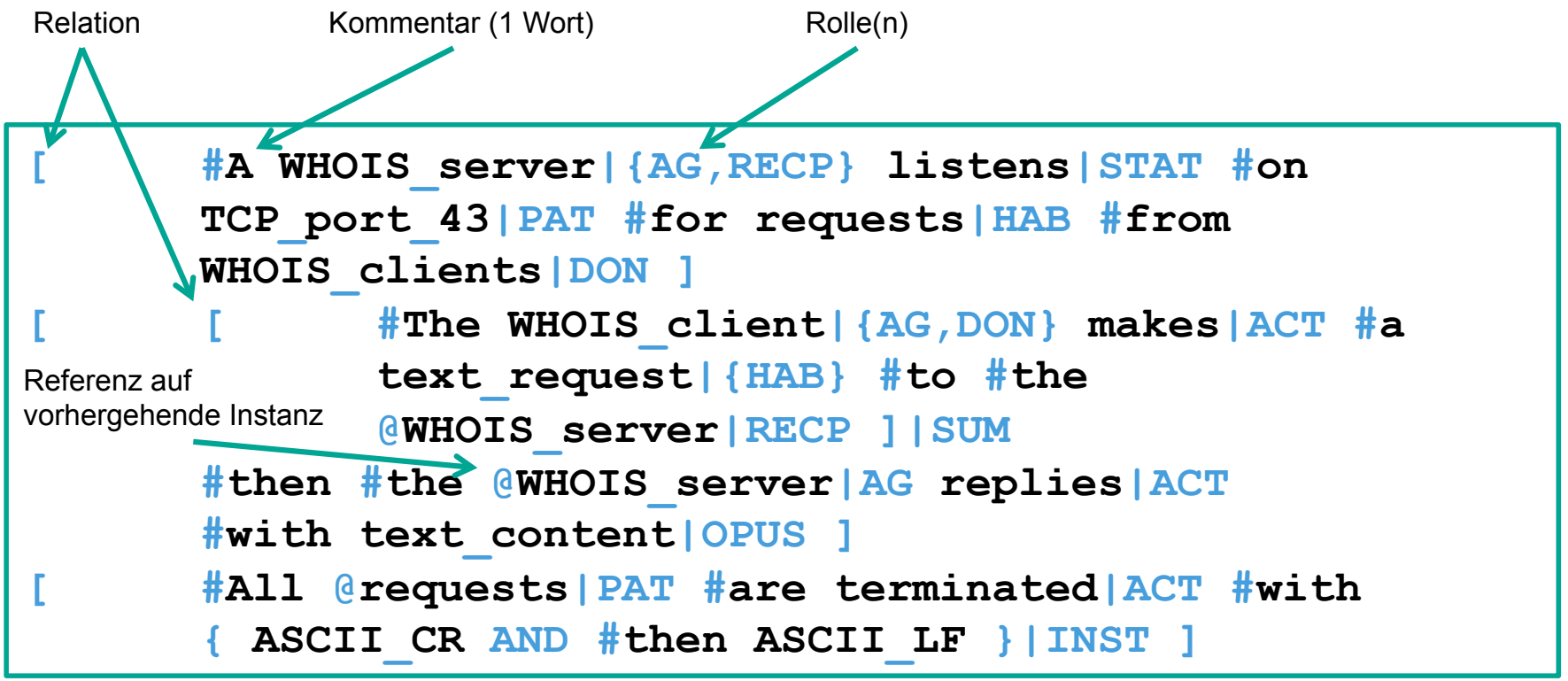
```
 A WHOIS server          listens       on
TCP port 43      for requests      from
WHOIS clients
        The WHOIS client          makes      a
       text request        to  the
        WHOIS server
 then  the  WHOIS server    replies
 with text content
 All  requests      are terminated      with
  ASCII CR      then ASCII LF
```

Quelle: IETF WHOIS Protocol Specification, RFC 3912, Ch. 2, “Protocol Specification”

(Gelhausen2007)

