S&N INVENT

Automatisierung von UI-Tests in SCRUM

Welche Prozesse, Tools und Praktiken passen zu meinen Bedürfnissen?

Johann Neufeld

28.04.2021

Inhaltsverzeichnis

1 Motivation und Notwendigkeit von automatisierten Tests

In der agilen Softwareentwicklung wachsen die Anforderungen an die Software sehr schnell. Das Testing muss an dieser Stelle mithalten. Oftmals ist die Qualitätssicherung der Flaschenhals, wenn es um eine schnelle Auslieferung der Software geht. In vielen Projekten werden die Implementierungen erst spät im Sprint fertig, sodass wenig Zeit für deren Überprüfung bleibt. Um weiterhin schnell Entwickeln und Ausliefern zu können, muss das Testing schneller, zuverlässiger und reproduzierbarer stattfinden.

In der Praxis findet man immer wieder sehr viele manuelle Tests, wenige Integrationstests und mit etwas Glück zumindest einige Unittests. Dieser Zustand wird auch „ice cream cone antipattern“ genannt. Das folgende Bild verdeutlicht den Namen und die Verteilung der Tests.

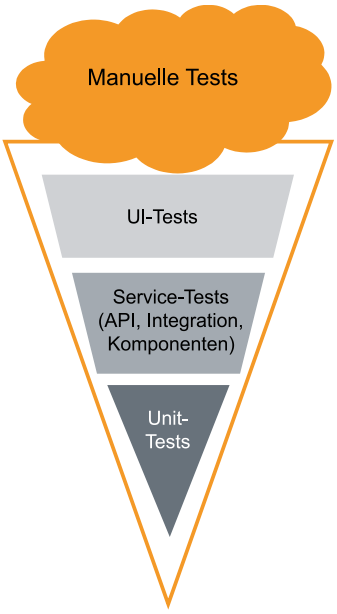

Abbildung 1: Ice Cream Cone Anti Pattern

Das manuelle Testen macht hier den Hauptbestandteil der Qualitätssicherung aus, obwohl es am längsten dauert und somit auch am meisten kostet. Um die Kosten zu verringern, bietet es sich an, die manuellen Testfälle auf ihre Automatisierbarkeit zu analysieren und die Testpyramide neu zu formen. Da die automatisierten UI-Tests zum Beispiel auch nachts ausgeführt werden können und in der Regel weniger Zeit brauchen, können sie Releasezyklen verkürzen und somit die Time-to-Market der Software(-features) verbessern.

Ein weiterer Nutzen der Testautomatisierung ist die Sicherstellung der Kompatibilität der eigenen Software zu sich ändernden Fremdkomponenten. Die meisten Programme nutzen Services, wie zum Beispiel ein Authentifizierungs-Service zur Authentifizierung der Nutzer. Solche Services werden regelmäßig aktualisiert, sodass hier sichergestellt werden muss, dass die eigene Software weiterhin kompatibel mit der neuen Version ist. Solche wiederkehrenden Tests bieten ein hohes Einsparpotenzial, da automatisierte Tests hier wiederkehrend ihren Nutzen zeigen.

Die Automatisierung von Testfällen ist mit Kosten verbunden und dem Risiko, dass Fehler trotzdem zu spät erkannt werden. Diese Risiken lassen sich durch eine geschickte Auswahl geeigneter Methoden und Tools reduzieren.

Eine performante Testautomatisierung baut auf folgenden Säulen:

- Maßgeschneiderte Prozesse
- Bedarfsgerechte Testabdeckung
- Verwendung von Patterns, Frameworks und Bibliotheken
- Aufbau und Nutzung einer Testinfrastruktur

2 Maßgeschneiderte Prozesse

Prozesse spielen für die Testautomatisierung eine wichtige Rolle. Passen sie nicht auf die Fertigkeiten der Mitarbeiter, oder werden sie nicht gelebt, wird viel Potenzial auf der Strecke gelassen. Die Prozesse zur Definition, Implementierung und Verifizierung der Tests sollten auf das Können und die Verfügbarkeit der Teammitglieder zugeschnitten sein. Für die zeitliche Einplanung der Aktivität gibt es folgende Möglichkeiten:

