

Akzeptanztests

Echtzeitsysteme 2 - Vorlesung/Übung

Peter Ulbrich
Fabian Scheler
Wolfgang Schröder-Preikschat

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander Universität Erlangen-Nürnberg

<http://www4.cs.fau.de/~{scheler,ulbrich,wosch}>
{ulbrich,scheler,wosch}@cs.fau.de



1

Übersicht

- Grundlegende Techniken
- Szenariobasierte Akzeptanztests
- Testen von verteilten Echtzeitsystemen
- Zusammenfassung



© {ulbrich, scheler, wosch}@cs.fau.de - EZL (SS 2010)

2

Grundlegende Techniken

- **Verfolgbarkeit**
 - Beziehung von Testfällen und Anforderungen
- **Formale Methoden**
 - Generierung von Testfällen
 - *Requirements Language Processor*
- **Prototypen**
 - Evaluation von Prototypen
- **andere Methoden**
 - statistische Methoden, Structured Analysis, Simulation



© {ulbrich, scheler, wosch}@cs.fau.de - EZL (SS 2010)

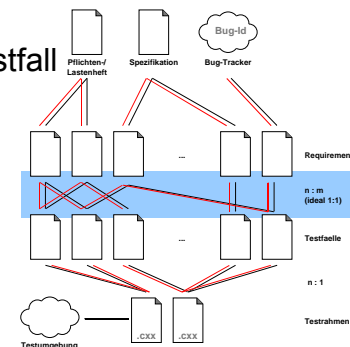
3

Verfolgbarkeit

- **verfolgbare** Software Requirements Specification (SRS)
 - eine Anforderung je Absatz
 - direkte Nummerierung der Anforderungen
 - Anforderungen werden z.B. mit „shall“ eingeleitet

- **Abbildung: Anforderungen ↔ Testfall**

- ✓ mind. 1 Test / Anforderung
- ✗ Abbildung ist nicht bijektiv
- ✗ Abbildung ist subjektiv

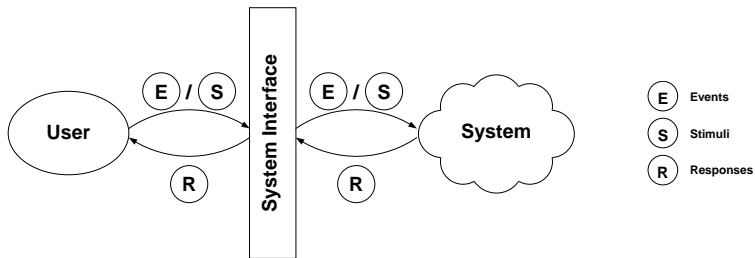


© {ulbrich, scheler, wosch}@cs.fau.de - EZL (SS 2010)

4

Szenarioanalyse

Systemstruktur

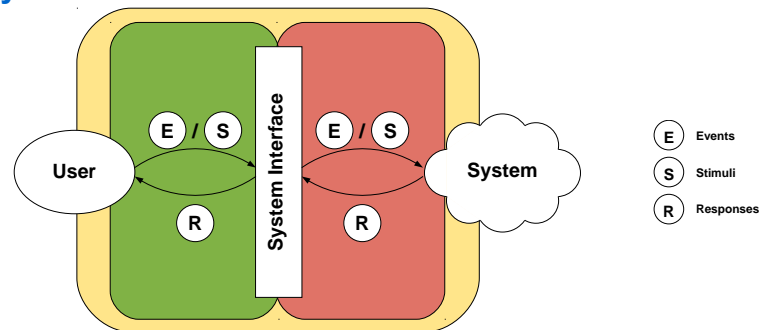


- Benutzer kommuniziert durch **Ereignisse / Stimuli**
- System reagiert mit **Antwort** bzw. **Zustandsänderung**
- gültige Folge von Eingaben: **Szenario**



