Akzeptanztests

Echtzeitsysteme 2 - Vorlesung/Übung

Fabian Scheler
Michael Stilkerich
Wolfgang Schröder-Preikschat

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

http://www4.cs.fau.de/~{scheler,mike,wosch}
{scheler,mike,wosch}@cs.fau.de

Übersicht

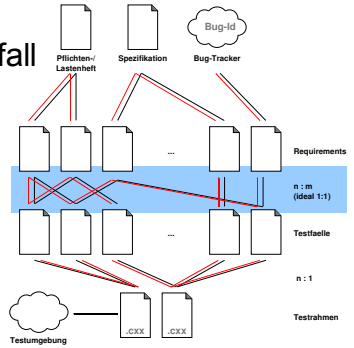
- Grundlegende Techniken
- Szenariobasierte Akzeptanztests
- Test-Driven Development (TDD)
- Testen von verteilten Echtzeitsystemen
- Zusammenfassung

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

Verfolgbarkeit

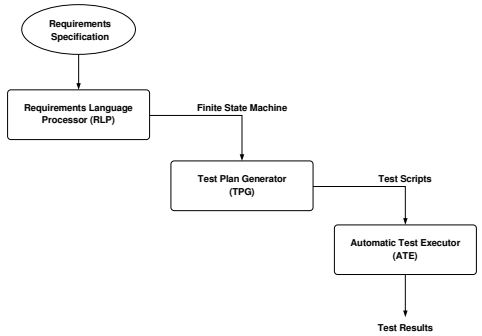
- **verfolgbare** Software Requirements Specification (SRS)
  - eine Anforderung je Absatz
  - direkte Nummerierung der Anforderungen
  - Anforderungen werden z.B. mit „shall“ eingeleitet
- Abbildung: Anforderungen ↔ Testfall



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

Formale Methoden

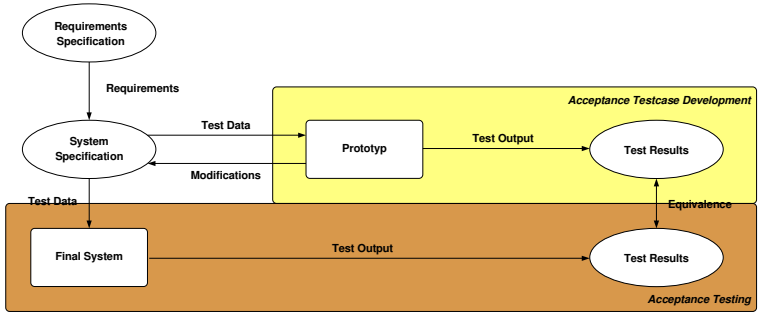
- Voraussetzung: **formale Beschreibung** der Anforderungen



- Zustandsmaschine: verhält sich wie das reale System
- ✔ Anforderungen: **vollständig**, **konsistent** und **eindeutig**
- ✗ geringe Akzeptanz formaler Methoden

Prototypen

- Entwicklung der Tesfälle gegen einen Prototypen
- Prototyp wird an das gewünschte Verhalten angepasst



- ✔ Erprobung neuer, riskanter Entwicklungsmethoden

andere Methoden

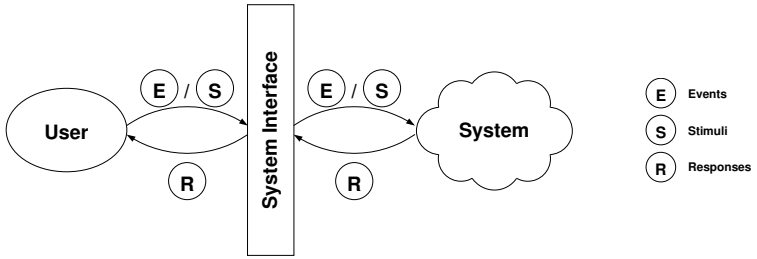
- **Zuverlässigkeitsmaße** für Software
  - Wahrscheinlichkeit für das Auftreten von Fehlern (MTBF)
  - Spezifikation von Benutzungsprofilen und Fehlern
  - Testfälle werden anhand des Benutzungsprofils erstellt
- **Structured Analysis**
  - existierende Systeme ohne SRS
  - Testfälle werden anhand des CFG und DFG erstellt
- **Simulation**
  - für Systeme, deren reale Interaktion nicht getestet werden kann
  - Simulation der Umwelt

