Testautomatisierung - Lessons learned

Andreas Schönknecht, TUI InfoTec GmbH, andreas.schoenknecht@tui.com

In unserem aktuellen Projekt haben wir das Thema Testautomatisierung nahezu seit Beginn des Projektes auf der Agenda gehabt und recht intensiv betrieben. Nach viel Arbeit und auch einigen Irrwegen haben wir mittlerweile einen sehr guten Stand erreicht:

Wir führen nur Minuten nach dem Commit einer Änderung am Source-Code fast 3000 automatisierte Testfälle aus. Das Testergebnis wird dem Entwicklerteam auf großen Bildschirmen an der Wand sofort sichtbar gemacht. Wenn Tests fehlschlagen ist die Suche und Behebung der Ursache daher unmittelbar möglich, maximal 30 Minuten nach dem Commit. Die Menge der Änderungen, die zu einem fehlschlagenden Test führen kann, ist wegen der zeitlichen Nähe zum Test gering, so dass die Behebung normalerweise schnell erfolgt.

Bei Gesprächen zum Thema Testautomatisierung mit Mitarbeiten anderer Projekte stellte sich für mich wieder einmal heraus, dass es bei diesem Thema weiterhin eine große Diskrepanz zwischen Wunsch und (einfacher) Machbarkeit gibt.

So scheint die naive Vorstellung immer noch recht weit verbreitet zu sein, man müsste nur das richtige Tool finden und schon geschieht von der Testfallspezifikation bis zum Bugfix alles automatisch. **Nicht zu unterschätzen ist erst einmal die Erkenntnis, dass sich nur die Testausführung und –protokollierung automatisieren lassen.**

Die Testfallspezifikation und Analyse der Auffälligkeiten muss weiterhin wie beim manuellen Testen erfolgen.

So ist die erste Frage zur Testautomatisierung an uns leider oft nicht die nach unserem Testprozess, sondern die nach den von uns eingesetzten Tools.

Ich habe mich eigentlich während meiner gesamten Berufszeit mehr oder weniger intensiv mit dem Thema befasst, vor meinem aktuellen Projekt zugegebenermaßen nur mit mäßigem Erfolg. Zumindest hatte ich aber eine Menge Erfahrung gesammelt, wie es nicht funktioniert.

Die Hauptursache für ein Scheitern der Testautomatisierung ist meines Erachtens, die Nichtintegration der Testautomatisierung in den eigentlichen Softwareentwicklungsprozess.
In allen vorherigen Projekten, an denen ich beteiligt war und in denen Tests automatisiert werden sollten, hatte man ein von den Programmierern separiertes Testteam aufgestellt, das sich um Testautomatisierung kümmern sollte.
Nicht selten bestand das Team auch nur aus einer einzelnen Person.

Aus meiner Sicht kann das aus verschiedenen Gründen nicht funktionieren:

1. Das (Automatisierungs-)Testteam bekommt neue Anforderungen nicht mit und fängt mit der Spezifikation von Testfällen erst an, wenn die neuen Features in einer Testumgebung deployed sind. Bis dann daraus automatisierte Testfälle entstanden sind, ist das zu testende Feature meist längst produktiv.
2. Das Testteam hat zu wenig fachliches Wissen bzw. zu wenig Zeit, um überhaupt aussagekräftige Testfälle erstellen zu können.
3. Das Testteam hat zu wenig technische Möglichkeiten, um das zu testende System in einen automatisiert testbaren Zustand zu versetzen.
4. Wenn die automatisierten Tests einen potentiellen Fehler finden, ist die Analyse für das Testteam sehr anstrengend. Die Wahrscheinlichkeit, dass es sich entweder um ein nun gewolltes Verhalten nach eine Anforderungsänderung handelt oder um instabile Testdaten bzw. eine instabile Testumgebung ist sehr hoch, das Testteam wird aber wegen der Punkte 1-3 eine vergleichsweise Lange Zeit benötigen, um dies herauszufinden.
5. Wenn die automatisierten Tests dann trotz der Punkte 1-4 einen Fehler finden, hat das Testteam Mühe jemanden zu finden, der das hören möchte. Denn dies bedeutet ungeplante Arbeit. Oft handelt es sich bei den gefundenen Fehlern auch um Kleinigkeiten, denn die schwerwiegenden Fehler sind vermutlich schon vorher gefunden worden. Weil die Durchführung der automatisierten Tests bzw. insbesondere die Ergebnisanalyse wegen Punkt 4 lange dauert, verzichtet man trotz automatisierter Tests nicht auf manuelle Tests.
6. Die von den automatisierten Tests gefundenen Kleinigkeiten werden vom Entwicklerteam nicht mit hoher Priorität behoben und bleiben lange im System. Dadurch schlagen die automatisieren Tests lange Zeit fehl und verdecken womöglich wei-

tere Fehler. Zumindest erschweren sie die Testergebnisanalyse zusätzlich.

