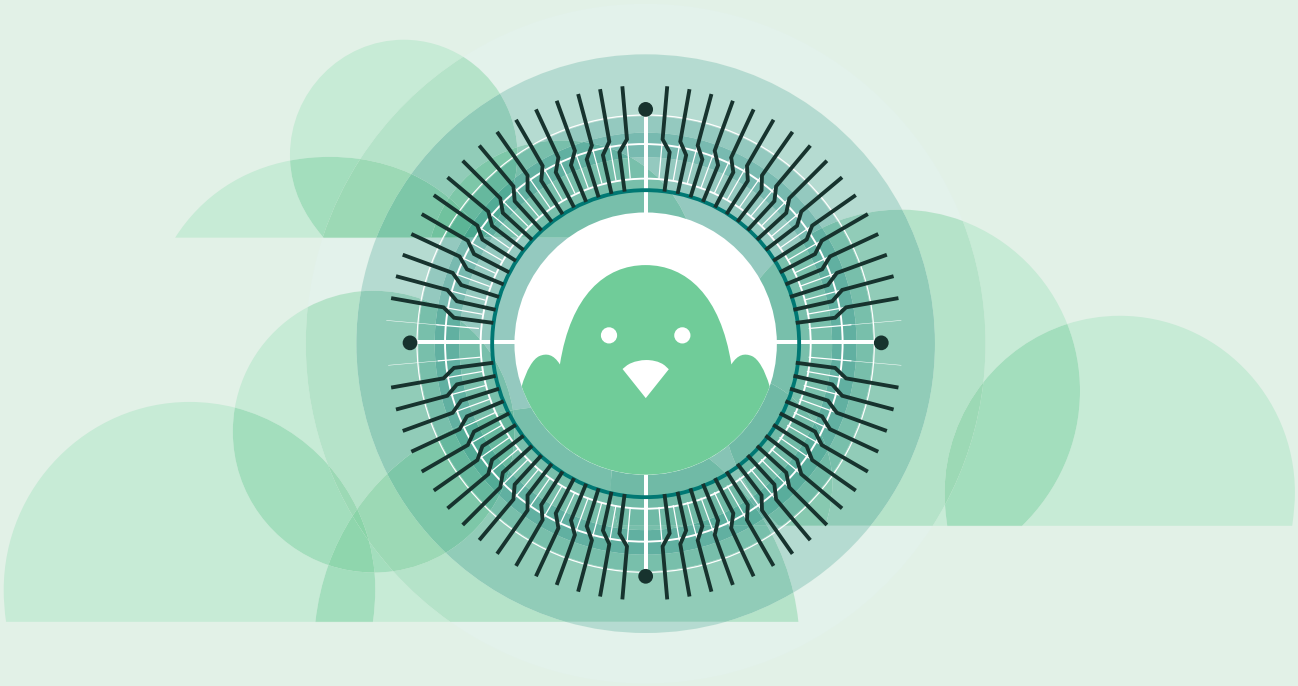

Testbirds

Testautomatisierung von UI-Tests mit Selenium und Appium

AUTOR:
Georg Hansbauer

Georg Hansbauer ist Geschäftsführender Gesellschafter der Testbirds GmbH mit Sitz in München. Vor seiner Zeit bei Testbirds sammelte er umfassende Erfahrung in diversen Großkonzernen. Er leitete dort u. a. Projekte im Feld des Enterprise Testing – von automatisierten Tests kompletter IT-Service-Desks bis hin zu Lasttests mit 15.000 simulierten Benutzern. Bei Testbirds verantwortet Hansbauer die Produktentwicklung sowie den Bereich Finanzen.

INHALTSVERZEICHNIS

1. Neue Plattformen und Prozesse erfordern ein Umdenken in der Qualitätssicherung

Vor nur wenigen Jahren wurde die eigene Website lediglich auf Desktopgeräten sowie einer Handvoll Browsern getestet. Mittlerweile steht die Qualitätssicherung wegen der enormen Menge an Mobilgeräten sowie der zunehmenden Verbreitung „smarter" Gegenstände vor gänzlich neuen Herausforderungen. Testautomatisierung entwickelt sich dabei zu einem wichtigen Qualitätsfaktor, jedoch steht vor allem im Bereich mobiler Applikationen ein Großteil der Unternehmen noch ganz am Anfang.

Dieses Whitepaper vermittelt einen Einstieg in das Thema Testautomatisierung und beleuchtet anhand praktischer Anwendungsfälle Herausforderungen sowie konkrete Einsatzmöglichkeiten. Es bietet wichtige Informationen für Entscheider, die über den Einsatz von Testautomatisierung nachdenken und dient gleichzeitig als Ressource für Entwickler und Anwender, die tiefer in das Thema einsteigen möchten.