1. Automatisierte Tests werden *nach* Fertigstellung der Anforderung umgesetzt.

   Während der Umsetzung des Features wird dieses manuell getestet und die Tests dokumentiert. Die dokumentierten Tests können anschließend automatisiert werden. Das hat den Vorteil, dass die automatisierten Tests für die Anforderung nicht mehrfach an den Entwicklungsstand angepasst werden müssen. Nachteilig ist, dass während der Entwicklung des Features einige Tests mehrfach manuell ausgeführt werden müssen. Diese Option eignet sich für Projekte, bei denen das Team keine große Erfahrung mit UI-Tests hat und es keinen dedizierten Testautomatisierer gibt oder ein solcher immer nur für kurze Zeitabschnitte verfügbar ist.

2. Automatisierte Tests werden *während* der Umsetzung der Anforderung umgesetzt

   Ein in vielen Projekten genutzter Prozess ist, dass während der Implementierung eines Features, die automatisierten Tests ebenfalls umgesetzt werden. Hier bietet es sich an, die Implementierung der Tests bereits in der Anforderung festzuhalten. Somit brauchen Design und Implementierung des Testfalls nicht separat verwaltet zu werden. Eine weitere Möglichkeit dies zu schaffen, ist die Testabdeckung in der Definition of Done festzulegen. Das kann aber auch den Nachteil haben, dass Umsetzungen nicht fertig werden, weil der Test noch nicht fertiggestellt ist, obwohl die Umsetzung tadellos funktioniert. Zusätzlich kann es einen dedizierten Tester weiter unter Zeitdruck setzen. Damit das funktionieren kann, sollte eine Strategie verfolgt werden, die es ermöglicht, Tests schnell und einfach umzusetzen. Zudem können die Entwickler bei der Erstellung einer Adapterschicht für die Zugriffe auf die Anwendung, Setup- und Teardown-Methoden und der Umsetzung weiterer Testmittel behilflich sein.

3. Automatisierte Tests werden *vor* der Umsetzung der Anforderung umgesetzt

   Die automatisierten Tests können anhand der Anforderungen entworfen und umgesetzt werden, auch bevor das zu testende System existiert. Die User Stories können dann zum Beispiel in Gherkin geschrieben werden, sodass sie gleichzeitig als Anforderung und Testbeschreibung dienen. Hier muss die Kommunikation zwischen Tester und Entwickler gut funktionieren, da Zugriffe auf Services oder Objekte zu erstellen sind, welche zu dem Zeitpunkt ggf. noch nicht existieren. Da Gherkin eher natürlichsprachlich ist, sollten auch Business Analysten und Product Owner in der Lage sein, solche Beschreibungen ggf. mit Hilfsmitteln zu verfassen. Der Vorteil ist, dass die Tests für die Prüfung der korrekten Arbeitsweise und auch für die Ermittlung des Implementierungsfortschritts genutzt werden können. Wer ganz sicher gehen möchte, kann noch zusätzlich manuell testen. Je umfangreicher die automatisierten Tests sind, umso mehr Vertrauen sollte in der Umsetzung stecken und umso weniger manuelle Tests sollten dann stattfinden.

Neben dem Design und der Implementierung der Testfälle gehören noch weitere Prozesse wie die Verifizierung des Tests, der Aufbau der Testumgebung, die Wartung der Testinfrastruktur als auch die Wartung der Tests und viele weitere zur Testautomatisierung. Diese Tätigkeiten können in den regelmäßig stattfindenden Scrum-Events geplant oder konkretisiert werden. So können im Review zum Beispiel auch die Testfälle geprüft werden. Das hat den Vorteil, dass sichergestellt wird, dass die Tests sich genauso verhalten, wie es das gesamte Team erwartet. Sollten weitere Prüfungen in den Testfällen benötigt werden oder mehr Szenarien getestet werden, dann sollte die Komplexität der Änderung auch in einem Refinement geschätzt und später in einem Planning eingeplant werden. Das Gleiche gilt auch für den Aufbau und die Wartung der Testinfrastruktur. Diese Vorgehensweise kostet natürlich Zeit in den Regelterminen. Es hat aber den Vorteil, dass die Aktivitäten gemäß den Anforderungen des Projektes priorisiert und eingeplant werden. Ein weiterer Vorteil ergibt sich durch die Diskussion über die Anforderung und den daraus entstehenden, möglicherweise auch kreativen Lösungsmöglichkeiten.

