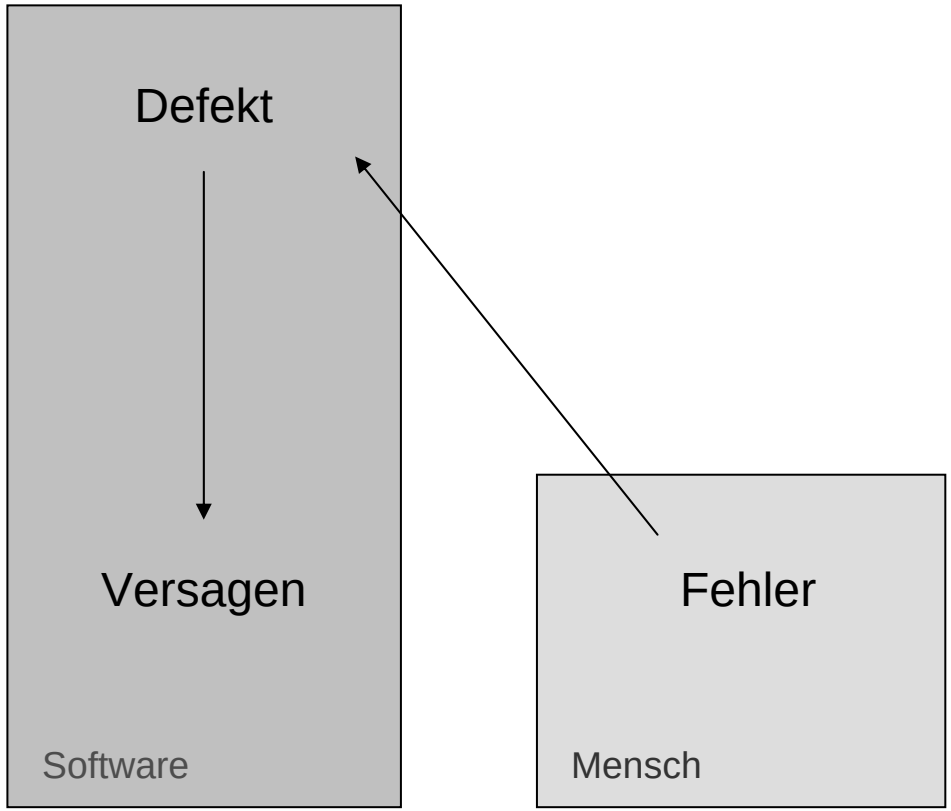
Seminar “Ausgewählte Beiträge
zum Software Engineering”

# Forschung zu Fehlerursachen – Stand der Überlegungen

Sebastian Jekutsch

Freie Universität Berlin
Institut für Informatik
Arbeitsgruppe Software Engineering

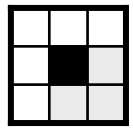
# Grundlegender Fehlerprozess



# Defekt-/Fehlervorhersage: Techniken

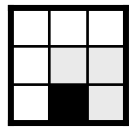
| | Produktbasiert | Prozessbasiert |
|---|---|---|
| Defekt-vorhersage (=Folge) | *Anti-Patterns, Codeanalyse* | *Fehlerzählung, Änderungsanalyse* |
| Fehler-vorhersage (=Ursache) | *Komplexität* | *Prozessbeurteilung, CMM, PSP* |

# Defektvorhersage, Produktbasiert

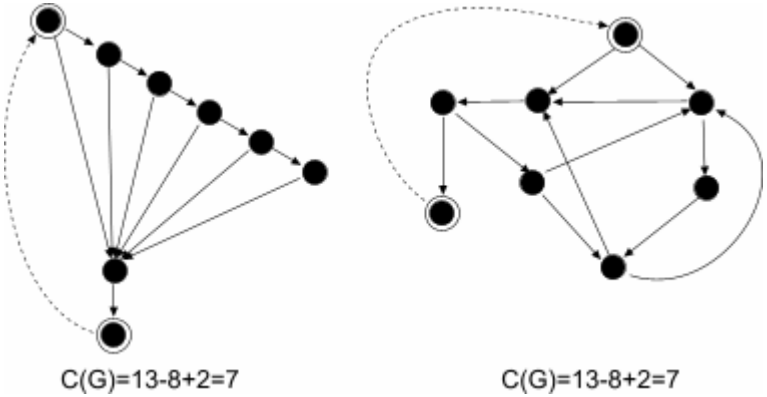


- Prüfen typischer Defektmuster
- Codeanalyse
  - Checklisten
  - Lint, etc.
  - Widersprüche im Code
- Anti-Patterns auch z.B. in Entwürfen
  - bzw. Nicht-Einsatz von Mustern

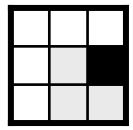
# Fehlervorhersage, Produktbasiert



- Komplexer Code = schwierig zu erweitern
- Beispiel: Zyklomatische Komplexität

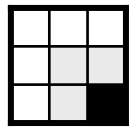


# Defektvorhersage, Prozessbasiert



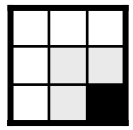
- Diverse Techniken wurden entwickelt
  - Defektzählung: Welche Defekte tauchen auf und ist das normal?
  - Änderungszählung: Wie viele Änderungen welcher Art wurden gemacht?
  - Testumfang: Wurden schon genug Defekte gefunden?

# Fehlervorhersage, Prozessbasiert



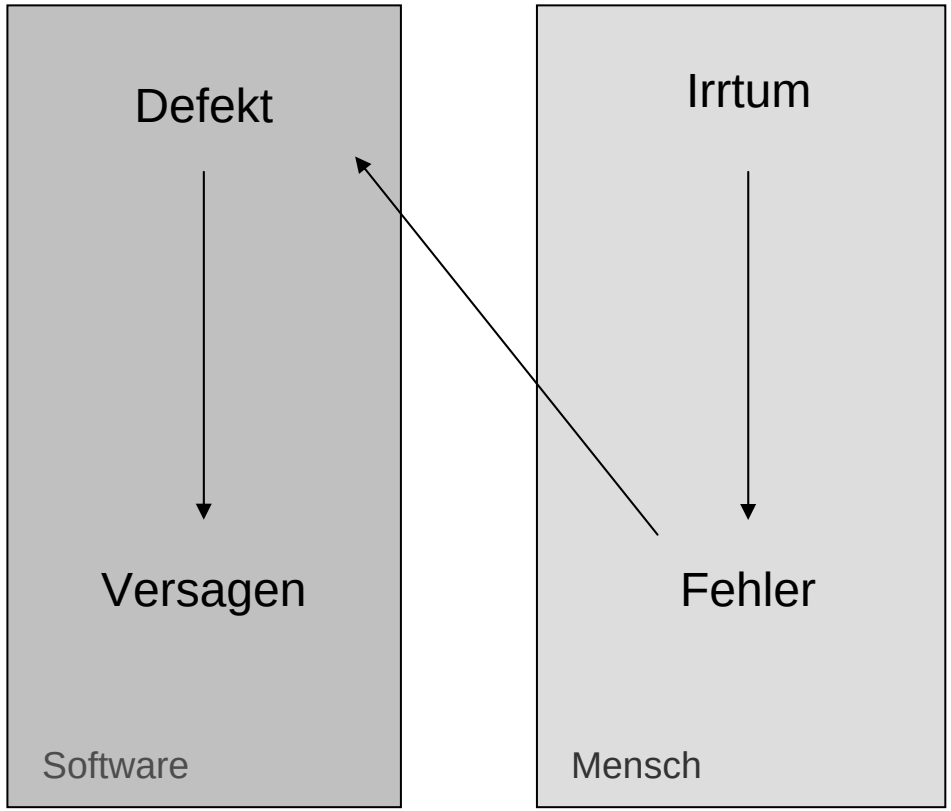
- Untersuchung des Prozesses, wie sehr er Fehler begünstigt/vermeidet
- Im Groben: z.B. CMM
- Im Kleinen: Persönlicher Software Prozess (PSP)
  - systematische Aufzeichnung von (unerwünschten) Ereignissen und Lernen aus den Zusammenhängen udn Umständen
- Ansonsten hat jedes Prozessmodell eine Fehlervermeidungskomponente
  - aber meist keine Fehlervorhersage