Das klassische Wasserfall-Modell mit seinen starren Entwicklungs- und Testphasen wird zunehmend durch agile Methoden[1] ersetzt. Kurze Iterationen, oft Sprints genannt, sorgen für häufige und ebenso kurze Testphasen. Durch den Ansatz der Continuous Integration[4] wird die Software in kleinen Zyklen ständig auf den neuesten Stand gebracht. Parallel zur Entwicklung findet auch das Testen regelmäßig im Rahmen von kurzen Zeitabständen statt (sog. Continuous Testing) und stellt interne sowie externe Teams vor neue Herausforderungen.

Zudem sollten sowohl bestehende Komponenten als auch neue Bestandteile kontinuierlich getestet werden, was die Kosten schnell in die Höhe treiben kann. Eine Mischung aus manuellen sowie automatisierten Tests ist daher oft empfehlenswert. Während neue Features einfach und schnell manuell geprüft werden können, verspricht die Automatisierung von Tests der Kernfunktionen eine ressourcenschonende und damit langfristig kostengünstigere Option. Wiederkehrende Regressionstests[7] sind damit prädestiniert für die Testautomatisierung.

Eine Voraussetzung für die agile Entwicklung ist zudem die reibungslose Zusammenarbeit zwischen Software-Entwicklern (Development) und IT-Betrieb (Operations), was unter dem Schlagwort DevOps zusammengefasst wird. Ein zentraler Gedanke von DevOps ist die Automatisierung von Vorgängen bei Deployment und Tests, um die Zusammenarbeit einfacher zu gestalten.

Testautomatisierung entwickelt sich somit zu einem essentiellen Bestandteil der agilen Software-Entwicklung. Doch welche konkreten Einsatzmöglichkeiten gibt es in der praktischen Umsetzung?

In diesem Whitepaper werden Ansätze für folgende Fragen aufgezeigt:

1. Wie können Entwickler manuelle Testaufwände durch UI-Automatisierung mit Selenium und Appium reduzieren?
2. Wie sollten QA Manager automatisierte Tests in den bestehenden Entwicklungszyklus integrieren?
3. Wie kann eine geeignete Testinfrastruktur dafür aussehen?

2. Testautomatisierung

2.1. Einsatzgebiete

Zu Beginn stellt sich häufig die Frage, ob manuell oder automatisiert getestet werden sollte. Zunächst ist es nötig zu entscheiden, wann die manuelle Variante wirtschaftlicher ist. Bei straffen Projektdeadlines ist kaum Zeit zum Testen vorhanden, was Testautomatisierung als die ideale Lösung erscheinen lässt. Dennoch ist es kein Allheilmittel. Im Gegensatz zu manuellen Tests ist die Einstiegshürde für die Automatisierung vergleichsweise hoch. Die Erstellung der entsprechenden Testfälle ist aufwändig und damit kostenintensiv. Hinzu kommt, dass die nötige Expertise im Team vorhanden sein oder extern hinzugezogen werden muss. Nach dem initialen Setup besteht die Arbeit dann vor allem in der Wartung sowie Erweiterung der bereits vorhandenen Skripte, was wiederum vergleichsweise günstig ist. Automatisierte Tests können damit sehr gut als fester Bestandteil des Entwicklungsprozesses integriert werden.

Ein weiterer Vorteil von Testautomatisierung ist die Reduktion menschlicher Fehler. Diese Methode eignet sich vor allem für kritische Kernanwendungen wie Login, Registrierung, Buchungsprozess und Kaufvorgang, da sich diese Komponenten in der Regel nur geringfügig ändern.

Manuelle Tests bieten hingegen die Möglichkeit, subjektive Eindrücke von Benutzern, wie z.B. hinsichtlich Usability, harmonischer Gestaltung oder sinnvoller Strukturierung, zu beurteilen. Diese Informationen sind beim Anwendungsdesign und vor allem in der frühen Entwicklungsphase unerlässlich und sollten auch bei der finalen Produktversion nicht fehlen.

Insbesondere im agilen Umfeld mit kurzen Entwicklungszyklen ist schnelles Feedback für die Entwickler durch automatisierte Tests bei der Fehlersuche enorm hilfreich, da die Tests unmittelbar nach Codeänderungen ausgeführt werden. Mithilfe von Continuous Integration-Systemen[3] kann dieser Prozess effizient automatisiert werden. Die Ergebnisse werden daraufhin allen Beteiligten transparent zur Verfügung gestellt.