Die bereits bestehenden Prozesse sind keineswegs in Stein gemeißelt, sondern können in den regelmäßig stattfindenden Retros analysiert und verbessert werden. Auch der Reifegrad der Qualitätssicherungsprozesse kann hier analysiert und Tätigkeiten zur Erreichung der nächsthöheren Stufen abgeleitet werden. Diese abgeleiteten Tätigkeiten können dann wiederum in den folgenden Sprints eingeplant werden, sodass ein fortwährender Verbesserungsprozess gewährleistet ist.

3 Bedarfsgerechte Testabdeckung

Die Abdeckung von Anforderungen oder sogar von Code durch die automatisierten Tests ist von zentraler Bedeutung und spielt für die Qualitätssicherung eine große Rolle. Mit automatisierten UI- oder Schnittstellentests lassen sich neben funktionalen noch weitere nicht-funktionale Anforderungen abdecken. So kann auch die Einhaltung von Last- und Performance-Anforderungen oder auch Security- und Recovery-Vorschriften geprüft werden. Ob und welche Anforderungen automatisiert getestet werden, sollte schriftlich festgelegt werden. Für Security-Tests bieten sich sogenannte Evil-User-Stories an, bei denen das Verhalten des Systems bei böswilligem Verhalten des Anwenders beschrieben ist.

Ist die Abdeckung sehr gering, so ist das Risiko für Regressionen hoch. Bei einer hohen Testabdeckung reduziert sich der Aufwand für manuelle Tests sowohl für Releases als auch für den Sprint. Genauso reduziert sich die Wahrscheinlichkeit, dass neue Bugs durch Weiterentwicklungen eingeführt werden, sodass sich auch die Gesamtkosten reduzieren. Wie die folgende Abbildung verdeutlicht, werden die Kosten größer, je höher die Tests in der Testpyramide angesiedelt sind.

