Nico Orschel

(nico.orschel@aitgmbh.de)
ist Software Process Consultant, Autor und Referent im Umfeld Microsoft ALM bei der AIT GmbH & Co. KG Stuttgart und wurde von Microsoft als Most Valuable Professional (MVP) für Visual Studio ALM ausgezeichnet. Er hilft Unternehmen, auf Basis von Microsoft Visual Studio Team Foundation Server effizienter Software zu entwickeln und zu testen und so ein höheres Qualitätsniveau bei kürzeren Release-Zyklen zu erreichen.

Ein Dialog unter Fremden: Testautomatisierung in der Praxis

Montagmorgen 8:15 Uhr, das E-Mail-Programm zeigt den Empfang einer neuen Mail. Absender: Unbekannt. Betreff: Testautomatisierung. Stellen Sie sich vor, Sie sind der Empfänger dieser Mail. Die Situation könnte nicht skurriler sein, denn Ihr Chef hat Sie erst vor wenigen Tagen mit einem neuen strategischen Projekt betraut. Sie als langjähriger Mitarbeiter sollen Testautomatisierung in alten und neuen Softwareprojekten des Unternehmens einführen bzw. verbessern. Das Problem: Sie kennen das fachliche Umfeld, sind aber Einsteiger in der Testautomatisierung und möchten nichts falsch machen. Der Artikel dokumentiert den Verlauf einer imaginären Unterhaltung zwischen dem unbekannten, erfahrenen E-Mail-Absender und Ihnen, dem Fragesteller. Im Verlauf des Gesprächs diskutieren Sie beide die Grenzen, Chancen und Erfahrungen bei der Umsetzung von Testautomatisierung im Alltagseinsatz. Die Schwerpunkte liegen dabei auf Arbeitsteilung im Team, dem richtigen Test- und Aufwandsmix bei Legacy Software und Neuentwicklungen sowie Anforderungen an Programm- und Testcode.

Das bin ich

Sie werden es nicht glauben, aber lassen Sie mich am Anfang meiner Reise zur Testautomatisierung beginnen. Vor wenigen Tagen habe ich relativ überraschend von meinem Vorgesetzten die Aufgabe bekommen, das Thema Testautomatisierung für bereits existierende Produkte und Neuentwicklungen aus unserem Haus zu begleiten. In meiner aktuellen Position arbeite ich als Tester bereits seit mehr als 5 Jahren. Als Tester kenne ich deshalb mein fachliches Umfeld gut, schreibe ohne große Mühe meine Testfälle und organisiere diese effizient in Testplänen. Zu Beginn haben wir unsere Testpläne noch in Excel organisiert. Sie können sich sicherlich vorstellen, dass das eine Menge Arbeit verursacht hat. In den letzten drei Jahren haben wir unseren Testmanagementprozess von Excel auf den Microsoft Team Foundation Server (TFS) und den dazugehörigen Microsoft Test Manager (MTM) umgestellt. Für uns war das eigentlich keine große Umgewöhnung, haben doch unsere Entwickler und Projektmanager den TFS schon wesentlich länger im Einsatz. Die tägliche Arbeit mit dem MTM verläuft bis jetzt gut. Wir können schnell und einfach unsere Testfälle erfassen, die Tests ausführen und die Ergebnisse und Testfälle mit den Beteiligten teilen. Im Rahmen unserer ersten Gehversuche haben wir mit MTM die Testausführungen aufgezeichnet und beim wiederholten Test abgespielt und so etwas Zeit gespart. Für einfache Schrittfolgen hat dies recht gut funktioniert und so beim Management den Wunsch nach mehr Automatisierung geweckt.

Testautomatisierung

Bling *Sie haben eine neue E-Mail*
Absender: *unbekannt@gmx.net*
Empfänger: *tester@gmail.com*
Datum: *01.08.2013 8:15*
Betreff: *Testautomatisierung*
Nachricht:
Hallo Fremder,
Sie kennen mich nicht. Ich hatte gerade das Bedürfnis, mich mit einem Fremden austauschen zu wollen. Mein Thema ist aber schon ungewöhnlich – Testautomatisierung. Beschäftigen Sie sich zufällig auch mit diesem Thema? Wenn ja, dann würde ich mich freuen, mit Ihnen das Thema zu erörtern.
Als Berater unterstütze ich viele Kunden bei der Umsetzung und Planung von Testautomatisierung. Durch verschiedene Kundenprojekte, Blogartikel, Tweets und Fachzeitschriften konnte ich verschiedene Trends im Softwaretestbereich beobachten. In den Medien wurde in den vergangenen Jahren viel zum Thema agile Prozesse und damit oft einhergehend Unit-Testing veröffentlicht. Testautomatisierung wird dabei oft mit Unit-Testing gleichgesetzt. Aus meiner Sichtweise ist dies jedoch zu kurz gedacht.
Was verstehen Sie unter Testautomatisierung? Was gehört aus Ihrer Sichtweise dazu?

