Testautomatisierung – out of the box?

Autoren: **Dierk Engelhardt**
imbus

Datum: 30.07.2012

Version: 2.2

Status: in Arbeit

imbus AG
Kleinseebacher Str. 9
91096 Möhrendorf
Tel. 09131/7518-0
Fax 09131/7518-50
info@imbus.de,
www.imbus.de

Inhalt

1 Ausgangssituation

Der Wunsch ist so alt wie die Testautomatisierung selbst: Statt manuell zu testen spult ein Testautomat die Testfälle vollautomatisch ab – ganz ohne weitere Arbeit.

Die Realität zeigt jedoch: Die erforderlichen Testskripte müssen erstellt werden und – was noch viel entscheidender ist – zukünftig an neue Release-Stände, Änderungen usw. angepasst werden. Lohnt sich da Testautomatisierung in der Praxis überhaupt? Oder ist der Traum vom „Vollautomaten“ ausgeträumt?

Testautomatisierung lohnt sich - wenn man ihr mit dem richtigen Konzept zu Leibe rückt! Im folgenden Artikel werden die möglichen Konzepte am Beispiel der GUI-Testautomatisierung erläutert.

Testautomatisierung soll Aufwand, Zeit und Geld durch mehr Effizienz beim Testen sparen, d.h. am besten gleichzeitig Entwicklungskosten verringern, Entwicklungszeiten verkürzen und die Produktqualität spürbar erhöhen – soweit die Theorie.

Kosten im Test lassen sich spürbar durch die Reduzierung des Aufwands in der Testdurchführung verringern. Das bedeutet in der Regel, dass in Relation teure manuell durchgeführte Tests durch automatisiert durchgeführte Tests ersetzt werden. Die Alternative der Verringerung der reinen Anzahl der Tests durch intelligente Auswahl der richtigen Tests hat oft nicht die erwünschten Einsparungseffekte. Im Gegenteil: Immer häufiger sorgt die Produktstrategie dafür, dass die Anzahl der notwenigen Tests weiter steigt. Besondere Aufwandstreiber sind dabei die Unterstützung von mehreren Sprachen, Zielsystemen oder Browsern durch die zu testenden Systeme. Dies führt geradezu zwangsläufig zur Erhöhung der Anzahl der Tests.

Gerade diese Aufwandstreiber machen jedoch eine Automatisierung von Tests besonders attraktiv, da sie oft die mehrfache Ausführung ähnlicher Tests bedeuten, die sich nur in der Verwendung verschiedener Daten bzw. in der getesteten Variante des Systems unterscheiden.

Sollen gleichzeitig anspruchsvolle Qualitätsziele erreicht werden, sind damit in der Regel auch mehr auszuführende Tests verbunden

2 Rahmenbedingungen beim Einsatz einer Testautomatisierung beachten

Eine weitere Motivation für die Automatisierung von Tests entsteht durch den zunehmenden und weit verbreiteten Trend hin zu agileren Prozessen. In diesem Zuge werden sog. Continuous Integration Systeme eingesetzt: Diese erzeugen zyklisch neue Zwischenversionen innerhalb des Entwicklungsprozesses und sind in der Lage für jede Zwischenversion Tests anzustoßen. Diese Tests müssen dann natürlich automatisiert durchgeführt werden, da ein manueller Test hiermit nicht mehr Schritt halten kann.

Damit es zu Kosteneinsparungen kommt, muss der zu erwartende Nutzen einer Testautomatisierung den damit verbundenen Aufwand auf Dauer übersteigen. Ist das nicht der Fall oder ist der Nutzen marginal, lohnt sich die Investition nicht. So banal diese Aussage auch ist, wird leider immer wieder eine Kosten-Nutzen-Betrachtung im Rahmen der Einführung einer Testautomatisierung unterlassen oder nicht wichtig genug genommen. In der Tat, lohnt sich nicht in jeder Situation eine Testautomatisierung. Zudem führt auch nicht jede Art und Weise, in der eine Testautomatisierung erstellt wird, zum Ziel. Beim Aufwand sind zwei prinzipiell Kostenblöcke zu beachten:

- **Einmal anfallende initiale Kosten**: Aufwände für die Erstbeschaffung von Werkzeugen, die Konzeption der Testautomatisierung und der damit verbundene KnowHow-Aufbau und die initiale Implementierung der Testautomatisierung
- **Fortlaufende Kosten für den Betrieb**: Aufwände für die Wartung der Werkzeuge, die Pflege und Optimierung der Testautomatisierung, der Ausbau des KnowHows und die laufende Erweiterung der Implementierung

Die fortlaufenden Kosten sind der Schlüssel für den dauerhaften Nutzen einer Testautomatisierungslösung. Werden diese Kosten bei der Kosten-Nutzen-Betrachtung nicht gebührend beachtet, ist der dauerhafte Nutzen stark gefährdet. Letztendlich sind dabei vier Faktoren für einen dauerhaften Erfolg entscheidend:

- **Hoher Automatisierungsgrad**: Je weniger manuelle Tests zusätzlich zu den automatisierten Tests durchgeführt werden müssen, desto größer ist der Effekt.
- **Wartbarkeit**: Die laufenden Kosten der Pflege, der Änderungen und der Erweiterungen müssen gegenüber einer reinen manuellen Testlösung deutliche Einsparungen an Aufwand, Zeit und Kosten bringen. Erfahrungsgemäß ist die Wartbarkeit direkt von der gewählten Testautomatisierungsarchitektur abhängig.
- **Geschwindigkeit**: Die Anfangsinvestition muss sich durch die Einsparungen möglichst schnell rechnen.
- **Optimale Investition**: Art und Umfang der Anfangsinvestition muss den dauerhaften Nutzen der Investition berücksichtigen.