# Unsere Herangehensweise:



- Prozessbasierte Fehlervorhersage
- Und zwar im Kleinen: Betrachtung…
  - … des einzelnen Programmierers
  - … und seines Vorgehens = Mikroprozess
- Mittel: Analyse der Tätigkeiten
  - Entdecken von „Fehlerepisoden“ in der Arbeit
  - Suche nach positiven Korrelationen mit nachträglicher, ungeplanter Korrektur dieser Arbeit
    - = Defektbehebung
    - oder zumindest etwas zu vermeidendes
- *Frage: Gibt es typische Mikroprozessmuster, die mit großer Wahrscheinlichkeit fehlerfördernd sind?*
  - oder gar fehlererzeugend

# Erster Ansatz: Menschliche Schwächen



# Typische menschliche Schwächen (1)

- Es gibt typische Denk- und Urteilsfehler
  - in Kognitionspsychologie umfangreich untersucht
- Beispiel: Bestätigungsneigung (confirmation bias)

| | | | |
|---|---|---|---|
| **E** | **4** | **7** | **K** |

- Hypothese: *„Wenn auf der Vorderseite ein Vokal steht, steht auf der Rückseite eine gerade Zahl.“*
- Welche zwei Karten decken Sie auf, um diese Hypothese zu prüfen?

# Typische menschliche Schwächen (2)

- Die meisten Menschen decken die E und die 4 auf.
- Korrekt ist E und 7.

| | | | |
|---|---|---|---|
| **E** | **4** | **7** | **K** |

- Bestätigungsneigung: Der Mensch versucht eher, Hypothesen zu bestätigen als sie zu widerlegen.
- Nachgewiesen wurde dieser Effekt beim Testen von Software
  - Kurz: Es werden bevorzugt Fälle getestet, die bei der Spezifikation auch erwähnt wurden.
  - B. Teasley, L.M. Leventhal, R.M. Clifford, and D.S. Rohlman: Positive test bias in software testing by professionals
- Ansonsten keine Arbeit dieser Art bekannt.
  - Und Defekte verhindert dies leider auch nicht.

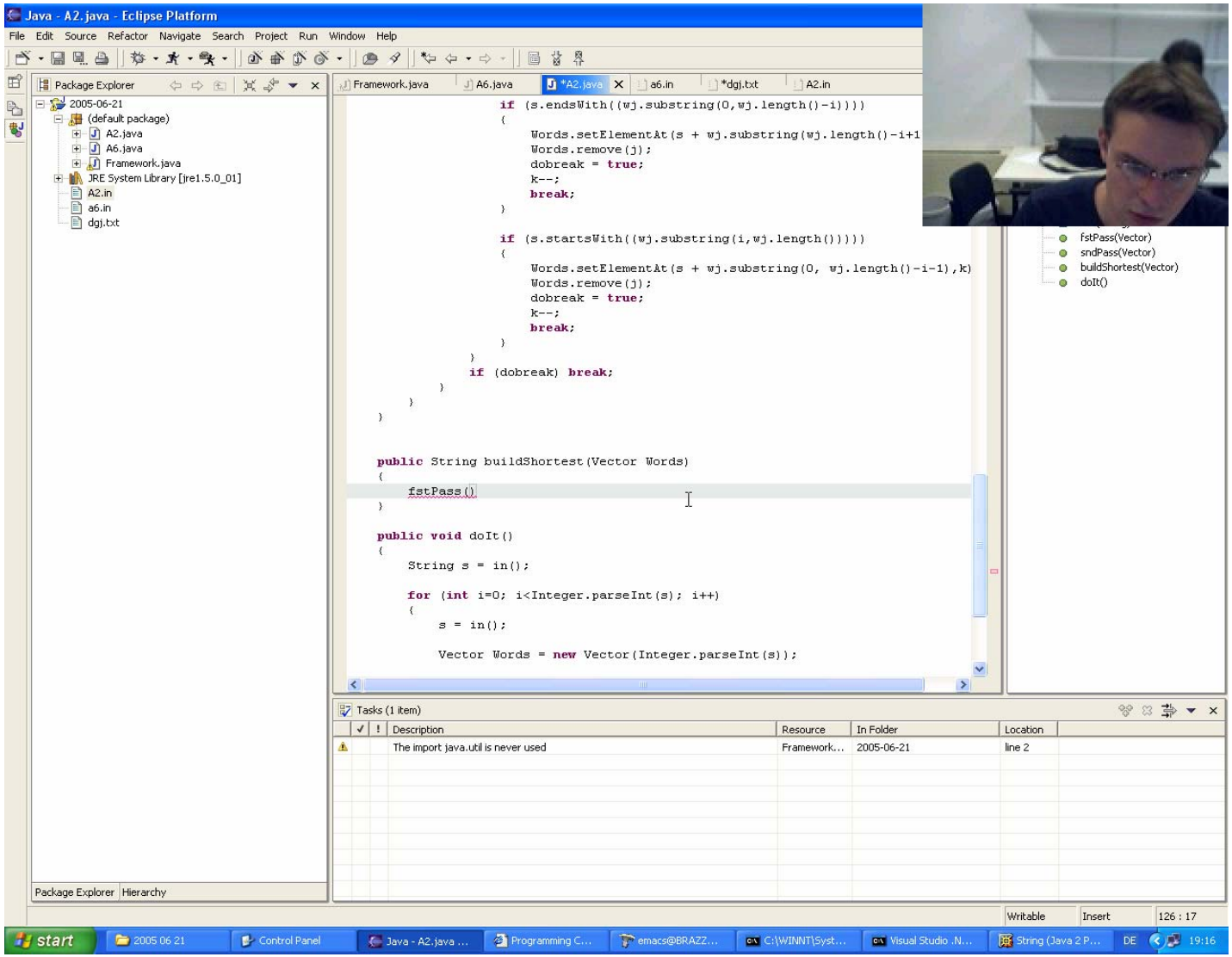
# Typische menschliche Schwächen (3)

- Es gibt verschiedene Arten von menschlichen Schwächen
  - Fehler in der Wahrscheinlichkeitsrechnung
  - Fehler im logischen Schließen
  - Hypothesenfehler bei komplexen, dynamischen Systemen
  - Fehler in der Informationsaufnahme (Filtern)
  - u.s.w.
- Querverweise aufs Programmieren habe ich nicht bilden können.
- Es gibt aber eine Untersuchung zu Logikfehlern beim Algorithmusentwurf
  - Timm Grams: Denkfallen und Programmierfehler

# Typische menschliche Schwächen (4)

- Das führte also zu nichts...
- Ausflucht auf handwerkliche Fehler...
  - z.B. Copy-Paste-Change
  - z.B. Nicht-Nachtesten einer Änderung
- ... und irgendwie suspekt erscheinendem Verhalten
  - z.B. Trial-and-Error-Vorgehen
  - z.B. keine Refactorings

