Testen Web-basierter Systeme mittels strukturierter, graphischer Modelle – Vergleich anhand einer Fallstudie

Fevzi Belli[+], Christof J. Budnik[+], Michael Linschulte[+], Ina Schieferdecker[*]

[+)]: Universität Paderborn, Fakultät für Elektrotechnik, Informatik und Mathematik,
Institut für Elektrotechnik und Informationstechnik, Fachgebiet „Angewandte Datentechnik“

[*)]: Technische Universität Berlin, Fakultät IV - Elektrotechnik und Informatik,
Lehrstuhl „Entwurf und Testen von Telekommunikationssystemen“

Abstract: Es wird ein ereignisbasierter Ansatz zur Generierung von Tests für Web-basierte kommerzielle Systeme vorgestellt. Die Modellierung des Prüflings erfolgt durch Ereignissequenzgraphen, in denen vollständige Wege Tests definieren. Dieser Ansatz wird beispielhaft an einer Fallstudie präsentiert und mit einem zustandsbasierten Ansatz verglichen.

1 Einleitung

Als Folge des stark wachsenden E-Commerce, wie z.B. des Online-Handels, ist in den vergangenen Jahren die Anzahl an Web-Applikationen oder auch Web-basierten Systemen enorm gestiegen. Gerade weil jene Applikationen entscheidend für solche Geschäftsprozesse sind, ist damit auch die Sorge um deren Qualität und Zuverlässigkeit verbunden. Dieses liegt zum einen an der offenen Nutzungsmöglichkeit der Systeme durch beliebige Nutzer und zum anderen ist die Ausführungsumgebung mit den vielen unterschiedlichen Browsern, Web-Servern und Betriebssystemen sehr heterogen. Gleiches gilt für die Komponenten, die durch verschiedene Technologien wie ASP, Java oder PHP realisiert sein können. Des Weiteren findet die Möglichkeit der Erfassung von Daten mittels so genannter Web-Formulare in vielen komplexen Web-Applikationen eine breite Anwendung, die sogar dynamische Strukturänderungen innerhalb der Applikation hervorrufen kann. All diese Eigenschaften stellen somit auch für die Prüfung und im speziellen den Test neue Anforderungen. Aufgrund der unterschiedlichen Qualitätsmerkmale von Web-basierten Systemen und der damit verbundenen Probleme beschränkt sich diese Arbeit auf den Test der Funktionalität einer Web-Applikation. Der nächste Abschnitt fasst die verwandten Arbeiten im relevanten Bereich zusammen, bevor Abschnitt 3 unseren ereignisorientierten Ansatz zum Testen Web-basierter Systeme einführt. Abschnitt 4 stellt eine Fallstudie dar, die diesen Ansatz mit einem anderen, zustandsbasierten Ansatz vergleicht. Einen Überblick über geplante weitere Arbeiten gibt Abschnitt 5 und schließt den Beitrag ab.

2 Verwandte Arbeiten

Die Modellierung und das Testen interaktiver Systeme mittels zustandsbasierter Modelle hat eine lange Tradition [1], [2]. Diese Ansätze analysieren den Prüfling und leiten aus Nutzeranforderungen Sequenzen von Nutzerinteraktionen ab, die als Testfälle genutzt werden. [3] führt dafür ein vereinfachtes zustandsbasiertes, graphisches Modell ein; dieses Modell wurde in [4] erweitert, um nicht nur erwünschte, sondern auch unerwünschte Situationen abzubilden. Der Entwurf und Test von Web-basierten Systemen stellt eine komplexe Aufgabe dar, zu dessen Hilfe Modelle und Techniken eingesetzt werden, die sowohl die Konzeption, Navigation als auch das Design berücksichtigen. Die meisten dieser Modelle sind graphisch; beispielsweise sei die Erweiterung von UML erwähnt, um Web-Applikationen darzustellen [5], [6]. [7] bedient sich ebenfalls eines UML-Modells, um eine Web-Applikation zu analysieren und zu testen. Die Grundlage bildet dabei ein Meta-Modell, das die Konzepte einer Web-Applikation beschreibt. Das Modell einer bestimmten Applikation ist infolgedessen eine Instanz dieses Meta-Modells. [8] entwickelt eine Methode zur Testfallgenerierung auf Basis eines objektorientierten Modells. Hierbei wird das zu testende System aus drei verschiedenen Perspektiven betrachtet: der Objekt-, Struktur- und Verhaltensperspektive, die in unterschiedlichen Diagrammen modelliert werden. Darauf aufbauend werden Testbäume erstellt und anschließend daraus Testfälle generiert. Der Ansatz in [9] befasst sich mit dem Black-Box-Test von Web-Applikationen. Er berücksichtigt das zustandsabhängige Verhalten von Web-Applikationen und schlägt eine darauf gezielte Lösung des Zustandsexplosionsproblems vor. Diese Technik erlaubt, wie der in dieser Arbeit vorgestellte Ansatz auch, eine Strukturierung des betrachteten Systems. Anhand einer Fallstudie werden beide Verfahren miteinander verglichen.

