
# Experimentbeispiele

# An Experiment about Static and Dynamic Type Systems

## Doubts about the Positive Impact of Static Type Systems on Development Time

# Objective

- Evaluierung des Einflusses von statischen und dynamischen Typsystemen auf Entwicklungszeit
- Verschiedene Argumente für und gegen statische Typsysteme
- Keine Forschungshypothese, sondern Forschungsfrage (nicht explizit genannt)

# Variablen (1)

- Unabhängige Variablen:
  - Typsystem
    - 2 Stufen (statisch und dynamisch)
    - 2 gleiche Programmiersprachen mit unterschiedlichem Typsystem
  - Aufgabe
    - 2 Stufen (Scanner und Parser)
- Abhängige Variablen:
  - Entwicklungszeit

# Variablen (2)

- Tool-erfahrung

| Variable | Kontroll-Technik | | Gemessen/Gesichert | |
|---|---|---|---|---|
| | Wie | Warum | Wie | Warum |
| Erfahrung | Matching | Erfahrung spielt große Rolle bei Entwicklungszeit | Interview | Um möglichst genaue Einschätzung von Erfahrung |
| Programmier-sprache | Konstant | Spielt große Rolle bei Entwicklungszeit | Neue Sprache | Garantiert gleiche Erfahrung aller Probanden |
| Tool-erfahrung | Konstant | Spielt große Rolle bei Entwicklungszeit | Neues Tool | Garantiert gleiche Erfahrung aller Probanden |

# Material

- Purity, ein paar Bibliotheken
- PurityIDE
- -> beides kontrolliert am besten die Einflüsse von Sprache und IDE

- Spezifikation eines Parsers in kontext-freier Grammatik
- Geheimhaltungserklärung

# Aufgabe

1. Scanner:
   - Scannt Wort und entfernt Sonderzeichen
   - Characters als Tokens
2. Parser:
   - Bekommt Wort als Input
   - Gibt wahr oder falsch aus, abhängig davon, ob Wort Teil der Grammatik ist

# Design

- Latin Square
- Within-Subjects nicht sinnvoll, da man in einem Typsystem denkt
- Zeitaufwand zu lang

# Probanden

- 49 Bachelor-Studenten
- Erfahrung mit formalen Sprachen und Java
- Unerfahren mit Parser-Implementierung

# Ausführung

- Erst Interview zur Erstellung der Gruppen
- 16 Stunden Training (dynamisch)
- 18 Stunden Training (statisch -> braucht mehr Erklärung)
- 27 Stunden, frei verteilt auf 4 Arbeitstage
- Probanden durften kein Material mit nach Hause nehmen

# Deviation

- There are no deviations to report

# Using Students as Subjects

An Empirical Evaluation

# Objective

- Studenten werden oft herangezogen in Experimenten, und dann Aussagen über Experten getroffen
- Eignung von Studenten, um Aussagen über Experten treffen zu können
- To what extent are students capable of imagining how industry professionals work in a complex requirements engineering decision process?

# Variablen

- Unabhängig:
  - Status (Student [2 Sichten], Experte)
  - Daten von Experten aus anderem Artikel
- Abhängig:
  - Auswahl von Anforderungen
- Störvariablen:
  - Nicht klar erkennbar

# Material/Task

- Fragebogen
  - Studentensicht auf Anforderungen
  - Expertensicht aus Sicht der Studenten auf Anforderungen

- Auswahl von Anforderungen aus zwei Sichten

# Design

- Between-Subjects (Studenten vs. Experten)

# Probanden

- Studenten aus Requirements-Engineering-Kurs
- Durchschnittsalter: 26
- Verschiedene Kulturen
- Praktische Erfahrung: 0-2 Jahre, 2 Studenten mit 7 Jahren

# Ausführung/Deviation

- Anforderungsauswahl über 4 Wochen
- Auswahl aller Studenten wurde besprochen
- Jede Woche konnten Anforderungen angepasst werden

- There are no deviations to report

# An Empirical Study of the Effects of Personality in Pair Programming using the Five-Factor Model

# Objective

- Effektivität von Pair-Programming in Ausbildung verbessern
- Fokus auf Persönlichkeitstypen
- Hypothese:

  Unterschiede in Persönlichkeit beeinflussen Effektivität von Studenten, die Pair-Programming angewendet haben

