Ein generativer Ansatz für den automatisierten Software-Test

Anne Göthlich, Karin Eisenblätter, Michael Kroll, Johannes Schad, Heike Vocke

iSAX GmbH & Co. KG
Weinbergstraße 15
01129 Dresden
{anne.goethlich, karin.eisenblaetter, michael.kroll, johannes.schad, heike.vocke}
@isax.com

Abstract: Testen beansprucht 25 - 40% des Arbeitsaufwandes bei einer Software-Entwicklung. Ein großes Potential für die Qualitätssteigerung und Kostenreduzierung bietet ein höherer Formalisierungs- und Automatisierungsgrad beim Testen. Die Optimierung des Testprozesses wird anhand eines Fallbeispiels eines mittelständischen Softwareunternehmens mit Methoden der generativen Software-entwicklung vorgestellt. Der Einsatz einer formalen Beschreibungssprache für Testfälle ermöglicht dabei die Automatisierung eines kompletten Prozessschrittes und reduziert Spielräume bei der Interpretation von Testfällen.

1 Motivation

Das Testen ist eine wichtige Arbeitsstufe in der Software- und Systementwicklung. Bereits mit den aktuellen Anforderungen an die Flexibilität und Sicherheit von IT-Systemen wird die Bedeutung dieses Arbeitsschritts weiterhin zunehmen. Derzeit entfallen durchschnittlich 25-40% des Entwicklungsaufwands auf Testprozesse. Diese laufen unserer Erfahrung nach selten nach einheitlichen Mustern ab, sind immer teuer, zeitaufwendig und werden unter großem Zeitdruck durchgeführt. Damit werden häufig unentdeckte Qualitätsmängel oder Produktivitätsverluste bei Kunden riskiert. Aus Gründen der Qualitätssicherung und Wirtschaftlichkeit ist es daher notwendig, Software- und Systemtests zu optimieren, zu automatisieren und dafür entsprechende Werkzeuge zu entwickeln.

Dafür notwendige Testwerkzeuge sind auch heute bereits in einer Vielzahl am Markt erhältlich. Sie sind jedoch

- meist konzipiert für begrenzte Einsatzfelder,
- nicht in den Entwicklungsprozess integriert oder integrierbar,
- aufgrund hoher Lizenzgebühren für KMU wirtschaftlich nicht sinnvoll.

Die Anforderungen an qualitativ hochwertiges und wirtschaftliches Testen werden jedoch in Zukunft dramatisch steigen, wenn für stärker individualisierte Produkte und Leistungen sowie personalisierte Dienste immer stärker differenzierte und komplexere Softwareprodukte in immer kürzerer Zeit angepasst und damit jedes Mal neu getestet werden müssen. Kunden in der IT-Branche / Softwareentwicklung sowie in der Industrie und im Dienstleistungssektor, die eigene Softwareentwicklung für ihre Produkte integriert haben (z.B. Automatisierung, Anlagenbau, Telekommunikation), müssen verstärkt Testprozesse in der Software- und Systementwicklung optimieren und automatisieren.

So geht es generell um

- die Erhöhung der Effizienz und Qualität des Testens,
- die Beherrschung der Komplexität und Dynamik der Testfälle,
- die Trennung von Testspezifikation und Implementierung der Testskripte,
- effektiveres Testen mit methodischem Vorgehen,
- die Erhöhung des Automatisierungsgrades bei Softwaretests.

Damit können auch bei hohem Zeitdruck in der Entwicklungs- und Testphase steigende Qualitätsanforderungen zuverlässig erfüllt und Wirtschaftlichkeit erreicht werden.

2 Aktuelle Herausforderungen im Bereich Software-Test

Um tiefergreifende Probleme und Herausforderungen in der Praxis zu ermitteln und daraus Verbesserungspotenziale ableiten zu können, wurde bei unseren Kunden eine entsprechende Analyse durchgeführt. Die Befragung konzentrierte sich auf die Umsetzung von Testprojekten in der Praxis, wobei insbesondere Projekte mit automatisierter Testausführung im Fokus standen. Insgesamt wurden 15 Testexperten zu 9 Projekten bei 4 Kunden befragt.

Von Interesse war dabei insbesondere,

- in welchen Testprojekten was, wann und wie (automatisiert) getestet wird,
- welchen Anteil die Automatisierung im Vergleich zum manuellen Test einnimmt,
- welche Probleme dabei auftreten und welche Lösungsansätze hierfür bereits bekannt sind oder umgesetzt werden.

2.1 Ergebnisse der Analyse

Die in der Befragung erhobenen Informationen wurden in Bezug auf allgemein und explizit genannte störende Faktoren und nachteilige Situationen sowie bekannte und angewandte Ansätze/Prinzipien zur Vermeidung und zur Auflösung der beschriebenen nachteiligen Situationen analysiert, im Kontext der beschriebenen Anforderungen verdichtet und Herausforderungen an den (automatisierten) Software-Test abgeleitet.

Zur Übersichtlichkeit sind die Herausforderungen entsprechend des Testprozesses in die Bereiche Testspezifikation, Testausführung und Testmanagement gegliedert.

2.1.1 Testspezifikation

Unzureichend spezifizierte Testbasis. Die Testbasis - z.B. Anforderungen, Anwendungsfälle - ist für die Testspezifikation nicht ausreichend definiert, z.B. kann sie nicht vollständig oder nicht detailliert genug spezifiziert sein sowie nicht eindeutig oder veraltet sein. Hinzu kommt, dass die Testbasis nicht ausreichend formalisiert ist und dass ihre Qualität stark in Abhängigkeit des Bearbeiters variiert. Eine unzureichend spezifizierte Testbasis vergrößert den Interpretationsspielraum zwischen Anforderung und Testfall, welcher sich negativ auf die Testausführung auswirkt.