Neben den oben genannten Faktoren sind auch technische Aspekte und die Auswahl des geeigneten Werkzeugs zu beachten. Einer der häufigsten Fehler ist, dass die Auswahl des Werkzeugs völlig im Vordergrund steht, was zu einer vorschnellen Beurteilung einer Automatisierungslösung führt und ein dauerhafter Nutzen nicht eintreten kann. Denn nicht jede technisch machbare Lösung ist auch wirtschaftlich die richtige. Oft zeigt sich im Nachhinein, dass die teuer erstellten Testskripte auf Dauer nicht laufen oder die Wartungskosten unterschätzt wurden oder von den Testern im operativen Testbetrieb nicht beherrscht werden.

Bei der Konzeption einer Automatisierungslösung – wenn diese Konzeption überhaupt erfolgt – werden die meisten Fehlentscheidungen getroffen, die sich auf lange Sicht als Nutzenkiller erweisen. Hier gilt es, vor allem die Wartbarkeit der Lösung zu beachten und welche Methoden und Architekturen der Testautomatisierung für das organisatorische Umfeld und das Testobjekt geeignet sind.

3 Die richtige Architektur wählen

Eine Testautomatisierungsarchitektur beschreibt die unterschiedlichen Komponenten, die für eine Testautomatisierung verwendet werden, und das Zusammenspiel dieser Komponenten. Der Bedarf einer bewussten Architektur steigt mit der Notwendigkeit, eine zunehmende Anzahl von Komponenten zu orchestrieren. Zu Beginn der Testautomatisierung steht allzu häufig nur die Auswahl eines passenden Automatisierungswerkzeugs im Vordergrund, so dass die wirkliche Anzahl der Komponenten beim Einsatz einer Testautomatisierung unterschätzt wird und keine bewusste Architekturentscheidung vorgenommen wird.

Ob agile Entwicklung oder klassisches V-Model, ob Web-Applikation oder Legacy-System: Von Capture-Replay über schlagwortbasierte Automatisierung und die Verwendung vorgefertigter Bibliotheken bis hin zum modellbasierten, generierten Test – für jede Einsatzsituation gibt es eine passende Testautomatisierungsarchitektur.

Damit der oben genannte Nutzen erzielt werden kann, sollten die folgenden Anforderungen und ihr Erfüllungsgrad durch die zu wählende Architektur betrachtet werden:

1. **Robust**
   Gegen geänderte Anforderungen bis zu kleineren Änderungen an der Bedienoberfläche muss eine Testautomatisierung so unempfindlich wir möglich sein, da sie sonst jede Art der Änderung ebenfalls nachvollziehen muss.
2. **Wartbar**
   Änderungen sollen mit geringst möglichem Aufwand machbar und alle zu ändernden Stellen müssen leicht identifizierbar sein, im Idealfall: Jede beliebige Änderung an dem zu testenden System zieht nur Änderungen an definierten und leicht auffindbaren Stellen in der Testautomation nach sich. Diese Änderungen müssen im Idealfall ohne Seiteneffekte durchführbar sein.
3. **Einfach**
   Der Einsatz muss schnell und einfach erfolgen können. Eine intensive Einarbeitung in Erstellung und Verwendung ist nicht unbedingt notwendig.
4. **Kurzfristig günstig**
   Es sollen Quick-Wins erzielbar sein, d.h. erste Erfolge sollen sich schnell und kostengünstig einstellen.
5. **Langfristig kostensparend**
   Auf die Dauer sollen die Einsparungseffekte erhalten bleiben oder besser noch verstärkt spürbar werden.
6. **Werkzeugunabhängig**
   Ändern sich Softwaresysteme technisch, aber nicht fachlich, z.B. beim Übergang zu einer Weboberfläche, dann sind die vorher passenden Werkzeuge oft nicht mehr verwendbar. Eine Testautomatisierung soll beim Wechsel des Werkzeugs nicht vollständig und in allen Teilen neu erstellt werden müssen.
7. **Arbeitsteilig**
   Oft wird im Test das Spezialwissen unterschiedlicher Personen genutzt, z.B. steuert der Fachbereich die zu testenden Szenarien und Abläufe bei, die ein Testdesigner in Testspezifikationen abbildet, die wiederum ein Testautomatisierer im Testautomatisierungswerkzeug implementiert. Dieses Vorgehen soll nicht behindert, sondern unterstützt werden.