Benutzersicht und Schnittstellensicht

Systemstruktur



- Benutzersicht**
- Schnittstellensicht**
- externe Systemsicht**



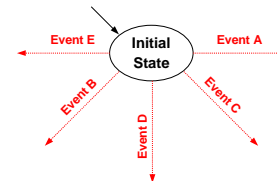
Szenariobäume

1. initialer Systemzustand



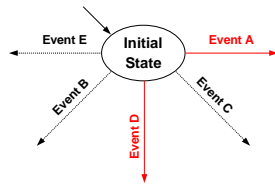
Szenariobäume

1. initialer Systemzustand
2. **Domänenexperte:**
mögliche Ereignistypen



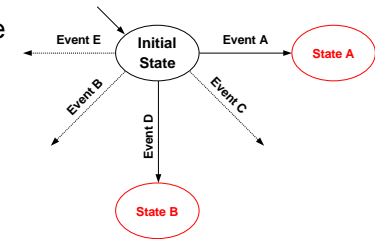
Szenariobäume

1. initialer Systemzustand
2. **Domänenexperte:** mögliche Ereignistypen
3. **Benutzer:** Eingabeereignisse



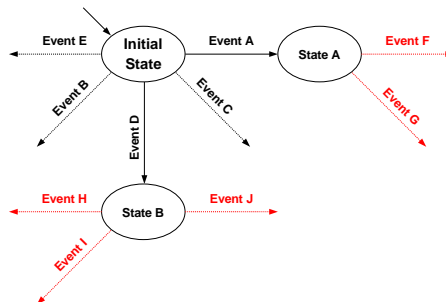
Szenariobäume

1. initialer Systemzustand
2. **Domänenexperte:** mögliche Ereignistypen
3. **Benutzer:** Eingabeereignisse
4. neue Zustände



Szenariobäume

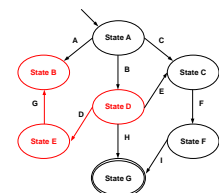
1. initialer Systemzustand
2. **Domänenexperte:** mögliche Ereignistypen
3. **Benutzer:** Eingabeereignisse
4. neue Zustände
5. zurück zu Schritt 2



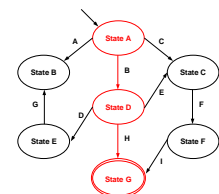
- eher DFA als Baum
- bis zu einem Fixpunkt
- Zustände sind durch den Benutzer beobachtbar
- Tracing von Benutzereingaben
- Pfad des DFA ↔ Szenario

Basispfade

- Pfad über mind. 1 Kante
- gültige Szenarien
 - bestehen aus Basispfaden
 - beginnen an einem Startzustand
 - enden an einem Endzustand
- Testfallgenerierung
 - für jedes gültige Szenario
 - mit Hilfe der Basispfade



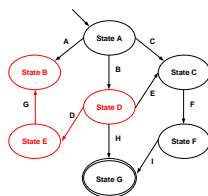
Basispfad



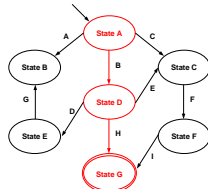
gültiges Szenario

Basispfade

- Pfad über mind. 1 Kante
- gültige Szenarien
 - bestehen aus Basispfaden
 - beginnen an einem Startzustand
 - enden an einem Endzustand
- Testfallgenerierung
 - für jedes gültige Szenario
 - mit Hilfe der Basispfade
- ✓ Vorteil
 - Einbeziehung des Benutzer
 - Entwickler/Tester müssen keine Domänenexperten sein