3 Modellierung und Test Prozess

Diese Arbeit nutzt *Ereignissequenzgraphen* [4] für die Modellierung von Systemverhalten, insbesondere für Interaktionen mit dem System aus Nutzersicht. Ein *Ereignis* ist hierbei ein extern beobachtbares Phänomen, wie Nutzereingaben, Systemantworten etc.

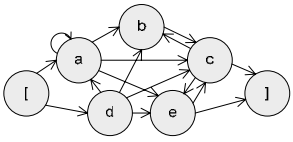


Abbildung 1: Beispiel eines ESG mit ε='[', γ=']'

Def. 1: Ein *Ereignissequenzgraph* ESG=(V,E,I,O) ist ein gerichteter Graph (V,E) mit einer endlichen Menge von Knoten $V \neq \emptyset$, einer endlichen Menge von Kanten $E \neq \emptyset$ und $E \subseteq (I \cup V) \times (V \cup O)$, der endlichen Menge $I \neq \emptyset$, $I \cap V = \emptyset$ von Eingangsknoten ε und der endlichen Menge $O \neq \emptyset$, $O \cap V = \emptyset$ von Ausgangsknoten γ (s. Abb. 1).

Operationen Web-basierter Systeme werden typischerweise über Ein-/Ausgabegeräte mittels Tasten, Checkboxen, etc. gesteuert und beobachtet. Zur Darstellung dieser Nutzer-System-Interaktionen werden die Knoten des ESG als Ereignisse interpretiert. Diese Ereignisse führen interaktiv zu Sequenzen von Ein- und Ausgaben.

Def. 2: Sei (V,E,I,O) ein *ESG*. Jede Knotensequenz $\langle v_0,...,v_k\rangle$ wird *gültige Ereignissequenz* (g*ES*) genannt, wenn $(v_i, v_{i+1}) \in E$, für $i=0,...,k-1$ gilt. Eine *vollständige Ereignissequenz* (v*ES*) hat die Form $\langle \varepsilon, v_0,...,v_k, \gamma\rangle$, wobei $\langle v_0,...,v_k\rangle$ eine gES ist, $\varepsilon \in I$ und $\gamma \in O$.

Def. 3: Seien $ESG_1 = (V_1,E_1,I_1,O_1)$ und $ESG_2 = (V_2, E_2,I_2,O_2)$ ESGs. ESG_1 kann über einen Knoten $v \in V_1$ durch ESG_2 *strukturiert* werden, wenn gilt: Sei $N^+(v) \subseteq O_1 \cup V_1 \setminus \{v\}$ die *Menge aller Nachfolger von v* und $N^-(v) \subseteq I_1 \cup V_1 \setminus \{v\}$ die *Menge aller Vorgänger* von *v*. Dann muss es eine Abb. von ESG_2 nach ESG_1 mit $O_2 \rightarrow N^+(v)$ und $I_2 \rightarrow N^-(v)$ geben (s. Abb. 2).