Welche der Anforderungen mit welcher Gewichtung in die Beurteilung einer Architektur eingeht, ist jeweils individuell für jede Projektsituation festzulegen, denn nicht für jedes Projekt ist z.B. ein arbeitsteiliges Vorgehen im Test erforderlich oder realisierbar.

Grundlegend lassen sich die folgenden vier Alternativen bei der Realisierung einer Testautomatisierung unterscheiden, die mit Hilfe des Erfüllungsgrades der Anforderungen beurteilt werden können:

3.1 Capture/Replay

Die später auszuführenden Tests werden im Laufe einer manuellen Ausführung mitgeschnitten (Capture) und in für das Automatisierungswerkzeug ausführbaren Code gespeichert. Dieser Code wird dann später wieder vom Werkzeug ausgeführt (Replay). Eine Überarbeitung der so erzeugten Skripte geschieht nicht oder nur selten.

Vorteile	Nachteile
• Kurzfristiger Start möglich • Automatisierung schnell fertig gestellt • Geringer Aufwand	• Automatisierung kann erst erstellt werden, wenn Testobjekt zur Verfügung steht • Keine Trennung der Aufgaben • Wartung betrifft immer Testfall, Daten und Umsetzung auf einmal • Kleine Änderungen im Testobjekt können verheerende Auswirkungen haben • Extrem hohe Wartungsaufwände • Funktioniert nur, wenn sich das zu testende System weder funktional noch an der Oberfläche fortentwickelt

Typische Anwendungsfälle wären die wiederholte Durchführung der exakt gleichen Testsequenzen z.B. im Rahmen der Abnahme genau eines Releases oder das Regressionstesten von Refactoring-Aktivitäten.

Beide Anwendungsfälle zeichnen sich dadurch aus, dass man auf der einen Seite nennenswerte Einsparungen bei der manuellen Durchführung von Tests hat, auf der anderen Seite das zu testende System keinerlei Änderungen der Funktionalität oder der Oberfläche erfährt. Man darf nur nicht dem Trugschluss erliegen, diese Testautomatisierung beim nächsten funktional und/oder GUI-mäßig veränderten Release in der gleichen Art und Weise wieder nutzen zu können.

3.2 Datengetrieben

Aus den automatisierten Abläufen der Testfälle werden die Testdaten extrahiert und das Skript hinsichtlich der Datenverwendung verallgemeinert. Die Tests bestehen jetzt aus Abläufen und Datentabellen. Soll eine andere Kombinationen von Daten (= ein neuer Testfall) für einen bestehenden Ablauf getestet werden, muss lediglich ein weiterer Datensatz hinzugefügt werden. Im Gegensatz zum Capture/Replay wird hier Softwareentwicklung (im Sinne einer Überarbeitung der aufgezeichneten Skripte) notwendig, allerdings verteilt sich jetzt der Aufwand auf viele Testfälle.

Vorteile	Nachteile
• Getrennte Verwaltung und Pflege von Testimplementierung und Testdaten (erste Aufgabenteilung) • Immer noch relativ günstig • Verfeinerung des Capture/Replay-Ansatzes • Wartungsaufwände verteilen sich auf mehrere Testfälle	• Automatisierung kann zum großen Teil erst erstellt werden, wenn Testobjekt zur Verfügung steht • Überarbeitung der Skripte erfordert Know-how • Testlogik ist mit der Testimplementierung verwoben • Immer noch sehr hoher Wartungsaufwand • Komplexität der Skripte steigt (manchmal erheblich)

Da es sich um eine Fortentwicklung des Capture/Replay-Ansatzes handelt, sind die typischen Anwendungsfälle ähnlich. Ein konkretes Beispiel: Durch die Änderung der Mehrwertsteuergesetzgebung (d.h. Berechnung der zu zahlenden Mehrwertsteuer) muss die Preisberechnung eines Moduls eines ERP-Systems regressionsgetestet werden. Mit der Äquivalenzklassenmethode werden hierfür gut 1.300 Testfälle bestimmt, welche alle mit der identischen Testsequenz durchzuführen sind. Dieses Szenario „schreit“ geradezu nach dem datengetriebenen Ansatz.

3.3 Schlüsselwortbasiert

Die Testabläufe werden aus Bausteinen, den Schlüsselworten, zusammengesetzt (siehe auch [FAU04]). Diese Schlüsselworte beschreiben einzelne Aktionen des Tests, also z. B. Eingaben, Bedien- und Prüfaktivitäten am Testobjekt. Die Schlüsselworte werden katalogisiert und mehrfach verwendet. Die Trennung von Testlogik und Testdaten, wie im datengetriebenen Test, wird auf die Schlüsselworte übertragen, d.h. Schlüsselworte erhalten eine Datenschnittstelle und die Testfälle benötigen mindestens die Daten, die die darin verwendeten Schlüsselworte benötigen. In der Testautomatisierung werden nur noch einzelne Schlüsselworte implementiert, die später zur Laufzeit in der Reihenfolge, wie in den spezifizierten Abläufen festgelegt, abgearbeitet werden.

Innerhalb der schlüsselwortbasierten Methode sind zwei Realisierungsansätze zu beobachten:

3.3.1 High-Level

Die Logik der Testfälle wird durch Schlüsselworte auf einer einzigen Ebene dargestellt.