Gemeinsame Nachteile

- ✗ der spätere Benutzer wird nicht einbezogen
  - Testfälle aus Sicht des Entwicklers
  - Subjektivität: Testfälle niedriger Qualität
- ✗ Testfallentwicklung: separater Prozess
  - höherer Aufwand (Personal, Zeit und Geld)

Szenarioanalyse

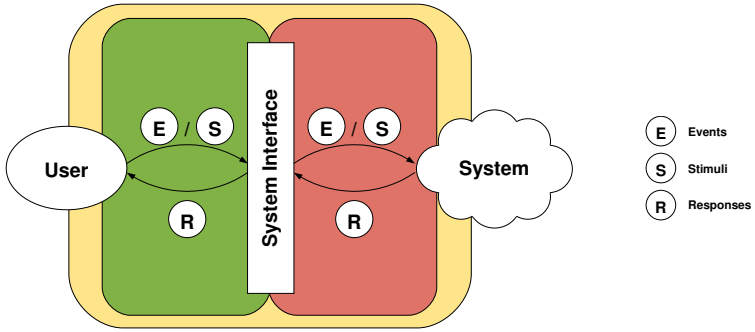
- Systemstruktur



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

Benutzersicht und Schnittstellensicht

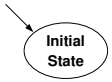
- Systemstruktur



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

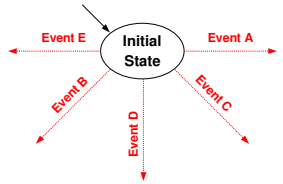
Szenariobäume

1. initialer Systemzustand



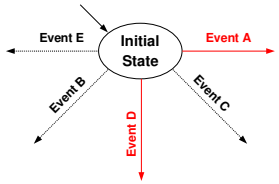
Szenariobäume

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



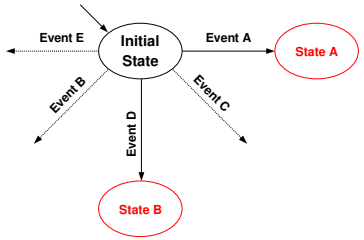
Szenariobäume

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



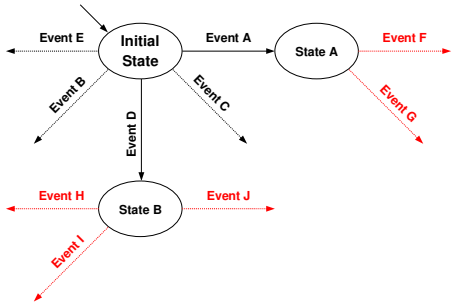
Szenariobäume

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



Szenariobäume

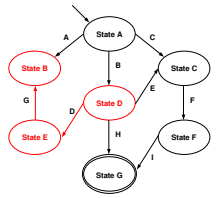
1. initialer Systemzustand
2. **Domänenexperte**: mögliche Ereignistypen
3. **Benutzer**: Eingabeereignisse
4. neue Zustände
5. zurück zu Schritt 2



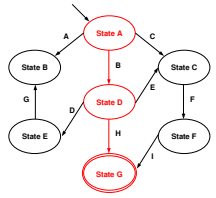
- eher DFA als Baum
- bis zu einem Fixpunkt
- Zustände sind durch den Benutzer beobachtbar
- Tracing von Benutzereingaben
- Pfad des DFA ↔ Szenario

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



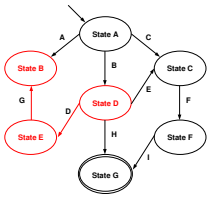
Basispfad



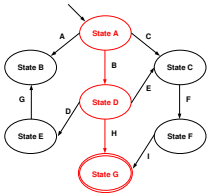
gültiges Szenario

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
  - Entiwckler/Tester müssen keine Domänenexperten sein



