WHITEPAPER

Bitech.AG

Kunden-Information 07/22

Einstieg in die Testautomatisierung

Business Technologies

Dieses Schriftstück richtet sich an Abteilungs- und Projektleiter, die für die Entwicklung und Pflege bestehender Anwendungen verantwortlich sind und bis jetzt über keine Erfahrungen mit der Testautomatisierung verfügen

- Warum Testautomatisierung?
- Funktionale Eigenschaften und Merkmale
- Technische Eigenschaften und Merkmale
- Unsere Leistungen

Bitech Fachexperte

Thomas Barleben

Teamleiter SAP Business Technologies

Mobil +49 21713 / 72 30-0

thomas.barleben@bitech.ag

SAP Silver Partner

Warum Testautomatisierung?

Es gibt unterschiedliche Gründe, die zu einer Entscheidung führen, in die Testautomatisierung einzusteigen. In der Regel kommt man auf diesen Gedanken, wenn fehlerhafte Software ausgeliefert wurde und dadurch womöglich wirtschaftlicher Schaden entstanden ist. Die Reaktionen darauf sind in der Regel ähnlich; man stellt immer wieder die Fragen: Wurde die Anwendung richtig getestet? Warum hat man bei den Tests diesen Fehler nicht entdeckt? Was kann man machen, um die Qualität der Auslieferungen zu verbessern?

Die Antworten bzw. die Reaktionen sind ebenfalls ähnlich. Man erhöht die Anzahl der manuellen Tests und erstellt Checklisten und Protokolle. Doch irgendwann stößt man an seine Grenzen und stellt fest, dass es kaum möglich ist, die gesamte Anwendung zu testen. Man beschränkt sich im Test darum häufig hauptsächlich auf die Funktionen, die zuletzt geändert wurden, da hier häufig die Fehlerquelle vermutet wird.

Bei den automatisierten Tests unterliegt man hier keiner Einschränkung. Die automatischen Tests können vollständig und nach jeder Änderung ausgeführt werden. Für neue Funktionen werden neue Tests entwickelt, sodass die Testabdeckung und die Qualität der Anwendung konstant bleiben.

⊕ Pro: Höhere Testabdeckung bei komplexen Anwendungen möglich

Bei manuellen Tests können dagegen die Fehler ausgeliefert werden, die sich in Funktionen befinden, die durch die aktuellen Änderungen eigentlich nicht betroffen sein sollten (Stichwort: Wechselwirkung von letzten Änderungen). Richtet dieser Fehler wirtschaftlichen Schaden an, beginnt die o.g. Diskussion von Neuem. Man ist in einer Sackgasse angekommen. Die Akzeptanz der Software stagniert oder wird zum Abwärtstrend.

⊕ Pro: Auslieferung von fehlerhafter Software verhindern

Die ausgelieferten Fehler sind aber nicht der einzige Grund, warum in eine Testautomatisierung investiert werden sollte. Wenn man mit der Entwicklung einer Anwendung beginnt, ist die Welt noch in Ordnung. Die Anwendung ist überschaubar und nicht allzu komplex. Man liefert die erste Version aus und entwickelt weiter. Die Anwendung ist jetzt schon etwas komplexer und man muss schon mehr aufpassen. Die Komplexität der Anwendung und des Quellcodes nehmen zu. Man liefert die nächste Version der Anwendung.

Nicht jede Auslieferung verläuft ohne Probleme. Das eine oder andere Mal werden Fehler mitgeliefert. Nach ein paar Zyklen hat man sich den ein oder anderen blauen Fleck geholt und es werden Funktionen identifiziert, die besonders risikobehaftet sind. Ab hier werden die Änderungen an diesen Stellen vermieden oder gar nicht mehr gemacht. Die Anwendung ist in ihrem Leistungsumfang dann oftmals im weiteren Produktlebenszyklus eingeschränkt. Früher oder später wird entschieden, die Anwendung neu zu schreiben, damit man auch die Anforderungen abdecken kann, die mit der aktuellen Version nicht mehr umsetzbar sind. Mit der neuen Anwendung wird man aber die gleichen Fehler, nur auf eine andere Art, wiederholen. Auch hier ist man in einer Sackgasse angekommen.

