Nondeterministic Coverage Metrics as Key Performance Indicator for Model- and Value-based Testing

03. Februar 2011
David Faragó (farago@kit.edu)

Institute for Theoretical Computer Science; Logic and Formal Methods

TAV 31

Testing in modern IT-Contexts

Übersicht

- Value-based Software Engineering
  - KPI & VBSE
  - Value-based Testing
  - $_{opt}$ ROI des Testens
- Modell-basiertes Testen
  - MBT
  - $_{opt}$MBT-Methoden
- Überdeckungskriterien
- Nichtdeterminismus
  - Nichtdeterministische Systems
  - Arten des Nichtdeterminismus
  - Überdeckungskriterien für Nichtdeterminismus
  - Stand der Technik
  - n-choices
  - Quantisierung
- Zusammenfassung

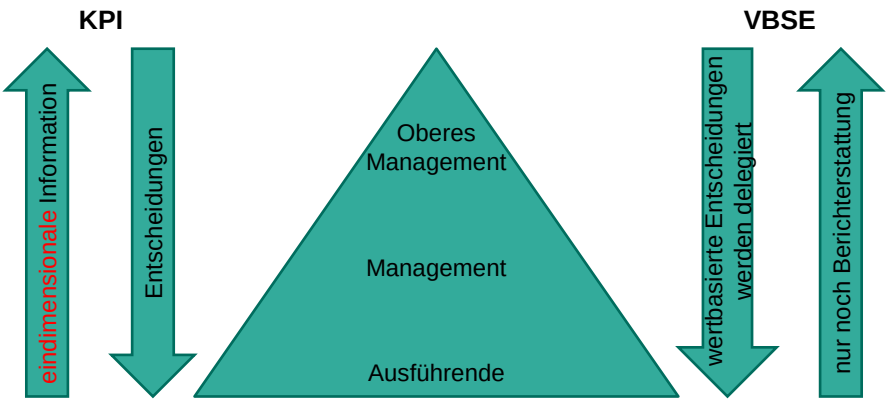
Value-based Software Engineering

KPI | **VBSE**

Key Performance Indicator:
- Kennzahl für Fortschritt oder Erfüllungsgrad
- z.B. ROI, Überdeckungskriterien, Sprint burndown

Value-based Software Engineering:
Gesamte SW-Entwicklung wertbasiert (jeglicher Nutzen, nicht nur ökonomisch)

Value-based Software Engineering



gefährlich: kann in falsche Richtung steuern, z.B. nur *defect detection percentage* und *loc per day*
⇒ *quick&dirty Programmierung und viele kleine Bugfixes*

Werte herunterbrechen und kommunizieren, z.B. mittels Priorisierung, Traceability und Risiko-Betrachtungen

Value-based Testing

- Wert des Testens steigern
- Werte (über Testsuite, Bugtracker) Paretoverteilt $\Rightarrow$ Priorisierung
- Anforderungs-basiertes Testen:
  gewichte Anforderungen, um wichtiges früher und intensiver zu testen
- Risiko-basiertes Testen:
  - Risikobelastung (Schadenswahrscheinlichkeit x Schadensgröße) ist KPI für Qualitätsmängel und Fortschritt
  - priorisiere Testfälle & Bugfixes um Risikobelastung zu minimieren

$_{opt}$ ROI des Testens

maximiere ROI[3,12]:
ROI = (Gewinn-Kosten)/Kosten

Kurzfristig:
- Bugerkennung erspart Nachbesserung
- erhöhte Planungssicherheit durch Risikoabschätzung

Langfristig:
- verbesserte Prozesse

Durch Konformität:
um Qualität zu erzielen
- Präventionskosten
- Bewertungskosten

Durch Nichtkonformität:
verursacht durch Qualitätsmangel
- interne Fehlerkosten
- externe Fehlerkosten

Modellbasiertes Testen (MBT)

- Die beste Umsetzung von anforderungsbasiertem Testen ist es, aus den Anforderungen Blackbox-Tests zu generieren[2].