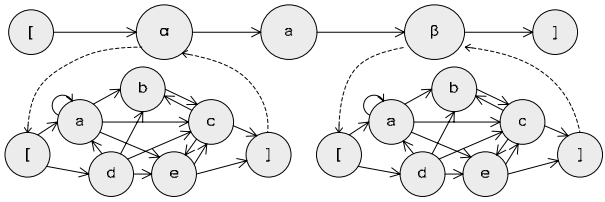
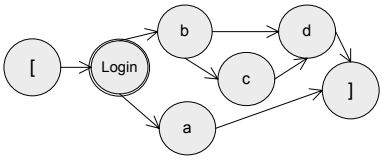


Abbildung 2: sESG mit Strukturierung von α und β durch den ESG aus Abb. 1

Hierbei repräsentieren die Knoten im ESG nicht mehr nur einfache Ereignisse, sondern werden selber wieder durch ESGs beschrieben. Man spricht jetzt von einem *strukturierten* ESG (s*ESG*). Die strukturierte Modellierung in verschiedenen Ebenen ist wichtig für den Entwurf, um die Knotenmenge zu begrenzen und damit die Komplexität der Testfallgenerierung zu reduzieren. Web-basierte Systeme haben zusätzlich die Eigenschaft, dass die Ereignisfolgen in Abhängigkeit der Benutzereingaben variieren können. Aus diesem Grund wird das ESG-Modell bei der Beschreibung der Testeingabedaten um Entscheidungstabellen (ET) [10] erweitert, welche durch einen Doppelkreis im ESG kenntlich gemacht werden. Abb. 3 zeigt exemplarisch einen ESG mit zugehöriger ET zur Generierung möglicher Testeingabesequenzen für den Login an einem System mittels Passwort („PWD"). „Administrator" und „Benutzer" stellen dabei Mengen zulässiger Passwörter dar.



Login			
PWD∈Administrator	T	F	F
PWD∈Benutzer	F	T	F
a	X		
b		X	
Fault			X

Abbildung 3: Erweiterung ESG um Entscheidungstabelle

Der Testprozess basiert auf der Testfallgenerierung nach [10], welche die minimale Überdeckung von Ereignissequenzen einer bestimmten Länge durch vollständige Ereignissequenzen vorsieht (s. Algorithmus 1). Die strukturelle Ersetzung eines Knotens eines ESG in höheren Abstraktionsebenen in einer Testsequenz wird dabei durch die vollständigen Ereignissequenzen des strukturell verfeinerten ESG substituiert. Daraufhin werden die Testsequenzen mit den verschiedenen Testeingabedaten aus den Regeln der zugehörigen Entscheidungstabellen generiert und schließlich am System durchgeführt.

Die verschiedenen Ereignissequenzen des ESG sollten das System in einen Ausgangsknoten führen können. Wenn immer das nicht möglich ist, wird ein Test abgebrochen

und als fehlerhaft markiert. Auf eine Web-Applikation bezogen, kann das Folgeereignis aus folgenden Ursachen nicht erreichbar sein:

- Das Ereignis zuvor konnte zwar ausgeführt werden, das System jedoch bricht ab, weil Webseiten nicht geladen werden können oder gänzlich fehlen.
- Das System zeigt z.B. eine Fehlermeldung an, die zuvor bestätigt werden muss.
- Die Dateneingabe hat eine andere Struktur als erwartet dynamisch aufgebaut.
- Es wurden gültige Daten eingegeben, deren Abhängigkeiten aber nicht berücksichtigt.

Algorithmus 1: Testgenerierung
n:= Zahl an interaktiven Komponenten des Systems
l:= geforderte Länge der Testsequenz
FOR Komponente 1 TO *n* DO
Generiere seinen ESG [11]
FOR k:=2 TO *l* DO
Generiere alle minimialen vES der Länge k [10]
FOR jede vES mit sESG DO
Generiere mehrere komplettierte vES unter Auflösung der strukturierten Kanten mit vES des strukturierenden sESG
FOR jedes komplettierte vES DO
Generiere datenerweiterte komplettierte vESs mit Eingabedaten der ET

Dieses Fehlermodell definiert das Testorakel zur Bewertung der Testausführungen. Es ist einfach, aber mächtig genug, um Entwurfsfehler und Dateneingabefehler zu erkennen, solange die Testsequenzen lang genug sind und alle Eingabedaten aus den ET getestet werden.