Mit freundlichen Grüßen
Der Unbekannte

Nach dieser Mail war ich erstmal sehr überrascht und verwirrt zu gleich. Es ist schon ein großer Zufall, dass ich mich wirklich gleichzeitig mit dem Thema beschäftige. Die erste Frage ist äußerst interessant: Was ist Testautomatisierung?

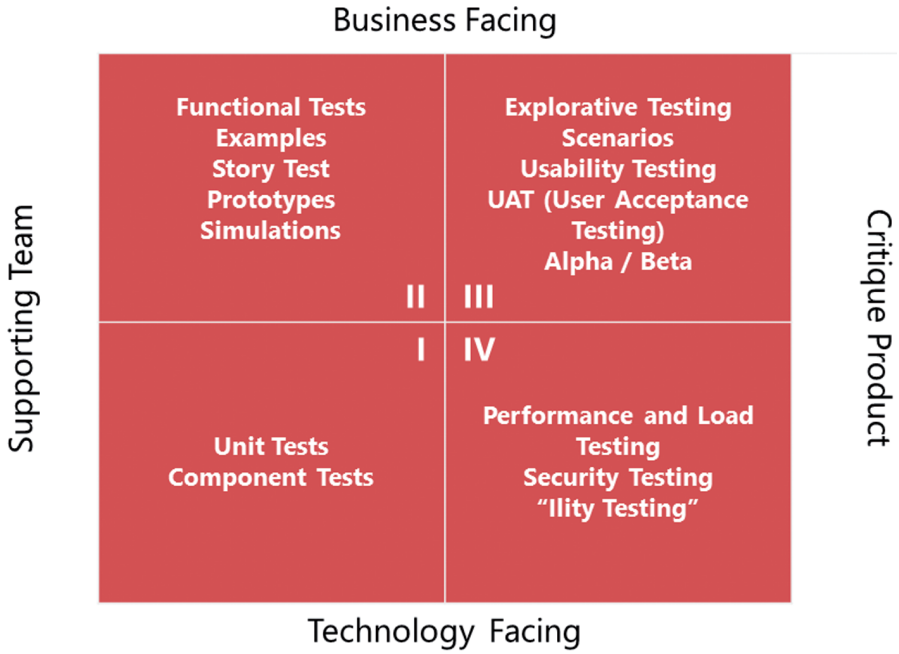

Abb. 1: Agile Testing-Quadranten.

Was gehört eigentlich alles dazu? Testautomatisierung würde ich definieren als die Automation von Testaktivitäten. Die Automation umfasst dabei nicht nur das automatische Abarbeiten von Schritten, sondern auch die Prüfung des Testergebnisses. Die Prüfung des Testergebnisses entscheidet über Bestehen oder Fehlschlagen der Tests.

Vor ein paar Wochen habe ich mir zur Vorbereitung auf meine neue Aufgabe ein Buch mit dem Titel „Agile Testing“ (vgl. [Cri08]) gekauft. Im Buch wird ein interessantes Konzept zur Strukturierung von Testarten vorgestellt, die Agile Testing Quadrants (siehe Abbildung 1).

Das Konzept gruppiert grundlegend die Testarten nach verschiedenen Dimensionen: technologieorientiertes, teamorientiertes, produktorientiertes und fachorientiertes Testen. Aus der Kombination der Dimensionen ergeben sich vier Sektoren (von I links unten bis IV rechts unten).

Das Thema Unit-Testing aus der Mail des Unbekannten deckt also nur einen Bruchteil eines umfassenden Testprozesses ab (siehe Sektor I). Betrachte ich unseren eigenen Testprozess, dann fällt auf, dass unsere Testaktivitäten primär im Quadranten III angesiedelt sind.