Basispfad

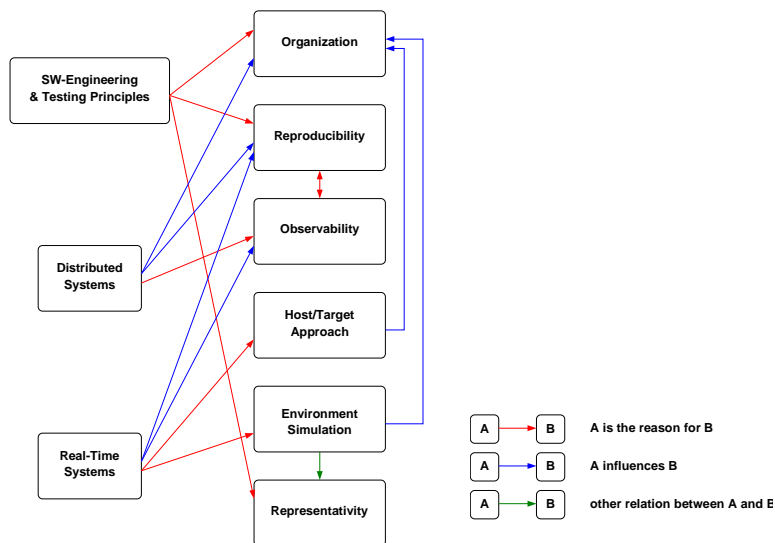


gültiges Szenario

Testen von verteilten Echtzeitsystemen

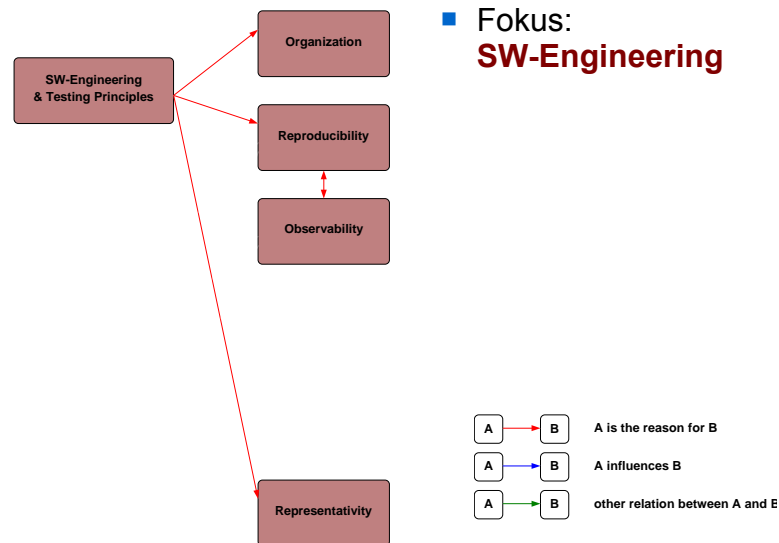
- Herausforderungen beim Testen von
 - **Echtzeitsystemen**
 - starke **Kopplung zur Umgebung**
 - **Voranschreiten der realen Zeit** ist nicht vernachlässigbar
 - **Umgebung kann nicht gezielt gesteuert werden**
 - **verteilten Systemen**
 - hohe **Komplexität**
 - **Beobachtung** des Systems ist schwierig
 - **Reproduzierbarkeit** des Systemverhaltens
 - **fehlende globale Zeit**: kein eindeutiger globaler Zustand

Problemfeld Testen

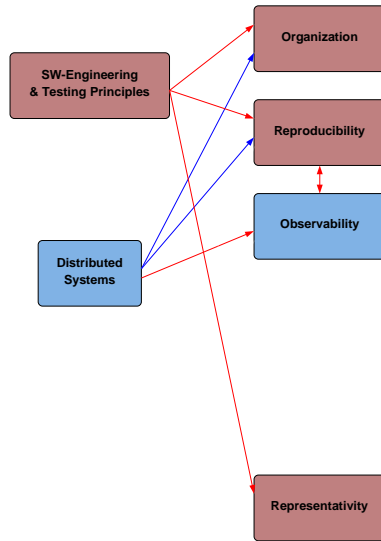


Problemfeld Testen

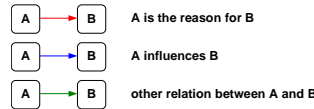
- Fokus: **SW-Engineering**



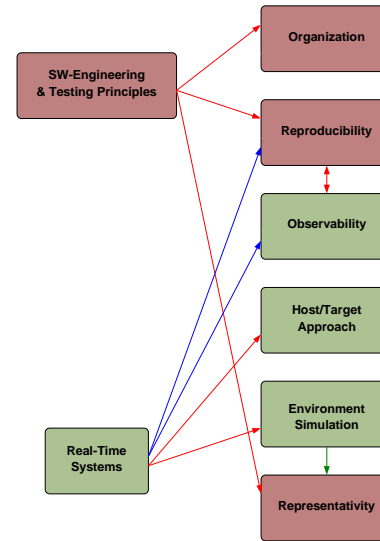
Problemfeld Testen



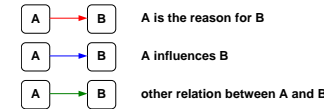
■ Fokus: **verteilte Systeme**



Problemfeld Testen



■ Fokus: **Echtzeitsysteme**



Organisation

Entwicklungsmethodologie

- Entwicklungsphasen
- Zwischenprodukte
 - Evaluation
 - Eingabe für die nächste Entwicklungsphase