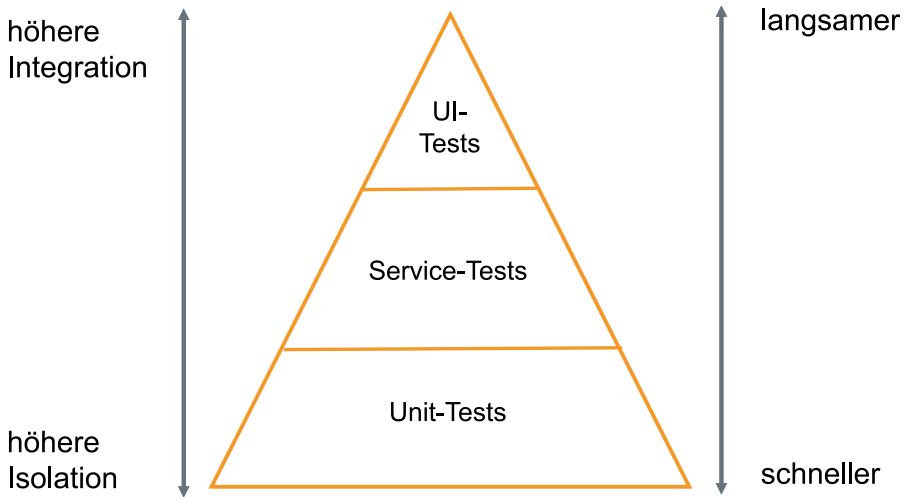

Abbildung 2: Testpyramide

Eine zu hohe Testabdeckung durch UI- Tests ist also aufgrund ihrer Geschwindigkeit und Kosten nicht erstrebenswert. Da sie die Spitze der Testpyramide darstellen, sollten sie auch nicht jeden Grenzfall abdecken, sondern sich eher auf die Hauptanwendungszwecke bzw. User Stories beschränken. Damit UI-Tests und Service-Tests nicht das Gleiche prüfen und somit unnötig Testzeit und -Ressourcen verbrauchen, sollte genau festgelegt werden, was in welcher Schicht geprüft wird. Dies könnte zum Beispiel im Refinement der jeweiligen Story oder auch in Sonderrefinements zum Thema „automatisiertes Testen“ abgestimmt werden.

Eine Messung der Code-Abdeckung ist in der Regel mit weiterer Werkzeugunterstützung und Eingriffen in das Testobjekt möglich, erfordert jedoch etwas Aufwand. Für eine hohe Abdeckung des Codes sollte auf effizientere Werkzeuge wie Unit- oder Integrationstests zurückgegriffen werden.

Die Testabdeckung der UI-Tests kann auf mehrere Arten ermittelt werden.

Wird *Behaviour Driven Development*, wie in Kapitel 2.3 kurz erläutert, als Vorgehensmodell angewendet, so gibt es für jede Anforderung jeweils mindestens einen Test, sodass die Testabdeckung hervorragend ist. Für die grafische Aufbereitung der Abdeckung bieten manche Testautomatisierungswerkzeuge wie zum Beispiel Serenity Berichte an, welche Auskunft über die Testergebnisse sowie über die Abdeckung der Anforderung geben.

Wird ein *Testmanagementwerkzeug* wie zum Beispiel XRAY for JIRA eingesetzt, so kann auch dieses Auskunft über den Teststatus und die Testabdeckung geben. Voraussetzung dafür ist, dass die Testfälle mit ihren Anforderungen verknüpft sind und die Testergebnisse der automatisierten Tests dem jeweiligen Test vorliegen. Ist dies gegeben, so lassen sich Berichte komfortabel generieren und entsprechende Maßnahmen ableiten.

Sind die *Tests lediglich in einem Wiki*, wie Confluence, gepflegt, so lässt sich die Testabdeckung nur mit erheblichem manuellem Aufwand ermitteln. Auch hier muss die Verfolgbarkeit zwischen Anforderungen und Testfällen gegeben sein, bzw. hergestellt werden. Mittels geschickt gesetzter Meta-Informationen und Makro-Kombinationen lassen sich die Verknüpfungen dann auswerten, sodass ermittelt werden kann, welche Anforderungen abgedeckt sind und welche (noch) nicht.

4 Verwendung von Patterns, Bibliotheken und Frameworks

Jede Methodik für die Umsetzung von automatisierten Testfällen hat ihre Vor- und Nachteile. Während die Nutzung von Record- und Replay-Tools schnell gute Ergebnisse liefern, sind die generierten Testskripte in der Regel schwer wartbar und schlecht übertragbar. Die Programmierung von Skripten oder Testschritten ist anfangs mit einer höheren Investition verbunden, kann langfristig aber einige Vorteile haben. Einzelne Testschritte sind schnell und einfach implementiert. Sind diese Schritte in übersichtlichen Strukturen abgelegt, sodass sie einfach zu finden und wiederverwendbar sind, so lassen sich auch schnell komplexe Tests mit vielen Testschritten umsetzen. Der Schlüssel für eine schnelle Testfallumsetzung und somit einer erfolgreichen Testautomatisierung liegt also in ihrer Struktur.

Auf der UI-Testebene eignet sich die Anwendung des PageObject-Patterns, wo für jede Seite oder auch für jede Komponente eine Abstraktionsschicht erstellt wird. Dieses Pattern führt zu stabileren und wartbaren Tests, was eine der größten Herausforderungen in der Testautomatisierung ist[1]. PageObjects bilden eine Abstraktionsschicht für alle technischen Zugriffe. Hier sind die Zugriffe auf die Inhalte, wie zum Beispiel Button und Labels, definiert. Ein Vorteil dieses Patterns ist die hohe Wartbarkeit. Da der technische Zugriff nicht im Testskript, sondern im PageObject definiert ist, muss bei einer Änderung des Elements nur das PageObject, anstatt vieler Testskripte, angepasst werden, wie das folgende Bild verdeutlicht.