Vorteile	Nachteile
• Grundbausteine und Logik der Automatisierung können sehr effektiv erstellt werden, bevor das Testobjekt zur Verfügung steht • Verwaltung der Testfälle, Tabellen und Schlüsselworte kann durch einfache Werkzeug unterstützt werden • Langfristiger Nutzen • Wartungsfreundlich • Bessere Zugänglichkeit durch Fachtester	• Sorgfältige Planung / Konzeption und anfängliche Investition nötig • Da die Schlüsselworte in der Regel fachlich geprägt sind, können die Bausteine in der Implementierung für sich noch eine gewisse Komplexität beinhalten

Der typische Anwendungsfall ist die Automatisierung immer wieder durchzuführender Regressionstests, d.h. Test der Kernfunktionalität des zu testenden Systems in Anlehnung an die wichtigen Geschäftsprozesse, um auch die unerwünschten und unvorhersehbaren Nebenwirkungen

von Änderungen an der Software fernab der manuell durchgeführten Tests zu entdecken. Dieser Anwendungsfall erfordert es, dass die automatisierte Regressionstest-Suite mit möglichst geringem Aufwand an zukünftige Versionen des zu testenden Systems anpassbar ist.

3.3.2 Hierarchisch

Zusätzlich zum High-Level-Ansatz besteht die Möglichkeit, Schlüsselworte innerhalb von Schlüsselworten aufzurufen. Es entstehen somit mehrere Ebenen von aufrufenden und aufgerufenen Schlüsselwörtern. So lassen sich immer wiederkehrende Abläufe in einem Schlüsselwort höherer Ebene zusammenfassen, z. B. kann das Login an einem System, eigentlich bestehend aus mehreren Einzelschritten, in einem Aufruf zusammengefasst werden:

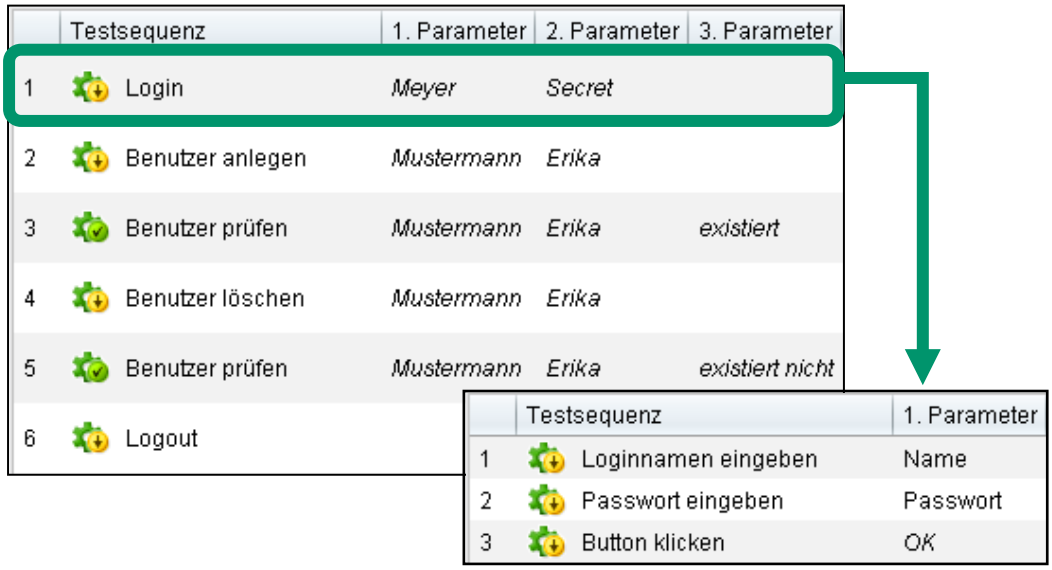

Abb.1: Loginprozedur abgebildet über zusammengesetzte Schlüsselworte auf zwei Ebenen am Beispiel des Werkzeugs Test*Bench*.

Es werden jetzt nur noch die Schlüsselworte automatisiert, die nicht weiter zusammengesetzt sind. Dies erhöht den Anspruch an die Ausführungsebene, an das Bausteinmanagement und die Ausführungsebene im Vergleich zu einer nicht-hierarchischen High-Level Methode.

Vorteile	Nachteile
• Fachliche Ebene und technische Ebene können problemlos gleichzeitig abgebildet werden • Test-First-Ansatz auf hoher Ebene wird möglich, langsame Verfeinerung • Ausgezeichnete Robustheit und Wartbarkeit • Geringe Aufwände für Testimplementierung • Erlaubt durch bessere Übersicht größere Teams	• Tester müssen für das Vorgehen qualifiziert sein • Erfordert sorgfältige Planung, geeignete Werkzeugunterstützung und anfängliche Investition

Die typischen Anwendungsfälle sind identisch mit dem schlüsselwort-basierten Ansatz.