# Beobachtungen, um auf Ideen zu kommen



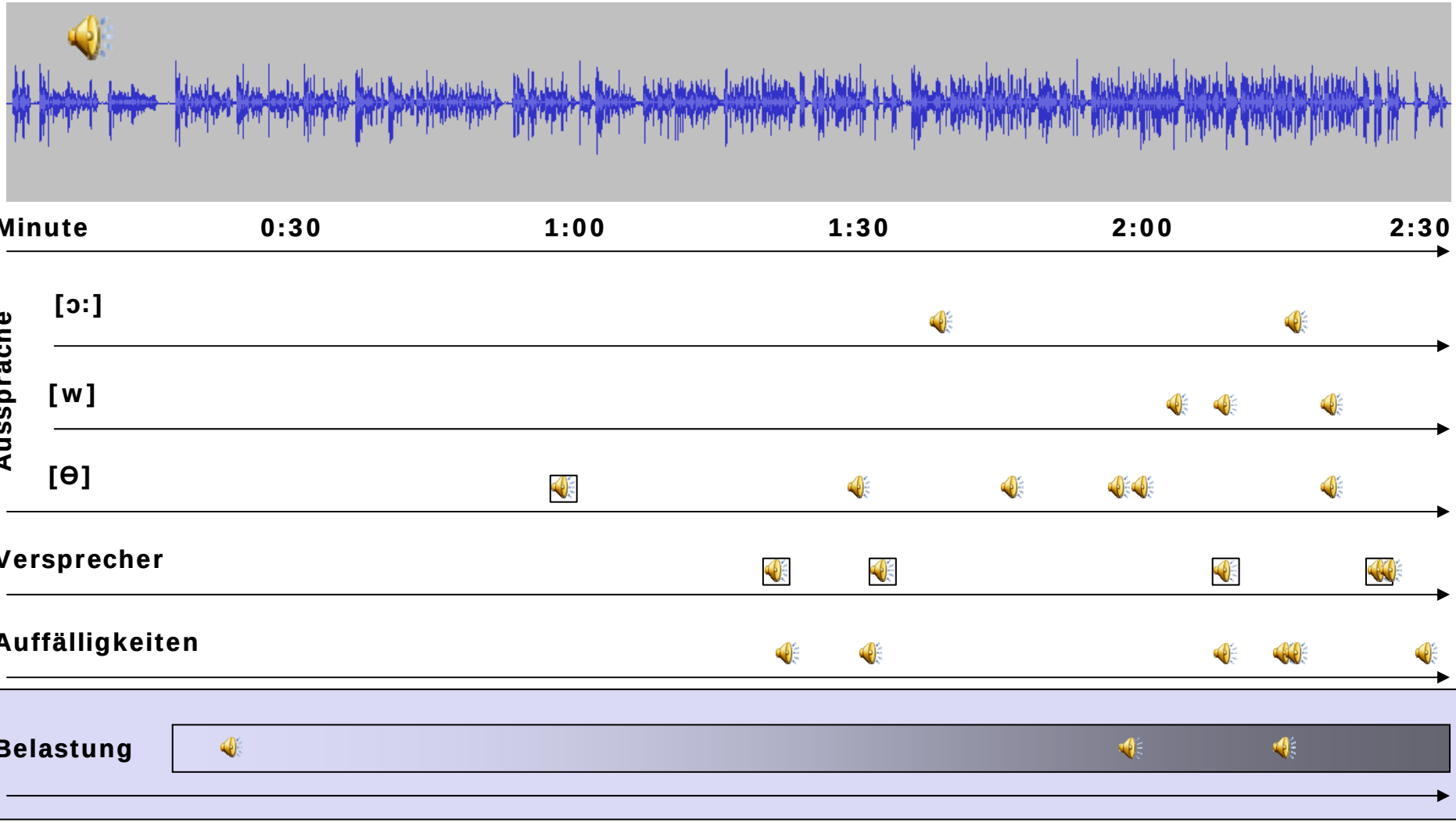
# Zweiter, aktueller Ansatz: Mentale Belastung

- Verhalten bei schwierigen Problemen und ungünstigen Umständen ist anders als „sonst“
  - Ähnliche Unterschiede bei guten und schlechten Programmierern
- Nicht mehr die Ursachen (oder gar die Folgen), sondern die Symptome beobachten
  - *Frage nun: Gibt es typische Mikroprozessmuster, die mit großer Wahrscheinlichkeit fehlerbegleitend sind*

# Verdeutlichung: „Die zwei Cousinen“

*Auf dem Landsitz North Cothelstone Hall von Lord und Lady Hesketh-Fortescue befinden sich außer dem jüngsten Sohn Meredith auch die Cousinen Priscilla und Gwyneth Molesworth aus den benachbarten Ortschaften Nether Addlethorpe und Middle Fritham, ferner ein Onkel von Lady Hesketh-Fortescue, der neunundsiebzigjährige Jasper Fetherston, dessen Besitz Thrumpton Castle zur Zeit an Lord Molesworth-Houghton, einen Vetter von Priscilla und Gwyneth Molesworth, vermietet ist.*
[...]

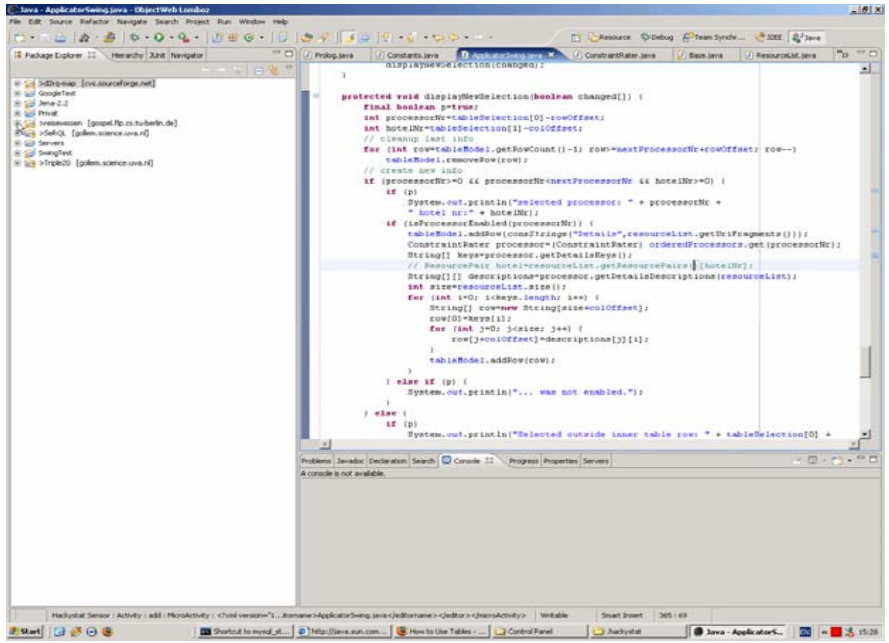
# Verdeutlichung: „Zwei Cousinen“ analysiert



# Zurück zum Programmieren

Meine Hypothesen:

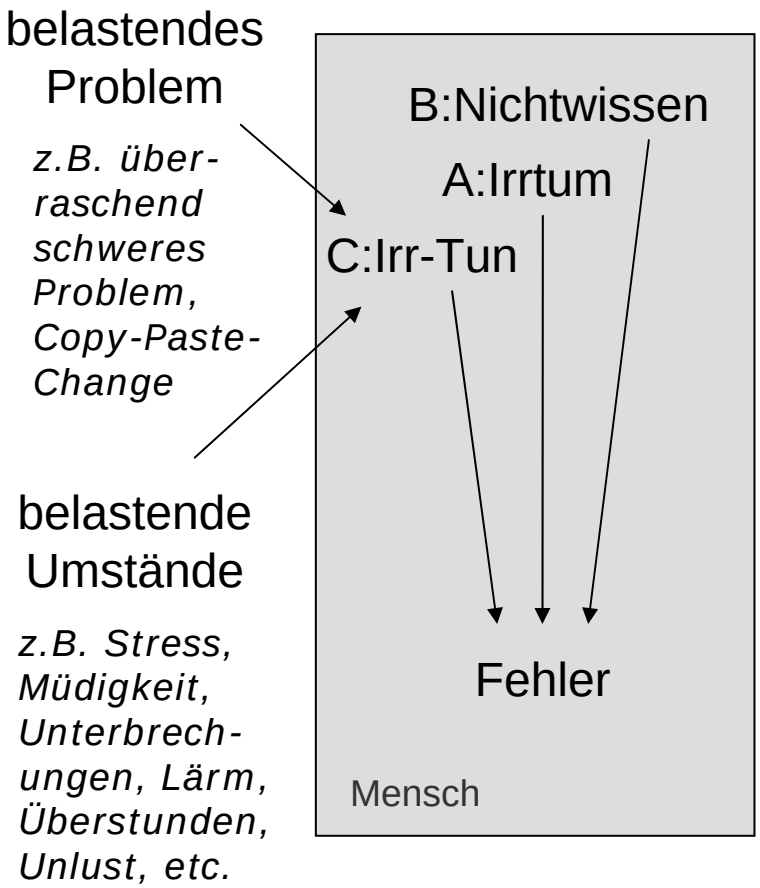
1. Hypothese: Mentale Belastung lässt die Fehlerwahrscheinlichkeit steigen
2. Hypothese: Mentale Belastung ist im messbaren Mikroprozess erkennbar (als Symptom)