Grundsätzlich ist im Einzelfall zu prüfen, welche Testart sowohl praktikabel als auch wirtschaftlich ist. Je öfter getestet wird, desto eher kann es sich lohnen, ein Testskript zu entwerfen und dieses bei Änderungen anzupassen. In der Regel sind beide Testarten sinnvoll. Es stellt sich jedoch die Frage, welche Methode für welche Komponenten besser geeignet ist.

Manuell

- Tests werden nur selten durchgeführt
- Software ändert sich stark und häufig (hoher Wartungsaufwand des Testskriptes)
- Frühes Entwicklungsstadium der Software
- Fehlendes Know-How hinsichtlich Automatisierung
- Menschliche Komponente soll getestet werden (Verhalten durch Testfälle nicht zu simulieren)
  - Explorative Tests
  - Usability-Tests

Automatisiert

- Hoher Bedarf an Regressionstests
- Wiederkehrend hoher Zeitaufwand bei manuellem Testen
- Kurze Release-Zyklen
- Systemumgebung soll überwacht werden (Live Monitoring)
- Große Kosteneinsparungen bei der Testausführung
- Skalierung, unter anderem auf zusätzliche Testumgebungen
- Einbindung von Tests direkt in den Entwicklungsprozess (Continuous Integration)

Abbildung 1: Einsatzszenarien für manuelle und automatisierte Tests

2.2. Beispiel APIs: Selenium und Appium

Für das Erstellen automatisierter Tests sind spezielle Frameworks notwendig. Im Folgenden werden die beiden wichtigsten Vertreter vorgestellt: Selenium[8] für browserbasierte Tests auf Desktopumgebungen sowie Appium[2] für mobile Browser und Applikationen.

Einführung Selenium

Selenium ist ein Framework, um Tests von Webseiten und mobilen Apps zu automatisieren. Es simuliert dabei Nutzeraktionen und steuert somit den Browser fern. Dies beinhaltet Aktionen wie das Aufrufen einer Seite, Scrollen, das Klicken einzelner Elemente oder die Eingabe in Textfelder.

Damit bietet Selenium zahlreiche Möglichkeiten den Ablauf eines Webseitentests zu automatisieren. Diese Kommandos können in verschiedenen Programmiersprachen verwendet werden. So entsteht ein Testskript, das die oben genannten Aktionen hintereinander ausführt und Ergebnisse evaluiert, um die Funktionen einer Webseite zu überprüfen.

Ein großer Vorteil von Selenium ist die Kompatibilität mit gängigen Systemen als wichtige Anforderung für Tests auf Desktopgeräten.

Client APIs	Plattformen	Browser
› Java	› Windows	› Chrome
› PHP	› Linux	› Firefox
› Python	› OS X	› Internet Explorer (ab Version 7)
› JavaScript	› Android	› Safari
› Und viele mehr	› iOS	› Und viele mehr

Abbildung 2: Kompatibilität von Selenium

Lokales Testen

Die breite Unterstützung verschiedener Browser sowie Plattformen wird unter anderem durch die Architektur von Selenium ermöglicht:

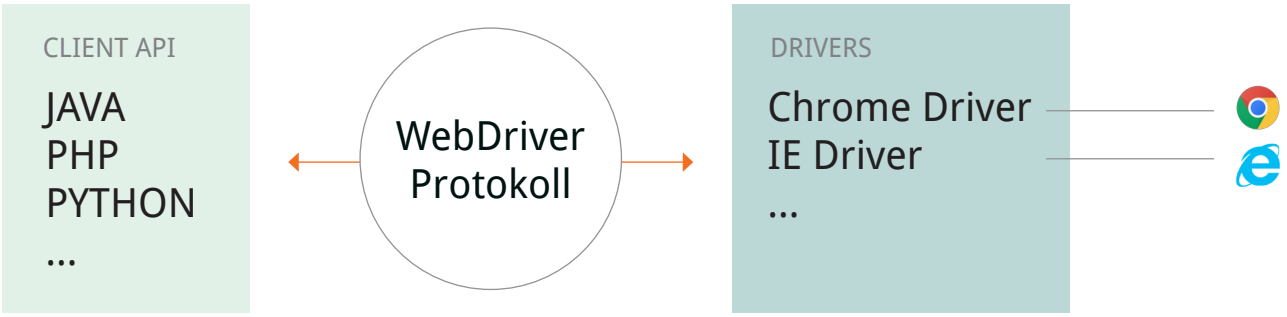

Abbildung 3: Lokales Testen mit Selenium durch Fernsteuerung des eigenen Browsers

Abbildung 3 verdeutlicht die beiden wichtigsten Komponenten von Selenium:

- **Selenium WebDriver**: Ein inzwischen durch den W3C standardisiertes Interface, das Kommandos via HTTP entgegennimmt. Hierfür existieren jeweils browser- und plattformspezifische Implementierungen (ChromeDriver, IE Driver ...), welche auf dem gleichen System ausgeführt werden wie der zu testende Browser. Die jeweilige WebDriver-Implementierung startet den Browser und führt die gewünschten Aktionen darin aus.