3.4 Modellbasiert

Die Spezifikation der Tests erfolgt auf Ebene von Modellen (oftmals mit Hilfe der UML). Aus diesen Modellen werden Testfälle einschließlich der Testautomatisierung und eventuell der Sollergebnisse generiert. Nach Änderungen und der dadurch notwendigen Aktualisierung der Modelle kann eine Testautomatisierung jederzeit neu generiert werden. (Näheres siehe [ROS10]).

Vorteile	Nachteile
• Testmodelle können sehr früh erstellt werden • Abgeleitete Testfälle können jederzeit wieder generiert werden • Wartung beschränkt sich auf die Überarbeitung von Modellen und kleinen Implementierungsbausteinen • Für bestimmte Testaufgaben kann eine sehr große Testabdeckung mit geringem Aufwand erreicht werden	• Erfordert Mitarbeiter, die im Erstellen von Testmodellen ausgebildet sind • Steuerung der Generatoren hinsichtlich der Ergebnismenge der Tests ist zuweilen schwierig • Hoher Prozessreifegrad notwendig • Wartungsaufwände für Generatoren kommen hinzu oder diese müssen passend eingekauft werden

Typische Anwendungsfälle für modellbasiertes Testen sind zustandsbasierte Tests wie man sie z.B. beim Test von Protokoll-Stacks findet, oder Geschäftsprozess-Tests.

Sind die zuvor genannten Ansätze schon länger bekannt und für die skizzierten Anwendungsfälle erprobt und bewährt, so gibt es hinsichtlich der Bedeutung des modellbasierten Tests durchaus verschiedene Meinungen in der Tester-Community. Der Autor dieses Artikels vertritt hierzu die Einschätzung, dass modellbasiertes Testen für bestimmte Testaufgaben eine Zukunft hat. Dies mag auch ausreichend Motivation dafür sein, sich schon heute mit dieser innovativen Methode zu beschäftigen und eventuell damit verbundene Pionieraufgaben als positive Herausforderung zu begreifen

3.5 Beurteilung der Alternativen

Beurteilen wir nun die gezeigten Methoden hinsichtlich der gestellten Anforderungen, zeigt sich das folgende Bild:

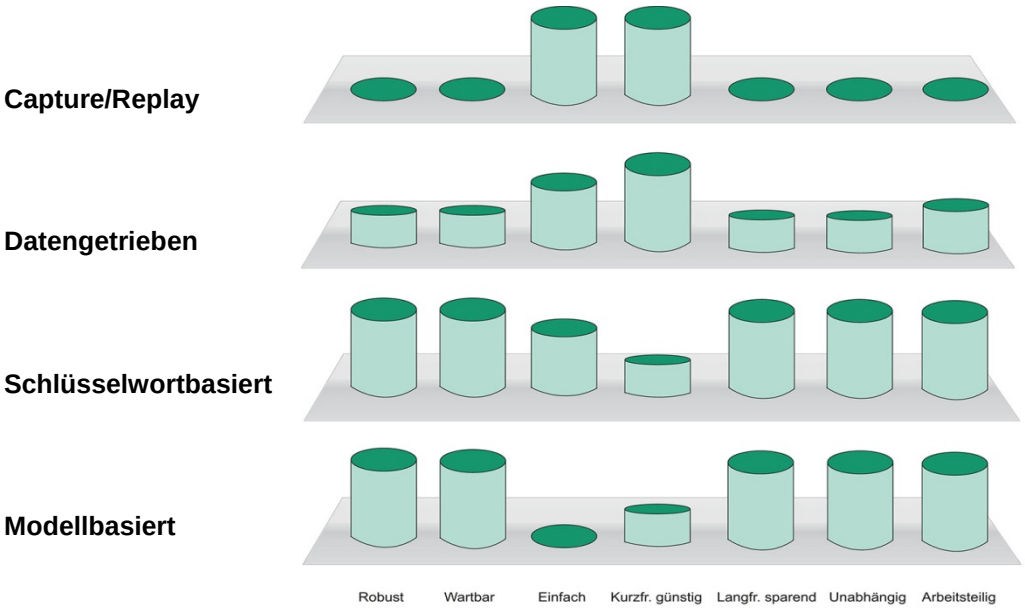

Abb.2: Testautomatisierungsmethoden und ihr Erfüllungsgrad der Anforderungen.

Es wird erkennbar, dass der schlüsselwortbasierte Ansatz die günstigste Verteilung hinsichtlich des Erfüllungsgrades der Anforderungen hat. Die für einen langfristig die Kosten senkenden Einsatz wichtigsten Anforderungen Robustheit und Wartbarkeit haben die höchste Bewertung, ebenso, wie die Anforderungen, die eine gewisse Investitionssicherheit hinsichtlich Werkzeugentscheidung (unabhängig) und Teamgröße (arbeitsteilig) bedeuten. Tatsächlich wird dieses Vorgehen als „state of the art“ in der Testbranche angesehen und hat sich vielfach in Testprojekten bewährt.

In letzter Zeit werden immer häufiger modellbasierte und schlüsselwortbasierte Methoden in Kombination angewandt (Informationen zu **„Managed Model Based Testing“** siehe [TiT11]). Aus den Modellen werden schlüsselwortbasierte Testfälle generiert, die dann nach dem oben beschriebenen Vorgehen der schlüsselwortbasierten Methode automatisiert werden. Dies verbindet den Vorteil der Ausdrucksstärke der Modelle mit der einfacheren Anwendbarkeit der Schlüsselworte. Werden bei der Generierung nur „bekannte“, d.h. schon automatisierte Schlüsselworte erzeugt, kann nach der Generierung sofort automatisiert getestet werden.