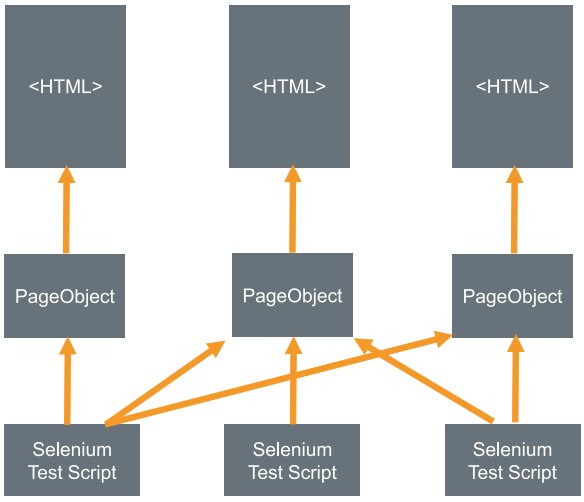

Abbildung 3: Page Object Pattern

Ein weiteres Prinzip für die Sicherheit der Tests ist das „Fail-Gracefully“-Prinzip[2]. Wenn ein Testfall fehlschlägt, dann sollte er die SUT in einem brauchbaren Zustand verlassen. Der Test sollte auch direkt wieder ausführbar sein. Um das zu ermöglichen, werden geeignete Setup- und Tear-Down Routinen benötigt. Diese können für jeden Testfall, für jede Testsuite

[1] Vgl. (Perforce)
[2] Vgl. (Testautomation Patterns, 2021)

oder sogar für den gesamten Testlauf erstellt werden. Mit diesen Methoden sollte der jeweilige Test einen testbaren Zustand herstellen, ihn testen und hinter sich wieder aufräumen.

Während bei Behaviour Driven Development die automatisierten Tests sehr gut nachvollziehbar sind, so sind sie es nicht immer bei den anderen Entwicklungsvorgehensmodellen. Um die Nachvollziehbarkeit zu gewährleisten, bieten sich Dokumentationen an. Die Erstellung solcher Dokumentationen kostet zusätzliche Zeit. Mit dem Einsatz geeigneter, notfalls selbstentwickelter Hilfsmittel kann die Erstellung der Dokumentation automatisiert werden.

Mit der Zeit häufen sich in der Testautomatisierungslösung genutzte Zugriffe auf fremde Dateien, die entweder vom zu testenden System generiert oder importiert werden können. Solche Zugriffe auf zum Beispiel CSV-Dateien, werden zudem in der Regel in mehreren Projekten benötigt. So ist es naheliegend, die Zugriffe allgemeingültig in einer eigenen Bibliothek zu sammeln. Das hat den Vorteil, dass die Methoden nur einmal entwickelt werden und zeitgleich für eine Vielzahl an Projekten zur Verfügung stehen können. Zusätzlich können dort auch Zugriffe auf Authentifizierungssysteme und Reportings an Testmanagementsysteme hinterlegt werden.

Für die Testautomatisierung gibt es eine Vielzahl an weit verbreiteten Frameworks wie TestNG, Junit, Selenium, Cucumber, Jbehave, GalenFramework. Diese Frameworks bieten eine Reihe an Vorzügen, wie zum Beispiel eine Reportgenerierung, bereits vorhandene Prüfungsmethoden und definierte Test-Lifecycles. Bevor man das Rad also neu erfindet, bietet es sich an zu prüfen, ob das Problem nicht bereits von anderen gelöst wurde.