Absender: *tester@gmail.com*
Empfänger: *unbekannt@gmx.net*
Datum: *01.08.2013 19:01*
Betreff: *Re: Testautomatisierung*
Nachricht:
Hallo Unbekannter,
Ihre Mail hat mich sehr überrascht. Es ist schon ein großer Zufall, dass ich mich gerade ebenfalls mit dem Thema Testautomatisierung in meinem Job befasse.
Das Thema Testautomatisierung scheint mir inhaltlich sehr breit zu sein. Generell verstehe ich unter Testautomatisierung die Automatisierung von Testaktivitäten und Prüfung der Ergebnisse.
Um den Überblick über das breite Feld der Testarten zu erhalten und konkret Testarten inhaltlich für einzelne Projekte zu planen, verwende ich die „Agile Testing Quadrants“ von Brian Marrick (vgl. [Cri08]). Aus dem Konzept habe ich für mich Komponententests, funktionale Tests, User Acceptance Testing, Szenarien sowie Performance- und Load-Testing als sehr gute Bereiche für die Testautomatisierung abgeleitet.
Wenn ich die vielen Bereiche sehe, dann habe ich die Befürchtung, dass kleine Entwicklungs- und Testteams überfordert sein könnten.
Wie verteilen und organisieren Sie die Arbeit in Ihren Teams?

Viele Grüße
Der Tester

Teammix

Mittlerweile ist wieder ein Tag vergangen und meine Gedanken drehen sich immer noch um die Organisation meines Teams. Wie verteile ich die Arbeit auf die verschiedenen Personen? Wer kann was leisten? Wer sollte was leisten können?

Aktuell setzt sich das Team nur aus „klassischen“ Testern zusammen. Testen bedeutet hier vereinfacht dargestellt, Testfälle erstellen, Testfälle ausführen und Ergebnisse kommunizieren. Mit dem neuen Thema Testautomatisierung kommen aber plötzlich Code, Unit-Tests, Oberflächentests mit Visual Studio CodedUI, Schnittstellentests oder Performancetests auf das Team zu.

Absender: *unbekannt@gmx.net*
Empfänger: *tester@gmail.com*
Datum: *02.08.2013 8:43*
Betreff: *Re: Re: Testautomatisierung*
Nachricht:
Hallo Tester,
Es freut mich, wieder von Ihnen zu hören. Das Thema Teamorganisation ist aus meiner Erfahrung spannend, höchst individuell und unterbewertet zugleich.
Wie sich bereits in den vorherigen E-Mails sicherlich herausgestellt hat, ist die Testautomatisierung zu großen Teilen von Softwareentwicklungsaktivitäten geprägt. Kurz gesagt: „Eine Softwareentwicklung in der Softwareentwicklung.“ Es wird Quellcode geschrieben, welcher wiederum Code testet. Aus meiner Sichtweise ergibt sich daraus, dass für eine langfristig erfolgreiche Testautomatisierung gute bis sehr gute Entwicklungsfähigkeiten notwendig sind. Es wird für diese Aufgabe eigentlich der klassische Entwickler benötigt. In der Realität finden aber Entwickler Softwaretesten nicht besonders attraktiv. Die enorme Bedeutung dieser Aufgabe zeigen z. B. spezielle Jobrollen bei den großen IT-Giganten (Google, Microsoft, etc.). Diese großen Firmen nennen diese Rolle beispielsweise „Engineers for Testing“. Diese Mitarbeiter haben eine Ausbildung zum Entwickler, welche um die Testexpertise erweitert wird, sodass sich diese Experten sehr elegant zwischen den Welten der Tester und der Entwickler bewegen können.
Nun stellt sich die Frage: Brauche ich jetzt keine klassischen Tester mehr oder muss ich Tester zu Programmierern umschulen? Tester haben ja aufgrund des fachlichen Hintergrundes selten mit Visual Studio, C++, C# etc. zu tun. Meine Antwort heißt hier klar: nein. Tester haben eine sehr wichtige Aufgabe bei Testautomatisierungsprojekten. Sie haben hier den großen