4 Exkurs: Die Abstraktionslücke

Von der Idee eines Testfalls zu dessen Umsetzung in einem Testautomaten ist es oft ein weiter Weg. Auf der einen Seite steht z.B. eine funktionale Anforderung, deren korrekte und vollständige Implementierung zu belegen ist. Auf der anderen Seite liefern die automatisierten Testskripte eine sehr technische Umsetzung der ursprünglichen Testidee, die sehr stark von nicht-testrelevanten Eigenschaften des Testobjekts und Testautomaten abhängen kann.

Viele Testteams profitieren davon, dass es einerseits Personen gibt, die die fachliche Seite des Testobjekts besonders kennen (Fachtester) und andererseits Personen, die die technischen Aspekte des Testens besonders gut beherrschen (Technische Tester). Erstere sind in der Regel die Personen, die die notwendigen Tests bestimmen und auch spezifizieren. Letztere sind in der Regel die, welche eine Testautomatisierung erstellen. Die Fachtester werden die Tests eher abstrakter und auf Ebene der Businesslogik beschreiben, da dort tiefes Verständnis über die Funktionsweise des Testobjekts vorliegt. Im Gegensatz dazu müssen die Testautomatisierer bei der Erstellung der Testautomatisierung jeden noch so kleinsten Schritt berücksichtigen und im Skript hinterlegen. Zusätzlich müssen sie noch die auftretenden technischen Probleme lösen, wenn sich z.B. bestimmte GUI-Elemente nicht reibungslos ansprechen lassen. Beide Gruppen betrachten ein Testobjekt demnach aus anderer Perspektive und mit einem sehr unterschiedlichen Detailgrad, d.h. sie sprechen in ihren Testbeschreibungen „unterschiedliche Sprachen“. So versteht der Testautomatisierer die fachlich spezifizierten Tests womöglich nicht in allen Teilen korrekt, so dass die zugehörigen automatisierten Tests in vielleicht entscheidenden Nuancen anders ablaufen werden. Im Gegenzug ist der Fachtester auch nicht in der Lage, die Implementierung der automatisierten Tests auf fachliche Korrektheit zu prüfen.

Diese Lücke im Verständnis zwischen beiden Gruppen wird in letzter Zeit immer häufiger als **„Abstraktionslücke“** bezeichnet. Formale Spezifikationsmethoden, wie der hierarchische Schlüsselwort-basierte Ansatz helfen diese Lücke entscheidend zu schließen. Vonnöten ist dabei eine Werkzeuglandschaft, die die jeweiligen Perspektiven der Benutzer ohne Medienbruch und ohne extra Aufwände in Einklang bringt.

5 Frameworks – wartbar und wieder verwendbar

Der entscheidende Erfolgsfaktor einer Testautomatisierung ist deren Wartbarkeit. Die Architekten einer Automatisierungslösung müssen vermeiden, dass jede Änderung an beliebiger Stelle des Testobjekts auch gleich Änderungen auf der Ebene der Testautomatisierung bedeuten und damit den Einsatz eines oft raren und teuren Testautomatisierers nötig machen. Gleiches gilt natürlich analog auch umgekehrt. Weiterhin gilt es, das Auftreten der oben beschriebenen Abstraktionslücke zu vermeiden.

Die Lösung des Problems ist die Fortentwicklung des hierarchischen Ansatzes, indem eine weitere dritte Ebene an der Basis eingeführt wird: In der Regel lassen sich Softwaresysteme mit Benutzeroberflächen aus Sicht des Tests grob in drei logische Ebenen untergliedern: Businesslogik,

grundsätzliches GUI-Layout und technische Ebene. Über die Hierarchieebenen lassen sich die Ebenen abbilden: Auf oberster und abstraktester Ebene der Testspezifikation wird die **Businesslogik** beschrieben. Der Fachtester spezifiziert Schlüsselworte in seiner Fachsprache und beschreibt die Funktionalität aus seiner fachlichen Sicht. Er muss auf technische Besonderheiten des Testobjekts oder der Testautomatisierung keine Rücksicht nehmen.

Auf der mittleren Ebene, der **Navigations-Ebene**, wird der logische Ablauf mit Hilfe von Menüs, Dialogen und Zuständen des Testobjekts dargestellt, d.h. die fachliche Ebene wird auf die Gegebenheiten der Testschnittstelle (z.B. der Masken) abgebildet. Z.B. wird aus einem Schlüsselwort der Businesslogik „Benutzer anlegen“ im ersten Unterschlüsselwort der Navigationsebene ein „Menü öffnen: Administration→Benutzer→Neu“. Erfahrungsgemäß entstehen im Rahmen der Navigationsebene wenige überschaubare Navigationsschlüsselworte pro Maske oder Zustand.