Etwas komplexeres Beispiel
Whois Protokoll annotiert

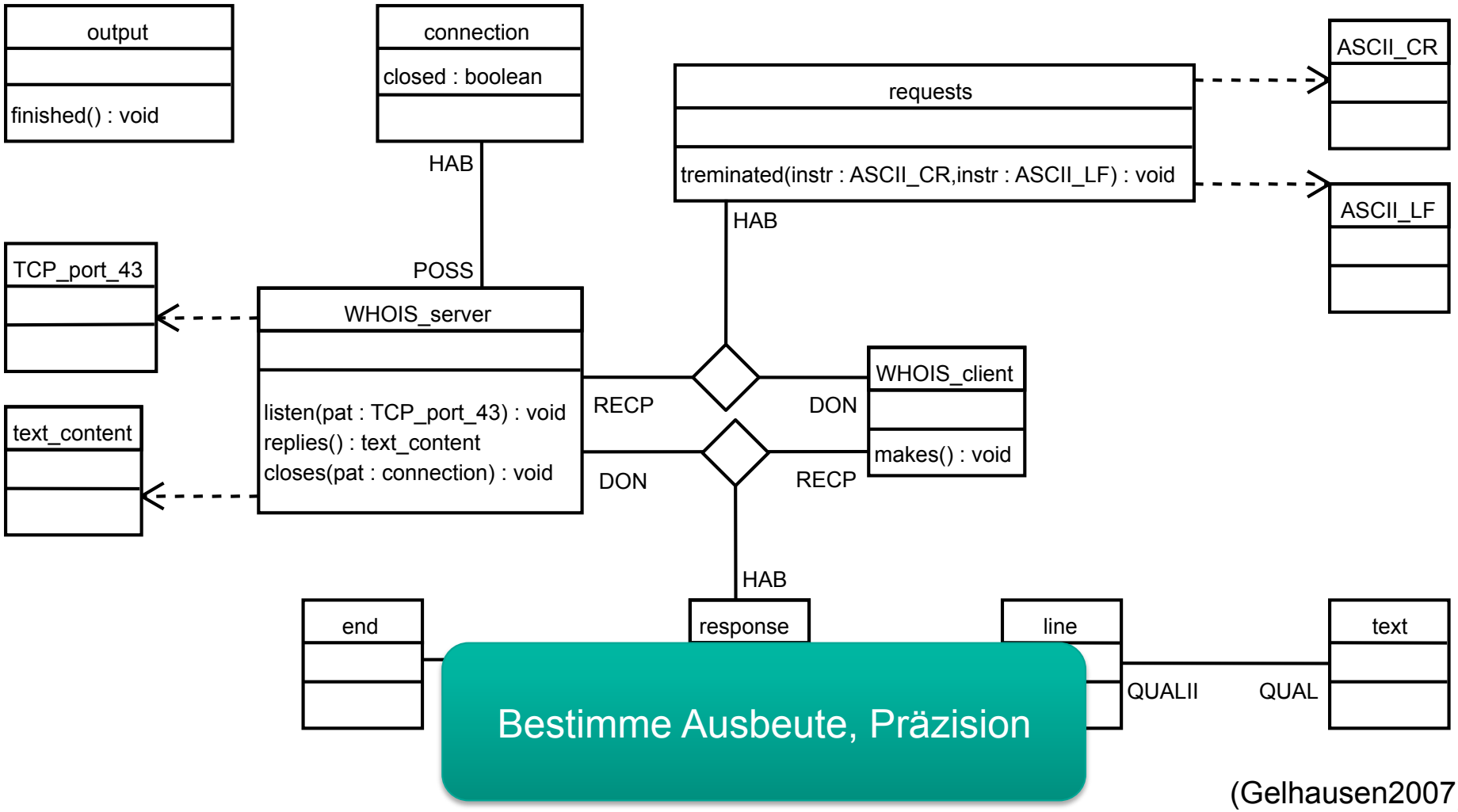
- Annotationssprache SAL_E



Quelle: IETF WHOIS Protocol Specification, RFC 3912, Ch. 2, “Protocol Specification”

(Gelhausen2007)

Whois Protokoll: Übersetzung in UML mit SAL_Emx



(Gelhausen2007)

Beispiel: Wettlaufdetektion

- Datenwettläufe sind unsynchronisierte Zugriffe auf gemeinsame Variablen
- Schwierig zu finden.
- Heutige Wettlaufdetektoren sind unbrauchbar
  - Erzeugen tausende falscher Warnungen.
  - Überwältigen Programmierer damit.
- Warum falsche Positive?
  - Ad-hoc Synchronisation (Warteschleifen)
  - Unbekannte Synchronisations-Bibliotheken
- Beitrag: wie man ad-hoc Synchronisation erkennt und unbekannte Synchroniationsbibliotheken behandelt, und damit falsche Positive drastisch reduziert.

Was ist ein Datenwettlauf?