- **Selenium Client API**: Die Schnittstellen für die populärsten Programmiersprachen werden von den Testskripten angesprochen. Die Client API übersetzt diese Aufrufe in WebDriver-Kommandos, welche an den WebDriver gesendet und von diesem im Browser ausgeführt werden.

Selenium Grid

Das Selenium Grid erlaubt es, Testskripte auf anderen Systemen als dem eigenen auszuführen. Dabei werden ein Hub und mehrere Nodes zu einem Grid zusammengeschlossen. Dies sieht wie folgt aus:

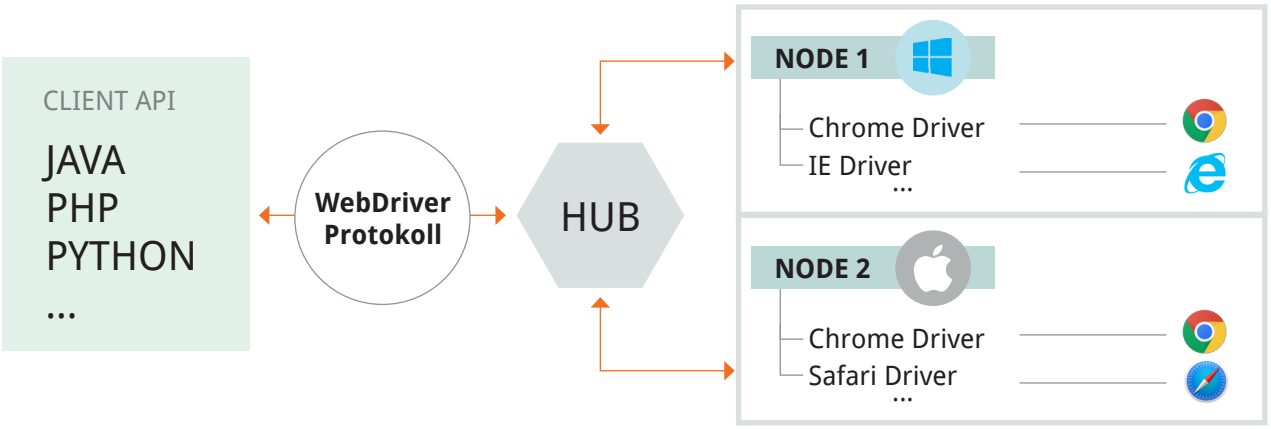

Abbildung 4: Testen im Selenium Grid auf verschiedenen Systemen

Die Nodes befinden sich auf verschiedenen Maschinen mit möglicherweise unterschiedlichen Betriebssystemen (Windows, Linux und OS X). Sie sprechen die dort installierten WebDriver an, die die Kommandos wiederum in den einzelnen Browsern ausführen.

Hub und Nodes werden über das Netzwerk miteinander verbunden. Testskripte kommunizieren in diesem Fall nur mit dem Hub, der wiederum die Anweisungen an den aktuell genutzten Node verteilt. Die Auswahl des Systems und Browsers, auf denen getestet wird, geschieht über die sogenannten DesiredCapabilities (vgl. Kapitel 4, Seite 13).

Das Selenium Grid eignet sich vor allem für fortgeschrittene Anwendungsfälle. Es bietet den Vorteil, andere Maschinen als die intern vorhandenen für das Testen einzusetzen. Dies ist insbesondere in einem Continuous Integration-Prozess von Vorteil, in dem die Tests beispielsweise von Jenkins angestoßen werden.

Appium

Bei Appium handelt es sich um eine Erweiterung von Selenium für das Testen von mobilen Apps und Webseiten auf Android und iOS. Es ermöglicht, Selenium-Tests in mobilen Browsern auszuführen sowie native Apps zu testen. Appium erweitert dazu die Funktionalität des WebDrivers, um auf mobilen Geräten wichtige Aktionen, wie beispielsweise Multitouch oder das Drücken der Hardwaretasten, auszuführen. Appium kann problemlos in ein existierendes Selenium Grid integriert oder als eigenständiges Framework benutzt werden.