4 Fallstudie: Ereignisbasierter Test eines Web-basierten Systems und exemplarischer Vergleich mit einem zustandsbasierten Ansatz

An der Universität Paderborn entsteht derzeit in Kooperation mit ISIK Touristik GmbH das Buchungs- und Verwaltungssystem ISELTA. Das Ziel dieses Systems ist es, Hotelanbietern die Möglichkeit zu geben, dass ihre Zimmerkontingente direkt über das Internet gebucht werden können. Dafür können die Anbieter ihr Hotel mit den jeweiligen hotelspezifischen Eigenschaften und Ausstattungen im ISELTA-System erfassen. Als Beispiel soll hier ein Auszug zur Verwaltung der Zimmertypen herangezogen werden (Abb. 4).

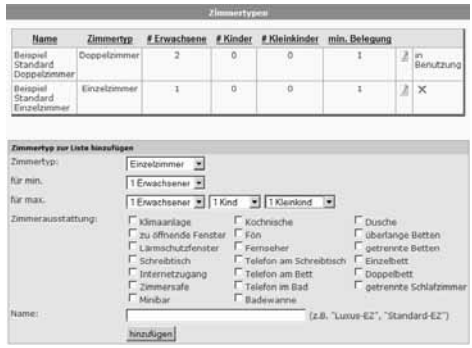
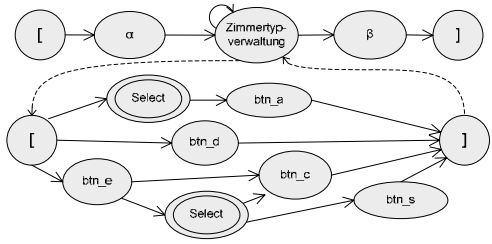


Abbildung 4: Zimmertypverwaltung im System „ISELTA“

Im oberen Teil der Abb. sieht man eine Liste mit bereits angelegten Zimmertypen. Der untere Teil bildet das entsprechende Formular zum Anlegen von Zimmertypen.

Ereignisbasierter Ansatz

Die grundlegende Struktur des Anlegens, Bearbeitens und Löschens modelliert der sESG in Abb. 5. Die Knoten „Select“ (Knoten mit Doppelkreis) beinhalten jeweils eine Entscheidungstabelle für die Dateneingabe und deren Abhängigkeiten.



Legende:
btn_a = Button hinzufügen
btn_d = Button löschen
btn_c = Button abbrechen
btn_s = Button speichern
btn_e = Button bearbeiten

Abbildung 5: sESG zur Zimmertypverwaltung

Muss eine Bedingung nicht berücksichtigt werden, um das Nachfolgeereignis zu bestimmen, so wird ein „Don’t Care“-Term in Form eines „-“ eingefügt. Die Verletzung einer der drei zu berücksichtigenden Bedingungen ist ausreichend, so dass die ET um die ausgegrauten Regeln konsolidiert werden kann.

ET Select:								
Zimmertyp ∈ {EZ, DZ, Su, Ap, St, Bu}	-	-	-	-	-	-	-	-
SelMin ∈ {1,2,3,4,5,6 }	-	-	-	-	-	-	-	-
SelMaxErw ∈ {1,2,3,4,5,6}	-	-	-	-	-	-	-	-
SelMaxKd ∈{1,2,3,4,5,6}∨∅	-	-	-	-	-	-	-	-
SelMaxKleinKd ∈{1,2,3,4,5,6}∨∅	-	-	-	-	-	-	-	-
Zimmerattribute ∈{…}∨∅	-	-	-	-	-	-	-	-
Name ∈{string}	T	F	T	F	T	F	T	F
SelMin <= SelMaxErw + SelMaxKd + SelMaxKleinKd	T	T	F	F	T	T	F	F
Name ∉ existierender Zimmertypname	T	T	T	T	F	F	F	F
btn_a	x							
btn_s	x							
btn_c	x	x	x	x	x	x	x	x