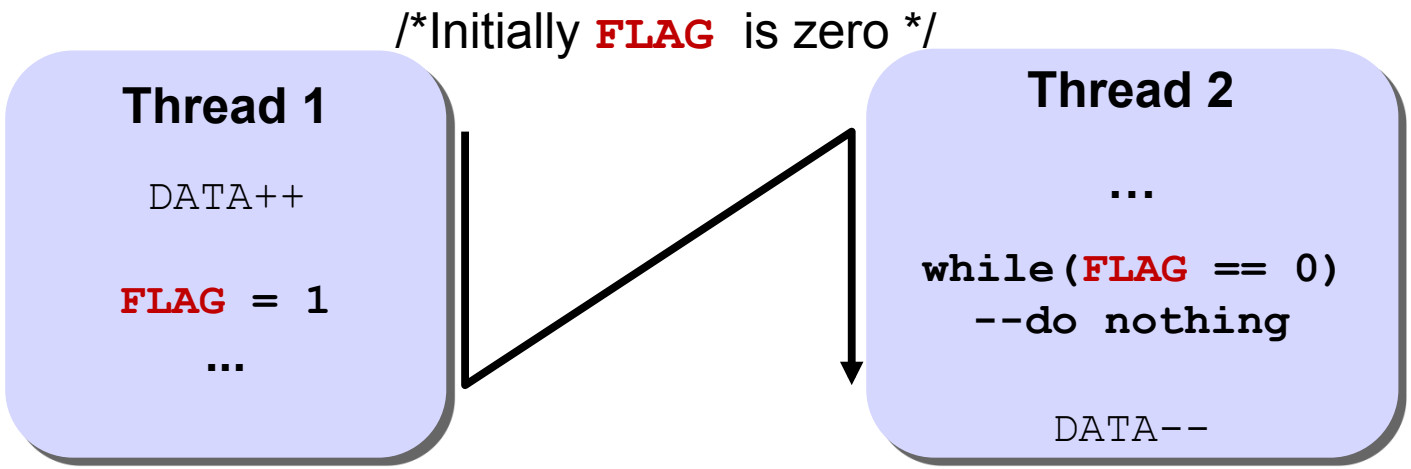
- Zwei oder mehr Zugriffe auf die selbe Variable, wobei mindestens einer schreibt..

Faden 1	Faden 2
X = 0	
X++	T = X

T=0 or T=1?

Ad-hoc (User-defined) Synchronization

- Synchronization constructs implemented by user for performance reasons
  - High level synchronizations (e.g. task queues)
  - Spinning read loop instead of a library wait operation



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

Test Suite – data-race-test

- 120 Testfälle (2-16 Fäden)
  - Programme mit und ohne Wettläufe (geschrieben mit Pthread)
    - Schwierige Fälle dabei
  - Warteschleifendetektion (spin)
    - Enfernt 24 falsche Positive und ein falsches Negatives
  - Ohne Information über Pthread Bibliothek (nolib)
    - Nur ein falsche Warnung zusätzlich

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

Ergebnisse

Program	Para. model	LOC	Racy Contexts			
			Helgrind+ lib	Helgrind+ lib+spin	Helgrind+ nolib+spin	DRD
			0	**0**	0	0
			0	**0**	0	0
			0	**0**	0	0
canneal	POSIX	29,31	0	**0**	0	0
freqmine	OpenMP	10,279	153.4	**2**	2	1000
vips	GLIB	1,255	50.8	**0**	0	858.6
bodytrack	POSIX	9,735	36.8	**3.6**	32.4	34.6
facesim	POSIX	1,391	113.8	**0**	0	1000
ferret	POSIX	2,706	111	**2**	47	214.6
x264	POSIX	1,494	1000	**19**	28	1000
dedup	POSIX	3,228	1000	**0**	2	0
streamcluster	POSIX	40,393	4	**0**	1	1000
raytrace	POSIX	13,302	106,4	**0**	0	1000

Happens-before detector
false positives are slightly increased in 4 cases

Beispiel: Auto-Parallelisierungs-Benchmark

- Um zu zeigen, dass automatische Parallelisierungstechniken funktionieren, stellen wir einen Benchmark zusammen.
- Besteht aus Paaren: seq. Implementierung und hand-parallelisierte Version (oder Versionen).
- Daran kann man z.B. das Konzept der Autofutures oder andere Parallelisierungsmuster testen.
  - Wurden alle Parallelisierungsmöglichkeiten gefunden?
  - Wurden sie richtig umgesetzt?
  - Wie schnell läuft die Parallelisierung?

Zusammenfassung

- Der Gebrauch von Benchmarks in der Softwaretechnik könnte deutlich höher sein (ist aber nicht die einzige Evaluierungstechnik).
- Mit realistischen Benchmarks bekäme man zuverlässige und vergleichbare Ergebnisse.
- Benchmarks beschleunigen den Fortschritt: sie sondern die schlechteren Ansätze aus, lenken die Aufmerksamkeit auf das Wichtige.
- Wenn die Werkzeuge ausgereift sind, dann erst überprüfe Brauchbarkeit mit Menschen (dem teuren Experiment).
- Literatur: Susan Elliott Sim, Steve Easterbrook, and Richard C. Holt. Using Benchmarking to Advance Research: A Challenge to Software Engineering, Proceedings of the Twenty-fifth International Conference on Software Engineering, Portland, Oregon, pp. 74-83, 3-10 May, 2003.

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

Vince Lombardi
Berühmter US Football Trainer

Barcelona gegen Manchester United: Wer spielt besser?