⊕ Pro: Komplexere Funktionen in den Griff bekommen und die Kontrolle darüber behalten

Technologiewechsel beenden oft die Laufzeiten von Anwendungen. Dabei spielt es keine Rolle, ob die Anwender mit der Software zufrieden waren oder nicht. Wenn das Ende einer Anwendung, die über die Testautomatisierung verfügt, kommt, hat man mittels der definierten Testfälle eine aktuelle Beschreibung aller Funktionen der Anwendung dokumentiert. Damit kann ein Ersatz mit neuer Technologie und „Test Driven Development“ mit weniger Planung entwickelt werden.

⊕ Pro: Mit Testfällen liegt eine immer aktuelle Beschreibung der Funktionen der Anwendung vor

Das klingt alles gut; warum macht man das nicht von Beginn an? Die Testautomatisierung ist ein Invest, der erbracht bzw. bereits von Anfang an in einer Kalkulation der Entwicklungskosten integriert werden sollte. Man befindet sich in einer Konkurrenzsituation und muss für ein Projekt deutlich höhere Kosten angeben.

⊖ Kontra: Höhere initiale Kosten

Ist die Anwendung erst einmal entwickelt, besteht im weiteren Verlauf oftmals der Bedarf die Anwendung in ihrer Funktionalität durch Änderungen und Erweiterungen anzupassen. Diese Weiterentwicklungen machen Änderungen am bestehenden und mit den Tests ggf. abgedeckten Codes notwendig. Durch diese Änderungen muss man jetzt nicht nur den Code der Anwendung ändern/pflegen, sondern auch den Code der Tests. Das erhöht den Aufwand für die Weiterentwicklungen.

⊖ Kontra: Höhere Kosten in der Weiterentwicklung

Zwischenfazit

Die Testautomatisierung hat sowohl Vor- wie auch Nachteile. Sollte man also auf die Testautomatisierung setzen oder doch lieber darauf verzichten? Diese Frage würde ich nicht pauschal beantworten, sondern von der Nutzungsdauer, nicht aber von der Größe der Anwendung abhängig machen.

Funktionale Eigenschaften und Merkmale

Die Entwicklung einer Anwendung ist mit hohen Kosten verbunden. Bei einer Anwendung, die ich z.B. nur für die nächsten zwei Jahre benötige, rechnet sich der Aufwand und der damit verbundene Invest in eine Testautomation u.U. nicht.

Anders sieht es mit Anwendungen aus, deren Produktlebenszyklus nicht im Vorhinein festgelegt ist oder bei denen nicht mit Sicherheit abschätzbar ist, wie viele Erweiterungen dort noch notwendig sein werden. Hier stellt die Testautomatisierung einen Investitionsschutz dar.

Der Umfang und der mögliche Ersteindruck bzgl. der Komplexität einer Anwendung müssen dabei nicht unbedingt ausschlaggebend sein. Deckt eine eher weniger umfangreiche Anwendung einen bestimmten Geschäftsprozess ab und funktioniert bspw. zwei Jahre lang ohne Änderungen fehlerfrei, sollte diese von vornherein von der Testautomatisierung abgedeckt werden. So ist auch nach einer längeren Zeit sichergestellt, dass durch die Testautomatisierung und der damit verbundenen Dokumentation alle Abhängigkeiten und Besonderheiten betrachtet werden. Dies minimiert – auch nach Wechsel von Verantwortlichkeiten – das Risiko zur Fehleranfälligkeit deutlich.

Es gibt viele unterschiedliche Arten von automatisierten Tests. Deshalb ist es wichtig, bei dem Einstieg in die Testautomatisierung die Ziele, die man erreichen möchte, festzulegen. Die Testabdeckung sollte sich linear zu der Komplexität der Anwendung entwickeln. Die Komplexität der Anwendung lässt sich aber nicht wirklich messen. Ziel sollte also sein, mit der Auslieferung einer neuen Funktionalität auch mindestens einen neuen Test auszuliefern. Bei Änderung einer Funktionalität müssen auch die dazugehörigen Tests erweitert und angepasst werden. Eigentlich ist es nicht sinnvoll, eine Anwendung manuell zu testen, denn diesen Test müsste bei jeder weiteren Auslieferung wiederholt werden. Manuelle Tests sind reine Zeit- und somit Geldverschwendung.