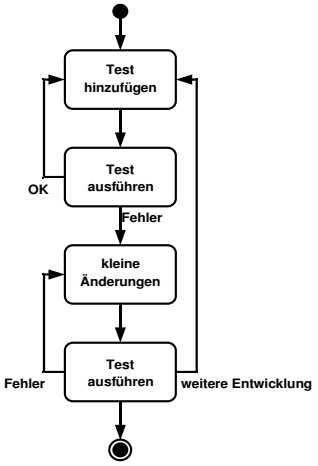
Basispfad



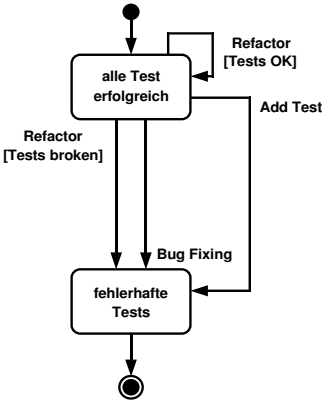
gültiges Szenario

TDD - Konzept

- Test First Development (TFD) + Refactoring



TFD



TDD

TDD - Idee

- TDD ist Methode des **Extreme Programming**
- zyklisches Entwicklungsmodell
  - klassisch: Requirements, Design, Implementierung, Test getrennt
  - hier: verwoben
- Tests sind Bestandteil des Abnahmetests

➔ klassisch:
- Anstieg erfüllter Anforderungen in der Testphase

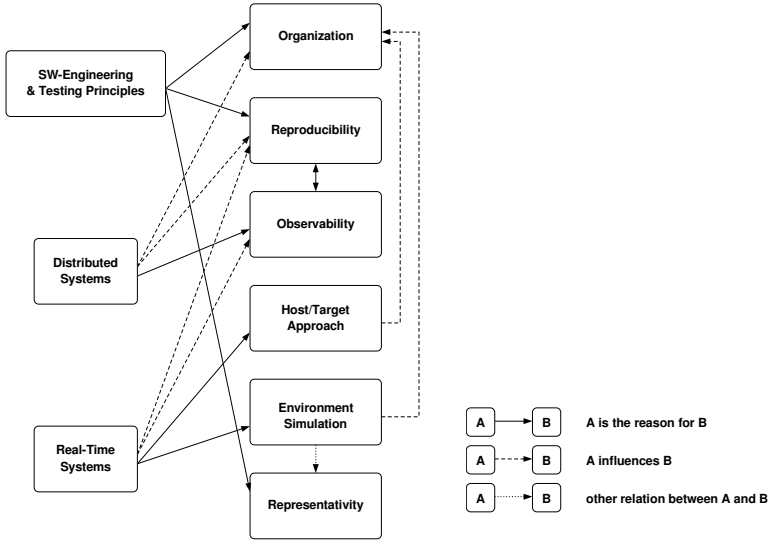
➔ hier:
- linearer Anstieg erfüllter Anforderungen während der Entwicklung

