

Vorlesung Fortgeschrittene Softwaretechnik

Wintersemester 2024/25

Prof. Dr. Stephan Diehl


Informatik

Universität Trier



Plan für die nächsten Wochen

Themenblöcke bisher: Testen, CI, Patterns, VR/AR+SE

	Datum	Thema/Inhalt	Übung			Dozent	Block
DO	12.12.2024	Empirische SE + Software Evolution				SD	Quantitative Studien
DI	17.12.2024	MSR, Empfehlungsdienste	Praxis BOA		Ausgabe LIT	SD	
DO	19.12.2024	???					
	FREI						
DI	07.01.2025	RG MSR/BOA	Übung BOA			SD	
DO	09.01.2025	Research Design + Quantitative Analyse				SD	
DI	14.01.2025	Qualitative Analyse	Praxis: QA 1		Ausgabe LIT	SD	Qualitative Studien
DO	16.01.2025	Praxis: QA 2 (gemeinsames Kodieren)				SD	
DI	21.01.2025	RG QA+SE	Übung QA			SD	
DO	23.01.2025	Info zu Portfolio				SD	
DI	28.01.2025	LLM + SE	Übung LLMSE		Ausgabe LIT	SD	LLM+SE
DO	30.01.2025	LLM + SE				SD	
DI	04.02.2025	RG: LLM+SE	Übung LLMSE			SD	
DO	06.02.2025	-----				SD	
DI	11.02.2025	Abgabe Expose				SD	
DO	13.02.2025	-----				SD	

Vorbereitung für praktischen Teil

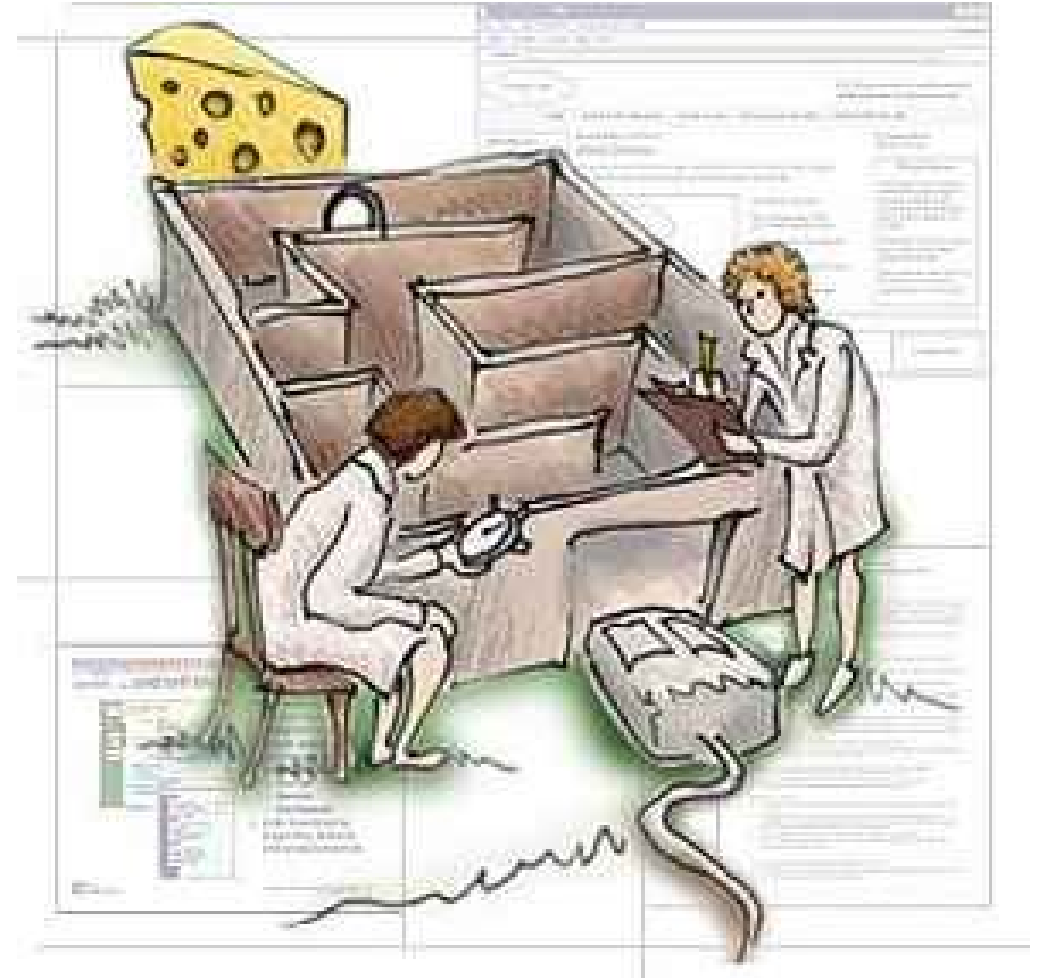
Beantragen Sie einen Benutzerzugang auf folgender Webseite. Geben Sie bitte Ihre Uni-Emailadresse an!