Die dritte und unterste Ebene, die sog. **Low-Level-Ebene**, bietet den Zugriff auf die Testautomationsschnittstelle des zu testenden Systems, z.B. umfasst diese Ebene bei einer GUI-Testautomatisierung alle notwendigen Schlüsselworte, um auf die GUI-Controls zugreifen zu können. Diese Schlüsselworte sind daher vom zu testenden System weitestgehend unabhängig (bis auf das eine oder andere Custom Control). Auf Basis dieser Low Level-Ebene können alle Schlüsselworte der Navigationsebene beschrieben und damit im weitesten Sinne „implementiert“ werden.

So wird der Testautomatisierer im obigen Beispiel das Navigations-Schlüsselwort „Menü öffnen: Administration→Benutzer→Neu“ als dreimaligen Aufruf des Low-Level-Schlüsselwortes „Select menu: menu item“ abbilden, wobei er für „menu item“ jeweils den passenden Menüpunkt einsetzt. Lediglich die Schlüsselworte dieser Low-Level-Ebene müssen automatisiert werden. Im obigen Beispiel demnach nur das Schlüsselwort „Select menu: menu item“. Damit ist die Aufwand für die Erstellung der reinen Testautomatisierungsskripte endlich und überschaubar und vor allem vom Komplexitätsgrad her erheblich geringer, als wenn ganze Testabläufe in einem Skript abgebildet werden müssen. Diese Verringerung der Komplexität ist eine weitere Voraussetzung für eine wartbare Testautomatisierungsarchitektur. Ebenso, dass die Skripte für den Testautomaten nur noch von den Controls des Testobjekts abhängig sind.

Dieser Ansatz vereinfacht die Erstellung von Testautomatisierungsframeworks, die vorgefertigte Automatisierungselemente zur Verfügung stellen. Ein Beispiel sind die sogenannten GUI-Frameworks. Hier werden kleinste Bedienaktionen, wie z.B. Klicken eines Buttons, Auswahl eines Menüpunkts oder Auswahl aus einer Liste, die eine gegebene Benutzeroberfläche bietet, als Low-Level-Schlüsselworte implementiert. Aus diesen Schlüsselworten lassen sich alle Abläufe in einer Applikation über die Navigations-Ebene und die Ebene der Businesslogik zusammenstellen. Automatisiert werden müssen jetzt nur noch die Low-Level-Schlüsselworte, deren Anzahl gering und endlich ist (je nach GUI-System sind ca. 100 Schlüsselworte).

Die letztendliche Abbildung des Tests auf eine Reihe von Controls der Low-Level-Ebene erzeugt einen hohen Grad an Wiederverwendung auf der Low-Level-Ebene, aber auch auf der Navigations-Ebene, was wiederum die Wartbarkeit erhöht und die Aufwände bei Änderungen bestmöglich minimiert.

Abb.3: Hierarchische Abbildung der Schlüsselworte mittels dreier Ebenen.

6 Test-First-Ansatz auf hoher Ebene wird möglich

Ändert sich nun die Businesslogik, muss lediglich die Navigations-Ebene zusätzlich angepasst werden, aber nicht mehr die Low-Level-Ebene, da die GUI-Controls in der Regel am stabilsten sind. Viele fachliche Änderungen am Testobjekt erfordern also keine Änderungen an der Programmierung der Testautomatisierung mehr. Mit diesem Vorgehen ist nun auch eine Rollenteilung zwischen fachlich versierten Testern und technisch geprägten Testern ohne Probleme möglich. In der Regel werden hierbei die Businesslogik von Fachtestern und die Navigations- und Low-Level-Ebene von den Technikern erstellt.

Ein weiterer Vorteil ist, dass die fachliche Testspezifikation auf Ebene der Businesslogik sehr frühzeitig beginnen kann, ohne dass das Testobjekt fertig gestellt sein oder als Prototyp vorliegen muss. Gleiches gilt für die Low-Level-Ebene, denn in der Regel ist frühzeitig bekannt, welche Controls an der Testschnittstelle zur Verfügung stehen. Diese Controls können somit unabhängig von den eigentlichen Funktionen des Testobjekts identifiziert und automatisiert werden. Die Erstellung der Navigations-Ebene ist dann der letzte Schritt bei der Verbindung von Businesslogik und Testobjekt.

Test Driven Development (TDD) und somit der Realisierung eines Test-First-Ansatzes auf Ebene von System- oder Annahmetests steht damit nichts mehr im Wege.

Mithilfe dieses Vorgehens ist der Einstieg in eine verteilte oder arbeitsteilige Projektstruktur erheblich einfacher möglich. Auch gelingt die Automatisierung von komplexen Tests in einer womöglich stark fachlich geprägten Organisation eines Testteams.

Der Einsatz einer auf das Management von Schlüsselworten spezialisierten Testmanagement-Lösung ist in einem solchen Umfeld unabdingbar. Die Schlüsselworte werden hinsichtlich Auffindbarkeit, Wartbarkeit und mehrfacher Verwendung erheblich leichter nutzbar. Der Mehrwert der Methode kann damit vollständig erschlossen werden.

