

Design für Testbarkeit

Stefan Jungmayr

FernUniversität Hagen, Praktische Informatik III
Feithstrasse 142, D 58084 Hagen
stefan.jungmayr@fernuni-hagen.de

STI-Jahrestagung

Kaiserslautern, 26. November 2000

Übersicht

- **Motivation**
- **Testbarkeit**
- **Testbarkeitsbewertung**
- **Konstruktive Ansätze**
- **Design für Testbarkeit**
- **Design für Testbarkeit - Strategie**
- **Zusammenfassung**

Motivation (I)

Testproblematik:

- **steigende Komplexität der Software**
- **Qualitätsanforderungen bei zunehmender Kritikalität**
- **Ressourcenbeschränkung: Aufwand, Zeit**

Lösungsansätze:

- **verbesserte Testkriterien**
- **Testautomatisierung**
- **automatisierte Testorakel**
- **Design für Testbarkeit**

Motivation (II)

	Ansätze zur Testkostenreduktion				wiederkehrende Einsparungen		
	Test-kriterien	Testauto-matisierung	automat. Orakel	Design für Testbarkeit	Erst-Test	Regr.test in Entwickl.	Regr.test in Wartung
Testkosten							
Testfallentwicklung	++			+	++	+	
Testausführung		++		+	+	++	++
Testauswertung		+	++	+	+	+	++
Fehlerlokalisierung				+	+		
Fehlerbeseitigung				+	+		

Motivation (III)

Interesse an Testbarkeit:

- **Manager:**
 - Bewertung der Testbarkeit als Entscheidungsgrundlage
 - Bereitstellung von Ressourcen, Schulungen
- **Entwickler:**
 - Design für Testbarkeit
- **Tester:**
 - Identifizierung von Testbarkeitsmängeln (Reviews)
 - Richtlinien

Testbarkeit (I)

Definition:

Testbarkeit ist der Grad, zu dem eine Softwareentität in einem bestimmten Kontext den Testvorgang ermöglicht.

Softwareentitäten:

- Anforderungen
- Architektur
- Entwurf
- Implementierung

Testbarkeit (II)

Testbarkeitsaspekte von Softwareentitäten:

- **Anforderungen:**
 - schriftliche Formulierung
 - testbare Formulierung
 - Aktualität
 - Stabilität
- **Architektur, Entwurf, Implementierung:**
 - Komplexität
 - Zerlegbarkeit
 - Verifizierbarkeit
 - Stabilität

Testbarkeit (III)

Testkontext:

- **Test-Rahmenbedingungen**
 - Budget des Testprozesses
 - geforderte Produktqualität
- **Geplante Nutzung der Software**
 - einmalig innerhalb einer bestimmten Anwendung
 - mehrmals (Wiederverwendung)
- **Angewandte Testkriterien**
- **Eingesetzte Testwerkzeuge**

Testbarkeitsbewertung (I)

0) Ist bekannt, was getestet werden soll?

1) **Gibt es Überflüssiges, dessen Test vermieden werden kann?**

- **Funktionalität, Größe, (Schnittstellen-)Komplexität, Überdeckbarkeit**

2) **Kann das Problem zerlegt werden?**

- **Werden viele andere Komponenten für den Test benötigt?**
- **Ist die Komponente steuerbar? Ist es einfach, Testresultate zu beobachten?**
- **Kann bei Testproblemen zwischenzeitlich Anderes getestet werden?**

3) **Kann die Softwareentität einfach getestet werden?**

- **Ist die Bewertung der Testresultate einfach?**
- **Können die Testresultate reproduziert werden?**
- **Sind die Testresultate repräsentativ?**
- **Ist Testautomatisierung (einfach) möglich?**

Testbarkeitsbewertung (II)

Metriken:

- **McCabe Komplexitätsmetrik, Domain/Range Ratio**
- **Prädikaten-, Entscheidungs-, Parameter-, Zusicherungsdichte**
- **ISO/IEC 9126-2:**
 - re-test efficiency
 - availability of built-in test function
 - test restartability
- **ISO/IEC 9126-3:**
 - completeness of built-in test function
 - autonomy of testability
 - test progress observability

Testbarkeitsbewertung (III)

Werkzeuge zur Metrikberechnung:

- **Logiscope (Verilog)**
 - execution paths
 - cyclomatic number
 - nesting levels
- **QStudio (QA Systems)**
 - Complexity
 - volume
 - modularity