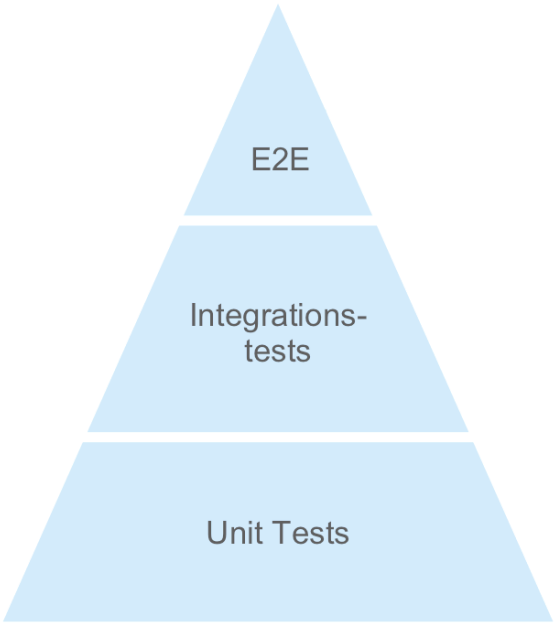
Deswegen hat wesentlich zum Erfolg der Testautomatisierung in unserem Projekt beigetragen, dass wir die Verantwortung dafür in das Entwicklerteam genommen haben. Wir hatten in der Anfangszeit zwar einen Berater mit dem Schwerpunkt Testautomatisierung im Team, aber explizit sollte er keine Tests automatisieren, sondern das restliche Team "nur" bei dieser Aufgabe unterstützen und sich um die nötige Testinfrastruktur kümmern. Nach ein paar Monaten war das Wissen dann aber im Team ausreichend verbreitet, so dass wir keinen dedizierten Mitarbeiter mehr für das Thema benötigen.

Die oben genannten Gründe für das Scheitern der Testautomatisierung wurden bei uns dadurch stark gemildert, denn:

1. Das Entwicklerteam erstellt die automatisierten Tests bereits während der Umsetzung einer neuen Funktionalität. Wenn die neue Funktion produktiv geht, sind die zugehörigen Regressionstests vorhanden.
2. Der Entwickler, der die neue Funktion implementiert, erstellt auch den automatisierten Test. Er hat das dafür notwendige fachliche Wissen. Andernfalls könnte er auch die eigentliche Anforderung nicht implementieren.
3. Das Entwicklerteam hat vollen Einfluss auf das zu testende System und kann die Schnittstellen und Architektur der Applikation entsprechend testbar gestalten.
4. Bei anschlagenden Tests wissen die Entwickler, was sie zuletzt geändert haben und können (wenn es sich nicht um einen Fehler handelt) die Tests bzw. die Testdaten anpassen.
5. Wenn ein Fehler gefunden wird, ist wegen des kurzen Testzyklus schnell ersichtlich, wodurch er verursacht wurde und das Entwicklerteam fühlt sich verantwortlich, den Fehler schnell wieder zu beseitigen.
6. Dies trägt auch zu einer deutlichen Verbesserung der Qualität der Applikation bei, denn auch Kleinigkeiten werden unverzüglich behoben.

Eine weitere häufige Ursache für das Scheitern der Testautomatisierung ist die falsche Wahl der Testschnittstelle. Hier sind wir auch in unserem Projekt zunächst einen Irrweg gegangen.

Für stabile, wartbare Tests sollte das zu testende System möglichst klein sein. **Daher sollte die Masse der Tests Unit-Tests sein, die nur einen sehr kleinen Ausschnitt testen.**



In Java ist eine Unit in der Regel eine Klasse bzw. ein Spring-Service.
Alle umliegenden Klassen sollten dann gemockt sein und die für den konkreten Test benötigten Testdaten bereitstellen. Da in einem Test nur ein sehr kleiner Ausschnitt der Applikation getestet wird, ist die Fehlersuche bei einem Fehlschlagen auch relativ einfach. Je nach Größe der zu testenden Unit ist auch die Bereitstellung von Testdaten nicht allzu schwierig. Die Testschnittstelle für Unit-Tests sind einzelne Funktionsaufrufe.

Für unsere Applikation haben wir aktuell 2371 Unit-Tests (Java und JavaScript), deren Ausführung etwa 2 Minuten benötigt.

