Modellbasiertes Testdesign – ein Erfahrungsbericht

Dr. Anne Kramer

sepp.med gmbh
Gewerbering 9
91341 Röttenbach
anne.kramer@seppmed.de

Abstract: Gerade in der IT-Branche stellt die rasant steigende Komplexität der Produkte große Herausforderungen an Entwicklung und Test. Die Methode des modellbasierten Testdesigns ist eine Möglichkeit, schnell und effizient auf Änderungen zu reagieren. Wir stellen diesen Ansatz vor und gehen detailliert auf unsere – durchweg positiven – Erfahrungen ein.

1 Einleitung

In der IT-Branche – wie sonst auch – ist Time-To-Market alles. Dabei nimmt die Komplexität der Software-Produkte rasant zu. Unter solchen Bedingungen sind Entwicklung und Test zu einer beträchtlichen Herausforderung geworden. Hinzu kommt, dass zunehmend iterative und agile Prozesse verwendet werden. Durch die teilweise hohe Anzahl an Iterationen ist es unabdingbar, Regressionstests schnell und effizient durchzuführen.

Zum Glück stehen uns zur Bewältigung dieser Aufgaben Prozesse, Methoden und Werkzeuge zur Verfügung. Diese Prozesse müssen sich ebenfalls stetig an die erweiterten Anforderungen anpassen, um die Qualität der Produkte und die Wirtschaftlichkeit des Prozesses selbst zu garantieren. Für die SW-Entwicklung wurden zahlreiche Ansätze entwickelt. Dazu zählen u.a. das modellbasierte Design und die Steigerung der Wiederverwendbarkeit durch Entwicklung modularer, objektorientierter Komponenten.

Die systematische Anwendung analoger Methoden im Test ist hingegen schwach ausgeprägt. Im sicherheitsrelevanten Sektor der IT-Entwicklung sind automatisierte entwicklungsnahe Tests und Dokumenten-basierte manuelle Systemtests nach wie vor Standard. Diese Methoden erweisen sich jedoch als immer schwerfälliger bzgl. der erreichten Qualität, der Effizienz und insbesondere der Wartbarkeit.

Im Weiteren stellen wir ein methodisches Konzept vor, welches bei sepp.med und einigen unserer Kunden zur Anwendung kommt. Kern des geschlossenen, werkzeuggestützten Ansatzes ist eine Methode des modellbasierten Testdesigns. Wir erläutern zunächst den Ansatz und gehen dann detaillierter auf die von uns gewonnenen Erfahrungen ein.

2 Testdesign-Modellierung und Testfall-Erstellung

Zunächst werden – wie immer – die Anforderungen an das neue Produkt analysiert. Anforderungen können dabei Änderungswünsche, Ergebnisse von Risiko- oder Impactanalysen, sowie Fehlerbehebungen sein. Alle Anforderungen werden in IBM Rational Rose Test-Use Cases zugeordnet, wo sie bei Bedarf auf Vollständigkeit geprüft werden können („tracing"). Die Uses Cases werden sodann als Zustandsautomaten modelliert, wobei alle enthaltenen Alternativen und Pfade betrachtet werden. Sofern dies gewünscht ist, können auch Vor- und Nachbedingungen, sowie Testdaten modelliert werden. So entsteht das Testdesign-Modell.

Mit Hilfe des Testfallgenerators *.getmore* werden dann aus dem Zustandsdiagramm unter Berücksichtigung der Vor- und Nachbedingungen automatisch Testfälle erzeugt. Jeder generierte Testfall bildet einen möglichen Pfad ab, den das System für den speziellen Use Case durchlaufen kann. Der Use Case selbst entspricht also einem ganzen Satz von Testfällen, die ein gemeinsames Testziel verfolgen: den Test verschiedener Kombinationen von Zustandsübergänge, welche einen spezifischen Arbeitsablauf abdecken. Die Abarbeitungsstrategie kann im Testfallgenerator ausgewählt werden. Je nach Einstellung erhält man die vollständige Abdeckung der Übergänge oder die vollständige Abdeckung aller möglichen Einzelpfade, wobei die Pfadlänge ebenfalls konfigurierbar ist.

Sobald die Testfälle generiert sind, werden sie von *.getmore* toolunabhängig exportiert und können als manuelle Testfälle in Standard-Testwerkzeuge (z.B. IBM Rational TestManager oder Mercury TestDirector) importiert werden. Für spezielle Werkzeuge (z.B. *.faust* von sepp-med) können auch automatisierte Testskripten generiert werden. Die Methode des modellbasierten Testdesigns ist aber unabhängig von der speziellen Wahl der Werkzeuge, welche in diesem Beitrag erwähnt werden.