Geringe Qualität der Testfälle. Fehlende Prozesse und Regeln bei der Testspezifikation führen zu Problemen bei der Qualität der Testfälle. So können zum Beispiel inkonsistente Formulierungen die Verständlichkeit und Lesbarkeit der Testfälle erschweren und ein nicht vorgegebener Detaillierungsgrad führt zu nicht ausreichend oder zu detailliert spezifizierten Testfällen. Dies erhöht den Interpretationsspielraum der Testfälle bei der Testausführung.

Mühsame Anpassung bei Änderungen. Ändern sich im Verlauf des Projekts Anforderungen, die Implementierung oder gar die Architektur der Anwendung, müssen diese Änderungen auch in den Testfallspezifikationen nachgepflegt werden. Dazu sind zunächst die betroffenen Testfälle zu identifizieren und anschließend anzupassen. Die Anpassung unterliegt einer hohen Fehleranfälligkeit, insbesondere, wenn sich Änderungen an vielen unterschiedlichen Stellen auswirken und Wechselwirkungen auftreten. Zudem ist in der Regel nur wenig Zeit verfügbar.

Enge Verflechtung mit Implementierung. Für die automatisierte Testausführung ist die Verknüpfung von Testobjekt und Testskript notwendig, damit das Testskript das zu prüfende Element des Testobjekts adressieren kann. Problematisch ist hier die teils mühsame Erarbeitung der Adressinformationen der zu testenden Elemente. Bei webbasierten Produkten muss beispielsweise die ID eines Elements manuell ermittelt werden, bei pixelbasierten Tests muss die Testumgebung (z.B. Bildschirmgröße) im Vorfeld bekannt bzw. vorhanden sein.

Ein weiterer Kritikpunkt ist die damit einhergehende Vermischung von Entwickler- und Testsicht. Werden seitens der Entwicklung Änderungen hinsichtlich der Adressinformationen vorgenommen, sind Anpassungen am Testskript notwendig – obwohl es sich im eigentlichen Sinn nicht um eine funktionale bzw. nichtfunktionale Änderung handelt.

2.1.2 Testdurchführung

Mangelnde Performance in der automatisierten Testausführung. Eine Herausforderung bei der automatisierten Testausführung ist die Performance. Diese wird unter anderem von der Qualität der Testskripte beeinflusst. Setzt man hier bei-

spielsweise effiziente Programmierparadigmen ein, laufen Testskripte schneller durch. Eine unzureichende Test-Infrastruktur oder eine sequentielle, nicht parallelisierbare Testausführung beeinflussen ebenfalls die Performance der automatisierten Ausführung. Desweiteren werden effektive Mechanismen zum Zurücksetzen der Testumgebung bzw. des Testobjekts in den Zustand vor der Testausführung benötigt. Für die Performance ist die Reihenfolge der Testausführung logisch zusammengehöriger Testfälle entscheidend (z.B. dass zuerst das Anlegen, dann das Bearbeiten und abschließend das Löschen eines Objekts getestet wird).

Hoher Aufwand für die Nutzung testunterstützender Werkzeuge. Automatisierte Tests sind ohne die Nutzung von Testadaptoren zur Steuerung der Anwendung sowie anderen Testwerkzeugen nicht denkbar. Zum Einsatz kommen dabei kundenindividuelle Testwerkzeuge oder Standardtestwerkzeuge. Dabei treffen wir auf eine allgemein bekannte Herausforderung: Kundenindividuelle Testwerkzeuge haben in der Regel einen hohen initialen Erstellungsaufwand, Standardtestwerkzeuge hingegen sind eher unflexibel und bedürfen Anpassungen. Eine weitere Herausforderung ist die fehlende Integration der Testwerkzeuge in den Test- bzw. Softwareentwicklungsprozess, die entweder technischen oder organisatorischen Hürden geschuldet ist.

Geringe Aussagekraft der Testergebnisse. Nicht erfolgreich durchgeführte Tests können verschiedene Ursachen haben - nicht nur tatsächliche Fehler im Testobjekt, sondern z. B. auch fehlerhafte Tests. Aus diesem Grund ist es notwendig, das Testergebnis zu analysieren und zu interpretieren, um die Fehlerursache bzw. Fehlerquelle finden zu können.

2.1.3 Testmanagement

Unzureichender Testfortschritt. Als weitere Herausforderung wird der – im Vergleich zur Entwicklung – unzureichende Testfortschritt benannt. Ursachen liegen in der Regel an nicht in ausreichendem Umfang zur Verfügung stehenden Ressourcen bei der Testausführung im manuellen Test bzw. bei der Testimplementierung im automatisierten Test.

Fehlende oder nicht aussagekräftige Testberichte. Der Test einer Anwendung sollte für alle Beteiligten transparent und aussagekräftig sein. Eine Aussage zum Reifegrad der Anwendung ist insbesondere für das Management wichtig. Eine weitere Herausforderung im Bereich Software-Test ist, diese Aussagen regelmäßig und bezogen auf die Art der Software mit angemessenem Umfang - beispielsweise durch aussagekräftige Testberichte - zu liefern.

2.2 Verbesserungspotenziale

Ausgehend von den in der Befragung erhobenen Informationen wurden Optimierungsbedarfe im (automatisierten) Software-Test identifiziert, um anschließend praktikable Lösungsansätze zu entwickeln.