+ MBT macht dies automatisiert
+ unterstützt VBSE auch durch Traceability und Priorisierung
+ frühe Verifikation
+ kann schnell auf Änderungen (Anforderungen, Risiken) eingehen
+ hohe Überdeckung
+ kann mit Nichtdeterminismus umgehen

Opt MBT-Methoden

Off-the-fly MBT: generiert —Test-suite→ Ausführung & Evaluierung

z.B. TGV, Spec Explorer

ineffizient, insbesondere bei Nichtdet. & Daten; keine dynamische Information

On-the-fly MBT: generiert —Test-schritt→ Ausführung & Evaluierung

z.B. TorX-Derivate, UPPAAL Tron, Spec Explorer

schwache Zielsuche für Testselektion

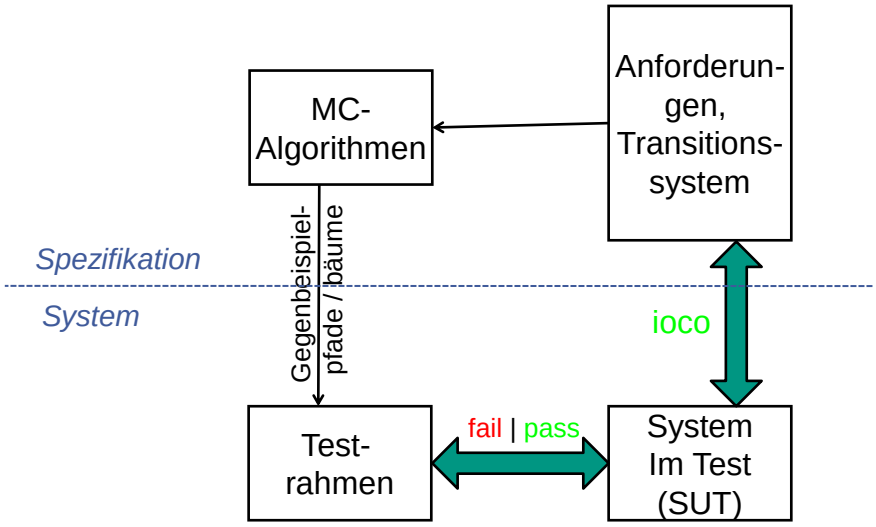
Lazy on-the-fly MBT: generiert —Teil-Test-sequenz→ Ausführung & Evaluierung

Initiiere Ausführung, z.B. beim Erreichen von

- Testzielpunkte
- einer bestimmten Tiefe
- Nichtdeterm. des SUTs

+ aussagekräftigere Testfälle
+ flexibler durch Reproduzierbarkeit und Wiederverwendbarkeit
+ Nichtdeterminismus des SUTs effizient behandelbar

Opt **MBT-Methoden**



Überdeckungskriterien

- % der überdeckten Elemente
  - Quelltext
  - funktionales Modell
  - Zustände
  - Transitionen
  - Bedingungen
  - Datenfluss
  - Nichtdeterminismus
- als KPI
- als Suchheuristik, d.h. für Test-Selektion, können Prioritäten automatisiert umsetzen

Nichtdeterministische Systeme

- Multiples Verhalten nach einem fixen Stimulus

$\Rightarrow$ Nichtdeterminismus auch modellieren

- bildet Realität am besten ab
- zur Unterspezifikation:
  - hilft, mit noch offenen Punkten umgehen zu können
  - erspart unnötige Umarbeiten
  - verringert die Initialkosten

Arten des Nichtdeterminismus

steuerbar
- Tester entscheidet
- modelliert durch input

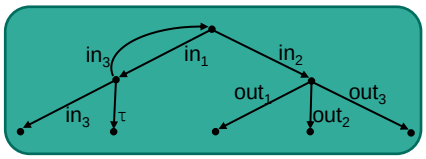
nicht steuerbar
System Under Test entscheidet

multiple Ausgaben

- nichtdeterministisches LTS
- nicht direkt observierbar

multiple identische Labels

interne Transition
τ



Überdeckungskriterien für Nichtdeterminismus

- multiples Verhalten überdecken

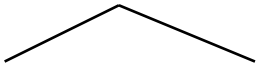
⇒ nichtdet. Testfälle müssen mehrfach ausgeführt werden