3 Erfahrungen und Nutzen

Der Hauptnutzen der Methode besteht in der gesteigerten Effizienz, der gesteigerten Qualität und der Vermeidung von Fehlern durch Automatismen. Letzteres hat gerade in sicherheitsrelevanten Prozessen massive Bedeutung (z.B. Regelwerk der FDA). Außerdem erhalten wir eine höhere Flexibilität und Qualität des Testdesigns. Damit sind wir in der Lage, immer komplexer werdende Produkte und Prozesse handhaben zu können (man denke nur an die Entwicklung von ganzen Produktfamilien und dem damit verbundenen Testaufwand).

Ein wichtiger Vorteil des modellbasierten Testdesigns besteht in der effizienteren Wartbarkeit der Testfälle. Änderungen werden an einer zentralen Stelle (im Testdesign) durchgeführt und dann über automatisierte Prozesse in zahlreiche Testfälle übertragen – eine Vorgehensweise, die auf Grund der Nachvollziehbarkeit der einzelnen Arbeitschritte weitaus weniger fehleranfällig ist als manuelle Änderungen einzelner Testfälle.

Durch den strukturierten Ansatz wird ein erhöhtes Abstraktionsniveau erreicht. Beim Erstellen des Modells werden Zusammenhänge und Abhängigkeiten zwischen Anforderungen geklärt. Widersprüche und fehlende Anforderungen fallen viel leichter auf. Im Übrigen erleichtert die graphische Darstellungsweise die Kommunikation erheblich. Da beim Review nicht einzelne Testfälle, sondern das Testmodell als Ganzes betrachtet wird, ist es sehr viel einfacher, den Überblick zu behalten.

Sofern vorhanden, können bestehende Design-Modelle der Entwicklungsabteilungen übernommen werden. Allerdings liegt der Fokus der generierten Testfälle bei Verwendung eines Entwicklungsmodells auf anderen Aspekten als bei der Erstellung eines spezifischen Testmodells. Während das Entwicklungsmodell der Verifikation dient („Funktioniert alles so wie spezifiziert?“), dient ein Testmodell mehr der Validierung („Erfüllt mein Produkt die Anforderungen?“). Daher eignen sich Entwicklungsmodelle eher für Unit- und Integrationstests, während für die Teststufen Systemtest und Kundenabnahmetest spezifische Testmodelle entworfen werden sollten.

Eine quantitative Abschätzung der Zeit- und Geldersparnis ist naturgemäß schwierig, da wir bei keinem unserer Projekte sowohl den modellbasierten als auch den „klassischen“ Ansatz durchgeführt haben. Auf Grund der diversen Erfahrungen, die wir mit verschiedenen Projekten gewonnen haben, lässt sich jedoch folgendes festhalten:

1. Die Aufwände für das Testdesign, also die Festlegung der Use Cases im modellbasierten Ansatz und die Aufteilung in Testfälle im klassischen Ansatz sind weitestgehend identisch.

2. Die Erstellung des Testmodells ist aufwändiger als die direkte Implementierung der Testfälle. Wir veranschlagen typischerweise einen Mehraufwand von 25%. Allerdings – und dies lässt sich so einfach in Zahlen fassen – deckt das Testmodell in der Regel mehr Aspekte ab und wir erhalten eine höhere Qualität der Tests.

3. Den eigentlichen Zeitgewinn erzielt man bei der Wartung bestehender Testfälle. Der Pflegeaufwand ist hierbei um 75% und mehr reduziert. Besonders anschaulich wird der Einfluss des Wartungsaufwandes an einem kleinen Projekt von insgesamt 50 Testfällen. Die Testfälle wurden modellbasiert erstellt. Noch vor Freigabe wurden dann formale Änderungen erforderlich, die alle 50 Testfälle betrafen. Durch den modellbasierten Ansatz konnten die Änderungen in einem Viertel der ursprünglich veranschlagen Zeit eingebracht werden.

In diesem konkreten Beispiel war der Wartungsaufwand gering, d.h. weit unter dem Gesamtaufwand. Typischerweise muss man aber mit Wartungsaufwänden von 400% des Erstaufwandes rechnen. Allgemein gilt: je mehr Testfälle gepflegt werden müssen und je mehr Iterationen ein Testfall durchläuft, umso größer ist die Ersparnis, die sich durch den modellbasierten Ansatz erzielen lässt. Im Mittel konnten wir Einsparungen von gut 50% gegenüber der „klassischen“ Methode (Entwicklung Dokumenten-basierter manueller Tests) erzielen.

4 Zusammenfassung

Unsere Erfahrungen mit modellbasierten Testdesigns sind durchweg positiv. Die Testqualität wird durch den strukturierten Ansatz deutlich verbessert. Verwendet man darüber hinaus den Testfallgenerator *.getmore*, kann die Effizienz erheblich erhöht und die Anzahl der Fehlerquellen reduziert werden. Die Vorteile werden umso spürbarer, je höher der Wartungsaufwand im Vergleich zum Erstaufwand ist.