Im nächsten Schritt werden anhand des ESG und der zugehörigen Entscheidungstabelle die Testfälle generiert. Als Testkriterium gilt die vollständige Kantenüberdeckung entsprechend Algorithmus 1. Die generierten Testfälle zeigt die folgende Tabelle links. Diese Testfälle beinhalten allerdings noch keine Eingabedaten und müssen anhand der Entscheidungstabelle ergänzt werden, siehe Tabelle rechts. Zu beachten ist, dass die Testfälle 1.2, 1.3, 1.4 Sequenzen darstellen, die nicht ausführbar sein sollten. Sie wurden durch Klammerung der Testfallnummer kenntlich gemacht. Nachdem die relevanten Testfälle aufgestellt wurden, kann der Testfall im übergeordneten ESG (s. Abb. 5) vervollständigt werden. Die entsprechende Testsequenz lautet: „α Zimmertypverwaltung Zimmertypverwaltung β“, wobei „Zimmertypverwaltung“ durch die Testfälle aus folgender Tabelle ersetzt wird.

Nr.	Sequenz	Sequenz mit Testeingaben
1	[Select btn_a]	[Zimmertyp=EZ SelMin=1 SelMaxErw=2 SelMaxKd=∅ SelMaxKleinKd=∅ Zimmerattribute=∅ Name=TestEZ btn_a]
		(1.2) [Zimmertyp=DZ SelMin=1 SelMaxErw=1 SelMaxKd=∅ SelMaxKleinKd=∅ Zimmerattribute=∅ Name=∅ btn_a]
		(1.3) [Zimmertyp=DZ SelMin=2 SelMaxErw=1 SelMaxKd=∅ SelMaxKleinKd=∅ Zimmerattribute=∅ Name=TestDZ btn_a]
		(1.4) [Zimmertyp=EZ SelMin=1 SelMaxErw=1 SelMaxKd=∅ SelMaxKleinKd=∅ Zimmerattribute=∅ Name=TestEZ btn_a]
2	[btn_d]	…
3	[btn_e btn_c]	…
4	[btn_e Select btn_c]	…
5	[btn_e Select btn_s]	…

Zustandsbasierter Ansatz

Abb. 6 stellt den entsprechend [9] erstellten endlichen Automaten (EA) des Vorgangs „Zimmertypverwaltung“ dar, der in Abb. 5 als ESG visualisiert wurde. Da die Notation für sich spricht, wird von einer ausführlichen Erklärung abgesehen und das Notwendigste angegeben: Die Knoten „ON“ und „OFF“ stellen den Start- und Endzustand dar, die restlichen Knoten die einzelnen Systemzustände, die innerhalb des Programmablaufs erreicht werden können. Die benötigten Eingaben, um einen bestimmten Zustand zu erreichen, attributieren die Kanten.

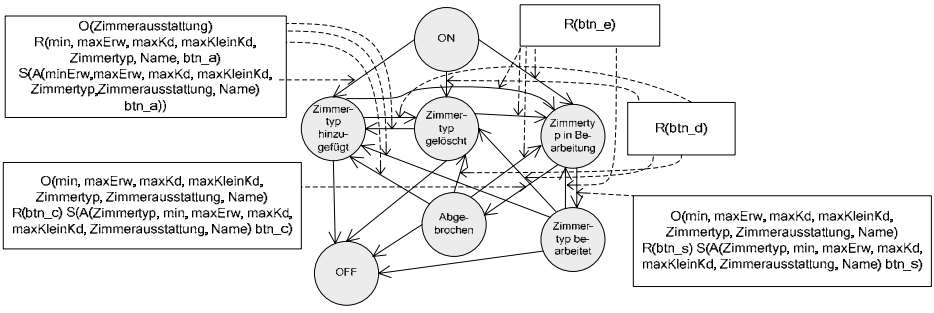


Abbildung 6: Annotierter Zustandsautomat zur Zimmertypverwaltung

Die Zustands- und Eingabe-Sequenzen, die aus dem EA generiert werden, zeigt die folgende Tabelle. Die Testfälle beinhalten allerdings keine konkreten Eingabedaten oder Vorschriften für die Bildung, sondern müssen nachträglich manuell selektiert werden.