- 100% Überdeckung evtl. nie erreichbar

Stand der Technik

- Testfälle konstant *k* mal ausführen

\+ sehr einfach umsetzbar
\+ funktioniert für alle Arten von Nichtdeterminismus
– manchmal zu häufig, manchmal zu selten
– keine Informationen



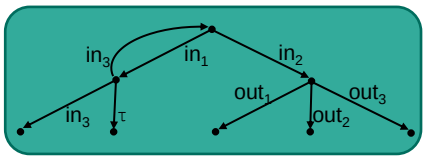
für Suchheuristiken für KPIs zu Nichtdeterminismus

n-choices

- Maß über Zuständen mit multiplen Ausgaben (*nichtdet. Zustände*)

- $n-\text{choices} := \frac{|\{s \in S | \text{von } s \text{ gab es } \geq n \text{ unterschiedliche Ausgaben}\}|}{|\{s \in S | s \text{ hat } \geq n \text{ unterschiedliche Ausgaben}\}|}$
- $n = 1$: wieviel nichtdeterministische Zustände wurden besucht
- $n = 2$: wieviel nichtdeterm. Zustände verhalten sich wirklich nichtdeterm.

- $\text{all-choices} := \frac{|\{\text{nichtdet. } s \in S | \text{alle Ausgaben von } s \text{ wurden gewählt}\}|}{|\{\text{nichtdet. } s \in S\}|}$
- wieviel Unterspezifikation vorhanden



n-choices

- 100% transition coverage $\Leftrightarrow$ 100% all-choices $\Leftrightarrow$ $\Leftrightarrow$ 100% 2-choices $\Leftrightarrow$ 100% 1-choices

+ recht einfach umsetzbar
+ mehr Information

für Suchheuristiken

für KPIs zu Nichtdeterminismus
(wie häufig Ausnahmefälle / Unterspezifikation)

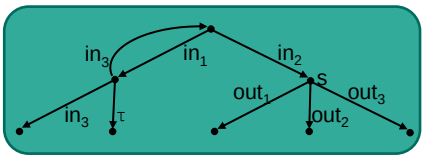
Quantifizierung (bei multiplen Ausgaben)

- Nichtdeterminismus ⇝ approximierter Probabilismus
- Zähle Anzahl Traversierungen
  $count(s)$ für Zustände, Transitionen

⇒ Approximiere Probabilismus,
z.B. $P_{\text{visit}}[out_1 \text{ from } s] = count(out_1)/count(s)$

\+ Umsetzung noch praktikabel

\+ viel Information
- für Suchheuristiken
- für VBSE und KPIs zu Nichtdeterminismus
  z.B. für anforderungs-basiertes und risiko-basiertes Testen, P[*s* from *s'*], Schadens-wahrscheinlichkeit, Risikobelastung



Quantifizierung (bei nichtdeterministischen LTSs)

- Aktueller Zustand: Menge $S_{\text{current}} \subseteq S$

⇒ nur Menge der potentiell überdeckten Zustände (Transitionen, ...)

– wenig aussagekräftig, z.B. nicht monoton steigend
– irreführende Suchheuristik

⇒ quantifiziere jeglichen Nichtdeterminismus

Quantifizierung (bei nichtdeterministischen LTSs)

- Wahrscheinlichkeitsverteilung P_{current} über Zustände

  $\forall s \in S: \; P'_{\text{current}}[s] = \Sigma_{\bar{s} \in S}(P_{\text{current}}[\bar{s}] \cdot P_{\text{visit}}[s \text{ from } \bar{s}])$

  s.o., für nicht observierbaren Nichtdeterminismus: gleichverteilt

  $\text{am Anfang: } P_{\text{current}}[s \in S_{\text{init}}] := 1/|S_{\text{init}}|, \; P_{\text{current}}[s \notin S_{\text{init}}] := 0$

- Wahrscheinlichkeiten P_{covered} über Zuständen (Transitionen, ...)