- [**https://boa.cs.iastate.edu/request/**](https://boa.cs.iastate.edu/request/)



Heute

- Empirische Softwaretechnik
- Forschungsmethoden





Realität

Phänomen

Theorie

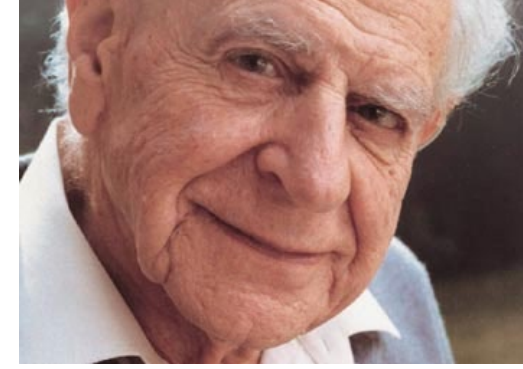
Wissenschaft

Terminologie

- **Empirie** (griech. „empeiria“ = die Erfahrung)
 - **Erfahrung** aufgrund **systematisch** in der Realität durchgeführter Beobachtungen, Versuche oder Befragungen
- **Phänomen** (griech. „phainomenon“ = Sichtbares, Erscheinung)
 - **mit den Sinnen wahrnehmbar** (im Gegensatz zu „Begriffen“=dem Gedachten)
- **Theorie** (griech. „theorein“ = „betrachten, schauen“)
 - geht zurück auf visuelle Beweise der Pythagoräer
 - Betrachtung der Wahrheit durch reines Denken
 - **bildet ein Modell der Realität**
 - bildet neue Erkenntnisse durch logische Schlussfolgerungen
- **Dogma** (griech. „dogma“ = Meinung, Lehrsatz)
 - Aussage, die von einer Gruppe von Menschen als grundlegend und **nicht verhandelbar** („heilig“) angesehen wird; nicht offen für neue Erkenntnisse



Wissenschaftstheorie (Karl Popper)



- Wissenschaftlicher Fortschritt geschieht dadurch, dass bestimmte Theorien durch Experimente widerlegt ("**falsifiziert**") werden.
- In einem evolutionsartigen Selektionsprozess setzen sich diejenigen Theorien durch, die "**wahrheitsnäher**" sind. Sicheres Wissen kann dabei allerdings nie erreicht werden; **alles Wissen ist vorläufig**.
- So können wir zwar nicht sicher wissen, ob eine Theorie wahr ist, aber sehr wohl, dass eine bestimmte Theorie falsch ist: nämlich wenn eine gegenteilige Beobachtung sie widerlegt.

Was ist Falsifikation ?



= Widerlegung von Hypothesen oder Theorien durch empirische Aussagen (Beobachtung, Experiment)

- Karl Popper:

- **Universelle Hypothesen sind falsifizierbar, aber nicht verifizierbar.**
- „Alle Schwäne sind weiß“ kann als vorläufige Hypothese akzeptiert werden, bis der erste nicht-weiße Schwan beobachtet wird.
- Je länger eine Hypothese Falsifikationsversuchen widersteht, als desto belastbarer wird sie angesehen.
- Beispiel: Newtons Theorie wurde falsifiziert; Einsteins Relativitätstheorie noch nicht

Die große Tragödie in der Wissenschaft ist, dass die schönsten Hypothesen von hässlichen Fakten erschlagen werden. [Thomas H. Huxley, 1825-1895]

Falsifikation ist in der heutigen Wissenschaftstheorie umstritten

- TED-Talk von Naomi Oreskes: „What makes science trustworthy?“

https://www.ted.com/talks/naomi_oreskes_why_we_should_trust_scientists/transcript

➔ Organized scepticism
Collective distrust

- “Evidence keeps you modest because you predict something, you test it, and the evidence sometimes shows you’re wrong. **Right now you have many celebrated scientists doing mathematical gymnastics about lots of untestable things:** string theory, the multiverse, even the theory of cosmic inflation. Once, in a public forum, I asked [physicist] Alan Guth, who originated the theory, “Is inflation falsifiable?” And he said it’s a silly question, because for whatever cosmological data an experiment gives us, a model of inflation can be found that accommodates it. **And therefore, inflation is in a very strong position because it can explain anything! But I see this as a very weak position because a theory of everything is sometimes a theory of nothing. There may be no difference between the two.”**




Bildquelle: Wikipedia

Quelle: Interview mit Astronom Avi Loeb, American Scientist, Februar 2021

<https://www.scientificamerican.com/article/astronomer-avi-loeb-says-aliens-have-visited-and-hes-not-kidding1/>

Vom Phänomen zur Theorie

„Die wissenschaftliche Methode“

- 
- Phänomen wird beobachtet (← Empirie)
 - Theorie wird aufgestellt, um das Phänomen zu erklären („Verstehen“)
 - Messungen und Experimente bestätigen oder verwerfen die Theorie (← Empirie)
 - Theorie wird angepasst

Empirische Softwaretechnik

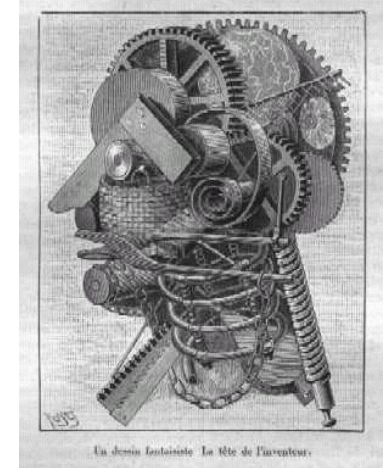
- **erkundet** Phänomene bei Erstellung und Einsatz von Software
- **bewertet** Werkzeuge und Methoden zur Software-Erstellung
- **testet** Theorien über Software und ihre Erstellung
- **bewertet** Eigenschaften von Software

Typische Fragestellungen

- Steigern die Techniken von XP (extreme programming) die Zuverlässigkeit von Programmen?
- Hängt der Wartungsaufwand für ein OO-Programm von der Vererbungstiefe ab?