2.3. Beispiel Testsuites: Jubula

Eine einfache und zugängliche Alternative zur klassischen Testerstellung mithilfe einer Programmiersprache ist Jubula, ein Open Source-Tool der Eclipse Foundation.

Jubula ermöglicht das Testen von Webseiten u.a. mit Selenium, ohne dass dafür Programmierkenntnisse nötig sind. Dabei werden die Tests in einer graphischen Benutzeroberfläche erstellt und einzelne Testschritte aus einer umfangreichen Bibliothek per Drag-and-Drop ausgewählt und organisiert.

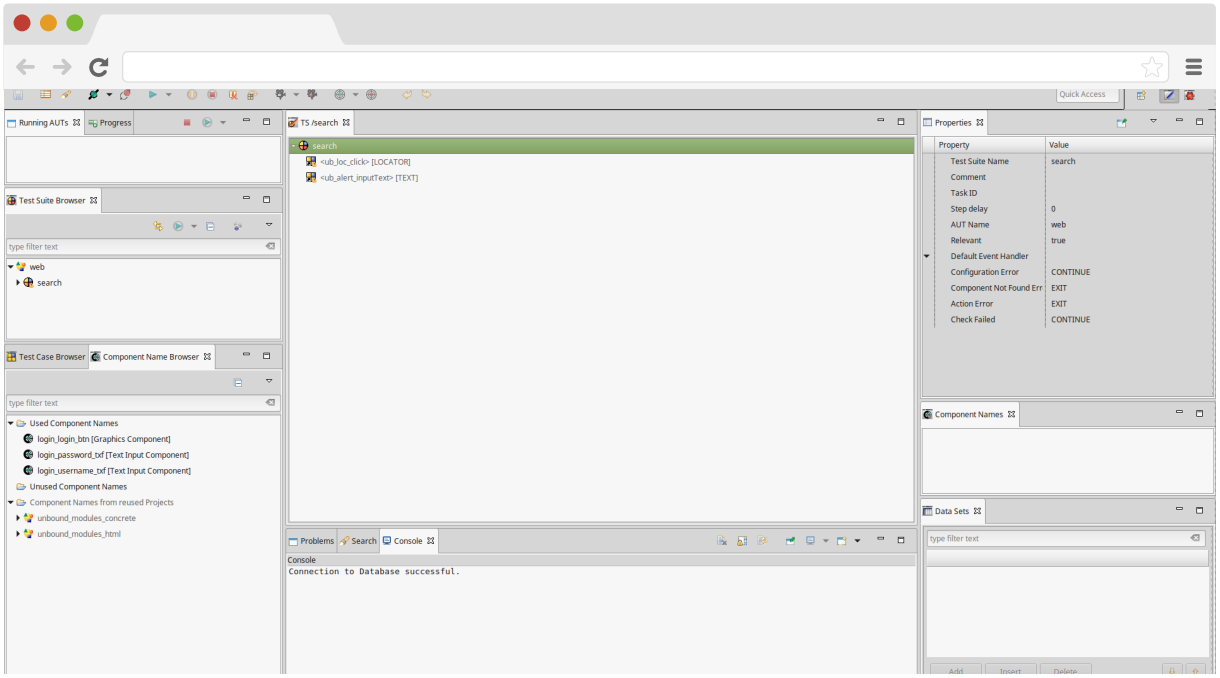

Abbildung 5: Benutzeroberfläche von Jubula

Bei der Testerstellung werden abstrakte Namen für die genutzten GUI-Elemente der Webseite wie z.B. „Benutzername-Eingabefeld“ verwendet. Erst wenn ein funktionaler Prototyp der Webseite vorhanden ist, werden diese mit den tatsächlichen technischen Elementen verknüpft. Daher kann die Erstellung von Tests mit Jubula zunächst unabhängig vom Entwicklungsstand der zu testenden Webseite oder Anwendung erfolgen.

Ebenfalls enthalten ist ein Reporting-Tool, das die Testergebnisse sammelt, ansprechend darstellt und im Fehlerfall mit automatisch generierten Screenshots dokumentiert.

Jubula basiert auf einer Client Server-Architektur, die auch verteilte Tests ermöglicht. Dadurch kann die Testausführung einfach in ein Continuous Integration-System eingebunden werden.

Neben Webseiten können mit Jubula auch Desktopanwendungen getestet werden, z.B. basierend auf den Swing-, SWT- oder JavaFX-Toolkits. Es werden auch verschiedene Plattformen wie Windows und Linux/Unix unterstützt.