# 1. Hypothese: Belastung ⇒ Mehr Fehler (1)

- Allgemein akzeptiert
- Beispiel: Gesetzliche Regelungen für Fahrpersonal
  - An jedem Arbeitstag dürfen 9 Stunden gelenkt und gearbeitet werden.
  - In einem Zeitraum von zwei aufeinander folgenden Wochen dürfen höchstens 90 Stunden gelenkt werden.
  - Nach einer Lenkzeit von 4,5 Stunden muss der Fahrer mindestens 45 Minuten Pause einlegen.
  - u.s.w.
- Beugt Müdigkeit vor
  - z.B. Sekundenschlaf
  - Müdigkeit ist eine Form mentaler Belastung

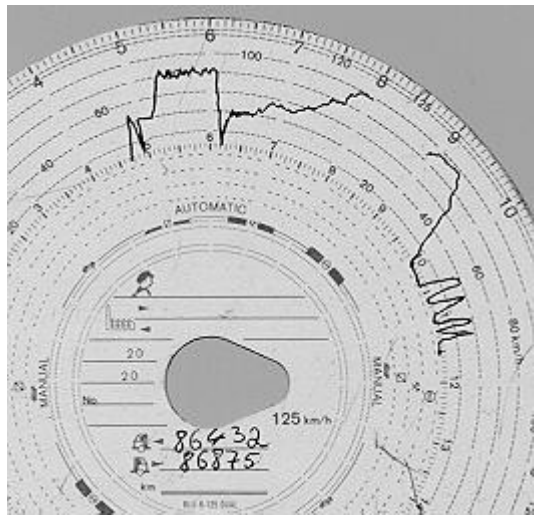
# 1. Hypothese: Belastung ⇒ Mehr Fehler (2)




- A-Fehler: Unbewusstes Falschwissen
  - Symptome identisch zu Richtigwissen
- B-Fehler: Bewusstes Nichtwissen
- C-Fehler: Unbewusstes Falsch*machen*
  - Man weiß es eigentlich besser
  - Problem wird oft nicht erkannt oder es wird verdrängt

# 2. Hypothese: Belastung am µP erkennbar (1)

- Deutlich problematischer…
- Beispiel: Woran erkennt man einen unsicheren (B) oder abgelenkten (C) Autofahrer?
- Beispiel: Fahrtenschreiber
  - Persönliche Aufzeichnung von Geschwindigkeit in der Zeit
    - = Mikroprozess
  - Wird kontrolliert
  - Ist aber eine Fehlervermeidung
    - … keine –vorhersage



# 2. Hypothese: Belastung am µP erkennbar (2)

- Achtung bei B-Fehlern (bewusstes Nichtwissen): Programmierer treffen oft Vorkehrungen
  - häufiges Testen
  - Problemzerlegung
  - Wissen nachholen
  - langsamer arbeiten
- Ist das ein Indiz (Symptom) für Probleme? oder ist das schon eine erfolgreiche Therapie?
- Kurzzeitige Überforderung wird aber oft überspielt
  - z.B. einfach raten (Trial-and-Error)
- C-Fehler (unbewusstes Falschtun) sind da eindeutiger
  - oft „dumme Fehler“ genannt
  - das Gegenteil ist das hochkonzentrierte „Fliegen“

# 2. Hypothese: Belastung am µP erkennbar (3)

- Wie sehen die Symptome aus?
  - Diskontinuität in der Arbeit, ungeplantes Vorgehen (C)
    - viele offene Stellen und Wechsel zwischen den Stellen
    - Schachtelungen der Teilprobleme
  - Ablenkungen, Erledigung nebensächlicher Dinge (B,C)
  - Wiederholungen in den Zielen und Tätigkeiten (C)
    - ewiges Suchen in der Dokumentation
    - Arbeiten häufig revidieren und neu erstellen
  - Auffälligkeiten als Folge von erkannten Fehlern (B)
    - viele Defektbehebungen
- = Chaos = hohe Entropie = *Mikroprozesskomplexität*
  - Es passiert Unwahrscheinliches
  - Experten gehen gradlinig = wahrscheinlich vor
  - A. E. Hassan, R. C. Holt: Chaos of Software Development

# 2. Hypothese: Belastung am µP erkennbar (4)

- Vermutlich sind die Symptome personenabhängig
  - Jeder reagiert anders
  - Was ist das „normale“ Verhalten?
- Andere Abhängigkeiten?