Stand der Technik in der Softwaretechnik



Technik

- stärkster Teil der Softwaretechnik
- viele Werkzeuge, Programmierparadigmen, Entwicklungsmethoden

Empirie

- viel Forschung zu Softwaremetriken
- viele Fallstudien
- in jüngster Zeit
 - Mehr systematische Experimente
 - Wichtige empirische Erkenntnisse
 - Verbesserte Methodik

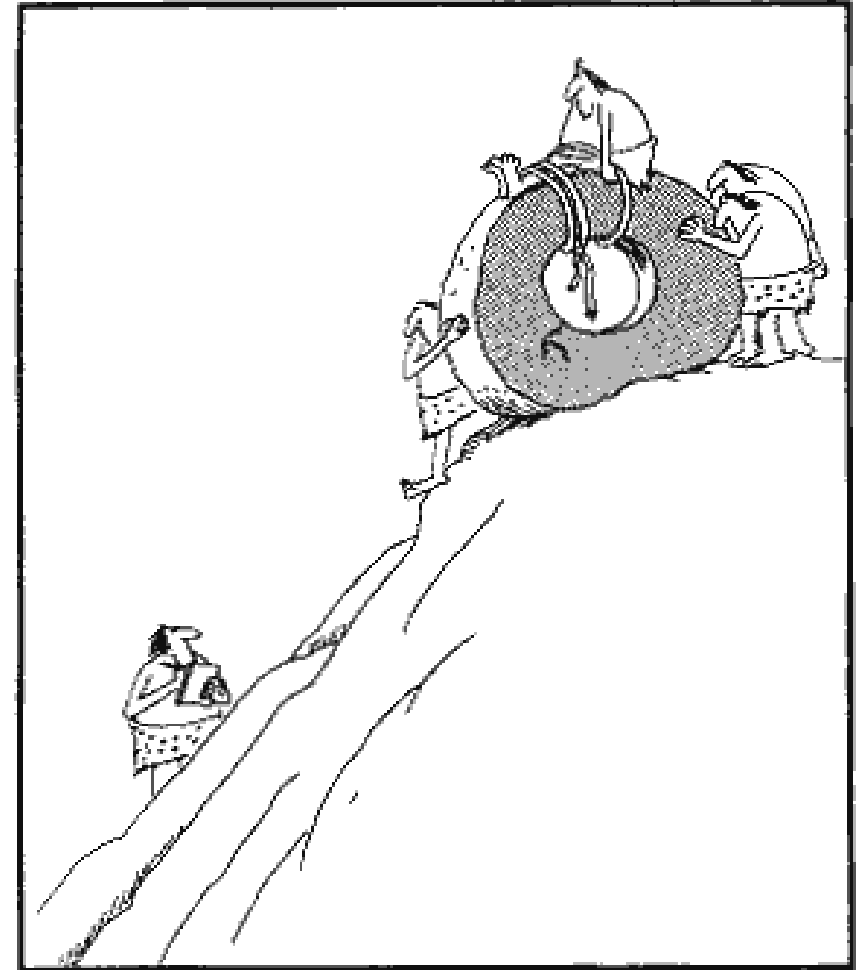
Theorie

- Gering entwickelt und meist nur qualitativ
- Formale Softwareprozessmodelle
- Erste ökonomische Modelle

Empirische Forschungsmethoden

- Fallstudie
- Feldexperiment
- Kontrolliertes Experiment
- Umfrage
- Metastudie

Illustriert an Beispielen zu TDD



Early experiments in transportation

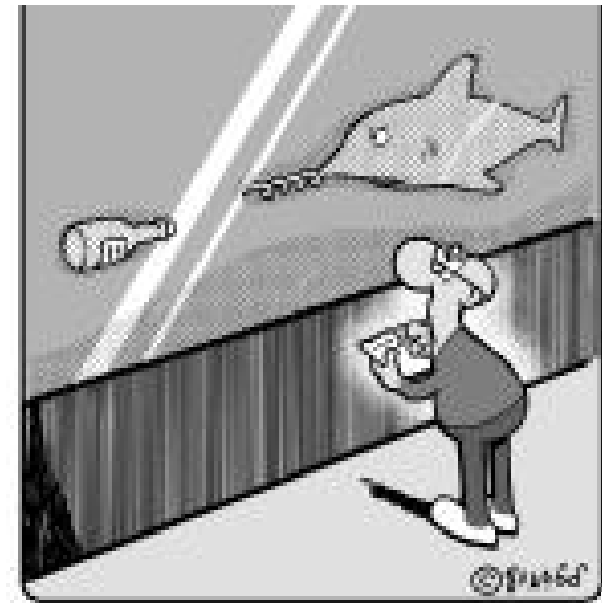
Fallstudie

- Genaue Beschreibung und Analyse:
 - **eines** Vorganges
 - **einer** Organisation
 - **eines** Ereignisses
- Nutzt verschiedene Informationsquellen:
 - Interviews
 - Dokumente
 - Messungen
- Anwendung
 - Illustration eines Werkzeuges (→ besser: Demonstration Study)
 - Machbarkeitsstudie
 - Abschätzung der Effizienz einer Technik
- Pro: Relativ einfach durchzuführen
- Contra: Ergebnisse schwierig zu verallgemeinern



Experiment

- Lat. „experimentum“ = Versuch, Beweis, Prüfung, Probe
 - **Unabhängige Variablen** werden variiert
 - Beispiel: Programmiermethodik
 - **Störvariablen** werden konstant gehalten oder ihre Auswirkungen neutralisiert
 - Beispiel: Fähigkeiten der Programmierer
 - **Abhängige Variablen** werden beobachtet und gemessen.
 - Dauer, Kosten der Entwicklung, Qualität der Software
- ➔ Die Wirkung der unabhängigen Variablen auf die abhängigen Variablen wird untersucht.