Je mehr Quellcodes wir haben, umso mehr Aufwand haben wir mit der Weiterentwicklung der Anwendung. Das bezieht sich auch auf den Quellcode der Tests, sodass eine unendliche Steigerung der Anzahl der Tests uns immer stärker ausbremst. Deswegen sollte man die Übersicht und die Kontrolle über die Tests behalten. Besser wäre es, die Tests von vornherein nach der Kritikalität für den Geschäftsprozess zu bewerten. Es kann sinnvoll sein, ein Test zu definieren, wegen geringer Kritikalität aber auf die Implementierung zu verzichten. Die Tests sollten auch gut strukturiert abgelegt werden. Wenn später die Frage kommt, wie eine Funktion abgebildet ist, sollte sich diese Frage anhand der definierten Tests selbst beantworten.

Im SAP Umfeld wird die Flexibilität der Anwendungen hochgeschätzt. Die Anwendungen bieten eine Vielzahl von Konfigurationsmöglichkeiten. Zusätzlich potenzieren diese Möglichkeiten den Testaufwand. Nehmen wir mal an, ich habe mit meinem Report drei Funktionen abgedeckt. Zu jeder Funktion habe ich einen Test geschrieben. Nun füge ich eine Konfigurationstabelle ein, in die ich einen Schalter mit drei unterschiedlichen Werten einpflegen kann. Um die Anwendung vollständig zu testen, müsste ich für jeden Konfigurationswert die drei implementierten Tests wiederholen. Die Anzahl der Test steigt damit auf 3 x 3 = 9 Tests.

Spielen Sie dieses Szenario mit weiteren Funktionen und Konfigurationsmöglichkeiten durch, werden Sie schnell feststellen, wohin das führt. Es werden sehr viele identische Tests benötigt, die sich lediglich in den Umgebungsparametern unterscheiden. Daraus folgt, dass wir die Wiederverwendbarkeit des Test-Quellcodes maximieren müssen, um unseren Implementierungsaufwand für die Tests zu reduzieren.

Mit einer steigenden Anzahl von Tests wächst auch die Notwendigkeit, dass die Tests zuverlässig funktionieren. Deren Ergebnisse sollten stets reproduzierbar und korrekt sein. Hier sollten bestimmte Gegebenheiten von vornherein konzeptionell abgefangen werden.

Als Beispiel können hier gesetzte Sperren auf (Datenbank-)Objekten herangezogen werden. Je nachdem, wie man die Tests implementiert, kann ein Test wegen einer gesetzten Sperre fehlschlagen. Das ist nicht unbedingt ein reproduzierbares Verhalten. Entweder sollten die Tests eigenständig keine Sperren setzen oder es muss sichergestellt werden, dass ein Test nur Sperren auf solche Objekte setzt, die von niemand anderem zu diesem Zeitpunkt gesperrt werden können.

Für die Reproduzierbarkeit ist zusätzlich wichtig, dass ein Test keine bestehenden Daten im System ändert. Auch wenn versucht wird, im gleichen Test den ursprünglichen Zustand wiederherzustellen, wird das bei 300 und mehr Tests nicht mehr funktionieren. Irgendwo schleicht sich ein Fehler ein und man ist letztendlich nicht mehr in der Lage, diejenige Testausführung zu identifizieren, die den Datenschiefstand verursacht hat.

Ziel: Tests dürfen bestehende Daten im System nicht verändern

Wurden viele Tests implementiert und der Testlauf zeigt 20 Fehler an, müssen die Ergebnisse des Testlaufs eingeordnet werden. Handelt es sich um eine oder womöglich mehrere Ursachen? Dazu ist eine ausführliche Fehlerausgabe notwendig. Ein Text wie: „Es ist ein Fehler aufgetreten“, ist hier weniger sinnvoll, da die Testframeworks über einen entsprechenden Indikator verfügen, sodass dieser Text kein Mehrwert hat. Die Frage, die sich bei der Auswertung der Ergebnisse stellt, ist: „Was ist schiefgelaufen?“. Aus diesem Grund sollte eher so etwas wie: „Geschäftspartner XY ist ungültig“ oder auch technische Hinweise zur weiteren Fehleranalyse (bspw. der Inhalt der BAPIRET-Tabelle) angegeben werden.