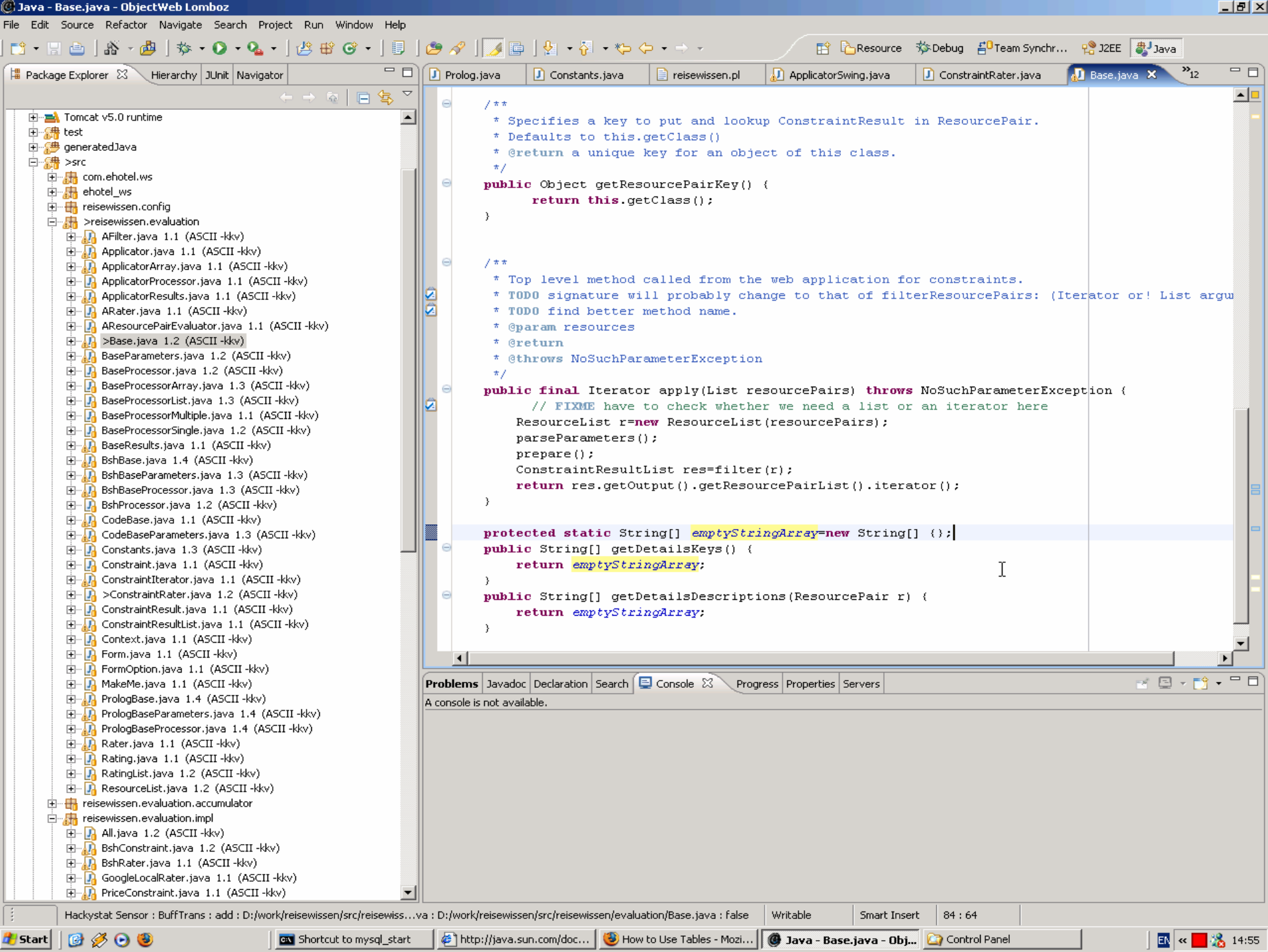
# Vision

- Ein Werkzeug bauen, das...
  ... typische Symptome entdeckt
  ... dem Programmierer bei der Diagnose hilft
  ... und daraufhin sogar eine Therapie vorschlägt
- Tipps auf vermutlich defekte Stellen
  - für Programmierer und Qualitätssicherung
    - Privatsphäre!
- Nachvollziehen der Gründe für einen Defekteinbau
- Vielleicht sogar Frühwarnsystem
  - kaum ohne Hilfe des Programmierers möglich
- Erkenntnisgewinn sowieso

# Methode: Analyse von Programmiersitzungen

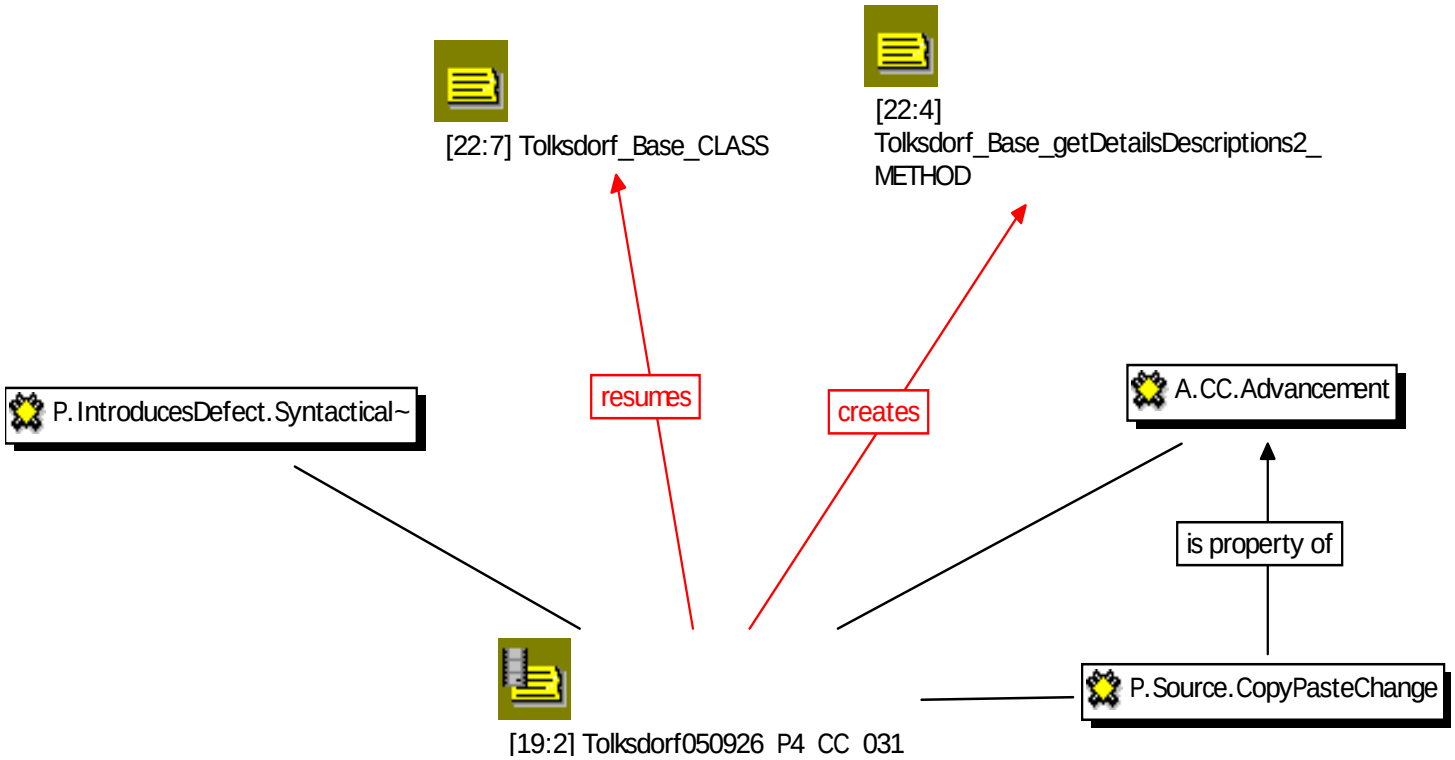
- Genaue Analyse von Programmiertätigkeiten
  - evtl. auf die Sekunde genau!
  - = Aufschreiben des Mikroprozesses
- Programmiertätigkeiten sind
  - Code ändern (verschiedenste Ausprägungen)
  - Kompilieren, Testen, Browsen
  - Pause machen, Nachdenken
  - Entwerfen
  - Diskutieren
  - Lesen, etc.
- Zunächst konzentrieren auf das leicht beobachtbare

- Es folgt ein Beispiel.