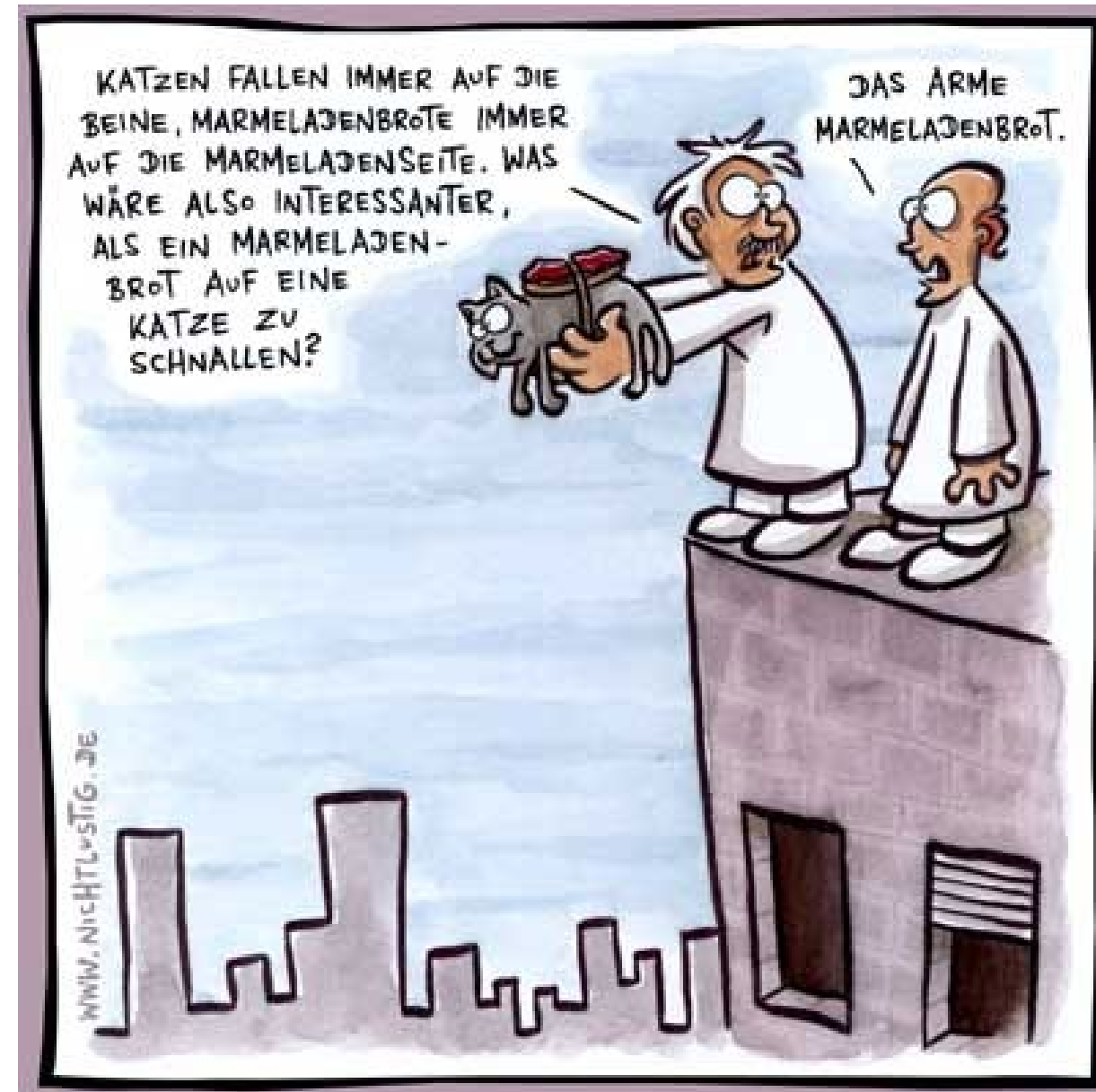


Kontrolliertes Experiment

- Im wissenschaftlichen Experiment wird durch planmäßiges Beobachten eines Sachverhaltes und dessen Veränderung unter **kontrollierten, wiederholbaren Bedingungen** eine Hypothese bestätigt oder widerlegt.
- In anderen Worten:
 - Planmäßige Manipulation der unabhängigen Variablen
 - Objektive Beobachtung der abhängigen Variablen
 - **Störvariablen werden kontrolliert**, d.h. konstant gehalten oder in ihrer Wirkung neutralisiert.
 - **Kausalität (Ursache-Wirkung-Beziehung) ist beobachtbar, aber nicht bewiesen!**
 - Wiederholbarkeit (dadurch werden Beobachtungen überprüfbar)

Laborexperiment

- Vom Menschen geschaffene, künstliche Versuchsanordnung
- Findet nicht in der Natur bzw. im realen Umfeld statt.
- Experimentator kann in das Experiment eingreifen.



TFP-Laborexperiment:

Beispiel

- „Experiment about test-first programming“,
Müller and Hagner, IEE Proceedings - Software, 149(5):131-136, 2002
- Untersucht den Einfluss von „Test-First“ auf die Entwicklungsdauer und Korrektheit von Programmen.
- Teilnehmer (Subjekte):
 - Studenten eines XP Praktikums
 - Programmiererfahrung reicht von Anfänger bis Profi
- Aufgabe:
 - Entwicklung der Hauptklasse einer Graphenbibliothek in Java
 - Gerichtete/ungerichtete Kanten
 - Gewichtete/ungewichtete Kanten
 - Operationen zum Hinzufügen/Löschen von Knoten und Kanten
 - Methodensignaturen vorgegeben

- Unabhängige Variablen
 - Testtechnik: test-first vs. „beliebig“
- Abhängige Variablen
 - Entwicklungsdauer und Programmkorrektheit (gemessen mit Akzeptanztest aus 20 Testfällen)
- Gruppen:
 - Experimentgruppe entwickelt mit test-first mit JUnit
 - Kontrollgruppe testet nach Belieben mit JUnit
 - Teilnehmer zufällig den Gruppen zugeordnet („randomisiert“)