  $\forall s \in S: \; P'_{\text{covered}}[s] = P_{\text{covered}}[s] + (1 - P_{\text{covered}}[s]) \cdot P_{\text{current}}[s]$

  $(\forall t \in T, \ldots \text{ analog})$

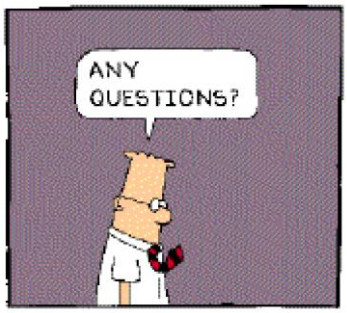
  $\text{am Anfang: } P_{\text{covered}}[s \in S_{\text{init}}] := 1/|S_{\text{init}}|, \; P_{\text{covered}}[s \notin S_{\text{init}}] := 0$

+ Information von oben auch für nichtdeterministische LTSs
− zu grobe Abschätzung?
− Berechnungen zu teuer?

Zusammenfassung

- KPIs:nicht isoliert, sondern innerhalb VBSE
- (nichtdeterministischen) Überdeckungskriterien gute KPIs innerhalb VBSE
- Quantifizierung nützlich bei multiplen Ausgaben

Vielen Dank!



Literaturverzeichnis

[1] Boris Beizer. *Software testing techniques, (2nd ed.)*. Van Nostrand Reinhold Co., 1990.

[2] Stefan Biffl, Aybke Aurum, Barry Boehm, Hakan Erdogmus and Paul Görnbacher, editors. *Value-Based Software Engineering*. Springer, Berlin, 2006.

[3] Rex Black. *Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing*. John Wiley & Sons, 2nd edition, 2002.

[4] Barry Boehm. *Value-based software engineering: reinventing earned value monitoring and control.* SIGSOFT Software Engineering Notes, March 2003.

[5] Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner. *Model-based Testing of Reactive Systems*. Springer, 2005.

[6] J. Bullock. *Calculating the value of testing*. Software Testing and Quality Eng., 2000.

[7] Margus Veanes et al. *Model-based testing of object-oriented reactive systems with Spec Explorer*. In Formal Methods and Testing, Springer, 2008.

[8] D. Farago. *Coverage criteria for nondeterministic systems*. testing experience, The Magazine for Professional Testers, pages 104-106, 2010.

[9] D. Farago. *Improved underspecification for model-based testing in agile development*. The 2nd International Workshop on Formal Methods and Agile Methods, 2010.

[10] D. Farago. *Model-based testing in agile software development*. In TAV 30, STT, 2010.

[11] Gorden Fraser and Franz Wotawa. *Test-case generation and coverage analysis for nondeterministic systems using model-checkers*. ICSEA, IEEE, 2007.

Literaturverzeichnis

[12] F.M. Gryna. *Quality and Costs, Juran's Quality Handbook*. McGraw-Hill, 1999.

[13] *KPI Library*. http://kpilibrary.com/home/. (November 2010).

[14] Roman Pichler. *Agile Product Management with Scrum: Creating Products That Customers Love*. Addison-Wesley, 2010.

[15] Jeff Smith. *The K.P.I. Book*. Insight Training & Development, Limited, 2001.

[16] Andreas Spillner and Tilo Linz. *Basiswissen Softwaretest: Aus- und Weiterbildung zum Certied Tester – Foundation Level nach ISTQB-Standard*. dpunkt, 3. edition, 2005.

[17] Mark Utting and Bruno Legeard. *Practical Model-Based Testing: A Tools Approach*, Morgan Kaufmann, 1 edition, 2007.

[18] Yuen-Tak Yu and Man Fai Lau. *A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions*. Journal of Systems and Software, 2006.