Ereignis-basierter Test grafischer Benutzeroberflächen – ein Erfahrungsbericht

Fevzi Belli, Mutlu Beyazit, Axel Hollmann, Michael Linschulte, Sascha Padberg

Universität Paderborn
Institut für Elektrotechnik und Informationstechnik
33098 Paderborn
{belli, beyazit, hollmann, linschulte, padberg}@adt.upb.de

1 Einleitung

Dieser Beitrag beschreibt die Erfahrungen, die unsere Gruppe bei Outsource-Testen einer grafischen Benutzeroberfläche im Auftrag eines Software-Hauses in einem kommerziellen Projekt gesammelt hat. Da das Projekt und unsere Mitwirkung einer strengen Geheimhaltungsverpflichtung unterliegen, können wir im Folgenden unsere Erfahrungen nur durch anonymisierte Daten darstellen. Wir hoffen jedoch, in unserem Beitrag bis zum Zeitpunkt der Tagung gemeinsam mit unserem Kooperationspartner mehr Informationen einließen lassen zu können.

Das getestete System ist eine komplexe Windows-Applikation mit ca. 120.000 Codezeilen, die über eine lokale Datenbank verfügt sowie über Web-Services Daten einer externen Datenbank lädt. Die Gesamtanzahl der GUI-Objekte betragen ca. 1.500.

Die Benutzeroberfläche wurde mittels Ereignis-Sequenz-Graphen [1,2] modelliert. Ein Ereignis-Sequenz-Graph ist ein gerichteter Graph, dessen Knoten die Ereignisse der zu testenden Software darstellen. Im Falle einer grafischen Benutzeroberfläche sind dies die Elemente der Oberfläche, die der Benutzer bedienen kann, wie z.B. Buttons, Eingabefelder, Tabellenreiter oder Checkboxen. Kanten zwischen den Ereignissen definieren legale Folgen von Ereignissen. Zwei durch eine Kante verbundene Ereignisse werden auch Ereignispaar genannt. Zudem gibt es jeweils einen ausgezeichneten Start- ([) und Endknoten (]). Ein Beispiel eines Graphen ist in Abbildung 1 gegeben.

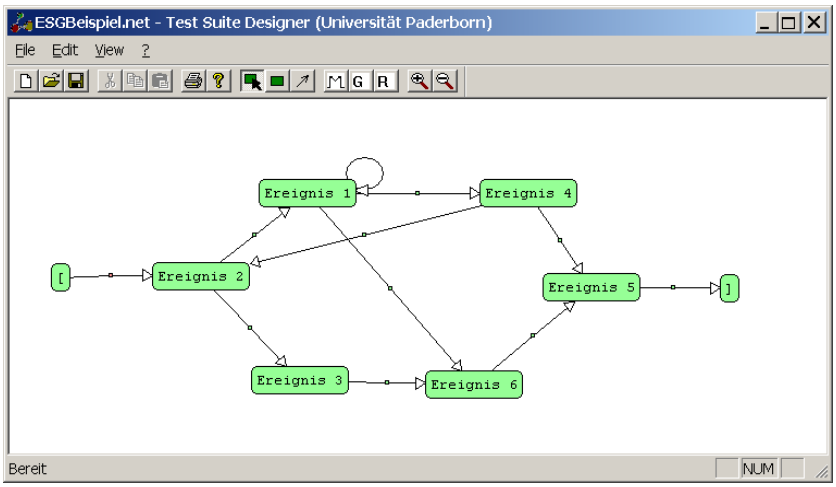

Abbildung 1: Modellierung eines Ereignis-Sequenz-Graphen mit Testtool

Aus einem Ereignis-Sequenz-Graph lassen sich Testfallsequenzen generieren, um beispielsweise alle Sequenzen von Ereignissen einer fixen Länge abzudecken. Für den Graph aus Abbildung 1 führt Tabelle 1 alle generierten Testsequenzen zur Überdeckung aller Ereignispaare auf. Die Testfälle sind in ihrer Gesamtlänge mittels eines Algorithmus zur Lösung des Chinese-Postman-Problems [3] minimiert worden.