5 Aufbau und Nutzung einer Testinfrastruktur

Das automatisierte Testen von Anwendungen auf dem lokalen Rechner stößt schnell an seine Grenzen. Spätestens wenn hunderte Tests jede Nacht oder nach jeder Codeänderung laufen sollen, reichen die Kapazitäten eines einzelnen Rechners nicht mehr aus. Hier stellt sich zunächst die Frage, wann und wo die Tests ausgeführt werden sollen.

Reichen die Kapazitäten eines einzelnen Rechners nicht mehr aus, dann können die Tests auf mehrere Rechner aufgeteilt werden. So können sie zum Beispiel auf den Rechnern der Entwickler des jeweiligen Projektes ausgeführt werden. Sollte auch dies nicht mehr reichen, dann bieten sich zentrale Rechner, im folgenden auch Testagenten genannt, an. Auf sie können die Tests aufgeteilt werden. Diese Testagenten können bei Bedarf dann beliebig erweitert werden, sind aber auch kostspielig.

Sollen die Tests nur sporadisch laufen, dann ist die Anforderung an die Testkapazität gering. Der Trend geht jedoch dahin, dass die Tests entweder jede Nacht oder sogar nach jedem Kompilieren der zu testenden Software die Qualität des Produktes testen. Je häufiger getestet werden soll, umso höher sind also die Anforderung an die Testinfrastruktur.

Ein nächtlicher Test hat den Vorteil, dass hier das zu testende System intensiv getestet werden und die benötigte Zeit gut auf die vorhandenen Testkapazitäten aufgeteilt werden kann. Nachteilig ist die relativ späte Rückmeldung über mögliche Regressionen an die Entwickler. Wenn die Tests direkt nach dem Bau des zu testenden Systems ausgeführt werden, dann erfolgt eine solche Rückmeldung deutlich schneller. Hierbei lässt sich die Verwendung der Testkapazitäten allerdings schlechter steuern, sodass die Tests gegebenenfalls, aufgrund von nicht vorhandener freier Testkapazität, länger dauern. Ein Kompromiss aus beiden Welten könnte sein, nach jedem Bau nur die wichtigsten Tests auszuführen und die aufwändigeren und zeitintensiveren Tests nächtlich auszuführen[3].

Eine weitere Möglichkeit die Testdauer zu verkürzen ist die Testinfrastruktur in der Cloud zu betreiben und somit besser skalieren zu können. So könnten für die zeitintensiven nächtlichen Seleniumtests, die Selenium Nodes, vervielfacht werden, sodass die Tests auf mehr Nodes gleichzeitig verteilt werden können, wie folgendes Bild verdeutlich.

[3] Vgl. (Hruska, 2020)