Bei Testspezifikation und Testausführung, aber auch im Testmanagement konnten diverse Verbesserungspotenziale identifiziert werden:

- Verbesserung der Qualität der Testbasis, beispielsweise von Anforderungen und Anwendungsfällen,
- Verbesserung der Qualität der Testfälle, insbesondere hinsichtlich Vollständigkeit, Korrektheit, Konsistenz und Aktualität,
- einfache Aktualisierung von Testfällen bei Änderungen,
- geringe Abhängigkeit von Testfällen bzw. Testskripten und der Implementierung,
- Verfügbarkeit von Testautomatisierern mit der entsprechenden fachlichen Qualifikation, Realisierung von Testfällen im vorgegebenen Zeit- und Budgetrahmen,
- gute Performance in der automatisierten Testausführung,
- geringer Anpassungsaufwand bei der Nutzung testunterstützender Werkzeuge, d.h. einfache Integration der Werkzeuge in den Entwicklungs- bzw. Testprozess,
- aussagekräftige Testergebnisse, insbesondere hinsichtlich der Ursache für das Fehlschlagen eines Testfalls (z. B. Fehler in der Testspezifikation, Fehler im Testskript, sporadisch fehlschlagende Testfälle),
- Gleichlauf zwischen Entwicklungs- und Testfortschritt,
- aussagekräftige Testberichte als Entscheidungsgrundlage für das Projekt- bzw. Produktmanagement.

Die Herausforderung besteht nun darin, einen Lösungsansatz zu finden, der möglichst eine Vielzahl der genannten Verbesserungspotenziale unterstützt.

2.3 Lösungsidee

Betrachtet man die bei unseren Kunden vorhandenen Herausforderungen genauer (vgl. Kapitel 2.1), so entsteht folgende Frage: Warum führen wir nicht eine formale, aber dennoch für alle Beteiligten verständliche Sprache ein, die wir sowohl im Bereich des manuellen als auch automatisierten Tests für die Testspezifikation nutzen können?

Die Vorteile einer solchen formalen Sprache im Bereich Software-Test sind

- eindeutige, nicht verschieden interpretierbare Testfälle (z.B. bei der Testausführung) aufgrund der festgelegten Detaillierung der formalen Sprache,
- sprachlich konsistente Testfälle durch Vorgabe der Syntax der formalen Sprache,
- zentrale Implementierung und damit reproduzierbare Testausführung für alle verwendeten Elemente der formalen Sprache, insbesondere im automatisierten Test.

Welche Herausforderungen können durch den Einsatz einer solchen formalen Sprache gelöst werden? Im Wesentlichen hat der Einsatz einer Testsprache folgende Vorteile:

Verbesserte Qualität der Testfälle: Eine formale Sprache zeichnet sich durch eine definierte Syntax aus. Diese Syntax trägt wesentlich zur Konsistenz der Testfälle bei, weil beim Einsatz einer formalen Sprache nur die vorhandene Syntax für die Spezifikation der Testfälle erlaubt ist. Gleichzeitig wird durch die Gestaltung der Syntax auch der Detaillierungsgrad der Testfälle vorgegeben und vereinheitlicht. So kann je nach Definition der Syntax der formalen Sprache beispielsweise "click button *Speichern*" ausgedrückt werden, nicht aber das grobe Szenario "save document".

Geringer Anpassungsaufwand: Beim Einsatz einer formalen Sprache – hauptsächlich im automatisierten Test – müssen insbesondere bei Akzeptanztests die Elemente der Benutzeroberfläche angesteuert werden können. Aus diesem Grund werden sie idealerweise an zentraler Stelle gepflegt und können dann von allen Testfällen genutzt werden. Dies erleichtert das Umsetzen nichtfunktionaler Änderungen wie zum Beispiel die Umbenennung eines Buttons in der Benutzeroberfläche.

Darüber hinaus können sich durch den Einsatz einer formalen Sprache in Kombination mit Regeln bzw. organisatorischen Maßnahmen unter anderem weitere Vorteile ergeben:

Universelle Einsetzbarkeit: Die formale Sprache ist nicht erst im Bereich Software-Test einsetzbar, sondern spätestens ab der Feinspezifikation, unter Umständen auch schon in der Anforderungsanalyse. Insbesondere bei der Beschreibung der Anwendungsfälle mit den entsprechenden Vor- und Nachbedingungen kann die formale Sprache benutzt werden. Diese Bedingungen sind später eine wesentliche Quelle für die Testfallspezifikation und können in diesem Fall einfach aus der Feinspezifikation in die Testspezifikation übernommen werden, da sie bereits mit Hilfe der formalen Sprache erstellt wurden.

Flexibilität: Testfälle, die mit einer formalen Sprache erstellt wurden, können sowohl manuell von einem Menschen verstanden und ausgeführt als auch von einer Maschine interpretiert und automatisiert oder teilautomatisiert ausgeführt werden. Werden alle Testfälle mit Hilfe der formalen Sprache spezifiziert, kann je nach Verfügbarkeit von Ressourcen, Zeit und Budget flexibel entschieden werden, welche Tests manuell oder automatisiert ausgeführt werden sollen.

3. Testprozess-Optimierung mit Testfall-DSL am Beispiel

Das mittelständische Unternehmen Gartesis (Name geändert) entwickelt Softwareprodukte im Bereich Produktions-Automatisierung und Systemintegration. Im Zuge einer Neuentwicklung hat Gartesis gemeinsam mit iSAX die Idee einer formalen Beschreibungssprache (Abschnitt 2.3) praktisch angewandt und konnte mit Methoden der generativen Softwareentwicklung seinen ursprünglichen Testprozess optimieren. Gartesis soll in den folgenden Abschnitten als Fallbeispiel dienen, um die Chancen und Risiken dieses Ansatzes zu untersuchen.

Im folgenden Abschnitt wird der ursprüngliche Testprozess von Gartesis dargestellt und es werden die Optimierungspotentiale beschrieben, welche zu einer Verbesserung des Testens genutzt wurden. Im zweiten Abschnitt wird das Konzept für die Verbesserung des Testprozess vorgestellt und es werden ausgewählte technische Grundlagen im Detail beschrieben.