- Ergebnis
 - Experimentgruppe (test-first) braucht etwas länger und die Programmkorrektheit ist deutlich schlechter
- Gründe
 - Experimentgruppe hat zu wenig oder zu einseitig getestet; möglicherweise falsches Gefühl der Sicherheit (Fragebögen).
- Mögliche Schwachpunkte des Experiments
 - Studenten, keine Profis
 - zu wenige Probanden
 - Technik war noch zu neu (Test-First noch nicht „in Fleisch und Blut übergegangen“)
 - keine direkte Überprüfung während des Experiments, ob wirklich gemäß Test-First entwickelt wurde
 - zu enge Aufgabenstellung

Feldexperiment

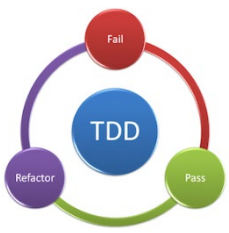
- Wird **in realer Umgebung** durchgeführt (ohne oder nur mit wenig Änderung der Umgebung)
- Anwendung
 - Umgebung kann „im Labor“ nicht realistisch nachgestellt werden
 - Benötigte Datenmenge kann nur in der Praxis gesammelt werden
 - Beobachtung muss über längeren Zeitraum erfolgen. (→ longitudinal study)
- Pro: realistische Ergebnisse
- Contra:
 - Kontext/Störvariablen oft schwer zu erfassen
 - oft hoher Aufwand
 - Erfordert Unterstützung durch Management

Feldexperiment: Beispiel

- Software-Archäologie
 - Eingriffsfreies Feldexperiment
 - Alle Beobachtungen erfolgen erst **im Nachhinein**
 - Basiert auf Daten, die das Projekt sowieso gesammelt hat (Versionsdatenbank, Fehlermeldungen, Mails, Bearbeitungszeiten, etc.)
 - Qualität der Daten oft schwer einzuschätzen
 - Oft fehlen Daten oder Zusatzinformationen



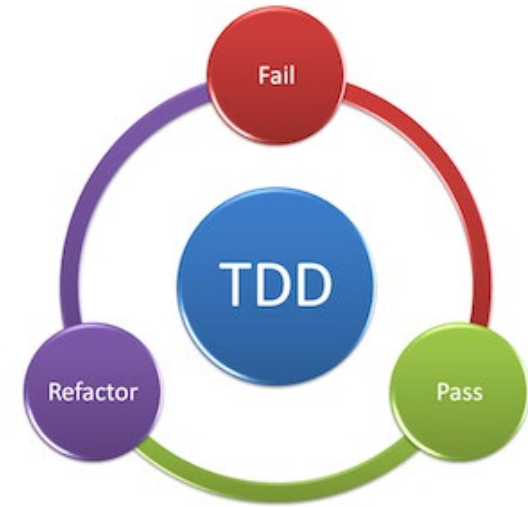
Bildquelle: ChatGPT

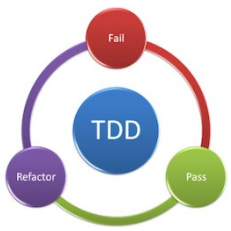


TDD-Feldexperiment

Beispiel

- Test-Driven Development (TDD)
 - Zuerst die Testfälle schreiben, dann implementieren („test-first“)
 - Häufiges, automatisches Ausführen aller Testfälle (JUnit)
 - Wichtige Technik bei XP
- Studie:
 - „Assessing Test-Driven Development at IBM“, Maximilien & Williams, ICSE 2003
 - Untersucht, wie sich TDD auf die Fehlerdichte in einem realen Projekt bei IBM auswirkt.

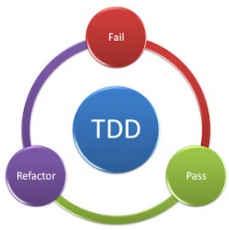




TDD-Feldexperiment

Beispiel

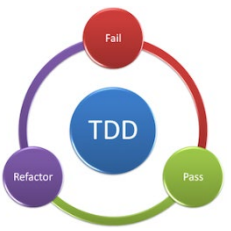
- Produkt JavaPOS (Java for Point of Sale)
 - Bibliothek von Java Beans zum Steuern von Geräten am Verkaufsort
- Bisher:
 - Versionen hatten zu hohe Fehlerdichte
 - Für jede Version existiert ein abschließender „functional verification test“ (FVT)
- Daher:
 - JavaPOS wurde von einem **neuen Team mit TDD** komplett neu entwickelt
- Später:
 - „alte“ Entwickler entwickeln aus der alten Version eine mit der Neuentwicklung funktional vergleichbare Version, aber ohne TDD.



Beispiel

TDD-Feldexperiment:

- Erhoffte Vorteile von TDD (Hypothesen)
 - **Niedrige Fehlerdichte** durch früheres und häufigeres Testen
 - Verkürzte Implementierungszyklen durch Automatisieren der Testläufe
 - Verbesserte Code-Integration durchlaufende Regressionstests
 - Höhere Testqualität durch Ansammeln vieler Testfälle
- Ergebnis:
 - Überarbeitete Software des „alten“ Teams
 - Tatsächlich: 7,0 Fehler/KLOC
 - Software des „neuen“ Teams
 - Tatsächlich: 3,7 Fehler/KLOC



TDD-Feldexperiment:

Beispiel

- „alte“ Entwickler
 - Viel Erfahrung mit früheren Versionen (Spezifikation und Code)
 - Lange Erfahrung mit Java
- „neue“ Entwickler
 - 7 von 9 ohne Erfahrung mit Spezifikation oder früheren Versionen
 - Einige hatten wenig Erfahrung mit Java
 - Junges, enthusiastisches Team
- Autoren führen Verringerung der Fehlerdichte allein auf TDD und ignorieren
 - unterschiedlichen Projektumfang (Neuentwicklung vs. Erweiterung)
 - Teams mit unterschiedlichen Vorkenntnissen
- → Kausaler Zusammenhang ist nicht schlüssig nachgewiesen!