# Variablen

- Unabhängig:
  - Persönlichkeitskombinationen (IPIP-NEO)
  - 2 Stufen: gleiche/verschiedene Persönlichkeit
- Abhängig:
  - Effektivität bei Pair-Programming (Benotete Leistung bei Test und 3 Aufgaben)
  - Zufriedenheit d. Studenten (Fragebogen)
- Störvariablen:
  - Motivation
  - Programmiererfahrung

# Material/Aufgabe

- Persönlichkeitsfragebogen
- Demografischer Fragebogen
- Programmiererfahrungsfragebogen
- Pair-Programming-Aufgaben (nicht weiter spezifiert)

# Probanden

- Studenten (BSc)
- Einführungskurs Informatik

# Ausführung/Deviation

- Persönlichkeitsfragebogen
- Demografischer/Programmiererfahrungs-fragebogen
- Aufgaben über das ganze Semester

# Empirical Studies of Programming Knowledge

# Objective

- Warum sind Experten besser als Anfänger?
- Do expert programmers possess programming plans and discourse rules?
- Programming plans:
  - Programmfragmente, die stereotypische Handlungssequenzen repräsentieren, z.B., Itemsuche in Liste
- Discourse rules
  - Coding conventions, z.B., Variablennamen

# Variables

- Unabhängig:
  - Pläne; 2 Stufen: konform oder nicht-konform
  - Aufgabe; 4 Stufen
  - Erfahrung; 2 Stufen
- Abhängig:
  - Korrektheit von Antworten
  - Antwortzeit
- Störvariable:
  - Motivation (5$ für Teilnahme)
  - Reihenfolge (Randomisierung)
  - Zeitdruck (kein Zeitlimit)

# Material/Aufgabe

- 4 verschiedene Programme in jeweils 2 Versionen in Pascal
- Auf Papier
- Fill-in-the-blank: Probanden sollten fehlende Codezeile ersetzen

# Probanden

- Studenten (1. Semester, oder mindestens 3 Programmierkurse bzw. Master)

# Ausführung/Deviation

- Probanden bearbeiteten Aufgaben (keine Erwähnung von Einführung, Fragebögen,…)
- Keine Abweichungen

# Understanding Exception Handling: Viewpoints of Novices and Experts

# Objective

- Herausfinden, wie Anfänger und Experten mit Exceptions umgehen

# Variablen

- Unabhängig: Erfahrungslevel
  - Anfänger
  - Experte
- Abhängig:
  - Exception-Handling-Strategien
- Störvariablen:
  - Erfahrung mit Java

# Material-Anfänger

- Beispiel-Fragen aus dem Interview-Leitfaden:
  - What approach do you follow to understand exception-flow information in a program?
  - When working with code (e.g., coding, testing, reviewing, and understanding) how often do you pay attention to the functionality associated with exception-handling?
  - Are there scenarios in which you avoid/ignore using exception-handling in your programs? (Yes/No) If yes, when do you do so? Why do you do this?
  - Are you satisfied with the way you approach understanding the program with respect to exceptions? (Yes/No/Maybe)

# Material-Experten

- Beispiel-Fragen aus dem Interview-Leitfaden:
  - How do you perceive error/exception-handling when you are coding/designing? Why?
  - When do you typically start thinking about exceptional conditions? Has your strategy changed over time? How? What factors caused these changes?
  - Have you worked with junior/novice developers? Have you observed any typical patterns in the way they handle the exceptions?
  - How did you learn about exception-handling? (Educated in the work place/college/learned on your own.)

# Probanden/Design

- Anfänger:
  - 8 Praktikanten, graduate students
  - Im Durchschnitt 2 Jahre Programmiererfahrung
- Experten
  - 6 Experten von 5 verschiedenen Firmen
  - Min. 5 Jahre professionelle Programmiererfahrung
- Jeder Proband wurde einmal befragt

# Ausführung

- Vorher Aufklärung, Erlaubnis über Audioaufzeichung und e-Mail-Kontakt
- Ca. 1 Stunde pro Proband

# Auswertung-Anfänger

- Zusammenfassung der Antworten der Probanden in 3 Kategorien
  - Approaching Exception Handling (ignore)
  - Using Exception Handling (debugging)
  - Perceiving Exception Handling (forced)

# Auswertung-Experten

- Zusammenfassung der Antworten der Probanden in 3 Kategorien
  - Approaching Exception Handling (handle right away)
  - Using Exception Handling (failing gracefully in unexpected situations)
  - Perceiving Exception Handling (forced)
- Strategie von Anfängern:
  - Ignorieren
  - Verallgemeinern
  - Fehlendes Logging/unvollständiges weiterreichen