3.1 Ausgangssituation

Gartesis setzt bei der Entwicklung eines neuen Softwareprodukts auf einen strukturierten Testprozess. In diesem Prozess soll geprüft werden, ob alle vorgegebenen Anwendungsfälle vollständig und korrekt implementiert sind.

Das Produkt wird über eine grafische Benutzeroberfläche bedient und alle Programm-Ausgaben erfolgen ebenfalls über diese Oberfläche. Daher können alle Anwendungsfälle als Interaktion des Benutzers mit der Benutzeroberfläche beschrieben und die Tests der Anwendungsfälle vollständig als GUI-Tests implementiert werden. Bei GUI-Tests wird die Benutzeroberfläche des zu testenden Systems mit simulierten Mausklicks und Tastaturanschlägen gesteuert. Anschließend wird geprüft, ob die Ausgaben der Benutzeroberfläche vorher definierten erwarteten Ergebnissen entsprechen.

3.1.1 Testprozess

Der Testprozess von Gartesis umfasste ursprünglich drei Rollen und vier Schritte:

Schritt 1: Der erste Schritt "Risiko Erfassung" wird von *Testdesignern* und dem *Architekten* durchgeführt. Der Architekt ist der Verantwortliche des Entwicklerteams und bildet die Schnittstelle zum Tester-Team. Den Gegenpart auf Seiten der Tester bilden die Testdesigner. Architekt und Testdesigner untersuchen gemeinsam neu implementierte oder veränderte Anwendungsfälle und erarbeiten eine Liste von Tests, die neu spezifiziert oder überarbeitet werden müssen. Dazu beschreibt der Architekt, wie die einzelnen Anwendungsfälle in der Anwendung realisiert wurden und welche Randbedingungen für jeden Anwendungsfall gelten. Ergebnis einer Besprechung zwischen Architekt und Testdesignern ist eine Liste von möglichen Nutzeraktionen und den jeweils erwarteten Reaktionen des Systems. Jede dieser Interaktionen könnte Programmfehler aufdecken. Das heißt, die Liste beschreibt die ungetesteten Risiken der aktuellen Implementierung. Aus diesem Grund wird die Liste als Risikoliste bezeichnet.

Schritt 2: Im zweiten Schritt des Testprozesses spezifizieren die Testdesigner eine Menge von Testfällen. Jeder Testfall ist nach einem festgelegten Muster aufgebaut und enthält eine bestimmte Menge von Informationen wie einen eindeutigen Namen und eine Liste von Testschritten und erwarteten Ergebnissen. Ein Testfall beschreibt mit konkreten Schritten wie ein oder mehrere Elemente der Risikoliste getestet werden sollen. Um eine vollständige Testabdeckung aller Anwendungsfälle zu erzielen, muss jedes Element der Risikoliste von mindestens einem Testfall getestet werden.

Schritt 3: Im dritten Schritt des Testprozesses werden die Testfälle an *Testautomatisierer* übergeben, welche die Aufgabe haben, die Testfälle in ausführbare Testskripte zu transformieren. Dazu übersetzen die Testautomatisierer die Testschritte und erwartete Ergebnisse in geeignete Befehle eines Testwerkzeugs für automatische Softwaretests. Nachdem ein Testautomatisierer ein Testskript erzeugt hat, legt er es in Vorbereitung auf den nächsten Schritt in einem speziellen Verzeichnis ab.

Schritt 4: Der vierte Schritt wird von einem Continuous Integration (CI) System ausgeführt, welches automatisch alle Testskripte erkennt und ausführt. Das Ergebnis der Ausführung ist ein Auswertungs-Dokument. Das Dokument enthält für jedes Testskript einen Eintrag, ob die Ausführung des Scripts erfolgreich war oder nicht. Bei einer erfolgreichen Ausführung entsprechen die Reaktionen des Programms genau den erwarteten Ergebnissen des Testfalls. Tritt ein Absturz auf oder wird ein unerwartetes Ergebnis erzeugt, erstellt das Testsystem eine Fehlerbeschreibung. Anhand der Fehlerbeschreibung kann der Testautomatisierer erkennen, ob es sich um eine Fehlfunktion des Programms handelt oder ob ein Fehler im Testskript vorliegt. Im Fall eines Programmfehlers benachrichtigt der Testautomatisierer das Entwicklerteam. Liegt der Fehler im Testskript kann der Testautomatisierer diesen selbst beheben.

3.1.2 Testfälle am Beispiel

Abbildung 1 zeigt ein typisches Beispiel für einen spezifizierten Testfall. Im oberen Teil stehen Testsuite, Name und Vorbedingungen. Im unteren Teil sind die einzelnen Testschritte und die erwarteten Ergebnisse aufgelistet. Die Testsuite ermöglicht eine Gruppierung von Testfällen in bestimmte Themengebiete wie etwa “Datenmanipulation” im Beispieltestfall. Die einzelnen Testschritte sind auf dem Abstraktions-Niveau individueller Eingaben auf der Benutzeroberfläche angesiedelt. Die erwarteten Ergebnisse einer Eingabe sind beispielsweise die Sichtbarkeit einzelner Elemente der Benutzeroberfläche oder die Existenz einer bestimmten Datei.

Bei der Spezifikation von Testfällen übersetzen Testdesigner informelle Beschreibungen der Risikoliste in konkrete Eingaben und messbare Beobachtungen auf der Benutzeroberfläche. Ein Testfall bietet eine exakte menschenlesbare Schritt-für-Schritt Dokumentation eines Tests. Er ermöglicht reproduzierbares manuelles Testen und dient als verbindliche Spezifikation bei der Implementierung von Testskripten.