Die nächsthöhere Testebene sind Integrationstests. Diese Tests benötigen bereits größere Teile der Anwendung bis hin zu Datenbanken. Hier ist es bereits deutlich aufwändiger, das zu testende System stabil zu halten. Die Testschnittstelle für Integrationstests kann zum Beispiel eine öffentliche Schnittstelle des zu testenden Systems sein.

Wir haben derzeit 438 Integrationstests. Die Ausführung benötigt etwa 10 Minuten.

Schließlich gibt es die Testebene der UI basierten Tests. Bei diesen Tests simuliert der Testautomat einen realen Benutzer und bedient die GUI des Systems (bei uns ist das der Browser). Hier zeigt die Erfahrung, dass solche Tests am wartungsintensivsten

sind, da (wenn man das Backend nicht wegmockt), zum einen das zu testende System sich in einem dem Tester bekannten Zustand befinden muss und zum anderen ist die Bedienung von GUIs für Tools immer noch kompliziert und fehleranfällig (auch wenn die Hersteller dieser Tools gerne etwas anderes behaupten).

Wir haben momentan 101 oberflächenbasierte Tests. Die Ausführung dauert etwa 15 Minuten, wobei ein großer Anteil dieser Tests mit einem gemockten Backend arbeitet, so dass tatsächlich nur die Funktionalität im Browser getestet wird, welches mit hierfür vorbereiteten Testdaten versorgt wird.

Da die UI Testtools (oft auch Capture/Replay Werkzeuge genannt) in begrenztem Umfang auch von Leuten ohne Programmierkenntnisse eingesetzt werden können, lassen sich Projekte oft dazu verleiten, sich vor allem auf diese Testebene zu konzentrieren. Dies rächt sich aber schnell durch die Probleme, diese Tests stabil und reproduzierbar zu halten.

In unserem Projekt hatten wir anfangs ebenfalls den Fehler gemacht, uns hauptsächlich auf UI basierte Tests zu konzentrieren. Der Pflegeaufwand war aber aus den oben genannten Gründen viel zu hoch, so dass wir nach und nach diese Tests durch Unit-Tests und einige Integrationstests ersetzt haben. Dieser Prozess ist noch nicht abgeschlossen.

Abschließend möchte ich noch einmal darauf hinweisen, dass Testautomatisierung Softwareentwicklung ist und die selben Skills erfordert. Je tiefer die Testebene liegt, umso mehr ähneln die Tests dem Programmcode. Insbesondere zur Erstellung der Unit-Tests muss der Programmcode der zu testenden Anwendung vorliegen und verstanden werden.
Um die zu testende Anwendung mit den benötigten Testdaten zu versorgen sind insbesondere für die höheren Testebenen Programme zu erstellen, welche die benötigten Testdaten an den Schnittstellen des Systems bereitstellen. Diese Schnittstellen müssen dem Testautomatisierer wohl bekannt sein. Idealerweise kann er die Schnittstellen sogar verändern, um sie für automatisierte Tests leichter bedienbar
zu machen.

Ich will nicht sagen, dass es unmöglich ist, einen automatisierten Regressionstest für ein eingekauftes Fremdsystem zu etablieren, aber wenn das weit weg von den tatsächlichen Entwicklern des Systems geschehen soll, ist das eine immense Herausforderung und nichts, das ein bis zwei Personen mit einer 20-Prozent-Zuordnung nebenbei bewerkstelligen können.

In unserem Projekt sind über die bisherige Projektlaufzeit hinweg einige hundert Bearbeitertage in die Testautomatisierung geflossen. Aber dafür haben wir nun ein System, bei dem spätestens alle zwei Wochen eine neue Version produktiv geht, für die kaum noch manuelle Regressionstests durchgeführt werden und für das kein Bug-Tracking-System im Einsatz ist, da es kaum Bugs zu managen gibt.

Auch wenn sich das nicht nachweisen lässt, bin ich darüberhinaus davon überzeugt, dass wir die investierte Zeit durch nicht gemachte manuelle Regressionstests und nicht notwendig gewordene Fehleranalysen wettgemacht haben.

In dem Artikel habe ich Risiken für den Erfolg von Testautomatisierung genannt und wie man ihnen begegnen kann: Durch Integration der Testautomatisierung in den Softwareentwicklungsprozess und durch Einhaltung der Testpyramide (viele Unit-Tests, einige Integrationstests, wenige UI-Tests). Der Umkehrschluss, dass dies unweigerlich zu erfolgreicher Testautomatisierung führt, gilt leider nicht. Man kann auch unwartbare Unit-Tests schreiben und wenn das Entwicklerteam nicht aus sich heraus den Nutzen von automatisierten Tests erkennt und deswegen das Erstellen und Pflegen der Tests vernachlässigt, wird es ebenfalls scheitern.