Testen

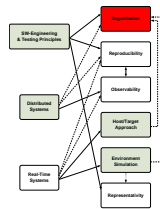
- folgt den Entwicklungsphasen
- z.B. Modul-, Integrations- und Systemtest

Echtzeitsysteme

- Entwicklungsmethodologie aber keine Testmethodologie

verteilte System

- höhere Komplexität
- Verteilung beeinflusst Organisation und Anzahl der Testphasen



Beobachtbarkeit

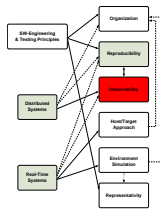
- Verhalten des Systems und der Umwelt muss erfasst werden können

Möglichkeiten der Beobachtung

- Ausgaben bzw. Ergebnisse
- Zwischenzustände und -ergebnisse

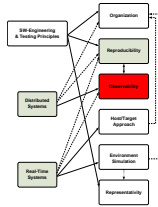
→ Zwischenzustände über

- zusätzliche Ausgaben (→ häufiges Übersetzen)
- Debugger



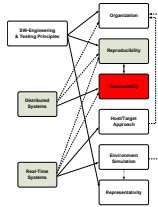
Probleme in verteilten Systemen

- **Ausgaben** beeinflussen das System
 - Ausgaben verzögern evtl. Prozesse
 - Nachrichtenreihenfolgen ändern sich
 - Timeout
 - ...
- **Debuggen**: globale Breakpoints unmöglich
 - perfekt synchronisierte, globale Uhr existiert nicht
 - wie soll man Prozesse gleichzeitig anhalten?
- bekanntes Phänomen: **Probe Effect**
 - muss vermieden oder kompensiert werden



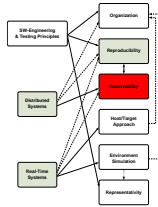
Probleme in Echtzeitsystemen

- neben den Ereignissen selbst muss auch der Zeitpunkt ihres Auftretens vermerkt sein
 - Verschärfung des Probe Effects
- verteilte Systeme
 - Probe Effect durch unabhängige Prozesse
- Echtzeitsysteme
 - Probe Effect durch voranschreiten der Zeit
 - auch einzelne Prozesse sind betroffen



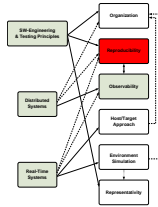
Probe Effect: Lösungsmöglichkeiten

- **Ignoranz**
 - der Probe Effect wird schon nicht auftreten
- **Minimierung**
 - Datenaufzeichnung hinreichend effizient
 - Kompensation der aufgezeichneten Daten
- **Vermeidung**
 - hinter der logischen Zeit verbergen
 - Datenaufzeichnung existiert auch im Produktivsystem
 - dedizierte Hardware für die Datenaufzeichnung



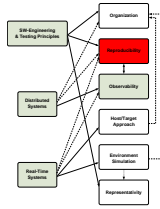
Reproduzierbarkeit

- **Regressionstests**
 - wurden die Fehler wirklich korrigiert
 - hat die Korrektur keine neuen Fehler erzeugt
- Voraussetzung: **Reproduzierbarkeit**
 - andernfalls: keine Aussage, ob der Fehler korrigiert wurde
 - dasselbe Symptom kann verschiedene Ursachen haben
- Reproduzierbarkeit besteht aus
 - **Beobachtbarkeit** aller relevanten Ereignisse
 - **Kontrollierbarkeit** des Systems



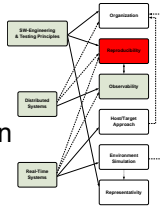
Probleme – Beobachtbarkeit

- **nicht-deterministische** Operationen
 - Abhängigkeiten z.B. vom Netzwerkverkehr
 - Zufallszahlen
- **ungenügendes** Vorabwissen
 - Fadensynchronisation
 - (asynchrone) Unterbrechungen
 - Zeitbasis der Systeme
- das sind **relevante Ereignisse**
 - sie beeinflussen den Programmablauf
 - hängen von der Anwendung ab
 - müssen identifiziert und beobachtet werden