Testsuite:	„Data Manipulation“
Name:	9999 – create new file
Vorbedingungen:	user is logged in
Testschritte	**Erwartete Ergebnisse**
1. doubleclick on the icon *“Microsoft Word 2010”*.	1. window *“Document1 – Microsoft Word”* is opened. 2. the main menu including the item *“File”* exists at the top of the window.
1. open menu *“File”* in the main menu and click the item “Save”.	1. dialog *“Save As…”* is opened.
1. select *“Own Documents”* on the left side. 2. enter the text *“testdoc”* into the field *“Document Name”*. 3. click on the button *“OK”*.	1. dialog *“Save As…”* is closed. 2. verify that the file *“testdoc”* exists in the folder *“Own Documents”*.

Abbildung 1: Beispiel eines manuell spezifizierten Testfalls

3.1.3 Optimierungspotentiale

Eine große Schwäche des Testprozesses ist der hohe Aufwand, der bei der Verwaltung von Testfällen und Testskripten entsteht. Testfälle und Testskripte enthalten annähernd die gleiche Information und unterscheiden sich lediglich im Grad der Formalisierung. Obwohl Testfälle in natürlicher Sprache verfasst werden, sind Testschritte und erwartete Ergebnisse durch die Ein- und Ausgabeoperationen des getesteten Systems vorgegeben. Testskripte nutzen die gleichen Operationen in der gleichen Reihenfolge und bilden damit nahezu ein Duplikat der Testfälle.

Der größte Nachteil dieses Ansatzes ist der unnötige Aufwand, der durch die manuelle Duplizierung von Informationen entsteht. Jeder neue oder veränderte Testfall muss manuell auf die zugehörigen Testskripte übertragen werden. Zusätzlich ergibt sich das Problem, dass bei der Übersetzung von Testfällen in Testskripte Fehler auftreten können. Missverständliche Formulierungen oder übersehene Änderungen im Testfall führen zu Testskripten die etwas anderes testen als es der Testdesigner beabsichtigt hat. Diese Fehler werden im besten Fall bei der Testauswertung entdeckt und erhöhen lediglich den Zeitaufwand. Im schlechtesten Fall führen sie zu unbemerkt ungetesteten Funktionen und verstecken Lücken in der Testabdeckung.

3.2 Ein Generativer Ansatz

Der ursprüngliche Testprozess in der Softwareentwicklung von Gartesis zeichnet sich durch die manuelle Duplizierung von Informationen aus. Jeder Test wird von Hand in einem Testfall und einem Testskript beschrieben und jede Änderung muss zuerst am Testfall vorgenommen und anschließend manuell in das zugehörige Testskript übernommen werden. Dieser Prozess ist sowohl aufwändig als auch fehleranfällig, da bei der Überset-

zung von Testfall zu Testskript Missverständnisse auftreten oder Details übersehen werden können. Würde es gelingen, den Testprozess so anzupassen, dass die manuelle Duplizierung vermieden werden kann, könnte eine erhebliche Steigerung der Effizienz und Zuverlässigkeit beim Testen erzielt werden.

Gemeinsam mit Gartesis hat die iSAX ein Verfahren entwickelt, welches mittels domänenspezifischer Sprachen und Code-Generierung das manuelle Erstellen von Testskripten weitgehend überflüssig macht. Das Verfahren wird im Folgenden zuerst konzeptionell vorgestellt und anschließend dessen Umsetzung und Anwendung beschrieben. Der überarbeitete Prozess wird anhand eines Beispiels veranschaulicht und es wird vorgestellt, wie die beteiligten Rollen bei der Durchführung des angepassten Testprozesses unterstützt werden.

3.2.1 Der Testprozess mit automatischer Testskript-Generierung

Um die manuelle Erzeugung von Testskripten zu vermeiden, muss ein Weg gefunden werden, Testskripte automatisch von Testfällen abzuleiten. Dieser Prozess setzt eine automatische und korrekte Interpretation von Testfällen voraus, welche in natürlicher Sprache verfasst sind. Eine exakte automatische Interpretation natürlicher Sprache scheitert jedoch an Mehrdeutigkeiten bzw. Kontextabhängigkeiten, die jeder natürlichen Sprache innewohnen.

Da eine Interpretation natürlicher Sprache nicht möglich ist, setzt eine durch Testfälle getriebene Generierung von Testskripten die Formalisierung der Testfall-Spezifikation voraus. Der resultierende Testfall muss dabei für Menschen und Software gleichermaßen verständlich sein. Da die Sprache von Menschen benutzt werden soll um Testfälle zu spezifizieren, muss das manuelle Schreiben der Sprache effizient und fehlertolerant möglich sein. Die Sprache wird außerdem genutzt, um ausführbare Testskripte zu generieren und darf daher nur Konzepte enthalten, die auf Operationen geeigneter Testwerkzeuge abgebildet werden können.

Gelingt es, die Spezifikation von Testfällen so anzupassen, dass Testskripte automatisch erzeugt werden können, kann der Prozessschritt der Testskript-Implementierung entfallen zusammen mit dem Arbeitsaufwand und der Fehleranfälligkeit, die mit diesem Schritt einhergeht. Außerdem besteht die Möglichkeit, die Qualität von Testfällen durch die Vereinheitlichung der Sprache und das Verhindern von unspezifischen Aussagen deutlich anzuheben.