Kommt man mit einer reinen Textausgabe nicht weiter, muss man sich die Implementierung des Tests anschauen. Die Implementierung eines Tests kann auch mal komplexer ausfallen, sodass eine Beschreibung des Tests notwendig ist. Wenn der Fehler auftritt, kann man den Ist-Zustand im System analysieren. Wenn aber keine Beschreibung des Tests oder des Soll-Zustandes existiert, kann man den Fehler unter Umständen nicht benennen.

Beispiel: Ich implementiere einen Negativ-Test für die Anlage eines Interessenten

Solange eine Ausnahme ausgelöst wird, ist die Anwendung fehlerfrei und mein Test läuft erfolgreich durch.

Irgendwann wird mein Test im Testlauf auffällig. Nach der ersten Analyse stelle ich fest, dass der Interessent ohne den Fehler, den ich erwartet habe, angelegt worden ist. Nun stellt sich die Frage: Warum habe ich vor einem halben Jahr erwartet, dass er nicht angelegt würde? Welche Ausnahme habe ich hier erwartet? Der Test ist damit nicht mehr zu gebrauchen.

Zwischenfazit

Zusammenfassend kann man drei Ziele für die Testautomatisierung definieren

1. Hohe Testabdeckung
a. Nur Funktionen ausliefern, die durch automatische Tests abgedeckt sind.

2. Hohe Qualität der einzelnen Tests
a. Testergebnisse sollten reproduzierbar und korrekt sein.
b. Zu jedem Test gehört eine Beschreibung, was getestet wird.
c. Texte in den Testergebnissen müssen aussagekräftig und vollständig sein.

3. Einschränkung der Kosten
a. Maximale Wiederverwendbarkeit des Test-Quellcodes
b. Anzahl der vorhandenen Tests ggf. einschränken

Technische Eigenschaften und Merkmale

Arten der Entwicklung

Test Driven Development (TDD)

Die testgetriebene Entwicklung ist ein sehr gutes Mittel, um sicher zu stellen, dass die Testabdeckung mit der Komplexität der Anwendung wächst. Der Nachteil der testgetriebenen Entwicklung im agilen Umfeld ist, dass auch für die Zwischenergebnisse Tests geschrieben werden. Diese Tests werden bis zur endgültigen Fertigstellung noch mehrfach geändert und sind damit ein großer Kostentreiber. Die Tests werden von Entwicklern definiert und sind aus fachlicher Sicht ggf. sinnfrei.

Abdeckung: ⊕
Kosten: ⊖

Behavior Driven Development (BDD)

Eine verhaltensgetriebene Entwicklung ist gut geeignet, um die Anwendung aus Sicht des Endanwenders zu testen. Dieses Verfahren gehört zu den sogenannten Black-Box-Ansätzen. Dabei wird nicht geprüft, ob ein Testfall im Ergebnis den richtigen Wert annimmt, sondern ob die Benutzer-Ausgabe zu einer Reaktion eines Benutzers korrekt ist. Die Fehler, die nach der Auslieferung einer neuen Version der Software gemeldet werden, fallen genau in diese Kategorie. Gleichzeitig kann man intern Änderungen durchführen, ohne dass diese sich auf irgendwelche Tests auswirken. Insofern wirkt sich dieser Ansatz doppelt positiv auf die Kosten aus. Ein bekannter Vertreter dieses Ansatzes ist „Gherkin“. Nach Gherkin sieht die Definition eines Tests wie folgt aus:

 # Anmeldung
Szenario:

Benutzer meldet sich an. Gegeben sei der Benutzer „John“. Wenn der Benutzer sich anmeldet, wird ihm die Startseite angezeigt.

Die Definition der Tests wird in einer Textdatei abgelegt. Die Implementierung erfolgt nicht auf der Ebene der Tests, sondern auf der Ebene der Schritte. Dadurch ergibt sich eine sehr hohe Wiederverwendbarkeit des Quellcodes. Nach dieser Methodik ist es nicht möglich, einen Test zu implementieren, den man vorher nicht beschrieben hat.