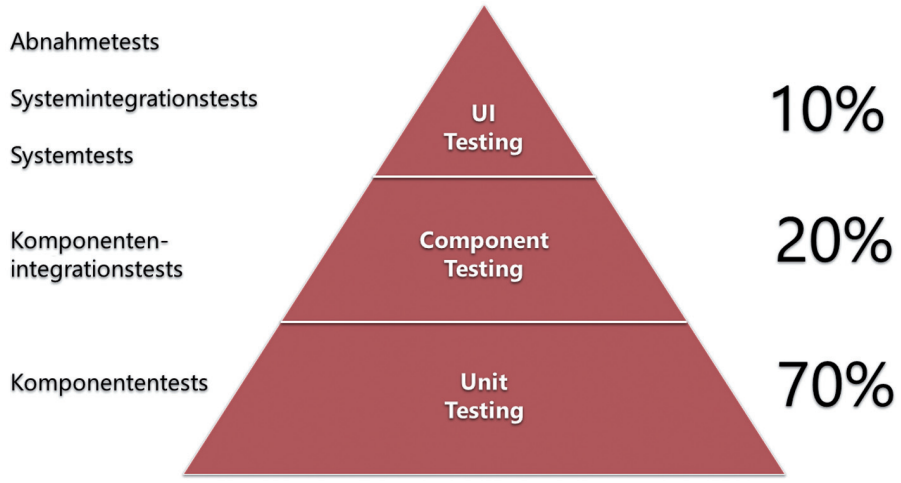

Abb. 2: Test Automation-Pyramide.

Vorteil, dass sie unbelastet von Technik und Entwicklung sind. Zu testen und fachliches Feedback im Kontext des Testautomatisierungsprozesses zu geben ist deshalb neutraler möglich. Die Testfälle können als Ergänzung zu den Anforderungen aufgefasst werden. Sie bilden damit ein wichtiges Fundament. Viele Entwickler verfallen bei Testautomatisierung gerne dem technischen Interesse, d. h. Automatisieren um zu Automatisieren, ohne konkrete Planung und aus konkreten Testfällen abgeleiteten Anforderungen. Ein guter Tester kann hier für eine gesunde Balance aus Technik, Aufwand und Nutzen sorgen.
Zum Schluss der heutigen Mail eine weitere Frage an Sie. Haben Sie sich schon Gedanken über den Umfang der einzelnen Testaktivitäten gemacht? Wenn nein, dann möchte ich Ihnen gerne die Test Automation-Pyramide von Mike Cohn mit auf den Weg geben (vgl. [Mou]).

Mit freundlichen Grüßen
Der Unbekannte

Testmix

Der Unbekannte hat Recht. Aktuell habe ich mir nur über das *Was* und *Wer*, aber nicht über das *Wie viel* Gedanken gemacht. Auf Grundlage des Stichwortes Test Automation-Pyramide (siehe ***Abbildung 2***) habe ich eine Internet-Recherche durchgeführt. Die Test Automation-Pyramide teilt die Tests für die Planung von Testaufwänden (bezogen auf Standardsoftware) in drei Klassen ein, Unit-, Komponenten- und oberflächenbasierende Tests. Unit-Testing kommt dabei mit ca. 70 % am gesamten Testaufwand eine besondere Bedeutung zu.

Gefolgt werden sie von den Komponentenintegrationstests mit ca. 20 % am Gesamtaufwand und den UI-Tests mit den verbleibenden ca. 10 %. Die interessante Frage ist hier: Warum haben Unit-Tests einen solchen dominanten Anteil gegenüber den UI-Tests? Brauche ich überhaupt die „höheren" Teststufen? Unit-Tests und Komponententests setzen ein hohes internes Wissen über die zu testende Software voraus, d. h., der Tester benötigt Wissen über Schnittstellen bzw. Programmcode. Diese Testmethoden haben generell den Vorteil, dass bei sich ändernden Schnittstellen oder Buildprozessen Werkzeuge wie z. B. Visual Studio frühzeitig auf potenzielle Fehler und notwendige Anpassungen hinweisen können. Werden dadurch Fehler früher erkannt und notwendige Anpassungen zeitnah durchgeführt, so führt dies direkt zu reduzierten Testpflegeaufwänden. Reduzierte Testaufwände bedeuten als Folge weniger Kosten, personelle Aufwände und frühere Liefertermine.