Tabelle 1: Generierte Testsequenzen zur Überdeckung aller Ereignispaare

ID	Testfall
1	[, Ereignis 2, Ereignis 3, Ereignis 6, Ereignis 5,]
2	[, Ereignis 2, Ereignis 1, Ereignis 6, Ereignis 5,]
3	[, Ereignis 2, Ereignis 1, Ereignis 1, Ereignis 4, Ereignis 2, Ereignis 1, Ereignis 4, Ereignis 5,]

2 Projektdurchführung

Der Testprozess gestaltete sich im Wesentlichen aus sechs Phasen (vgl. Abbildung 2): Zur Initiierung der Testarbeiten wurde im ersten Schritt ein Modul der Benutzeroberfläche des Systems ausgewählt. Anhand der Pflichtenhefte, die das vollständige Design aller Bildschirmmasken und deren Verhalten beschreiben, wurden die Modelle aufgestellt. Aus den Modellen wurden anschließend jeweils kürzeste Testfallsequenzen zur Überdeckung aller Ereignispaare generiert, und zwar anhand unserer Algorithmen und Tools, die in [3] beschrieben werden. Die Testfallausführung selbst erfolgte zunächst manuell. Gefundene Fehler wurden im Bugtrackingsystem *Mantis* [4] dokumentiert.

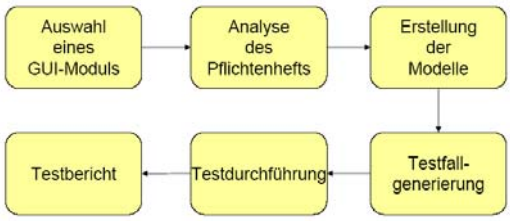

Abbildung 2: Testprozess

Tabelle 2 listet verschiedene Projektkennzahlen auf, die während des Testprozesses ermittelt wurden. Die Benutzeroberfläche wurde in Analogie zum Pflichtenheft in 12 logische Module unterteilt. Davon wurden im Rahmen des Projekts 4 Module getestet. Überraschend hoch fiel die Anzahl der aktivierbaren GUI-Elemente aus, die während der Erstellung der Modelle ermittelt wurden. Diese Anzahl beinhaltet sämtliche statisch lt. Pflichtenheft verfügbaren Elemente der Oberfläche, die ein Benutzer mittels Maus oder Tastatur bedienen kann, d.h. Buttons, Eingabefelder, Spaltenköpfe, Navigationshilfen, Karteireiter usw. Abbildung 3 zeigt die prozentuale Verteilung dieser aktivierbaren GUI-Elemente.

Tabelle 2: Projektkennzahlen

Bezeichnung	**Anzahl**	**getestet**
Module der GUI	12	4
Aktivierbare Elemente der GUI	1568	762
Anzahl ESG zur Überdeckung aller Elemente	154	77
∅ Anzahl Knoten je ESG	22	23
∅ Anzahl Kanten je ESG	36	39
Gesamtaufwand ESG-Modellierung (in Stunden)	89	53
∅ Modellierungsaufwand je ESG (in Minuten)	35	41
Generierte Testfälle	795	356
Aufwand Testdurchführung und Dokumentation (in Stunden)	-	84
Gefundene Fehler	-	78

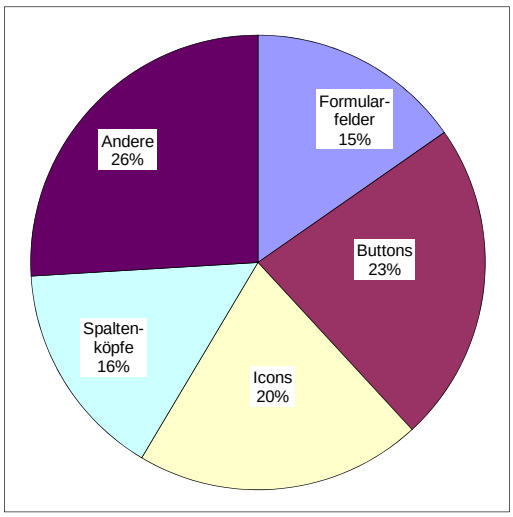

Abbildung 3: Prozentuale Verteilung der aktivierbaren GUI-Elemente

Wie leicht zu erkennen ist, nahm der Aufwand zur manuellen Testdurchführung mehr Zeit ein als die eigentliche Modellierung der Graphen, nämlich ca. 60% (84 Stunden zu 53 Stunden). Zukünftig ist geplant, die manuelle Testdurchführung zu automatisieren. Ziel ist es dabei, Test-Skripte aus den Graphen zu generieren, die durch eine entsprechende Testumgebung automatisiert ausgeführt werden. Ebenso sinnvoll ist ein Backtracking der gefundenen Fehler zurück in die Graphen, indem Ereignispaare, die Fehler detektiert haben, entsprechend im Graphen hervorgehoben werden. Im Idealfall muss sich der Testingenieur keine Gedanken mehr um die Aufstellung und Ausführung der Testfälle kümmern, sondern kann komplett auf Graphen-Ebene arbeiten.