- eine Art ***integrierte System-*** und ***Abnahmetest***

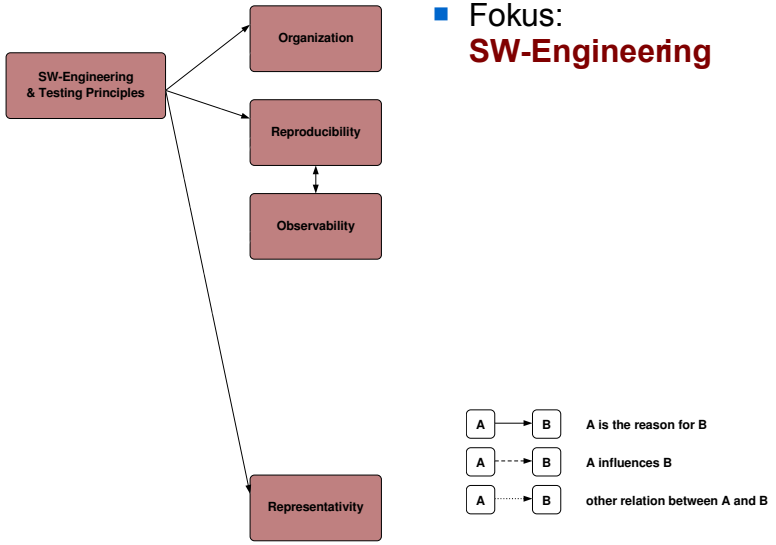
Testen von verteilten Echtzeitsystemen

- Herausforderungen beim Testen von
  - **Echtzeitsystemen**
    - starke Kopplung zur Umgebung
    - voranschreiten der realen Zeit ist nicht vernachlässigbar
    - die 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

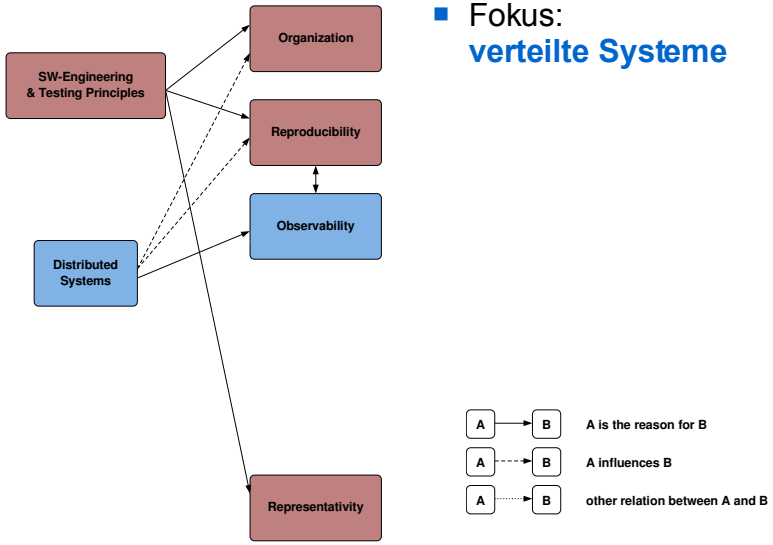


Problemfeld Testen



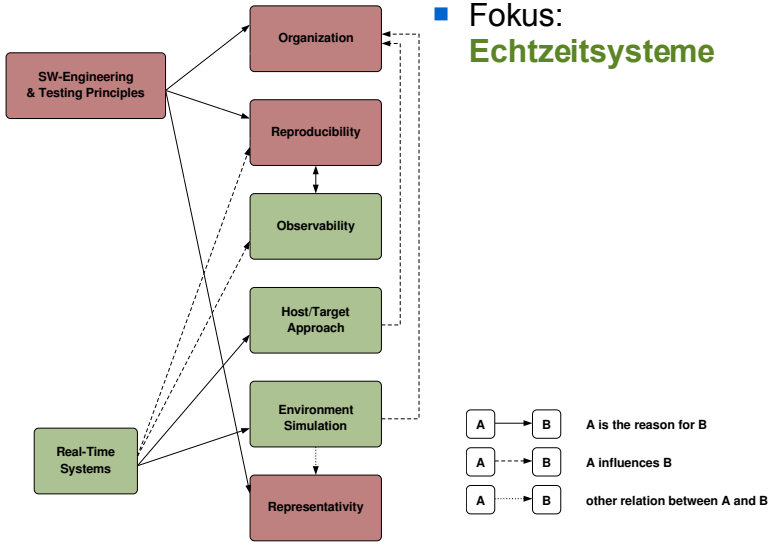
- Fokus:
  SW-Engineering

Problemfeld Testen



- Fokus:
  verteilte Systeme

Problemfeld Testen

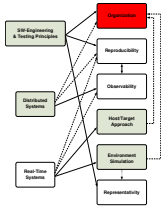


- Fokus:
  Echtzeitsysteme

Organisation

- Entwicklungsmethodologie
  - Entwicklungsphasen
  - Zwischenprodukte
    - Evalution
    - 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

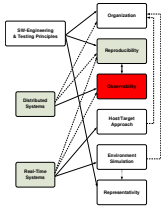


Beobachtbarkeit

- 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

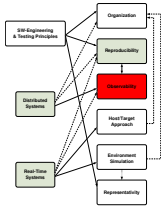
➔ ...