Java - Base.java - ObjectWeb Lomboz
File Edit Source Refactor Navigate Search Project Run Window Help
Resource Debug Team Synchr... J2EE Java
Package Explorer Hierarchy JUnit Navigator
Tomcat v5.0 runtime
test
generatedJava
>src
com.ehotel.ws
ehotel_ws
reisewissen.config
>reisewissen.evaluation
AFilter.java 1.1 (ASCII -kkv)
Applicator.java 1.1 (ASCII -kkv)
ApplicatorArray.java 1.1 (ASCII -kkv)
ApplicatorProcessor.java 1.1 (ASCII -kkv)
ApplicatorResults.java 1.1 (ASCII -kkv)
ARater.java 1.1 (ASCII -kkv)
AResourcePairEvaluator.java 1.1 (ASCII -kkv)
>Base.java 1.2 (ASCII -kkv)
BaseParameters.java 1.2 (ASCII -kkv)
BaseProcessor.java 1.2 (ASCII -kkv)
BaseProcessorArray.java 1.3 (ASCII -kkv)
BaseProcessorList.java 1.3 (ASCII -kkv)
BaseProcessorMultiple.java 1.1 (ASCII -kkv)
BaseProcessorSingle.java 1.2 (ASCII -kkv)
BaseResults.java 1.1 (ASCII -kkv)
BshBase.java 1.4 (ASCII -kkv)
BshBaseParameters.java 1.3 (ASCII -kkv)
BshBaseProcessor.java 1.3 (ASCII -kkv)
BshProcessor.java 1.2 (ASCII -kkv)
CodeBase.java 1.1 (ASCII -kkv)
CodeBaseParameters.java 1.3 (ASCII -kkv)
Constants.java 1.3 (ASCII -kkv)
Constraint.java 1.1 (ASCII -kkv)
ConstraintIterator.java 1.1 (ASCII -kkv)
>ConstraintRater.java 1.2 (ASCII -kkv)
ConstraintResult.java 1.1 (ASCII -kkv)
ConstraintResultList.java 1.1 (ASCII -kkv)
Context.java 1.1 (ASCII -kkv)
Form.java 1.1 (ASCII -kkv)
FormOption.java 1.1 (ASCII -kkv)
MakeMe.java 1.1 (ASCII -kkv)
PrologBase.java 1.4 (ASCII -kkv)
PrologBaseParameters.java 1.4 (ASCII -kkv)
PrologBaseProcessor.java 1.4 (ASCII -kkv)
Rater.java 1.1 (ASCII -kkv)
Rating.java 1.1 (ASCII -kkv)
RatingList.java 1.2 (ASCII -kkv)
ResourceList.java 1.2 (ASCII -kkv)
reisewissen.evaluation.accumulator
reisewissen.evaluation.impl
All.java 1.2 (ASCII -kkv)
BshConstraint.java 1.2 (ASCII -kkv)
BshRater.java 1.1 (ASCII -kkv)
GoogleLocalRater.java 1.1 (ASCII -kkv)
PriceConstraint.java 1.1 (ASCII -kkv)
Prolog.java Constants.java reisewissen.pl ApplicatorSwing.java ConstraintRater.java Base.java
/**
* Specifies a key to put and lookup ConstraintResult in ResourcePair.
* Defaults to this.getClass()
* @return a unique key for an object of this class.
*/
public Object getResourcePairKey() {
return this.getClass();
}
/**
* Top level method called from the web application for constraints.
* TODO signature will probably change to that of filterResourcePairs: (Iterator or! List argu
* TODO find better method name.
* @param resources
* @return
* @throws NoSuchParameterException
*/
public final Iterator apply(List resourcePairs) throws NoSuchParameterException {
// FIXME have to check whether we need a list or an iterator here
ResourceList r=new ResourceList(resourcePairs);
parseParameters();
prepare();
ConstraintResultList res=filter(r);
return res.getOutput().getResourcePairList().iterator();
}
protected static String[] emptyStringArray=new String[] {};
public String[] getDetailsKeys() {
return emptyStringArray;
}
public String[] getDetailsDescriptions(ResourcePair r) {
return emptyStringArray;
}
Problems Javadoc Declaration Search Console Progress Properties Servers
A console is not available.
Hackystat Sensor : BuffTrans : add : D:/work/reisewissen/src/reisewiss...va : D:/work/reisewissen/src/reisewissen/evaluation/Base.java : false
Writable
Smart Insert
84 : 64
Start
Shortcut to mysql_start
http://java.sun.com/doc...
How to Use Tables - Mozi...
Java - Base.java - Obj...
Control Panel
14:55

# Methode: Beispiel (2)

- Kodierung dieser Episode („Quotation“)



# Methode: Benutzte Codes

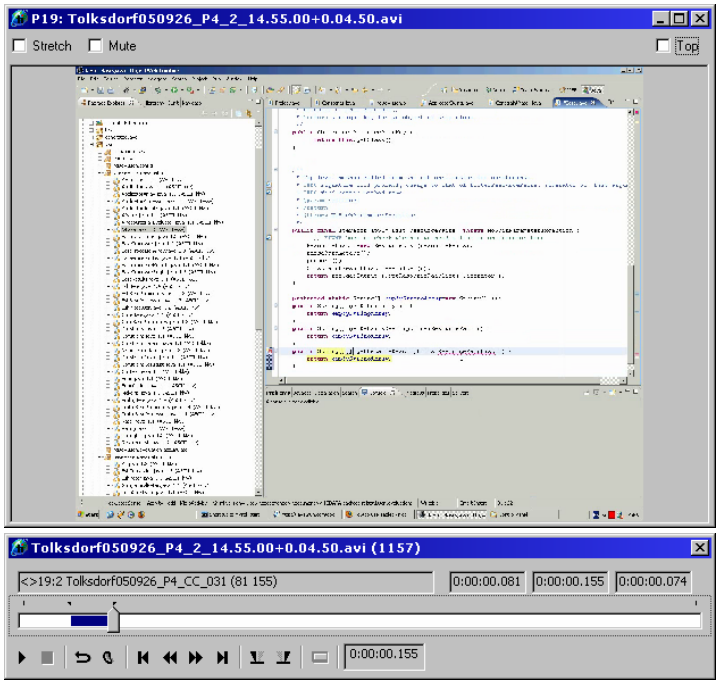
- Aktivitäten (etwa 50 verschiedene)
  - Browsen: in APIDoc, in Code
  - CodeChange:
    - Advancement, Betterment, Displacement, ...
    - IncludesTagAlong, IncludesReSpell, ...
    - Pause, Compile, Run, ...
  - Phasen:
    - Complete, ..., Compile, Run, PostFix, ...
  - Sonst:
    - Pause, Reading, Thinking (About...), Work
- Eigenschaften (20) + Werte (10)
  - IntroducesDefect, Speed, LeavesIncomplete, ...
- Ereignisse (12)
  - Interruption, CompilationWithDefect, RunningWithFailure,...