3. Testumgebungen für automatisierte Tests

3.1. Herausforderung Plattformvielfalt

Anwendungen müssen sowohl auf Desktopgeräten als auch mobilen Plattformen problemlos funktionieren. Hinzu kommt die Anbindung weiterer Software, wie etwa Java und Flash, die benötigt werden, um die Anwendungen ausführen zu können. Die Kombinationsmöglichkeiten unterschiedlicher Betriebssystem- und Softwarekombinationen sind enorm und dementsprechend kostet das Aufsetzen der entsprechenden Umgebungen viel Zeit und Ressourcen.

Virtuelle Maschinen[12] (VMs) können diesen Aufwand reduzieren. Bei Testumgebungen[10] für automatisierte Tests, die mithilfe von virtuellen Maschinen ausgeführt werden, ist der Konfigurations- und Wartungsaufwand nur geringfügig kleiner als bei physikalischen Umgebungen. Auch für manuelle Tests werden immer häufiger virtuelle Maschinen benötigt, da keine QA-Abteilung mehr in der Lage ist, die Vielfalt des Marktes mit einem eigenen Gerätepool abzubilden.

Besonders groß ist die Plattformvielfalt – und damit auch die Herausforderung für das Testen – bei mobilen Applikationen. Vor allem der Markt für Android-Geräte ist durch die Vielfalt an Herstellern und Betriebssystemversionen enorm fragmentiert. Auch iOS-Apps sollten auf zahlreichen unterschiedlichen Geräten sowie OS-Versionen reibungslos funktionieren.

Dementsprechend müssen Testumgebungen für automatisierte Tests eine ständig wachsende Anzahl an Geräte-, Browser- und Betriebssystemkombinationen abdecken und aufgrund der kurzen Entwicklungszyklen schnell für Tests bereitstehen. Es ist außerdem sinnvoll, zahlreiche Umgebungen parallel testen zu können, um die Dauer der einzelnen Testphasen zu minimieren.

3.2. Cloudtesting mit TestChameleon™

Als Software-as-a-Service-Lösung[9] ermöglicht es TestChameleon™ als Teil der Testbirds Cloud-Lösungen in wenigen Minuten eine Vielzahl an Testumgebungen für manuelle oder automatisierte Softwaretests zu erstellen. Hierzu gehören sowohl desktopbasierte Systeme als auch Mobilgeräte inklusive diverser Softwarekomponenten. Neben der Durchführung von manuellen Tests ermöglicht es TestChameleon™, mithilfe einer entsprechenden API, Testfälle in Selenium oder Appium automatisiert durchzuführen. Im Gegensatz zum Einsatz physikalischer Umgebungen entfallen hierbei Aufwände für das Aufsetzen sowie die Wartung.

Lösungsansatz für die Plattformvielfalt

Über einen Webkonfigurator wird bei TestChameleon™ die Anzahl an virtuellen Maschinen mit den gewünschten Betriebssystemen, Browsern sowie weiteren Komponenten bestimmt. Es kann eine Vielzahl an Softwarepaketen und Plugins, wie Java oder Flash, in verschiedenen Versionen ausgewählt werden.

TestChameleon™ wurde spezifisch für die Herausforderungen der modernen Qualitätssicherung entwickelt. Manuelle Tests, wie beispielsweise das Reproduzieren von Fehlern oder Usability-Problemen, können dabei über das HTML5-Frontend durchgeführt werden.

Zur Unterstützung von Automatisierungsprojekten stehen APIs für Selenium sowie Appium zur Verfügung. Einmal eingebunden wird anschließend die zu testende Umgebung definiert. TestChameleon™ fungiert dabei als Steuerungstool für die automatisierten Tests und kann mit Continuous Integration-Systemen wie Jenkins[1] verbunden werden.

Um ein größtmögliches Maß an Sicherheit zu gewährleisten, werden alle Testumgebungen von Grund auf neu erstellt. Außerdem findet keine Wiederverwendung der zuvor genutzten Systeme statt. Nach dem Test wird die Umgebung schließlich vollständig gelöscht. Eine Anbindung lokaler Netzwerke über eine standardisierte IPSec VPN-Lösung ist ebenfalls möglich.

Übersicht TestChameleon™

› Tool für die Erstellung virtueller Umgebungen als SaaS-Lösung in der Cloud
› Rund 2,5 Millionen Kombinationsmöglichkeiten aus Endgerät, OS und Softwarepaket
› Anbindbarkeit von internen Testservern
› Hohe Skalierbarkeit und individuelle Konfiguration
› Einsetzbar für automatisierte und manuelle Tests
› Serverhosting ausschließlich in Deutschland

Architektur und Technologie

In den meisten Fällen wird TestChameleon™ als Software-as-a-Service-Lösung eingesetzt. Es besteht ebenfalls die Möglichkeit einer On Premise-Installation im firmeneigenen Rechenzentrum.