- 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



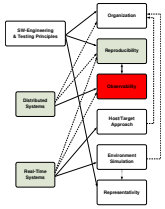
Probleme in Echtzeitsystemen

- neben den Ereignissen selbst muss auch der Zeitpunkt ihre Auftretens vermerkt sein

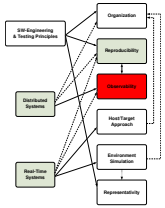
➔ Verschärfung des Probe Effects

- verteilte Systeme
  - Probe Effect durch unbhängige Prozesse
- Echtzeitsysteme
  - Probe Effect durch voranschreiten der Zeit

➔ auch einzelne Prozesse sind betroffen

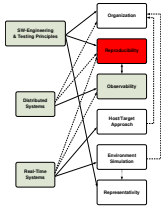


Probe Effect: Lösungsmöglichkeiten



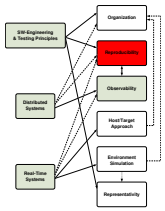
- **Ignoranz**
  - der Probe Effect wird schon nicht auftreten
- **Minimierung**
  - Datenaufzeichnung hinreichend effizient
  - Kompensation der aufgezeichneten Daten
- **Vermeidung**
  - hinter der logischen Zeit verbergen
  - Datenaufzeichnung existiert auch im Produktivsystem
  - dedizierte Hardware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

Reproduzierbarkeit



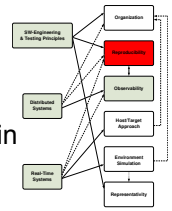
- Regressionstests
  - wurden die Fehler wirklich korrigiert
  - hat die Korrektur keine neuen Fehler erzeugt
- Voraussetzung: **Reproduzierbarkeit**
  - andernfalls: keine Aussage, ob der Fehler korrigiert wurde
  - dasselbe Symptom kann verschiedene Ursachen haben
- Reproduzierbarkeit besteht aus
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

Probleme – Beobachtbarkeit



- **nicht-deterministische** Operationen
  - Abhängigkeiten z.B. vom Netzwerk-Verkehr
  - 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



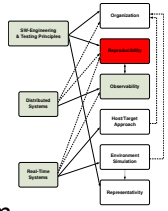
- Abspielen der relevanten Ereignisse
  - Einhaltung der Reihenfolge
  - zeitlich akkurat
  - schließt Unterbrechungen und externe Ereignisse ein

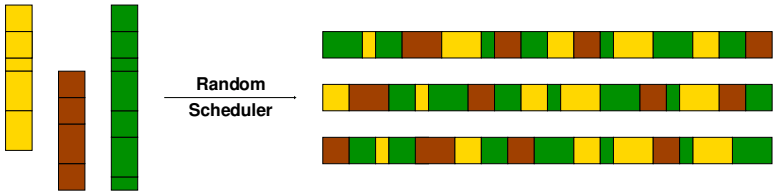
➔ *simulierte* statt *reale* Zeit
  - Zeitbasis muss sich nicht deterministisch verhalten

- Ansätze zur Kontrollierbarkeit
  - **sprachbasierter** Ansatz
  - **implementierungsbasierter** Ansatz

sprachbasierter Ansatz

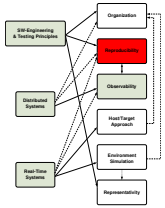
- statische Quelltextanalyse
  - Identifizierung von Ausführungsszenarios
- Erzwingen der Ausführungsszenarios
  - *Random Scheduler*
  - nebenläufiges Programm → sequentielles Programm
  - teste Sequentialisierungen statt nebenläufigem Programm