3.2.2 Die Testfall-DSL

Gemeinsam mit iSAX ist es Gartesis gelungen, eine domänenspezifische Sprache (DSL) zu entwickeln, die unter anderem die Dokumentationsfunktion klassischer Testfälle erfüllt und die sich automatisch in ausführbare Testskripte übersetzen lässt. Eine domänenspezifische Sprache ist eine formale, automatisch interpretierbare Sprache deren Vokabular und Grammatik auf einen bestimmten Anwendungsfall optimiert sind. Die Test-

fall-DSL von Gartesis und iSAX ermöglicht es, einfach lesbare Testfälle mit automatischer Transformation in Testskripte zu verbinden.

Formalisierte Testfälle haben eine ähnliche Struktur und enthalten die gleichen Elemente wie klassische Testfälle. Testschritte und Erwartungen werden mit einer vorgegebenen Menge von Satzschablonen formuliert, wobei es sich um Sätze handelt, die an bestimmten Positionen durch das Einfügen von Worten oder Zahlen angepasst werden können. Sobald alle variablen Positionen ausgefüllt sind, ist eine Satzschablone ein normaler Satz in einer natürlichen Sprache. Da die bekannte Struktur von Testfällen zum großen Teil erhalten wird sowie Testschritte und Bedingungen in normalen Sätzen beschrieben werden, sind formale Testfälle ähnlich leicht zu verstehen wie klassische Testfälle. Außerdem bleibt der Einarbeitungsaufwand für Testdesigner gering, da lediglich die Formulierung von Testschritten und Erwartungen auf eine bestimmte Menge vorgegebener Satzschablonen eingeschränkt wird.

```
testsuite DataManipulation

testcase CreateNewFile
description "This testcase verifies that the creation of a new file is possible."

precondition "user is logged in"
   start system machine1
   element LoginScreen is shown

   enter text "testname" into field username
   enter text "testpass" into field password
   click button OK
   element HomeScreen appears
   icon MicrosoftWord2010 appears
end

doubleclick icon MicrosoftWord2010
window Document1 is opened
menu MainMenu appears

choose item Save in menu MainMenu
dialog SaveAs is opened

select OwnDocuments
enter text "testdoc" into field DocumentName
click button OK
dialog SaveAs is closed
```

Abbildung 2: Formaler Testfall

Abbildung 2 zeigt einen formalen Testfall, der inhaltlich dem klassischen Testfall aus Abbildung 1 entspricht. In den ersten Zeilen wird der Testfall einer Testsuite zugeordnet und erhält einen Namen und Beschreibung. Anschließend wird eine Vorbedingung eingeführt und im unteren Bereich folgen Testschritte und Erwartungen.

Die Vorbedingung nimmt deutlich mehr Raum ein als im klassischen Testfall und reicht von **precondition** bis zu **end** neun Zeilen darunter. In den eingerückten Zeilen zwischen den beiden Schlüsselworten stehen Testschritte und erwartete Ergebnisse, die den Zielzustand der Vorbedingung in der getesteten Anwendung herstellen und überprüfen, bevor die darunter stehenden Testfälle ausgeführt werden. Der Satz "user is logged in" bezeichnet die Summe der erwarteten Ergebnisse der Vorbedingungen und wird vom Testdesigner angegeben, um Lesern des Testfalls eine Dokumentation der Vorbedingung auf fachlicher Ebene zu bieten. Die aufwändigere Beschreibung der Vorbedingungen im Vergleich zu klassischen Testfällen ist notwendig, da Vorbedingungen beim klassischen Ansatz auf einer fachlich-deklarativen Ebene angegeben werden. Um automatisches Testen zu ermöglichen, müssen Vorbedingungen automatisch hergestellt und geprüft werden können. Daher ist es nötig, konkrete Testschritte und erwartete Ergebnisse anzugeben.

Die Einbettung der Schritte zur Herstellung des Ausgangszustands in ein eigenständiges Vorbedingungselement ermöglicht das Erzeugen hilfreicher Fehlermeldungen, sollte die Vorbedingung nicht wie erwartet hergestellt werden können. Die Fehlermeldung kann hervorheben, dass es sich nicht um einen Fehler in den eigentlichen Testschritten handelt und so die Fehlersuche erleichtern.

Unterhalb der Vorbedingung folgen Testschritte und erwartete Ergebnisse. Unveränderliche Teile der Satzschablonen sind lila oder grün gefärbt, um sie einfach von variablen Teilen unterscheiden zu können. Der Testfall zeigt einige Beispiele für Satzschablonen wie z.B. "**enter text** #TEXT **into field** #FIELD", "**click button** #BUTTON" oder "**dialog** #DIALOG **is opened**". Testschritte und Bedingungen sind nicht mehr beliebige Texte, sondern Elemente einer vordefinierten Menge von Zeichenfolgen, welche von einem Computer automatisch erkannt und interpretiert werden können. Das Übersetzerprogramm, welches Testfälle in Testskripte umwandelt, ist in der Lage, jeden Testschritt und jede Bedingung einer eindeutigen Sequenz von konkreten Instruktionen zuzuordnen, die von einem Computer ausgeführt werden können.

Um einen formalen Testfall in ein ausführbares Testskript zu übersetzen, übergibt der Testdesigner den Testfall an das Übersetzerprogramm. Dieses liest den Testfall und ersetzt jeden Testschritt und jedes erwartete Ergebnis durch einen oder mehrere vom Computer ausführbare Befehle. Das Übersetzerprogramm ist so gestaltet, dass die resultierenden Testskripte die gleichen Testwerkzeuge nutzen, wie die manuell erstellten Testskripte des ursprünglichen Testprozesses. Dies minimiert die Zahl neu einzuführender Technologien und erleichtert die parallele Nutzung manuell und automatisch erzeugter Testskripte.