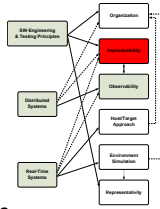
Kontrollierbarkeit

- **Abspielen der relevanten Ereignisse**
 - Einhaltung der Reihenfolge
 - zeitlich akkurat
 - schließt Unterbrechungen und externe Ereignisse ein
- **Simulierte Zeit** statt **realer Zeit**
 - Zeitbasis muss sich nicht deterministisch verhalten
- Ansätze zur Kontrollierbarkeit
 - **sprachbasierter** Ansatz
 - **implementierungsbasierter** Ansatz

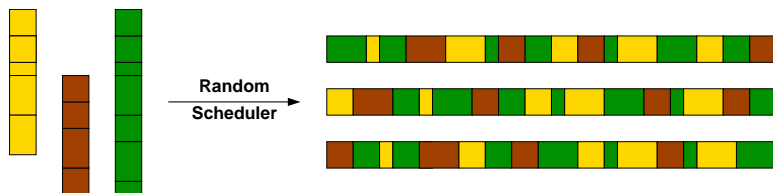


sprachbasierter Ansatz

- **statische Quelltextanalyse**
 - Identifizierung von Ausführungsszenarios
- **Erzwingen der Ausführungsszenarios**
 - **Random Scheduler**
 - nebenläufiges Programm → sequentielles Programm
 - teste Sequentialisierungen statt nebenläufigem Programm

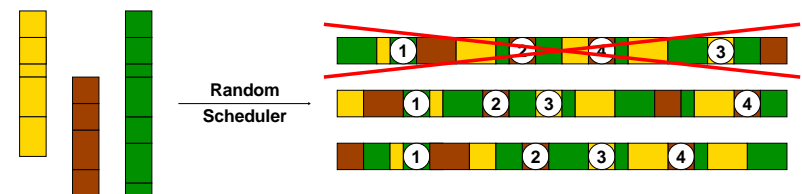
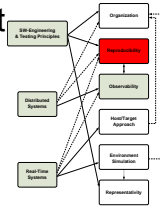


→ Vorgehen ist mit **Model Checking** vergleichbar



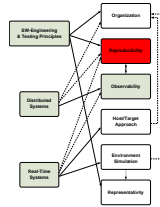
sprachbasierter Ansatz

- welche Sequentialisierungen werden getestet
 - (1) gemeinsame Daten werden synchronisiert
 - (2) Synchronisationssequenz ist eindeutig
 - alle Sequentialisierungen sind äquivalent
- (1) ist gegeben, (2) muss erzwungen werden



sprachbasierter Ansatz

- Erzwingen der Sequenz
 - Brich Hansen: atomare Monitor-Aufrufe
 - werden von **nebenläufigen Prozessen**
 - an **definierten logischen Zeitpunkten** getätigt
 - Quelltexttransformationen für
 - SEND/RECEIVE
 - Ada: Rendezvous
- Abhängigkeit vom Voranschreiten der Zeit
 - Zeitserver
 - Zugriff auf simulierte Zeit

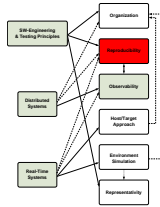


x asynchrone Unterbrechungen nicht adäquat behandelbar



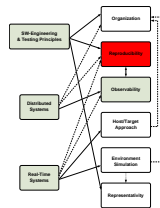
implementierungsbasierter Ansatz

- **Monitoring zur Laufzeit**
 - Aufzeichnung der relevanten Ereignisse
 - *Event Histories* bzw. *Traces*
- erneutes Abspielen dieser Traces
- ✓ existierende Lösungen für verteilte Echtzeitsysteme:
 - **vermeiden** Probe Effect
 - decken **Vielzahl von Ereignissen** ab
 - Systemaufrufe, Kontextwechsel, **asynchrone Unterbrechungen**, Synchronisation, Zugriff auf gemeinsame Variablen, ...