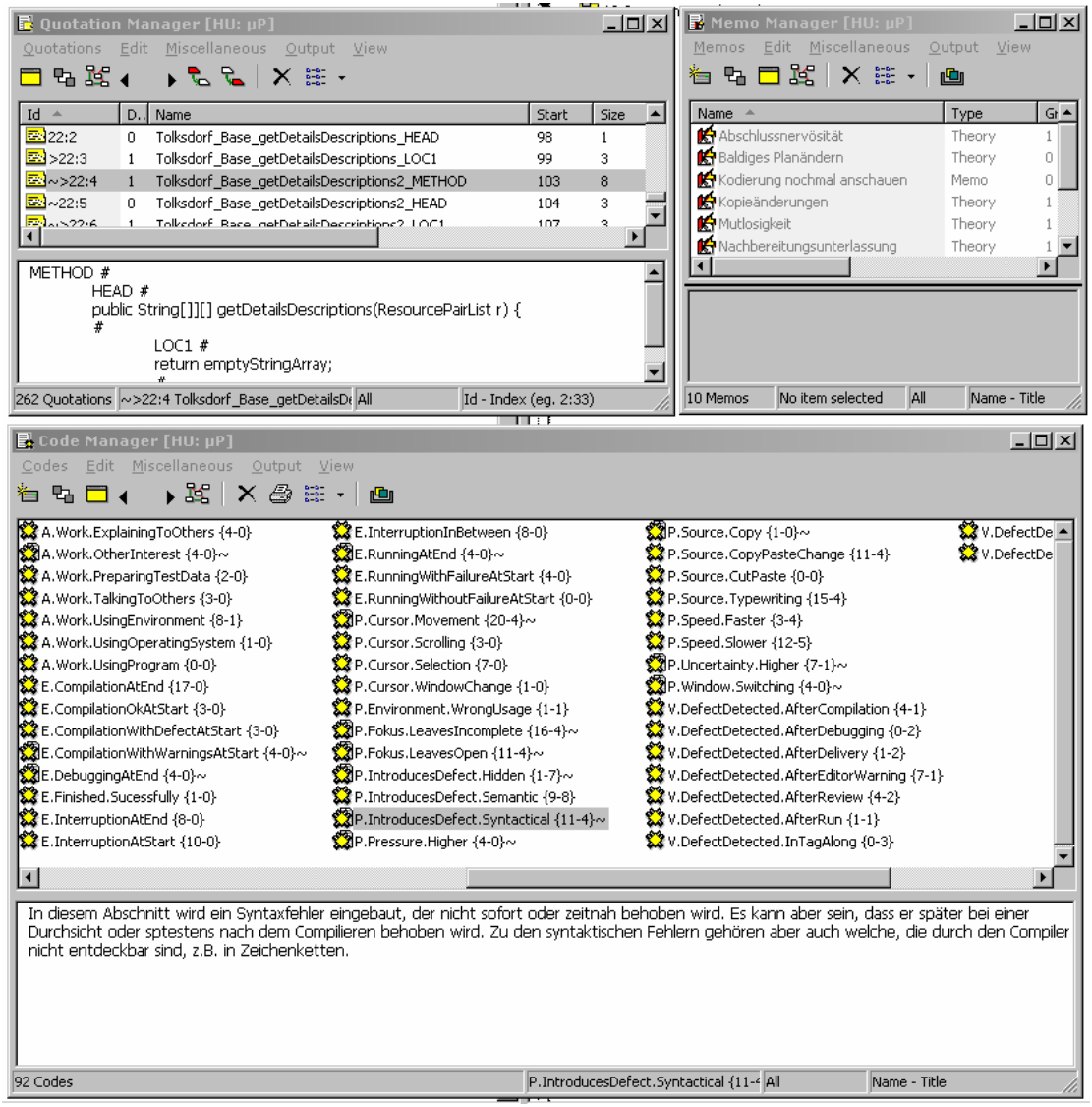
# Methode: Beziehungen, Netz

- Viele Aktivitäten beziehen sich auf Stellen im Code
- Codestellen sind auch Quotations
- Beziehungstypen Aktivität zu Codestelle:
  - creates, resumes, discards, changes, corrects
- Es ergibt sich ein großes Netz von Kodierungen und Beziehungen
  - man verliert den Überblick
  - aber man möchte Muster erkennen
    - eben jene Verhaltensauffälligkeiten
- Zweck: Beobachten von Defekterzeugungsphasen
  - im Vergleich zu „normalen“ Phasen

# Methode: Software

- Software: Atlas.Ti



# Methode: Automatisierung der Analyse

- Versuch einer automatischen Kodierung / Aufzeichnung
- Laufende Diplomarbeit
  - Server zur Sammlung von primitiven Ereignissen
    - derzeit: aus Eclipse
    - Studienarbeit: Erkennung bestimmter Unterbrechungen
  - Plug-in-Mechanismus zur Verarbeitung der Daten
    - z.B. Episoden wie Trial-and-Error
    - Echtzeitanalyse vorgesehen
  - siehe www.electrocodeogram.org
- Kommende X-Arbeiten
  - Visualisierung der Daten
  - Abspielmodus für eine Methode
- Visuelle und statistische Analyse der Daten

# Neu: Mikroprozesstheorie

- Entwicklung eines Vokabulars zur Beschreibung von Mikroprozessen (Codeänderungen und vieles mehr)
- Darauf aufbauend Episoden, Aktivitäten, Phasen (=Sätze)
  - inkl. Anomalien
- Belastungssymptome sind darin Prozessmaße (Metriken)
  - z.B. Maß an Mikroprozesskomplexität

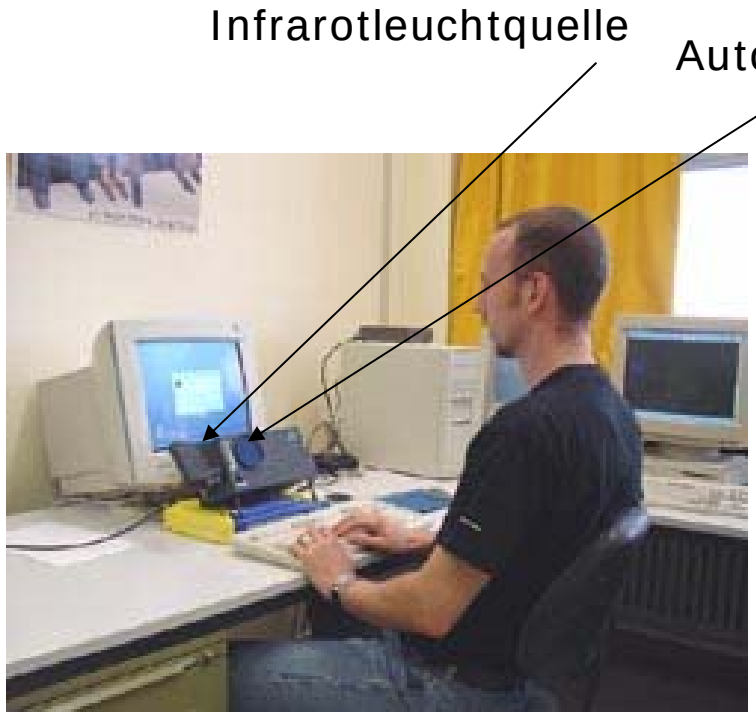
- Vermutlich Umbenennung: Nicht mehr „Mikroprozess“
  - „Software Development Behavior Patterns“?

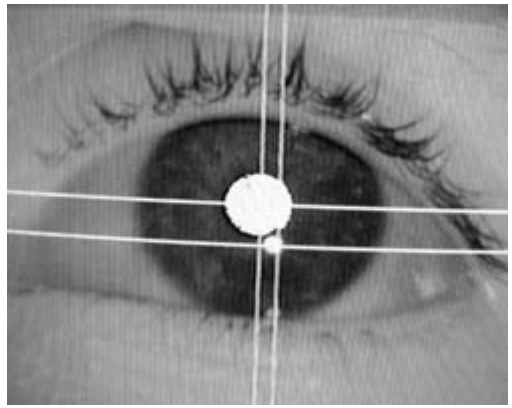
# Zusammenfassung

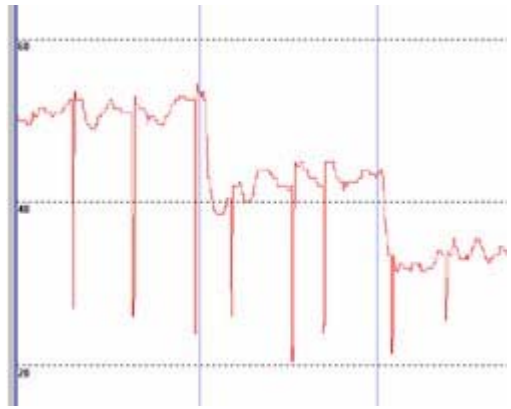
- Ziel ist Mikroprozessbeurteilung zur Fehlererkennung
- 1. Ansatz über menschliche Schwächen führte zu nichts
- Neuer Ansatz: Fehlerbegleitende Umstände identifizieren
  - B: Erkennen fehlervermeidender Strategien
    - ist allerdings schon die Therapie, also Problem beseitigt
  - C: Erkennen von Belastungssymptomen
    - Mikroprozesskomplexität, Chaos des Programmierens
- Forschungsmethode ist genaue Beobachtung
  - und Entdeckung von Korrelationen
  - möglichst automatisiert
- Heraus kommt hoffentlich ein Werkzeug, dass dem Programmierer bei der Selbstdiagnose hilft

# So geht‘s vielleicht auch: Pupillometrie (1)

- Pupillometrie als objektive Methode zur Messung von emotionalen Reaktionen
  - z.B. „mental overload“
  - Weite Pupille = hohe Belastung