7 Fazit

Testautomatisierung ohne vorherige Methoden- und Architekturentscheidung ist riskant und führt in der Regel mindestens zu einer nicht optimalen Nutzung der Möglichkeiten oder sogar zu einem negativen Kosten-Nutzen-Verhältnis.

Vielfach erprobt und bewährt hat sich bei der Testautomatisierung der schlüsselwortbasierte Ansatz. Welche Detail-Methode, high-level oder hierarchisch, dabei gewählt wird hängt von der jeweiligen Projektsituation und der Organisation des Testteams ab. Die verbesserte Wartbarkeit und auch der Grad an Wiederverwendung von Bausteinen der Testautomatisierung lassen die Teams langfristig einen höheren Grad an Testautomatisierung erreichen als mit anderen Methoden. Auch die Trennung von Business-Logik und Technik sorgt für bessere und reibungslose Zusammenarbeit von Fachbereichen und Automatisierungsspezialisten.

Sobald Testteams verteilt und/oder arbeitsteilig organisiert sind oder auch die Tests einen gewissen Komplexitätsgrad erreichen, empfiehlt sich der Einsatz eines Testmanagement-Systems mit einer auf schlüsselwortbasierten Test spezialisierten Komponente. Die Verwaltung und Nutzung der Schlüsselworte werden damit erheblich erleichtert. Bisher nicht besprochene Aspekte wie z.B. die Versionierung der Tests bis auf Ebene der Schlüsselworte wird damit möglich.

Das Testmanagement- und Testdesign-Werkzeug der imbus AG, Test*Bench*, unterstützt die Schlüsselwort-Methode direkt, ohne die sonst notwendigen Workarounds über angehängte Tabellen. Gerade durch die vielen Hilfsmittel, wie z.B. die Anzeige der Verwendung von Schlüsselwörtern, und die vollständig integrierte Versionierung unterstützt dieses Werkzeug ideal bei der Auswahl der richtigen Testarchitektur. Besonders hervorzuheben ist, dass von reiner Prosa-Beschreibung von Tests über Daten-getriebenen zu Schlüsselwort-basierten und Modell-basierten Tests eine sehr einfache Migration möglich ist und die Methoden situationsbedingt auch jederzeit gemischt angewendet werden können. Informieren Sie sich unter www.testbench.info.

8 Quellen

[FAU04] D.R. Faught, Keywords are the Key to Good Automation, siehe: www.tejasconsulting.com/papers/MUG-keywords.pdf

[Ros10] T. Roßner, C. Brandes, H. Götz, M. Winter, Basiswissen Modellbasierter Test, dpunkt.verlag 2010

[TiT11] Trends in Testing 2011: http://www.imbus.de/fileadmin/imbus_repository/Downloads/imbus_Allgemein/Veranstaltungen/tit2011/Flyer_trends_in_testing.pdf

9 Über imbus

imbus ist spezialisierter Lösungsanbieter für die Qualitätssicherung und das Testen von Software.

Unser Angebot umfasst Beratung zur Prozessverbesserung, Softwaretest Services, Testoutsourcing, Testwerkzeuge und Training.

Mit umfassendem Know-how, modernsten Werkzeugen und bewährter Methodik erhöhen wir die Zuverlässigkeit und die Performance von Softwareprodukten, softwareintensiven

Systemen sowie kompletten IT-Strukturen und sichern als herstellerunabhängiger Partner deren korrekte Funktionalität.

Seit 1992 steht das erfahrene und hochqualifizierte Team von imbus für durchgängige, den gesamten Lebenszyklus umfassende Software-Qualitätssicherung aus einer Hand.

Die in 20 Jahren erworbene Expertise aus rund 4.000 erfolgreichen Projekten bildet die solide Grundlage für die tägliche Arbeit unserer Experten, allesamt ISTQB® Certified Tester. Die entsprechenden Referenzprojekte finden Sie hier.

imbus ist mit derzeit über 190 Mitarbeitern an den Standorten Möhrendorf bei Erlangen, München, Köln, Hofheim bei Frankfurt, Norderstedt bei Hamburg und Shanghai/China vertreten.

imbus is a specialized solution provider for the quality assurance and testing of software.

Our portfolio includes consulting for process improvement, software testing services, test outsourcing, test tools, and training.

With our comprehensive know-how, the latest tools, and our proven methodology, we increase the reliability and performance of software products, software-intensive systems, and complete IT structures, and as a manufacturer-independent partner, we assure their correct functionality.

Since 1992, the experienced and highly-qualified imbus team has been synonymous with across-the-board software quality assurance from a single source that covers the entire lifecycle.

The expertise acquired from around 4,000 successful projects over a period of 20 years provides a solid foundation for the daily work of our experts, all of whom are ISTQB® Certified Testers. Here you can find the corresponding reference projects.

imbus is currently represented by more than 190 employees at locations in Möhrendorf near Erlangen, Munich, Cologne, Hofheim near Frankfurt, Norderstedt near Hamburg and Shanghai, China.

imbus AG
Kleinseebacherstr. 9

D-91096 Möhrendorf
Germany
Phone: +49 (0)9131 7518-0
Fax: +49 (0)9131 7518-50
info@imbus.de
www.imbus.de