Mit Hilfe von Tags können die einzelnen Tests frei kategorisiert werden, sodass bestimmte Tests anhand von bestimmten Tags von der Ausführung ausgeschlossen werden. Die Ablage der Tests in eigenen Dateien (oder Includes usw.) erleichtert die Verwaltung der existierenden Tests.

Qualität: ⊕
Kosten: ⊕⊕⊕

Zwischenfazit

Die vorgestellten Arten der Anwendungsentwicklung haben ihre Stärken und sie sind kombinierbar.

Abdeckung Qualität Kosten

TDD ⊕⊖
BDD ⊕⊕⊕⊕

Wenn wir einen Test nach BDD-Style vor der Implementierung definieren, würden wir testgetrieben entwickeln und es wäre sichergestellt, dass wir die Testabdeckung erhöhen. Den Kostentreiber von TDD im agilen Umfeld kann man organisatorisch abfangen, indem man festlegt, dass die Implementierung des Tests auf die endgültige Version der Funktion ausgerichtet sein muss. Solange die Implementierung nicht abgeschlossen ist, kann der Test mit einem „#in-arbeit“ Tag versehen und von der Ausführung ausgeschlossen werden.

Zusammensetzung der Tests

Es gibt unterschiedliche Arten von Tests. Wobei nicht alle Tests für jede Anwendung anwendbar sind und ggf. die Implementierungszeit sich je nach Anwendung unterscheiden kann. Die Tests werden häufig in Kategorien zusammengefasst und in Form einer Pyramide dargestellt.

Kategorie mögliche Implementierung:

Manuelle Test-Arbeitsanweisung
UI-Tests Selenium, ABAP Modultests*
Komponenten-Tests OPA, ABAP Modultests
Unit-Tests Qunit, Sinon, ABAP Modultests

Im Internet kursieren unterschiedliche Versionen der Pyramide und unterschiedliche Zahlen, die besagen, wie viele Tests von welcher Kategorie implementiert werden sollten. Gehen wir zunächst die Kategorien durch und prüfen, ob wir mit den Tests aus diesen Kategorien unsere Ziele erreichen können.

Manuelle Tests

Die manuellen Tests sind uns bekannt und für Stellen erforderlich, die von anderen Tests mangels Integration nicht abgedeckt werden können. Hier wird man weiterhin mit Checklisten arbeiten müssen. Die Ergebnisse der Tests sind nicht immer reproduzierbar, weil die Tests nicht isoliert sind. Man hat nicht immer die gleiche Ausgangsbasis was die Testdaten angeht, man scheitert ggf. an Sperren, die von anderen Benutzern gesetzt werden oder man hat nicht die Berechtigungen, die man für den Test braucht. Insgesamt muss man sagen, dass das die teuersten Tests sind.

E2E Tests

Die End-To-End-Tests stehen bei der Zuverlässigkeit des getesteten Codes an zweiter Stelle. Sie haben aber die gleichen Nachteile wie die manuellen Tests. Aus der Kategorie der automatisierten Tests sind es allerdings die einzigen Tests, die uns definitiv sagen können, ob unsere Anwendung funktioniert oder nicht. Da die E2E-Tests nicht isoliert ablaufen, testen wir damit die Anwendung nur zur aktuellen Konfiguration und zu den Berechtigungen des aktuellen Anwenders. Damit ist der Testumfang stark eingeschränkt.

Komponententests

Die Komponententests testen nicht die gesamte Anwendung, sodass aus den Testergebnissen nur bedingt abzuleiten ist, ob die Anwendung nun fehlerfrei ist oder nicht. Komponenten können z.B. sein: Front- und Backend. Die Gesamtheit der Komponententests kann die Fehler nicht identifizieren, bei der die eine Komponente nicht zu der anderen passt.