Tests können entweder manuell über das Webfrontend oder automatisiert per API erstellt werden. Die komplette Verwaltung erfolgt über einen Controller. Das Webinterface fungiert als relativ funktionsarmes Frontend. Zur Erhöhung der Kapazitäten werden lediglich weitere VM Hosts hinzugefügt, deren Allokation mithilfe eines entsprechenden Algorithmus erfolgt. Für On Premise existieren mehrere Treiber für Virtualisierungslösungen wie Libvirt/KVM oder VMware.

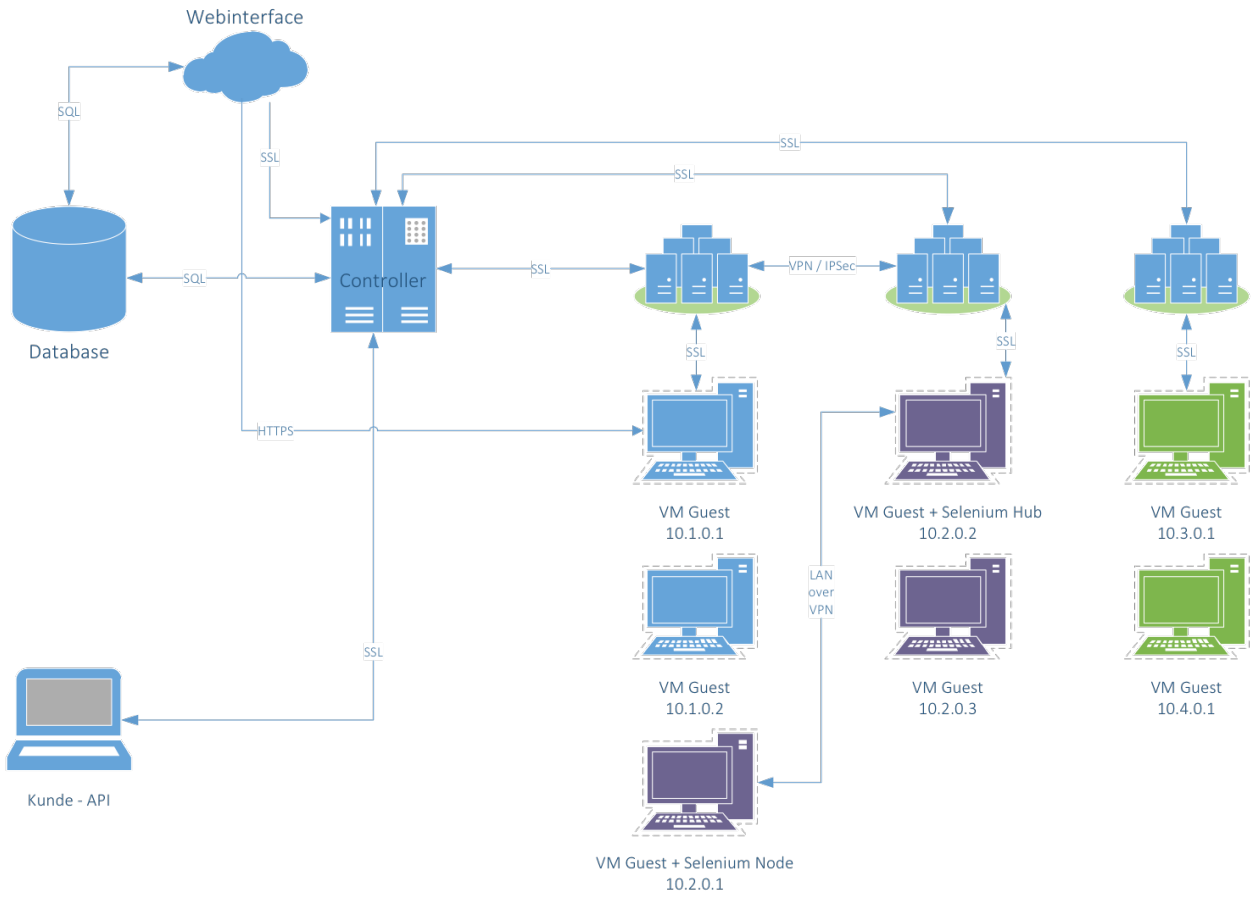

Abbildung 6: TestChameleon™ Architektur

Testablauf und Durchführung

Im Folgenden wird aufgezeigt, wie TestChameleon™ für manuelle Tests eingesetzt werden kann. Mithilfe des Webfrontends werden dabei individuelle VMs erstellt. Im Webinterface kann unter nest.testbirds.com ein Account erstellt werden, mit dem VMs erzeugt und gesteuert werden können.