3.2.3 Anwendung des generativen Ansatzes

Die Übersetzung von formalen Testfällen in Testskripte zeichnet sich durch eine sehr geringe Fehlertoleranz aus. Selbst geringfügige Abweichungen von den Regeln der Testfall-DSL führen zu Testfällen, die nicht in Testskripte übersetzt werden können. Für Testdesigner kann diese Eigenschaft leicht zu einem unüberwindbaren Hindernis werden, wenn ein scheinbar korrekter Testfall zu unverständlichen Fehlermeldungen führt.

Um die Konsequenzen dieser Fragilität für den Testdesigner abzuschwächen, wurde gemeinsam mit der Testfall-DSL ein spezieller Editor entwickelt, der den Testdesigner mit verschiedenen Hilfsfunktionen unterstützt. Die Wichtigste dieser Funktionen zeigt dem Testdesigner an jeder Stelle eines Testfalls an, welche Worte an dieser Stelle eingegeben werden können. In einem leeren Testfall wird von dieser Hilfsfunktion lediglich **testsuite** angezeigt, da jeder Testfall mit der Spezifikation der Testsuite beginnt. Möchte der Testdesigner einen neuen Testschritt hinzufügen, muss er lediglich seinen Cursor auf eine leere Zeile bewegen und die Hilfsfunktion zeigt eine Liste von möglichen Worten wie **doubleclick, enter** oder **dialog**. Auf diese Weise muss der Testdesigner die Sprache nicht exakt auswendig lernen, um sie benutzen zu können, sondern kann den gewünschten Schritt aus einer Liste von Möglichkeiten auswählen.

Eine weitere Hilfsfunktion ist das Anzeigen von Fehlern direkt im Editor. Ähnlich wie in gängigen Textverarbeitungsprogrammen werden unzulässige Zeichenfolgen im Editor rot unterstrichen. Auf diese Weise kann der Testdesigner seinen Fehler schnell korrigieren und sieht auf den ersten Blick, wo sich die fehlerhafte Stelle im Testfall befindet.

3.2.4 Ergebnisse der Testprozess-Optimierung

Mit der Testfall-DSL gelingt es Gartesis einen aufwändigen Schritt des Testprozesses einzusparen. Formalisierte Testfälle sind dank Satzschablonen und einer gewohnten Struktur sehr gut lesbar und erhalten daher die wichtige dokumentierende Qualität klassischer Testfälle.

Gleichzeitig können die neuen Testfälle automatisch in ausführbare Testskripte übersetzt werden und machen so das teure und fehleranfällige manuelle Erstellen von Testskripten überflüssig.

Mitgelieferte Hilfsfunktionen des Testfall-Editors erleichtern dem Testdesigner die Arbeit mit den neuen Testfällen und verhindern frustrierendes Fehlersuchen und aufwändiges Auswendiglernen von Satzschablonen der Testfall-DSL.

4. Bewertung des generativen Ansatzes

Die Einführung der Testfall-DSL hat zu einer tiefgreifenden Änderung des Testprozesses in der Softwareentwicklung von Gartesis geführt. Im neuen Prozess verfassen Testdesigner Testfälle nach eindeutigen formalen Regeln. Die Erzeugung von Testskripten ist automatisiert worden und die Rolle des Testautomatisierers ist im neuen Prozess nicht

mehr vorgesehen. Außerdem wurde mit der Testfall-DSL, dem Testfall-DSL-Editor und dem Übersetzer-Programm ein neues komplexes Werkzeug eingeführt, dessen Wartung und Erweiterung einen neuen Aspekt des Testprozesses darstellt.

Der wichtigste Vorteil automatisch generierter Testskripte liegt im reduzierten Aufwand, der durch Erstellung und Synchronisierung mehrerer Kopien gleichartiger Information entsteht. Neue Testfälle werden einmal geschrieben und bieten Dokumentation und ausführbares Testskript in einem einzigen Artefakt. Dieser Vorteil hat das Potential, die Gesamtkosten von Tests zu verringern bzw. den Testumfang bei einem vorgegebenen Kostenbudget zu erhöhen.

Mit der Einführung der Testfall-DSL wurde die Zahl der natürlichsprachigen Artefakte im Testprozess reduziert. Das ist ein Vorteil, da natürlichsprachige Beschreibungen grundsätzlich einen Interpretationsspielraum eröffnen, der bei vielen Schritten des Software-Entwicklungsprozesses zu Fehlern führen kann. Im Testprozess kann zum Beispiel der Testautomatisierer die Absicht des Testdesigners missverstehen und einen falschen Test erzeugen. Bei der Spezifikation von Anforderungen ist Präzision ein hohes Gut und jede Veränderung, die mit einer erhöhten Präzision bei vergleichbaren Aufwänden einhergeht, kann den Software-Entwicklungsprozess zu verbessern.

Weiterhin kann die Zahl manueller Schritte des Testprozesses verringert werden. Die manuelle Synchronisierung von Testskripten mit Testfällen setzt die wiederholte, zuverlässige Ausführung immer gleicher Arbeitsaufgaben voraus, bei denen jedes Detail entscheidend ist. Diese Art von Arbeit ist keine Stärke von Menschen und es schleichen sich sehr leicht Fehler ein, welche die Qualität und Aussagekraft von Tests empfindlich schwächen können. Computerprogramme eignen sich jedoch hervorragend für diese Art von Aufgabe. Daher ist eine Automatisierung im Sinne der Qualitätssicherung von großem Wert.

Die Testfall-DSL hat jedoch nicht nur Vorteile. Während ein großer Teil der Arbeit des Testautomatisierers entfällt, bleiben einige Aspekte, die auch weiterhin manuell ausgeführt werden müssen. Diese Aspekte werden im neuen Testprozess auf den Testdesigner übertragen. Indem der Testdesigner Einträge der Risikoliste in formale Testfälle verwandelt, übernimmt er die Aufgabe, eine natürlichsprachige informelle Beschreibung in Ausdrücke einer formalen Sprache zu transferieren. Dieser Schritt ist nicht automatisierbar, wurde vorher vom Testautomatisierer durchgeführt und setzt die Bereitschaft des Testdesigners voraus, den Umgang mit der neuen Technologie zu erlernen.