Umfrage

- sammelt Informationen durch Fragen an Repräsentanten einer bestimmten Zielgruppe
- gibt Einblick in den momentanen Zustand der Zielgruppe
- Repräsentanten vertreten die Zielgruppe durch entsprechende Merkmale, Verhaltensweisen und Einstellungen
- Beispiel:

Wie finden Sie die Vorlesung Softwaretechnik?

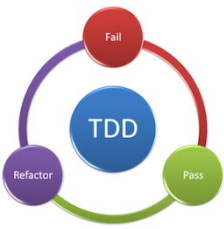
☆ ☆ ☆ ☆	noch keine Bewertung
☆ ☆ ☆ ☆	Sehr schlecht (keine Sterne)
★ ☆ ☆ ☆	Eher schlecht (1 Stern)
★ ★ ☆ ☆	Mäßig (2 Sterne)
★ ★ ★ ☆	Eher gut (3 Sterne)
★ ★ ★ ★	Sehr gut (4 Sterne)



Umfrage

- Fragen können sich auf subjektive oder objektive Sachverhalte beziehen
- Fragen werden schriftlich (Fragebogen) oder mündlich (Interview) gestellt
- Antworten sind immer subjektiv und nur begrenzt überprüfbar.
 - Beispiel: „Putzen Sie regelmäßig die Zähne?“ (besser: „Haben Sie heute morgen die Zähne geputzt?“)
 - „Was war der Schwierigkeitsgrad der Aufgaben?“
- Pro: einfach und relativ billig
- Contra:
 - Direkter Kontakt mit Zielgruppe notwendig
 - Verlässlichkeit bzw. Interpretation der Ergebnisse wg. Subjektivität schwierig





Beispiel

TDD-Umfrage:

- “Most Common Mistakes in Test-Driven Development Practice: Results from an Online Survey with Developers”, Aniche and Gerosa, Third International Conference on Software Testing, Verification, and Validation Workshops, 2010
- Online-Umfrage: 218 Programmierer

Mehrheit!

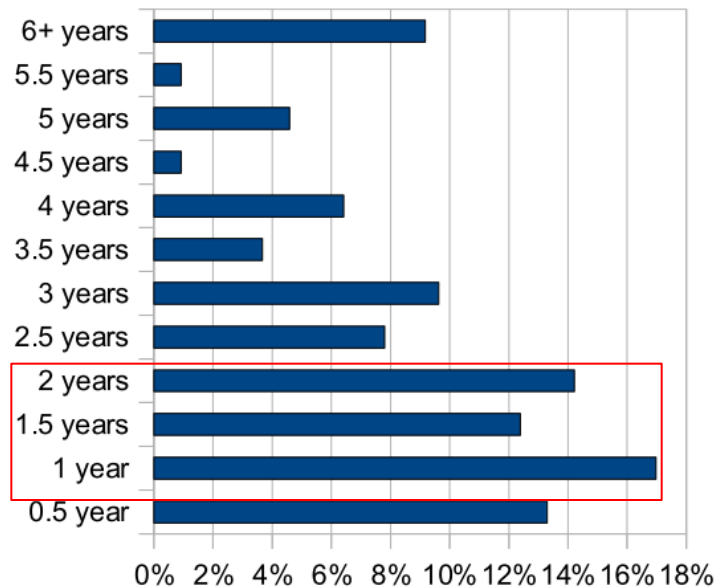


Figure 1. Programmers' experience in TDD

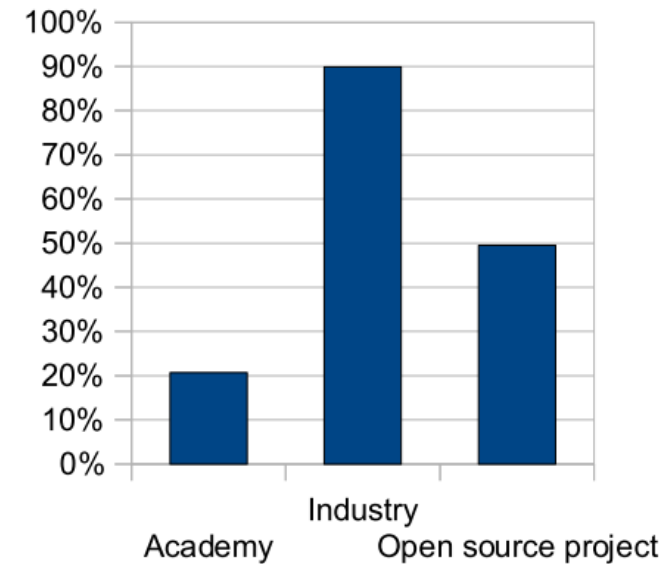
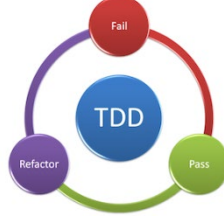


Figure 2. Where programmers practice TDD



TDD-Umfrage:

Beispiel

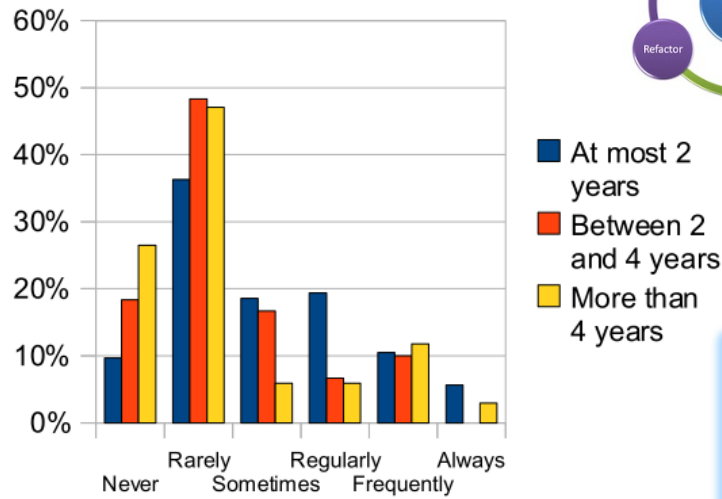


Figure 3. How often programmers forget to watch the test fail

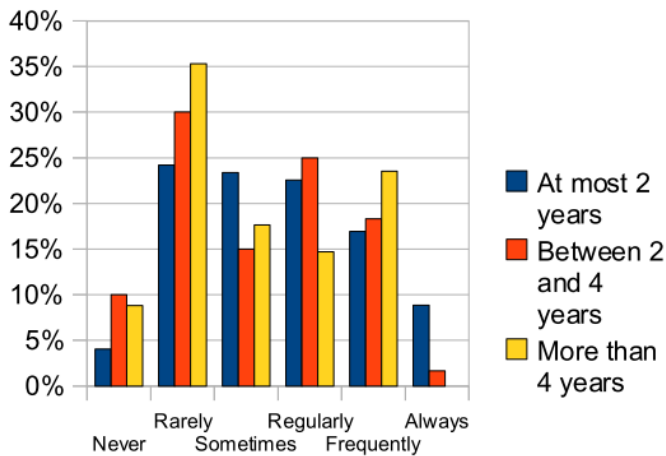


Figure 5. How often programmers refactor some other piece of code while working on a test