Die Herausforderung beim Testen in den höheren Teststufen liegt meistens in zwei Punkten: erstens wird eine lauffähige Software getestet und zweitens verfügt die zu testende Software über umfangreiche externe Abhängigkeiten (z. B. Datenbanken, Webserver). Das notwendige Wissen über Interna wie Klassenstrukturen oder Organisation und Quellcode nimmt im Verhältnis zu den „unteren" Teststufen ab. Aufgrund der Notwendigkeit einer laufenden Software ist das automatisierte Testen erst mit einem Zeitversatz möglich. Die höheren Schichten müssen für aufwandsoptimiertes Testen eine hohe Stabilität besitzen. Bezüglich des Testcodes gilt hier zu beachten, dass Fehler im Testcode oft nach Änderung und anschließender Ausführung erst zur Laufzeit feststellbar sind. Analysen gestalten sich entsprechend aufwändiger. Bedeutet das nun, alle Aufwände auf Unit-Testing zu konzentrieren? Meiner Meinung nach ist dem nicht so; die oberen Teststufen ergänzen Tests um andere inhaltliche Schwerpunkte. Ohne die oberen Schichten könnten z. B. keine End-To-End-Tests durchgeführt werden. Werden verschiedene Komponenten zusammen in Kombination getestet, dann können diese ein anderes Verhalten aufweisen. Das Gleiche gilt, wenn sich das Verhalten durch die Kombination von unterschiedlichen Benutzeraktionen verändert. Als Ergebnis bedeutet dies, dass die oberen Teststufen eine notwendige Ergänzung darstellen.

Absender: *tester@gmail.com*
Empfänger: *unbekannt@gmx.net*
Datum: *03.08.2013 19:01*
Betreff: *Re: Re: Re: Testautomatisierung*
Nachricht:
Hallo Unbekannter,
Vielen Dank für den Hinweis auf das Test Automation-Pyramidenkonzept. Eine dreiteilige Einteilung der Testarten und die prozentuale Gewichtung sind ein leicht verständliches Konzept zur Planung einer ganzheitlichen Testautomationsstrategie. Ich werde dies bei meinem Testplan berücksichtigen.
Für mich ergibt sich daraus die Anforderung, dass ich in meine Teststrategie auf jeden Fall die Entwicklerperspektive einbeziehen muss (Unit- und Komponententests).
Zusätzlich zu den unteren Teststufen sollten wir vom Testteam die oberen Teststufen adressieren. In Anlehnung an die vorletzte E-Mail werde ich einen neuen Kollegen mit Entwicklungshintergrund als Teamunterstützung suchen. Wir wollen schließlich eine nachhaltige Testautomatisierung aufbauen.
Es ist an dieser Stelle wieder interessant zu sehen, wie unsere aktuelle Testpraxis ist. Wenn ich aus der Vogelperspektive auf unseren aktuellen Testprozess schaue, dann konzentrieren sich aktuelle Aufwände auf manuelles oberflächenbasierendes Testen. Ich denke, hier haben wir viel Optimierungspotenzial im Hinblick auf die Test Automation-Pyramide (aktueller Fokus UI-Testing) und die Agile Testing-Quadranten (aktueller Fokus: Quadrant III). Unser Testen muss hier breiter aufgestellt und repriorisiert werden.

Haben Sie zufällig noch Tipps zum Umgang mit „alten“ und „neuen“ Projekten im Zusammenhang mit Testautomatisierung?

Viele Grüße
Der Tester

In meiner Selbstvorstellung habe ich bereits geschrieben, dass ich demnächst sowohl die Testautomatisierung für alte als auch neue Projekte begleiten soll. In der Vergangenheit hat mein Arbeitgeber bereits mehrere kleine Versuche unternommen, Testautomatisierung einzuführen. Es wurde oft probiert, wie in der Test Automation-Pyramide vorgestellt, Unit-Tests als das Fundament zu verankern. Es stellt sich jetzt für mich die Frage, ob dies wirklich immer der richtige Weg für eine wirtschaftliche Testautomatisierung ist.