Nr.	Sequenz
1	ON Zimmertyp gelöscht OFF
Eingabe	R(btn_d)
2	ON Zimmertyp hinzugefügt Zimmertyp gelöscht Zimmertyp hinzugefügt OFF
Eingabe	O(Zimmerausstattung) R(min, maxErw, maxKd, maxKleinKd, Zimmertyp, Name, btn_a) S(A(minErw, maxErw, maxKd, maxKleinKd, Zimmertyp,Zimmerausstattung, Name) btn_a)) R(btn_d) O(Zimmerausstattung) R(min, maxErw, maxKd, maxKleinKd, Zimmertyp, Name, btn_a) S(A(minErw, maxErw, maxKd, maxKleinKd, Zimmertyp,Zimmerausstattung, Name) btn_a))
…	…

Vergleich beider Ansätze
Zur Testgenerierung wurden die entsprechenden ESGs und EAs von je einem Studenten (getrennt voneinander) erstellt. Für diese Studenten waren die beiden Techniken fremd, d.h., der Einarbeitungsprozess wurde in den Vergleich mit einbezogen. Der benötigte Aufwand für diese nachträgliche Modellierung wurde festgehalten:

	Anzahl der Modelle	Einarbeitungszeit in min.	Erstellungszeit in min.	Gesamtzeit in min.
Tester A: ESG	10	28	274	302
Tester B: EA	12	73	329	402

Obige Tabelle zeigt, dass die Anzahl der Modelle sich kaum unterscheidet. Das ist auch nicht überraschend, da beide Modellierungstechniken eine hierarchische Strukturierung ermöglichen. Bei dem gemessenen Zeitaufwand ist festzustellen, dass Tester A sehr schnell mit der ESG Modellierungstechnik vertraut war und schon nach weniger als einer halben Stunde seinen ersten ESG erstellt hatte. Tester B hingegen war zu der Zeit immer noch mit der Einarbeitung beschäftigt. Darüber hinaus kam Tester B nicht ohne die BNF-Grammatik bei der Aufstellung der Kantenbeschriftungen aus.

Nach der Aufstellung der Modelle wurden die entsprechenden Testfälle generiert und am System ausgeführt. Bei dem Vergleich der beiden Methodiken ist zunächst einmal die unterschiedliche Testfallmenge anzumerken. Fünf Testfälle nach der EA-Methodik stehen 14 Testfällen nach der ESG-Methodik gegenüber. So können auf Grundlage der Entscheidungstabellen verschiedene Eingaben und Eingabeabhängigkeiten erfasst und systematisch zulässige und unzulässige Testfälle generiert werden. Ein Beispiel für die Bedingung, dass die minimale Belegung kleiner ‚gleich der maximalen Belegung ist, muss sowohl als zulässiger als auch unzulässiger Testfall durchgeführt werden. Durch die ESG-Systematik ist die Aufstellung dieser Testfälle gewährleistet, nicht jedoch bei der EA-Methodik. Eine Übersicht über die Anzahl der gefundenen Fehler innerhalb der betrachteten Module zeigt die folgende Tabelle.

Anzahl Fehler	Produktverwaltung	Zimmertypverwaltung	Zeitraumverwaltung	Preisverwaltung	Verpflegungspreisverwaltung	Gesamt
ESG	11	3	1	0	1	16
EA	8	0	0	0	1	9

Die meisten Fehler wurden dabei innerhalb der Produktverwaltung gefunden, da dieses mit Abstand das komplexeste Modul von allen ist. Insgesamt konnte die ESG-Methodik 16 Fehler und die EA-Methodik 9 Fehler finden. Einen Auszug der insgesamt entdeckten Fehler liefert die folgende Tabelle.