Weiterhin übernahm der Testautomatisierer implizit die Aufgabe, die Testfälle der Testdesigner zumindest oberflächlich auf deren Sinnfälligkeit zu prüfen. Offensichtliche Fehler oder fragwürdige Angaben konnten im Gespräch abgeklärt werden. Auf diese Weise wurde die Qualität des Testprozesses erhöht. Diese Prüfung fällt weg und der Testdesigner muss selbst versuchen, seine Testfälle genauer auf Fehler zu untersuchen. Allerdings sollte beachtet werden, dass es immer ein letztes Glied in der Kette gibt und es im ursprünglichen Testprozess die Testskripte waren, die keiner weiteren Prüfung mehr unterzogen wurden. Formalisierte Testfälle haben im Vergleich zu Testskripten so-

gar den Vorteil, dass sie nicht nur von ausgebildeten Entwicklern interpretiert werden können.

Neben den Auswirkungen auf den Testdesigner ist die Einführung der Testfall-DSL an sich eine zusätzliche Herausforderung im Testprozess. Einerseits entsteht durch die DSL zusätzliche Arbeit, wenn zu Beginn eines Projekts entweder eine neue DSL definiert oder eine bestehende DSL integriert werden muss und wenn im Laufe des Projektes Sprachkonstrukte wie die Menge der verfügbaren Satzschablonen überarbeitet und erweitert werden müssen. Außerdem ist die Testfall-DSL ein komplexes technisches System, welches selbst Fehler aufweisen kann.

5. Ausblick

Das Generieren von Testskripten aus einer Testfall-DSL bietet einige Optimierungsmöglichkeiten, die in Zukunft zu einer weiteren Verbesserung der Test- und Codequalität führen wird. Häufig können fachliche Operationen auf unterschiedliche Weise ausgeführt werden. Zum Beispiel kann ein Dokument sowohl durch Klicken auf den Menüeintrag *Datei Speichern* als auch durch betätigen der Tastenkombination *Strg + s* abgespeichert werden. Obwohl das Ergebnis das Gleiche ist, wird im Programm unterschiedlicher Code ausgeführt und es kann der Fall eintreten, dass ein Fehler nur bei einer der Möglichkeiten auftritt. Eine DSL gesteuerte Generierung von Testskripten kann systematisch für jede dieser Varianten einen separaten Test generieren. Dieser Vorteil zeigt eine Richtung für zukünftige Weiterentwicklungen an, bei der zunehmend früher im Software-Entwicklungsprozess auf formale Beschreibungen und Codegenerierung gesetzt wird, um mehr und mehr Eigenschaften des Softwareprodukts systematisch und automatisiert prüfen zu können.

Ein erster Schritt in diese Richtung könnte eine formale deskriptive Beschreibung von Vorbedingungen sein. In der aktuellen Fassung der DSL werden Vorbedingungen mit einem beliebigen Satz dokumentiert, müssen jedoch mit Testschritten und Bedingungen Schritt für Schritt hergestellt werden. Mit der Zeit werden sich typische Vorbedingungen wie *"Der Benutzer ist eingeloggt"* finden, die von Testdesignern immer wieder neu implementiert werden. An diesem Punkt sollten diese Bedingungen direkt auf Ebene der Testfall-DSL angeboten werden. Der Testdesigner kann aus einer wachsenden Menge von Standardvorbedingungen wählen und muss nur noch in ungewöhnlichen Fällen Vorbedingungen manuell herbeiführen.

Mit der zunehmenden Formalisierung des Testprozesses werden unterstützende Werkzeuge unerlässlich. Ohne die Unterstützung des Testfall-DSL-Editors, welcher mögliche Formulierungen aufzeigt und Fehler hervorhebt, wäre die Aufgabe, formale Testfälle zu erzeugen für den Testdesigner um einiges schwerer zu bewältigen. Diese Unterstützung kann in weiteren Schritten ausgebaut und durch neue hilfreiche Funktionen ergänzt werden. In einem ersten Schritt könnte die inhaltliche Unterstützung durch den Editor ausgebaut werden, um nicht nur das jeweils nächste Wort, sondern vollständige Satzschablonen einfügen zu können. Verbunden mit einer leistungsstarken Suchfunktion und auto-

matisch eingeblendeter Dokumentation der jeweiligen Testschritte und Bedingungen kann der Testfall-DSL-Editor zu einem leistungsfähigen Werkzeug ausgebaut werden.

Aufgrund der Flexibilität des eingesetzten Codegenerators, der leicht an vorhandene Standardtestwerkzeuge angepasst werden kann, ist der Lösungsansatz auch auf viele weitere Anwendungsszenarien erweiterbar.

Literaturverzeichnis

[Ru09] Rupp, C.: Requirements-Engineering und -Management: Professionelle, iterative Anforderungsanalyse für die Praxis. Hanser 2009.

[SBB12] Seidl, R.; Baumgartner, M.; Bucsics, T.: Basiswissen Testautomatisierung: Konzepte, Methoden und Techniken. Dpunkt.verlag 2012.

[SBS11] Sneed M. H.; Baumgartner, M.; Seidl, R.: Der Systemtest: Von den Anforderungen zum Qualitätsnachweis. dpunkt.verlag 2011.

[SL12] Spillner, A.; Linz, T.: Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB-Standard. dpunkt.verlag 2012.

[St12] Starke, G.: Effektive Softwarearchitekturen: Ein praktischer Leitfaden. Hanser 2012.