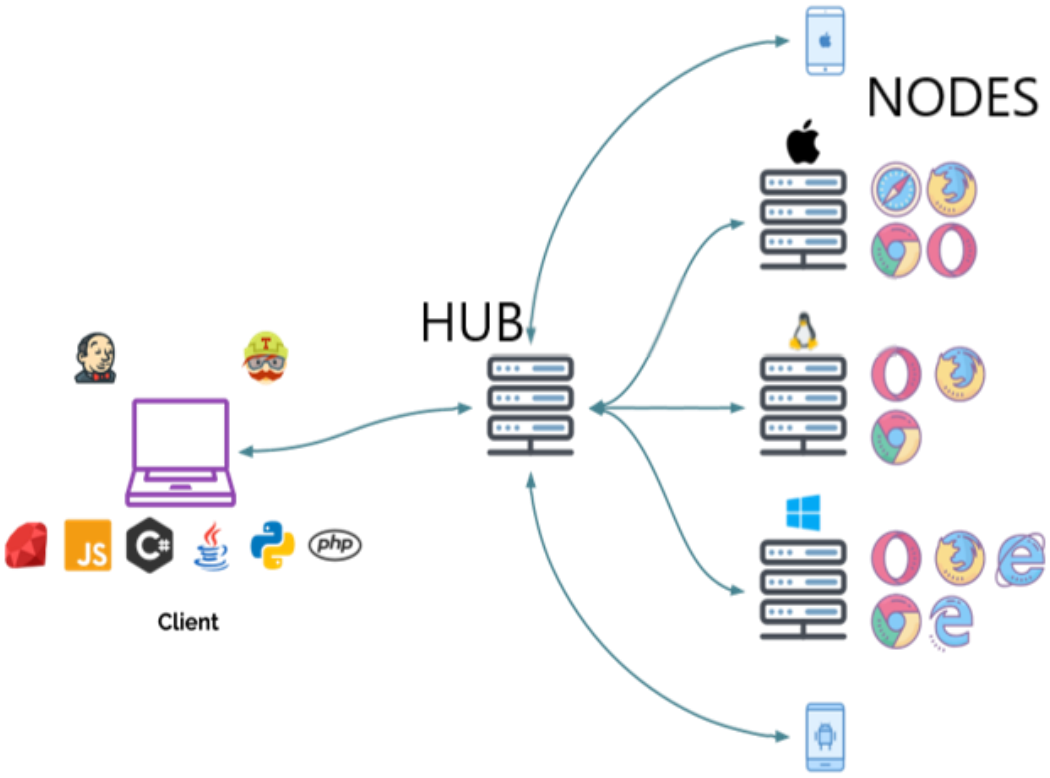

Abbildung 4: Selenium Grid

Durch den gleichzeitigen Test vieler Anwendungsfälle entsteht eine nicht unerhebliche Last auf das zu testende System. Dieses muss der Last standhalten oder mit ihr ebenso skalieren, sodass es sich anbietet, in einem solchen Szenario auch das zu testende System in einer Cloud zu betreiben.

6 Fazit

Die Testautomatisierung hat viele Stellschrauben und kann gut auf die eigenen projektspezifischen Anforderungen angepasst werden. Die richtige Einstellung dieser Stellschrauben ist nicht immer einfach, sie kann aber erfolgsentscheidend sein. Sie muss sowohl zum Team als auch zum Entwicklungsprozess passen. Da sich sowohl das Team als auch der Entwicklungsprozess im Laufe eines Softwareentwicklungsprojektes ändern können, sollte dann auch die Testautomatisierung dementsprechend angepasst werden.

Autor

Johann Neufeld

M Sc. Wirtschaftsinformatik

Consultant
QA Services

Über S&N Invent

S&N Invent ist ein bundesweit tätiges IT-Unternehmen mit einem umfassenden Leistungsportfolio über die gesamte Wertschöpfungskette des IT-Lifecycles. Wir entwickeln gemeinsam mit unseren Kunden Lösungen, setzen Projekte um und schaffen damit digitale Mehrwerte.

Das Leistungsspektrum reicht von klassischen Mainframe-Architekturen über moderne Java/JEE Web- und Portalarchitekturen bis hin zu neuesten Technologien im Cloud- und Mobile-Bereich. Agile Projekte mit hohem Qualitätsanspruch, gewährleistet durch modernes Continuous Quality Management, sind für uns in zahlreichen Projekten gelebter Standard.

Die S&N Invent GmbH ist ein Unternehmen der S&N Group. Insgesamt sind in den Gesellschaften der S&N Group ca. 380 feste Mitarbeiterinnen und Mitarbeiter an acht Standorten in Deutschland sowie dem Nearshore-Standort Budapest beschäftigt.

Damit können wir unsere Kunden mit umfassender Kompetenz und regionaler Nähe bestens betreuen. Gleichzeitig sind wir so in der Lage, große und komplexe Projekte in Time und Budget erfolgreich umsetzen zu können.

Abbildungsverzeichnis

Literaturverzeichnis

Hruska, W. (05. 12 2020). *Contiuous Testing*. Von https://continuoustesting.dev/2020/12/05/the-future-of-test-automation-must-be-intelligent/?utm_source=Coding_Jag&utm_medium=web&utm_campaign=Coding_jag_15/ abgerufen

Perforce, P. b. (kein Datum). *State of test automation 2020.*

Testautomation Patterns. (05. 01 2021). Von https://testautomationpatterns.org/wiki/index.php/FAIL_GRACEFULLY) abgerufen