Legacy-Software und Neuentwicklungen

Absender: *unbekannt@gmx.net*
Empfänger: *tester@gmail.com*
Datum: *05.08.2013 14:53*
Betreff: *Re: Re: Re: Re: Testautomatisierung*
Nachricht:
Hallo Tester,
Ich möchte gerne Ihre erste Frage aufgreifen. Im Alltag beobachte ich sehr viele Teams, welche Ihre gesamten Anstrengungen auf Unit-Tests konzentrieren. Diese Strategie ist aus meiner Sichtweise erstmal grundlegend für Neuentwicklungen vernünftig.
Vorteilhaft ist hierbei, dass Teams in den frühen Phasen die interne Softwarearchitektur der jeweiligen Anwendungen für Testautomatisierung optimieren können. Beispiele sind im .NET-Umfeld Schnittstellen oder Kapselung von Funktionalitäten in Modulen. Hier sieht man gut, dass die Entwicklung bereits sehr früh den Grundstein für das spätere Testfundament legt. An dieser Stelle kann ich Ihnen den Einsatz eines erfahrenen Softwarearchitekten empfehlen. Frühzeitig Architekturfehler erkennen und beheben spart später Zeit, Geld und Nerven. Ein Review durch einen Dienstleister ist ebenfalls eine interessante Alternative. Ein unabhängiger Blick auf die Software kann bisher nicht erkannte Schwachstellen aufdecken und neue Ideen in das Team bringen.
Einen guten Einstieg in das Thema Optimierung bietet „The Art of Unit Testing“ (vgl. [Art]). Das Buch eignet sich nicht nur, wie der Titel suggeriert, für Unit-Testing, sondern auch für Integrationstests. Die bisherigen Hinweise gelten primär für Neuentwicklungen, aber das bedeutet nicht, dass dies auch zwingend für andere Projekttypen optimal ist. Bei der Einführung von Testautomatisierung für „historische“ Projekte sehe ich die Situation ein wenig anders. Ein Großteil der historischen Projekte ist vor mehr als 10 Jahren entstanden. Bis zum heutigen Tag haben diese Produkte bereits sehr viele Anpassungen und Ergänzungen erfahren. Eine Grundvoraussetzung für eine aufwandsoptimierte Testautomatisierung wurde dabei aber leider sehr selten beachtet: Die Software muss auch eine testoptimierte Softwarearchitektur aufweisen. Eine modulare Bauweise oder Schnittstellen sind oft nur unzureichend umgesetzt. Fehlen bereits diese essenziellen Dinge, so wird eine spätere Testautomatisierung technisch sehr schwierig und verursacht hohe Kosten. An dieser Stelle gilt es, eine Entscheidung zu fällen. Entweder Sie bereinigen aufwändig die Schwächen in der Architektur oder Sie nutzen je nach Technologie Hilfswerkzeuge wie Visual Studio Fakes (vgl. [Msd]) oder Typemock Isolator (vgl. [Typ]), um Abhängigkeiten in der Architektur zu mindest für die Tests aufzulösen. Eine kleine Warnung zum Einsatz: Werkzeuge dieser Kategorie ermöglichen es, sehr elegant „untestbaren“ Code für Unit- und Integrationstests zugänglich zu machen. Die Gefahr lauert aber bei dem nicht zu unterschätzenden Aufwand für Pflege und Wartung der Tests und des damit verbundenen Isolationscodes. Diese Werkzeuge dürfen deshalb nur als kurzfristige Übergangslösung betrachtet werden. Langfristig sollten Sie eine optimierte Softwarearchitektur anstreben.
Für historische Projekte hat sich als guter Startpunkt für Testautomatisierungsprojekte die Konzentration auf die oberen Teststufen, System- und UI-Tests, herausgestellt. Service- und UI-Testing setzen eine nicht so tief greifende Isolation zwischen der zu testenden Software und den externen Abhängigkeiten voraus. Das Ziel ist hier, die Komplexität und den Aufwand für die Erstellung von automatisierten Tests aufgrund von komplexen Isolationscodes zu reduzieren. Ähnliches gilt, wenn notwendige Architekturänderungen verschoben werden müssen.
Durch den Aufbau dieser speziellen Testbasis werden zunächst die Basisfunktionen der zu testenden Software abgedeckt. Das Leitmotiv ist hier: ***Besser Integrations- und UI-Tests als keine automatisierten Tests****. Diese ersten automatisierten Tests bilden später wieder das Sicherheitsnetz für anstehende Refactoring- und Optimierungsmaßnahmen in den Folgeschritten. Ein Nebenziel dieser Strategie ist, sich nicht auf sehr aufwändige Unit-Tests ohne Mehrwert zu konzentrieren.*
Eine letzte Anmerkung zu Oberflächentests: Auch bei dieser speziellen Testkategorie müssen Sie Ihre Software im Vorfeld optimieren. Viele am Markt etablierte Testprogramme (Microsoft Visual Studio CodedUI / Ranorex / etc.) setzen UI-Schnittstellen wie MSAA (Microsoft Active Accessibility) bzw. UIA (Microsoft UI Automation) in Ihrer Software voraus (vgl. [Mic]). Ohne Optimierung investieren Sie auch hier viel Aufwand in eine instabile Testautomatisierung.
Ich hoffe, die Tipps können Sie vor folgeschweren teuren Entscheidungen bewahren.