Die oben beschriebenen Schritte zur Automatisierung erleichtern zudem Regressionstests, da bestehende und eventuell manuell erstellte Test-Skripte nicht aufwändig angepasst oder komplett neu geschrieben werden müssen. Parallel zur Weiterentwicklung der zu testenden Applikationen ist die Einführung eines Versionsmanagement-Systems für die Graphen geplant.

3 Diskussion und Analyse

Der gesamte beschriebene Testprozess erfolgte unabhängig von der Entwicklung der Software. Insbesondere die Tatsache, dass für sämtliche Szenarien im Pflichtenheft exakte Bildschirmmasken erstellt worden waren, ermöglichte eine unabhängige Modellentwicklung von der eigentlichen Software.

Die durchschnittliche Anzahl an Knoten je Graph lag im Gesamtprojekt bei ca. 22. Der kleinste ESG umfasste 6 Knoten; der größte 51 Knoten. Eine Analyse der Zeiten zur Modellerstellung ergab, dass die größeren Graphen im Durchschnitt nur wenig mehr Zeit benötigten. Dieses dürfte daran liegen, dass der größte Zeitaufwand in der Erfassung der Elemente der einzelnen Benutzeroberflächen und in der Analyse des Pflichtenheftes lag. Die eigentliche Modellierung des ESG ging deutlich zügiger von statten, zumal einige bestehende Modelle aufgrund ähnlicher Funktionalität übernommen werden konnten und nur angepasst werden mussten. Zur Beschleunigung des Prozesses wurde auch überlegt, die Designer der Benutzeroberfläche in den Test einzubeziehen. Allerdings stellte sich heraus, dass das Verständnis des ESG-Modells bzw. der Modellierung nicht ohne weiteres „übertragbar“ war. Obwohl das Konzept schnell verstanden war, herrschte doch immer wieder reger Diskussionsbedarf, ob eine bestimmte Kante des Graphen erlaubt oder nicht erlaubt sei. Trotz exakter Spezifikationen wurde hierdurch deutlich, wo Fehler oder Inkonsistenzen in der Spezifikation vorlagen. Jedoch konnten im Anschluss an die Modellierung auch viele Abweichungen zwischen Spezifikation und der getesteten Software aufgedeckt werden. Einige Beispiele für Fehler waren:

- Fehlende Spaltenköpfe/Buttons/Icons
- Falsche oder keine Sortierung von Tabellen bei Klick auf Spaltenköpfe
- Buttons/Icons ohne Funktion (z.B. „abbrechen“ oder „schließen“ bei modalen Fenstern)
- Systemabstürze bei Klick auf bestimmte Buttons
- Inaktive Buttons, die unter bestimmten Bedingungen aktiv sein sollten, es aber nicht waren

4 Fazit

Im Rahmen eines kommerziellen Projekts wurde die Benutzeroberfläche einer Software mittels Ereignis-Sequenz-Graphen getestet. Neben der systematischen Erfassung aller Elemente der Oberfläche ermöglichte dieses eine automatisierte Testfallgenerierung. Dadurch konnte eine Vielzahl von Fehlern aufgedeckt werden.

Weitere Ziele im Rahmen dieses Projektes sind die Automatisierung der generierten Testfälle mittels eines kommerziellen Testtools, so dass insbesondere Regressionstests automatisiert und damit deutlich schneller durchgeführt werden können. Des Weiteren ist die Messung der Codeüberdeckung (C_0-Test) geplant, die durch die aufgestellten Testfälle erreicht wird.

Literaturverzeichnis

[1] Belli, F., Finite-State Testing and Analysis of Graphical User Interfaces, Proceedings of 12th International Symposium on Software Reliability Engineering, ISSRE, S. 34-43, 2001

[2] Belli, F., Budnik, Ch. J., White, L., Event-based Modeling, Analysis and Testing of User Interactions: Approach and Case Study, Software Testing, Verification and Reliability, Vol. 16, Nr. 1, S. 3-32, John Wiley & Sons, 2006

[3] Belli, F., Budnik, C. J., Minimal Spanning Set for Coverage Testing of Interactive Systems, ICTAC, Lecture Notes in Computer Science, Vol. 3407, S. 220-234, Springer, 2004

[4] Mantis BugTracker, *http://www.mantisbt.org/*