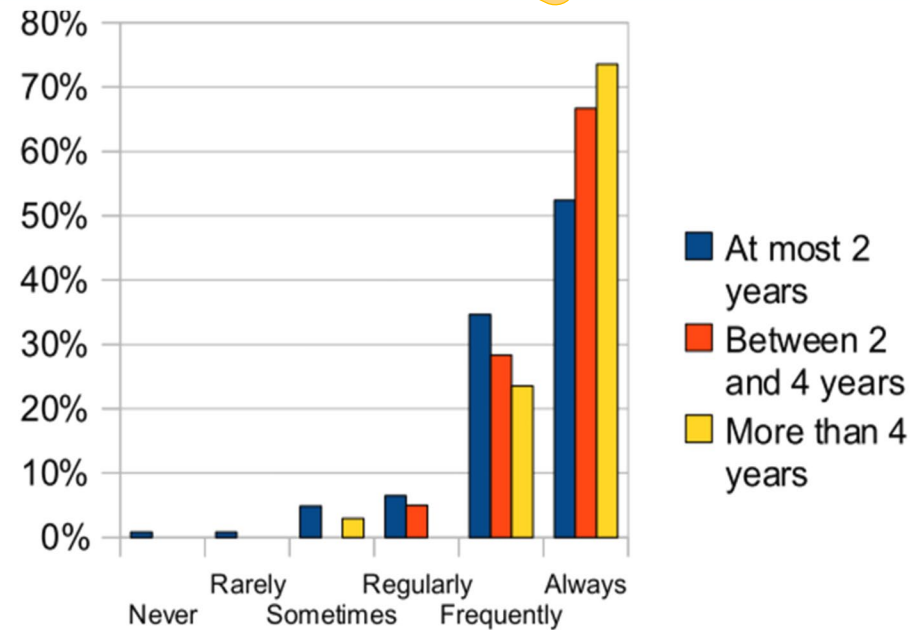


Figure 12. Programmers' opinion about TDD reducing defect density

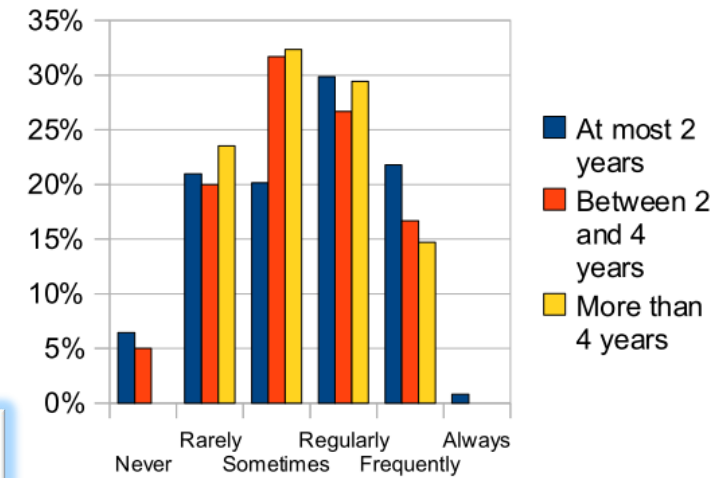


Figure 4. How often programmers forget the refactoring step

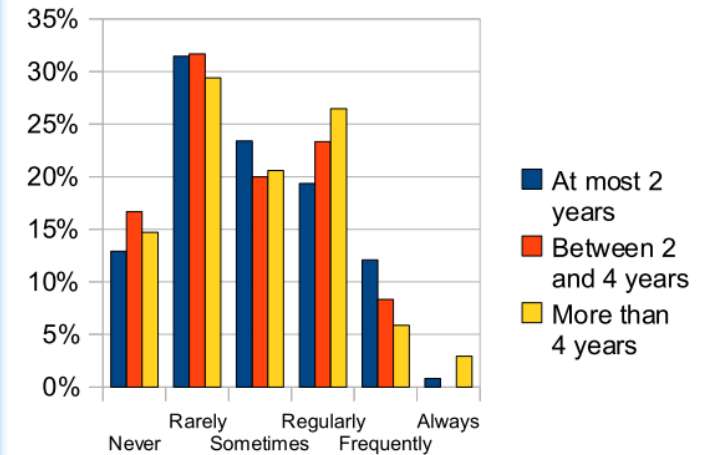


Figure 6. How often programmers write bad test names

Metastudie

- „**Studie über Studien**“
- Auswertung mehrerer bereits vorhandener Studien zu einem Thema
- Material ist Forschungsliteratur entnommen
- fasst nicht nur zusammen, sondern **vergleicht und analysiert** (im Gegensatz zu Überblicksartikel)
- bietet Orientierung und konsolidiert Wissen:
 - bestätigen sich Ergebnisse gegenseitig?
 - ergänzen sich Ergebnisse?
 - zu welchen Aspekten liegen noch keine Ergebnisse vor?
 - welche Ergebnisse widersprechen sich?
- Beispiel: *Es gibt zahlreiche Untersuchungen zu Paarprogrammierung. Diese sind teilweise widersprüchlich. In den meisten Fällen sind die Teilnehmer Studierende. Meistens haben die Teilnehmer Paarprogrammierung erst kürzlich erlernt und nur mit wenigen Partnern geübt. Langzeitstudien zu dem Thema gibt es nicht.*

Metastudie

- Pro:
 - vergleichsweise geringer Aufwand
 - Zeigt Ansatzpunkte für weitere Forschung auf
- Contra:
 - Zugrunde liegende empirische Studien müssen existieren
 - Kann Lücken und Mängel in den vorhandenen Studien nicht mehr ausgleichen

Spezialfall: **Metaanalyse**

- gemeinsame statistische Analyse verschiedener Studien (in der Regel Experimente)
- Ziel: neue quantitative Aussage
- Gibt es einen signifikanten Effekt, wenn man die Einzelergebnisse kombiniert?
- Wie groß ist der Effekt, wenn man die einzelnen Effektgrößen kombiniert?

Metastudie zu Pairprogramming

- J.E. Hannay, T. Dybå, E. Arisholm, D.I.K. Sjøberg, **The Effectiveness of Pair Programming: A Meta-Analysis**, Information and Software Technology (2009), doi: 10.1016/j.infsof.2009.02.001
- 18 Studien
- Die Autoren stellen fest, dass die Varianz zwischen den Studien hoch ist.
- Widersprüchliches Ergebnis:
 - Pair programming ist schneller für einfache Aufgabenstellungen (jedoch mit auffällig schlechterer Codequalität) und liefert besseren Code (jedoch mit deutlich höherem Aufwand).

Table 1
Characteristics of the Included Studies.

Study	Subjects	N _{Tot}	N _{Pair}	N _{Ind}	Study setting
Arisholm <i>et al.</i> (2007) ¹	Professionals	295	98	99	10 sessions with individuals over 3 months and 17 sessions with pairs <u>over 5 months (each of 1 day duration, with different subjects)</u> . Modified 2 systems of about 200-300 Java LOC each.
*Baheti <i>et al.</i> (2002) ²	Students	134	16	9	Teams had 5 weeks to complete a curricular OO programming project. Distinct projects per team.
Canfora <i>et al.</i> (2005) ³	Students	24	12	24	2 applications each with 2 tasks (run1 and run2).
Canfora <i>et al.</i> (2007) ⁴	Professionals	18	5(4)	8(10)	Study session and 2 runs (totalling 390 minutes) involving 4 maintenance tasks (grouped in 2 assignments) to modify design documents (use case and class diagrams).
Domino <i>et al.</i> (2007)	Professionals Students	88	28	32	Run as several sessions during a period of two months. Pseudo-code on "Create-Design" tasks Test-driven development.
*Heiberg <i>et al.</i> (2003) ⁵	Students	84(66)	23(16)	19(17)	4 sessions over 4 weeks involving 2 programming tasks to implement a component for a larger "gamer" system.
Madeyski (2006) ⁶	Students	188	28	31(35)	8 laboratory sessions involving 1 initial programming task in a finance accounting system (27 user stories).
Madeyski (2007)	Students	98	35	28	Java course project of developing a 27 user story accounting system over 8 laboratory sessions of 90 minutes each. Test-driven development.
Müller (2005) ⁷	Students	38	19	23	2 runs of 1 programming session each on 2 initial programming tasks (Polynomial and Shuffle-Puzzle) producing about 150 LOC.
Müller (2006) ⁸	Students	18(16)	4(5)	6	1 session involving initial design + programming tasks on an elevator system.
Nawrocki & Wojciechowski (2001) ⁹	Students	15	5	5	4 lab sessions over a winter semester, as part of a University course. Wrote four C/C++ programs ranging from 150-400 LOC.
Nosek (1998) ¹⁰	Professionals	15	5	5	<u>45 minutes to solve 1 programming task</u> (database consistency check script).
*Phongpaibul & Boehm (2006) ¹¹	Students	95	7	7	12 weeks to complete 4 phases of development + inspection.
*Phongpaibul & Boehm (2007)	Students	36	5	4	Part of a team project to extend a system. 13 weeks to complete 4 phases of development + inspection.
Rostaher & Hericko (2002) ¹²	Professionals	16	6	4	6 small user stories filling 1 day.
*Vanhanen & Lassenius (2005) ¹³	Students	20	4	8	9-week student project in which each subject spent a total of 100 hours (400 hours per team). A total of 1500-4000 LOC was written.
Williams <i>et al.</i> (2000) ¹⁴	Students	41	14	13	6-week course where the students had to deliver 4 programming assignments.
Xu & Rajlich (2006) ¹⁵	Students	12	4	4	2 sessions with pairs and 1 session with individuals. 1 initial programming task producing around 200-300 LOC.

Beispiel