Mit freundlichen Grüßen
Der Unbekannte

Fazit

Die letzte E-Mail vom Unbekannten war schon sehr umfangreich und hilfreich. Das Thema Testautomatisierung ist sowohl aus der technischen als auch prozessualen Perspektive kein Selbstläufer ohne Aufwand. Die Diskussion hat mir ermöglicht, neue Ideen und Denkanstöße für eine erfolgreiche Testautomatisierung mitzunehmen. Meine Woche der Testautomatisierung werde ich in einer E-Mail der Kategorie „Lessons learned“ abschließen.

Absender: *tester@gmail.com*
Empfänger: *unbekannt@gmx.net*
Datum: *07.08.2013 17:05*
Betreff: *Re: Re: Re: Re: Re: Testautomatisierung*
Nachricht:
Hallo Unbekannter,

Vielen Dank für Ihre wertvolle Zeit und die Tipps der letzten Tage.
Ohne die intensive Diskussion wäre ich das Thema Testautomatisierung sicherlich anders angegangen. Ich möchte gerne zum Abschluss der Woche die wichtigsten Punkte nochmals zusammenfassen:
Testautomatisierung ist ein Zusammenspiel

aus Entwicklern und Testern. Das Zusammenspiel der individuellen Stärken (Fokus der Entwickler auf Code und der Tester auf Fachlichkeit) sorgt für ein nachhaltiges Fundament und damit besseren Code in Tests und Anwendung.

Der zweite Punkt ist, dass Testautomatisierung mehr als Unit-Testing ist. Eine gute Testautomatisierung setzt, wie beim Team, das Zusammenspiel von mehren Testmethoden voraus. Die richtige Auswahl und Priorisierung schafft aus der Prozessperspektive das Fundament für die gesamten Anstrengungen und entscheidet über langfristigen Erfolg oder Misserfolg.

Als dritten wichtigen Punkt nehme ich mit, dass für Legacy Software („historische Software") andere Spielregeln gelten. Es ist hier soft sinnvoll, dass bei der Test Automation-Pyramide mit den oberen Teststufen begonnen wird. Ziel ist hier, Aufwand und Nutzen in ein wirtschaftliches Verhältnis zu setzen. Unit-Tests können hier einen nicht zu rechtfertigenden Aufwand bedeuten. Langfristig sollte hier der Anwendungscode für die Testautomatisierung optimiert werden.

Zu guter Letzt nehme ich mit: Eine nachhaltige Testautomatisierung bindet Aufwand auf allen Ebenen. Testautomatisierung macht sich nicht von allein, auch wenn Werkzeuge ein anderes Gefühl vermitteln. Planung, Wissen und Menschen sind extrem erfolgskritisch und können nicht durch Werkzeuge ersetzt werden. Werkzeuge können nur unterstützen, aber nicht testen.

Vielen Dank und ein schönes Wochenende wünscht
Ihr Tester

Referenzen

[Art] The Art of Unit Testing, siehe http://artofunittesting.com/.

[Cri08] L. Crispin, J. Gregory, Agile Testing: A Practical Guide for Testers and Agile Teams, Addison Wesley, 2008.

[Mic] Engineering Software for Accessibility eBook, siehe http://download.microsoft.com/download/5/0/1/501FF941-E93D-423F-868B-C7BB2EC08C56/engineering_for_accessibility_eBook.pdf.

[Mou] The Forgotten Layer of the Test Automation Pyramid, siehe http://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid.

[Msd] Isolating Code Under Test with Microsoft Fakes, siehe http://msdn.microsoft.com/en-us/library/hh549175.aspx.

[Typ] Typemock Isolator, siehe http://www.typemock.com/isolator-product-page.