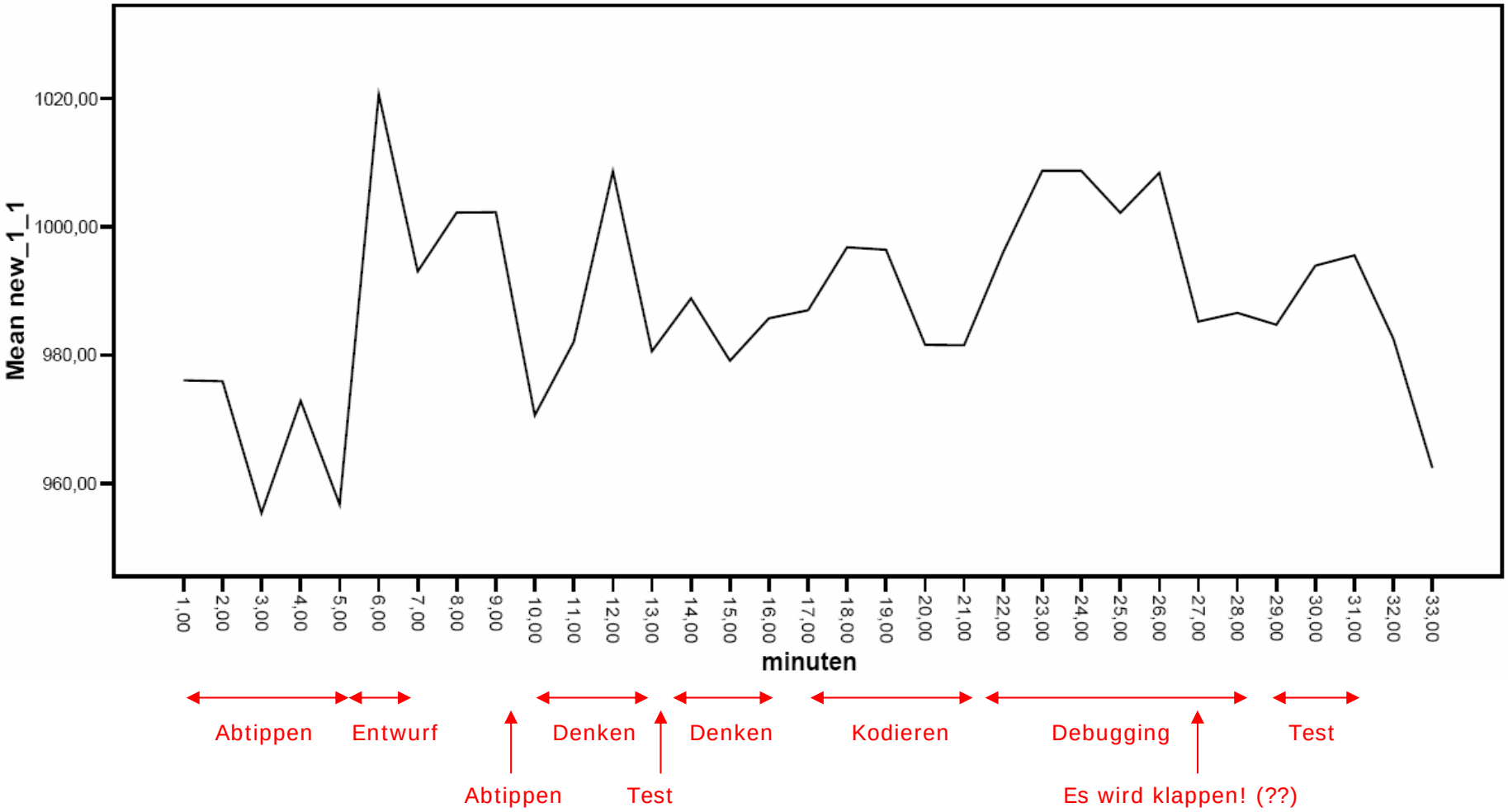




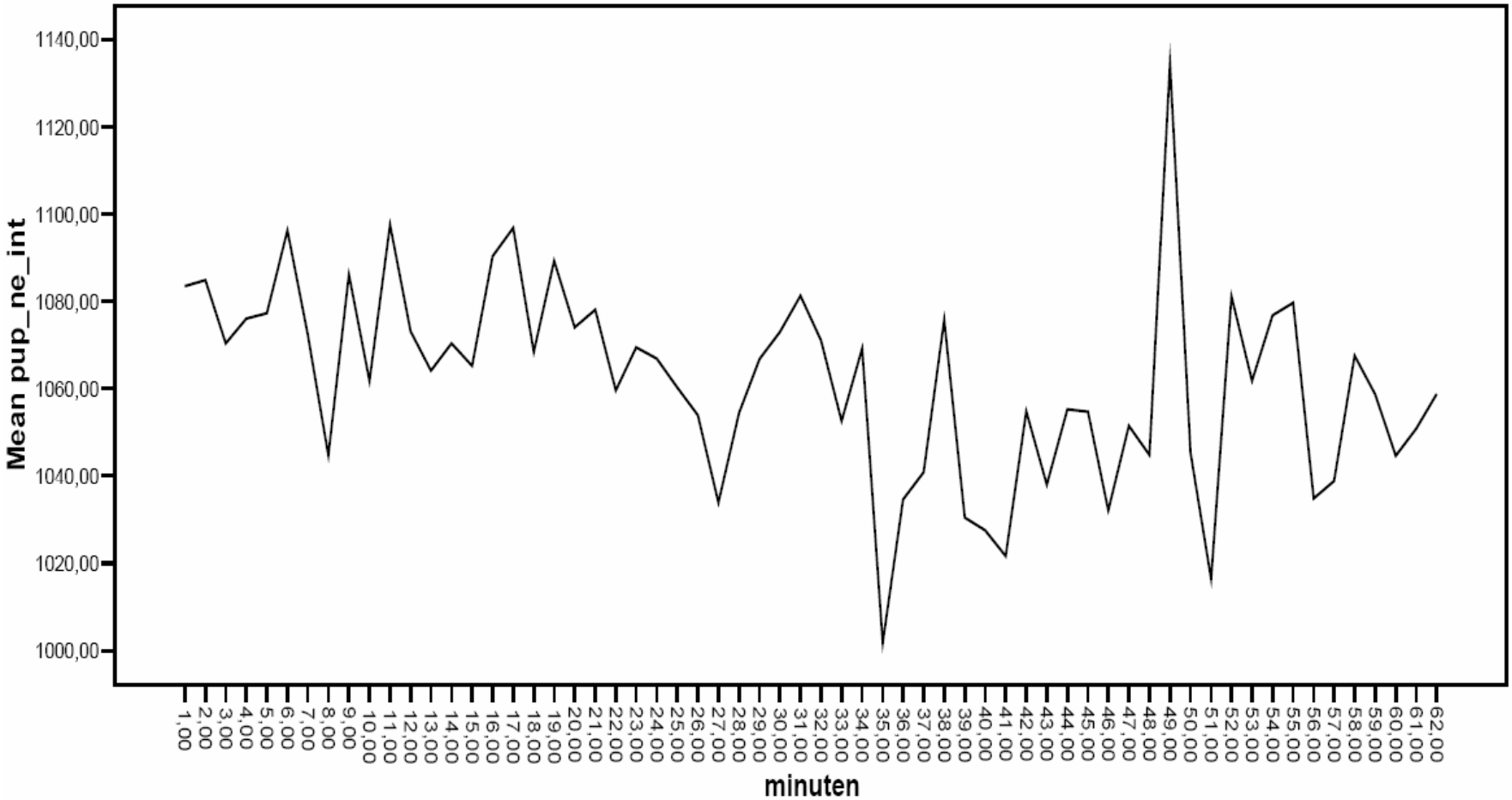
# Pupillometrie (2)

- Selbsttest mit einfacher ACM-Aufgabe



# Pupillometrie (3)

- Selbsttest mit nicht-trivialer ACM-Aufgabe
  - die absoluten Werte sollten nicht betrachtet werden



# Vielen Danke!