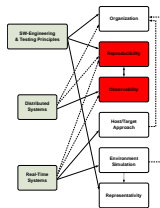
implementierungsbasierter Ansatz

- x **Spezialhardware** notwendig
- x **große Datenmengen** fallen an
 - Traces auf Maschinencodeebene
 - Eingaben, Zwischenergebnisse
- x **nur beobachtete Szenarien** können wiederholt werden
 - man kann nicht alles beobachten
 - evtl. passt der Trace nicht mehr zum geänderten System
- x Wiederholung & Aufzeichnung: auf demselben System
 - **Target wird benötigt**



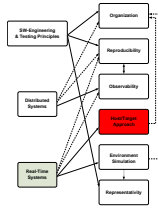
Beobachtbarkeit & Reproduzierbarkeit

- initiale Monitoringphase → **Probe Effect**
- Wiederholung → **Probe Effect**
- **Probleme**
 - getestetes Szenario ist anders
 - Wiederholung durch Probe Effect verfälscht
- **Annahme**: Probe Effect
 - x ist zwar **nicht-deterministisch** und
 - x **verändert Reihenfolge** und/oder **Timing** relevanter Ereignisse,
 - ✓ erzeugt aber (meistens) **keine neuen relevanten Ereignisse**.
- Probe Effect wird bei der Wiederholung reproduziert



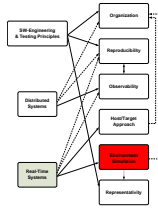
Host/Target Approach

- **Zielsystem** (engl. *Target*)
 - ist nicht direkt zugreifbar (z.B. Atomkraftwerk)
 - verfügt nicht über genügend Ressourcen
- Entwicklung: **Wirtssystem** (engl. *Host*)
 - Cross-Compiler
- Frage: **Wo testet man was?**
 - Modultests → Wirtssystem
 - Integrations- und Systemtests → Zielsystem
- **Problem:** wenig Testunterstützung auf dem Zielsystem
 - **Emulation** des Zielsystems



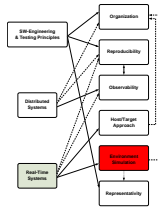
Umgebungssimulation

- engl. *Environment Simulator*
- ✓ **Vorteile**
 - Tests **ohne reales Objekt**
 - Tests von **Ausnamesituationen** (engl. *rare event situations*)
 - Simulation erlaubt
 - **gezieltere Kontrolle** als das physikalische Objekt
 - **einfachere Beobachtung** von Ein- und Ausgaben
 - **Erzeugung der Testdaten** bei der Testausführung
 - Korrektheit hängt von einer Sequenz von Ein- und Ausgaben ab
 - realistische Testdaten schwer bestimmbar



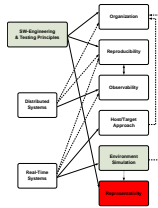
Umgebungssimulation

- ✗ **Nachteile**
 - hochgradig **anwendungsspezifisch**
 - zu simulierende **Umwelt ist sehr komplex**
 - **Validierung** des Simulators



Repräsentativität

- System sollte getestet werden
 - mit **realistischen Eingaben**
 - mit **realistischen Anwendungsszenarien**
- Problematik
 - (1) nicht alle Szenarien werden nachgestellt
 - ungenügendes Wissen über die Umwelt
 - Entwurf lässt bestimmte Szenarien außer acht (bewusst oder unbewusst)
 - (2) kombinatorische Explosion: zu viele Szenarien
- Verfahren
 - (1) ist **unbekannt** und kann nicht **gemessen** werden
 - (2) **Test Coverage:** wieviel wird durch die Tests abgedeckt
 - **Achtung:** Aussagen über Coverage sind **immer relativ niemals absolut**



Zusammenfassung

- Grundlegende Techniken
 - Verfolgbarkeit, formale Methoden, Prototypen
- Szenariobasierte Akzeptanztests
 - Szenarioanalyse, Szenariobäume
- Test-Driven Development
- Testen von verteilten Echtzeitsystemen
 - SW Engineering vs. verteilte Systeme vs. Echtzeitsysteme
 - Problemfeld
 - Organisation
 - Beobachtbarkeit & Reproduzierbarkeit
 - Host/Target-Approach
 - Umgebungssimulation
 - Repräsentativität



Literatur

- *Pei Hsia, David Kung, and Chris Sell.*
Software requirements and acceptance testing.
Anal. of Software Engineering, 3(0):291-317, 1997.
- *Werner Schütz.*
Fundamental issues in testing distributed real-time systems.
Real-Time Systems Journal, 7(2):129-157, 1994.