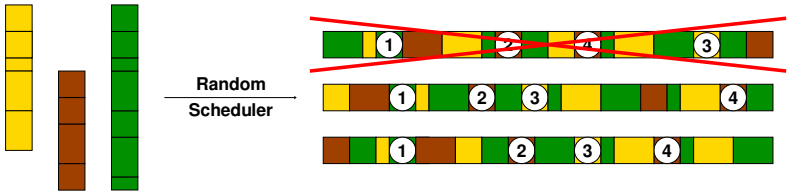




sprachbasierter Ansatz

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

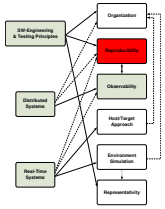




sprachbasierter Ansatz

- 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 Vorranschreiten der Zeit
  - Zeitserver
  - Zugriff auf simulierte Zeit

✗ asynchrone Unterbrechungen nicht adäquat behandelbar

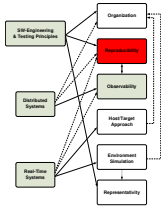


implementierungsbasierter Ansatz

- Monitoring zur Laufzeit
  - Aufzeichnung der relevanten Ereignisse
  - *Event Histories* bzw. *Traces*

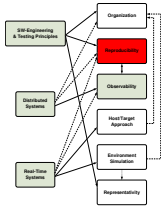
➔ erneutes Abspielen dieser Traces

✔ existierende Lösungen für verteilte Echtzeitsysteme:
  - **vermeiden** Probe Effect
  - decken **Vielzahl von Ereignissen** ab
    - Systemaufrufe, Kontextwechsel, **asynchrone Unterbrechungen**, Synchronisation, zurgriff auf gemeinsame Variablen, ...



implementierungsbasierter Ansatz

✗ **Spezialhardware** notwendig



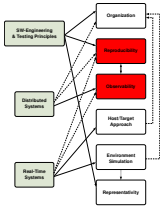
✗ **große Datenmengen** fallen an
- Traces auf Maschinencodeebene
- Eingaben, Zwischenergebnisse

✗ **nur beobachtete Szenarien** können wiederholt werden
- man kann nicht alles beobachten
- evtl. passt der Trace nicht mehr zum geänderten System

✗ Wiederholung & Aufzeichnung: auf demselben System
- **Target wird benötigt**

Beobachtbarkeit & Reproduzierbarkeit

- initiale Monitoringphase → **Probe Effect**
- Wiederholung → **Probe Effect**



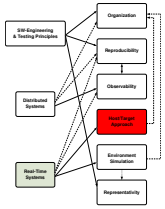
➔ **Probleme**
- getestetes Szenario ist anders
- Wiederholung durch Probe Effect verfälscht

- **Annahme**: Probe Effect
  - ✗ ist zwar **nicht-deterministisch** und
  - ✗ **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



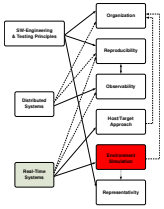
➔ Entwicklung: **Wirtssystem** (engl. *Host*)
- Cross-Compiler

➔ Frage: Wo testet man was?
- Modultests → Wirtssystem
- Integrations- und Systemtests → Zielsystem

- Problem: wenig Testunterstützung auf dem Zielsystem
  - ➔ **Emulation** des Zielsystems

Umgebungssimulation

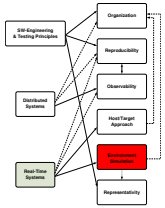
- engl. *Environment Simulator*



✔ **Vorteile**
- Tests **ohne *reales Objekt***
- Tests von **Ausnamesituationen** (engl. *rare event situations*)
- Simulator 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

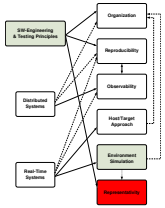
Umgebungssimulation

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



Repräsentatitivität

- 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 verteilen 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.
  Anals 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.