Reviews:

- **Checklisten**

Konstruktive Ansätze (I) - Hardware-Testbarkeit

Problem:

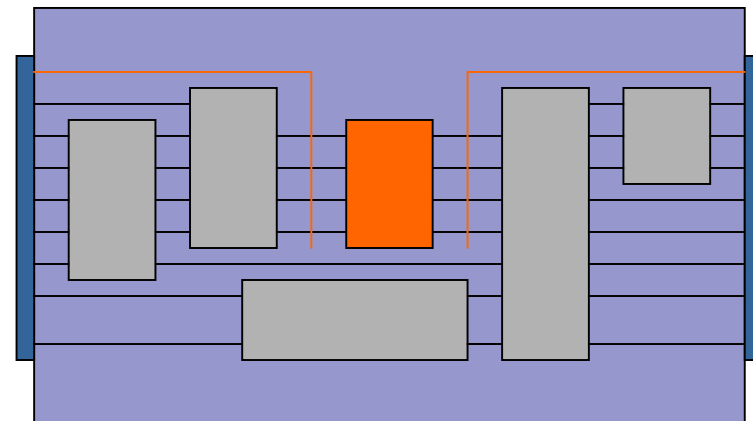
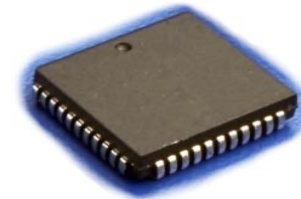
- jedes IC muss getestet werden
- Mangel an Steuerbarkeit
- Mangel an Beobachtbarkeit

Lösung:

- Boundary-Scan-Architektur
(IEEE 1149.1)

Besonderheit:

- Test auf Produktionsfehler



Konstruktive Ansätze (II) - Verteilte Echtzeitsysteme

Problem:

- **Beobachtung der Testergebnisse**
- **Beeinflussung der Testergebnisse durch Instrumentierung**
- **Reproduzierbarkeit der Testergebnisse**

Lösung:

- **geeignete Systemarchitektur**



Design für Testbarkeit (I) - Architekturebene

- **Abwägung von Modularität und Wiederverwendung**
- **Konzentration der Kontrollstruktur von Funktionen**
- **Einsatz von Standard-Testschnittstellen**
- **Testschnittstellen an semantisch einfach auswertbaren Stellen**
- **Abstimmung des Einsatzes der Vererbung mit Teststrategie**
 - **Delegation statt Vererbung / kein Neutest**
 - **Vererbung / selektiver Regressionstest**
 - **Vererbung / Testautomatisierung**

Design für Testbarkeit (Ila) - Entwurfsebene

- **Explizite Kontrollstrukturen**
- **Keine zyklischen Abhängigkeiten**
- **Methoden nach Möglichkeit als nicht-virtuell deklarieren**
- **Unmotivierten Einsatz von polymorphen Parametern vermeiden**
- **Implizite Abhängigkeiten vermeiden**
- **“Law of Demeter” berücksichtigen**

Design für Testbarkeit (IIb) - Entwurfsebene

- **Unmotiviertes Zustandsverhalten vermeiden**
- **Zustandstestfunktionen implementieren**
- **Informationsverlust durch “built-in”-Testeinrichtungen kompensieren**
- **Unerreichbare Ausgabewerte vermeiden**
- **Auslösen aller Ausnahmefälle (Exceptions) ermöglichen**

Design für Testbarkeit (III) - Code-Ebene

- **Verständlichkeit, Einfachheit vor “Eleganz”**
- **Variablenwiederverwendung vermeiden**
- **Unerreichbare Pfade vermeiden**
- **Rekursive Implementierungen von Algorithmen vermeiden**

Design für Testbarkeit - Strategie

- **Definition der Testbarkeitsanforderungen / Bewertung:**
 - für gesamte Architektur
 - für ausgewählte, kritische Komponenten
- **Wissen über Testbarkeit in Richtlinien verfügbar machen**
- **Testbarkeit früh berücksichtigen:**
 - konstruktive Richtlinien in Entwicklung einhalten
 - Reviews mit Beteiligung durch Tester durchführen

Zusammenfassung

- **Testbarkeit mit zunehmender Komplexität und Kritikalität wichtig**
- **Einige Metriken und Werkzeuge verfügbar**
- **Firmeninternes und -externes Wissen nutzbar machen**
- **Testbarkeit in Richtlinien und Checklisten berücksichtigen**