Da die E2E-Tests eine Inkompatibilität der Komponenten aufdecken sollten, sind wir bei den Komponententests schon sehr nah an der Aussage, ob eine Anwendung fehlerfrei ist oder nicht. Komponententests erlauben eine isolierte Ausführung und decken darüber hinaus alle Konfigurationen und Kombinationen aus Berechtigungen ab. Da ein Komponententest große Codeanteile abdeckt, ist der Aufwand für die gesamte Abdeckung hier geringer als bei den Unit-Tests.

Unit-Tests

Mit Unit-Tests testet man die kleinsten Einheiten, wie z.B. eine Methode oder einen Funktionsbaustein. Die Unit-Tests haben eine ähnlich gute Integration wie die Komponententests, sodass eine isolierte Ausführung auch hier möglich ist. Um die gesamte Anwendung mit Unit-Tests abzudecken, wird man für jede logische Verzweigung einen Unit-Test schreiben müssen. Die Ergebnisse der Modultests lassen nur bedingt auf die Funktionstüchtigkeit der Anwendung schließen, weil sie nur einen kleinen Teil der Anwendung testen. Gleichzeitig ist eine Abstraktion auf der Ebene des Endbenutzers hier schwer umsetzbar.

Zwischenfazit

Mit den automatisierten Tests wollen wir feststellen, ob unsere Anwendung nun fehlerfrei ist oder nicht. Diese Frage können uns definitiv nur die E2E-Tests beantworten, daher sollten diese Tests in erster Linie implementiert werden. Sobald wir in ein Bereich kommen, den die E2E-Test aufgrund schwacher Integration nicht mehr abdecken können, sollte man auf die Komponententests ausweichen. Mit Komponententests sollte man die restlichen Funktionalitäten abdecken. Auf die Modultests sollte man dagegen verzichten, weil diese zum einen den größtmöglichen Aufwand für die Implementierung darstellen und gleichzeitig am wenigsten darüber aussagen können, ob unsere Anwendung fehlerhaft ist oder nicht.

Fazit

Wann sollte man also in die Testautomatisierung einsteigen und welche Anwendungen eignen sich dafür? Jeder Zustand der Anwendung eignet sich für den Einstieg in die Testabdeckung. Ganz egal, ob die Anwendung schon seit Jahren in Linie gewartet wird oder man sich noch in der Konzeption befindet. Wenn Sie Ihre Software heute manuell testen, dann können Sie heute auch anfangen, die Tests dafür zu entwickeln. Die Architektur der meisten Anwendungen ist für die Testautomatisierung ungeeignet. Die Anwendungen müssen dafür angepasst werden. Bevor der erste, qualitativ hohe Test geschrieben werden kann, wird man einige Voraussetzungen schaffen müssen. Es wird eine Architektur benötigt, die es ermöglicht, nicht testbaren Code zu isolieren und für die Testlaufzeit ersetzen zu können. Sobald der erste Test läuft, wird man für die nachfolgenden Tests auf die bereits geschaffene Infrastruktur zurückgreifen können. Trotzdem wird der Aufwand für die ersten Tests sehr hoch sein und erst mit der Zeit immer weiter abnehmen.

Über den Autor

Richard Martens ist im Team Business Technologie tätig. Er hat seit 14 Jahren Erfahrung im SAP Bereich und verfügt über Zertifizierungen für SAP Certified Development Associate und ABAP with SAP Netweaver 7.0. Seien Aufgaben beziehen sich größtenteils auf die Konzeption und Entwicklung in SAP mit ABAP/ABAP OO und SAPUI5. Seine Kompetenzen reichen zusätzlich von SAP Web-IDE, SD, WM, MM, PP, SAP Netweaver Gateway/ OData, BSP, HTML5, CSS und JavaScript über Adobe LifeCycle Designer, CDS, BOPF, Design Thinking, Design-Patterns, Fiori Design Guidelines bis hin zu SAP Cloud. Herr Martens hat schon in den Branchen Gesundheitswesen, Metallverarbeitung, Industriedienstleistungen, Abfallentsorgung, Chemie und Pharma gearbeitet. Seit 2011 ist Richard Martens für die Bitech AG in Leverkusen tätig.

Bitech.AG
An der Schusterinsel 15
51379 Leverkusen

Tel. 02171 / 72 30 - 0
info@bitech.ag
www.bitech.ag

Bitech.AG