# Literatur

- Shah, Görg, and Harrold. Understanding Exception Handling: Viewpoints of Novices and Experts. TSE, 36(2), pp. 150-161, 2010.
- Nachfolgeexperimente:
  - Shah and Harrold. Exception Handling Negligence Due To Intra-Individual Goal Conflicts. CHASE, pp. 80-83, 2009.
  - H. Shah, Görg, and Harrold. Visualization of exception handling constructs to support program understanding. In *Proceedings of the 4th ACM Symposium on Software Visuallization,* pages 19–28, Sep 2008.
  - Shah, Görg, and Harrold. Why do developers neglect exception handling? In *Proceedings of the 4th International Workshop on Exception Handling, pages 62–68,* Nov 2008.

# The Relevance of Application Domain Knowledge: The Case of Computer Program Comprehension

# Objective

- Herausfinden, wie Vorhandensein von Domänenwissen Programmverständnis beeinflusst
- Hypothese:
  - Domänenwissen vorhanden -> Top-Down-Verständnis
  - Domänenwissen nicht vorhanden -> Bottom-Up-Verständnis

# Variablen

- Unabhängig: Domänenwissen in 2 Stufen, gemessen durch Selbsteinschätzung
  - Vorhanden (Buchhaltung)
  - Nicht vorhanden (Hydrologie)
- Abhängig: Programmverständnis
- Störvariablen:
  - Programmiersprache (COBOL als bekannte Sprache)
  - Komplexität des Programms (basierend auf LOC)
  - Reihenfolge

# Material/Aufgabe

- 3 Programme in COBOL
  - Einführungsprogramm
  - Programm aus Buchhaltung (bekannte Domäne)
  - Programm aus Hydrologie (unbekannte Domäne)

- Aufgabe
  - Quelltext verstehen, dabei Gedanken laut aussprechen

# Exkurs: Think aloud

- Probanden sprechen aus, was sie denken
- Protokollierung der Daten (schriftlich, audio, video und/oder screen capture)
- Schwierigkeiten:
  - Probanden versuchen, Quelltext zu erklären, statt ihre Gedanken auszusprechen
  - Probanden sprechen irrelevant erscheinende Gedanken nicht aus
  - Versuchsleiter muss ohne Beeinflussung eingreifen, wenn Proband nicht redet
  - Objektive Auswertung der Daten

# Probanden

- 24 professionelle Entwickler in Buchhaltungsdomäne, erfahren in COBOL

# Design

- 1-faktoriell, within-subjects
- Crossover

| Gruppe | Session 1 | Session 2 |
|---|---|---|
| A | kein Domänenwissen | Domänenwissen |
| B | Domänenwissen | kein Domänenwissen |

# Ausführung/Deviation

- Warm-Up-Aufgabe, um an Experiment zu gewöhnen
- Alle Probanden einzeln
- Keine Angabe, ob Audio- oder Videorecording

# Auswertung (1)

- Entwicklung eines Kodierhandbuchs, was Hypothese (Top-Down) und was Inferenz (Bottom-Up) ist
- Training von 2 unabhängigen Reviewern, die Statements als Hypothese oder Inferenz klassifizieren

# Auswertung (2)

- Übereinstimmung mittels Cohen's Kappa
- R:
  - `install.packages("psych")`
  - `library(psych)`
  - `reviewer1 <- c("i", "i", "h", "i")`
  - `reviewer2 <- -c("i", "h", "h", "i")`
  - `cohen.kappa(cbind(reviewer1, reviewer2))`
- Bei verschiedenen Ansichten Diskussion beider Reviewer mit Erstautor (unklar, ob Kappa vor oder nach Diskussion berechnet wurde)

# Literatur

- Think-Aloud-Protokoll:
  - Ericson, K.A. & Simon, H.A. (1980). Verbal Reports as Data. *Psychological Review, 87, 215–251.*
- Program Comprehension:
  - Teresa Shaft and Iris Vessey. The Relevance of Application Domain Knowledge: The Case of Computer Program Comprehension. Information Systems Research, 6(3):286–299, 1995.
  - M. P. O'BRIEN, J. BUCKLEY AND T. M. SHAFT. Expectation-based, inference-based, and bottom-up software comprehension. J. Softw. Maint. Evol.: Res. Pract. 2004; 16:427–447. (Nachfolgeexperiment zum besprochenen Artikel)