Nr.	Fehler
1	Zimmer anlegen möglich mit 0 Personen als Kapazität (Produktverwaltung)
2	Zimmername kann nur aus Leerzeichen bestehen (Zimmertypverwaltung)
3	Zimmertypname kann auf bestehenden Zimmertypnamen geändert werden (Zimmertypverwaltung)
4	Verpflegungspreis ohne spezifizierten Preis kann angelegt werden (Verpflegungspreisverwaltung)

5 Zusammenfassung und Ausblick

Web-basierte, kommerzielle Systeme verfügen oft über Formulare, die seitens des Benutzers ausgefüllt werden. Die Eingabedaten rufen häufig eine dynamische Strukturänderung hervor, was neue Anforderungen an die Prüfung des funktionalen Verhaltens von Web-Applikationen darstellt. In dieser Arbeit wurden strukturierte, ereignisorientierte Graphen zur Testfallgenerierung für Web-Systeme vorgeschlagen. Die gegenseitige Abhängigkeit der Testeingabedaten wurde explizit durch Entscheidungstabellen berücksichtigt. Eine Fallstudie demonstrierte den Einsatz des vorgestellten Ansatzes und verglich ihn mit einem zustandsbasierten Ansatz. Unterschiede beider Ansätze wurden durch generierte Testfälle und entdeckte Fehler aufgezeigt. Als weitere Arbeit ist geplant, Sequenzfehler zu lokalisieren, die nicht unmittelbar nach der Eingabe auftreten und sich oft erst im späteren Verlauf bemerkbar machen.

Referenzen

[1] Parnas, D.L.: On the Use of Transition Diagrams in the Design of User Interface for an Interactive Computer System. In Proc. of ACM Nat'l. Conf., ACM Press 1969; S. 379-385.

[2] Shehady, R.K.; Siewiorek, D.P.: A Method to Automate User Interface Testing Using Finite State Machines. In Proc. Int. Symp. Fault-Tolerant Computing, 1997; S. 80-88.

[3] White, L.; Almezen, H.: Generating Test Cases for GUI Responsibilities Using Complete Interaction Sequences. In Proc. of Int. Symp. on Softw. Reliability and Eng., IEEE Comp. Press, 2000; S. 110-119.

[4] Belli, F.: Finite-State Testing and Analysis of Graphical User Interfaces. In Proc. of Int. Symp. on Softw. Reliability and Eng. IEEE Comp. Press, 2001; S. 34-43.

[5] Conallen, J.: Building Web Applications with UML, Addison-Wesley, 1999.

[6] Hennicker, R.; Koch, N.: Modeling the User Interface of Web Applications with UML. UML 2001. LNI, vol. 7, 2001; S.158-172.

[7] Ricca, F., Tonella, P., Analysis and testing of Web applications. In Proc. of Int. Conf. on Softw. Eng. IEEE Comp. Press, 2001; S. 25-34.

[8] Kung, D.C.; Liu, C.-H.; Hsia, C.-T.: An Object-Oriented Web Test Model for Testing Web Applications. In Proc. of Asia-Pacific Conf. on Quality Software. IEEE Comp. Press, 2000; S. 111.

[9] Andrews, A.A.; Offutt, J.; Alexander, R.T.: Testing Web applications by modelling with FSMs. Software and System Modeling, vol. 4(3), 2005; S. 326-345.

[10] Belli, F.; Budnik, C. J.: Minimal Spanning Set for Coverage Testing of Interactive Systems. In Proc. of Int. Coll. on Theoretical Aspects and Computing, LNCS, vol. 3407, 2004; S. 220-234.

[11] Belli, F.; Budnik, C. J.; White, L.: Event-based Modeling, Analysis and Testing of User Interactions: Approach and Case Study. The Journal of Software Testing, Verification and Reliability. John Wiley & Sons, vol. 16(3), 2006; S. 3-32.