Manuell wird mit wenigen Klicks eine einzelne, individuell konfigurierte VM erstellt. Hierzu müssen beispielsweise das Betriebssystem, Browser und weitere Software ausgewählt werden (vgl. Abbildung 7).

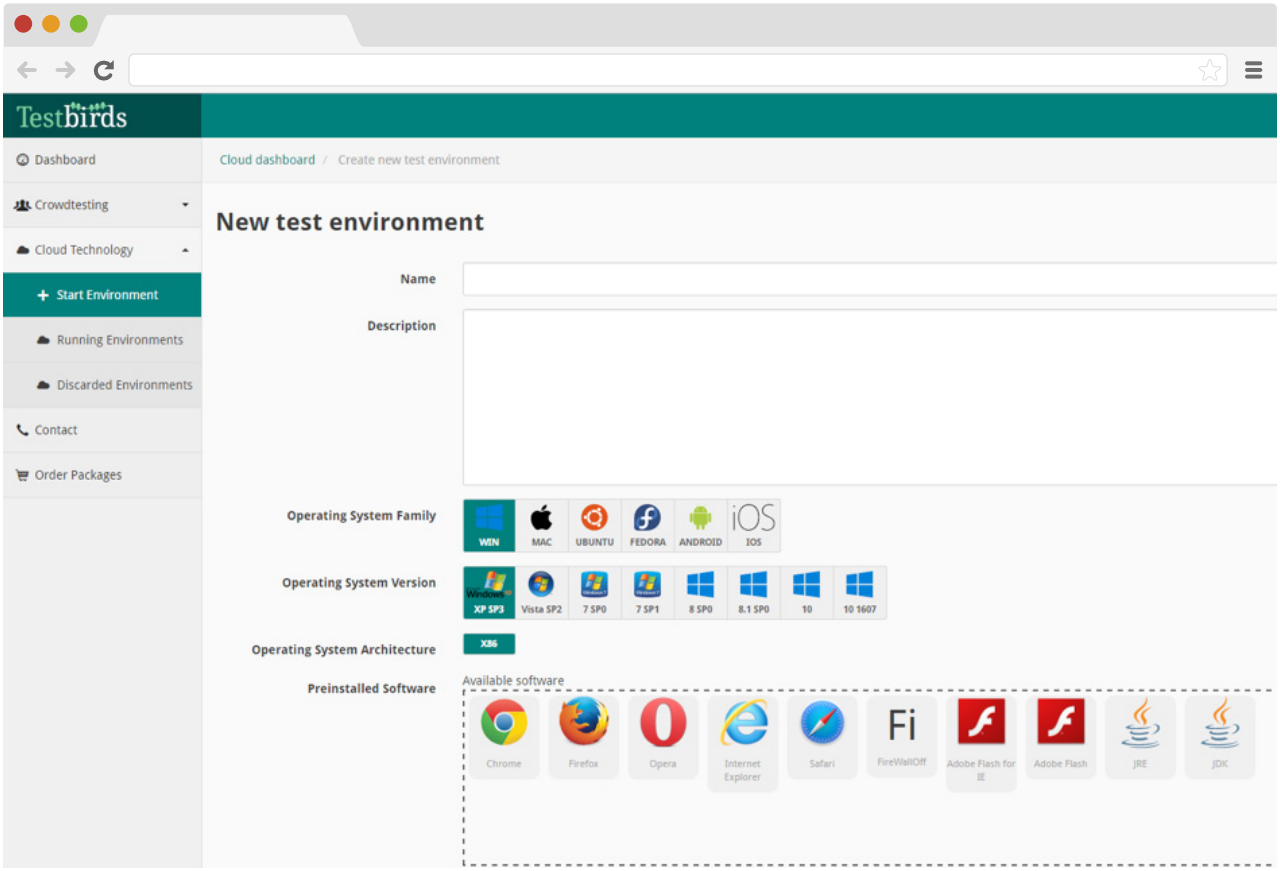

Abbildung 7: Konfigurator zur Erstellung einer Testumgebung

Die Umgebung ist anschließend innerhalb weniger Minuten verfügbar. Der Zugriff erfolgt entweder direkt im Browser via WebVNC (reines JavaScript, ohne Installation von Plugins) oder durch einen externen VNC Client (vgl. Abbildung 8).

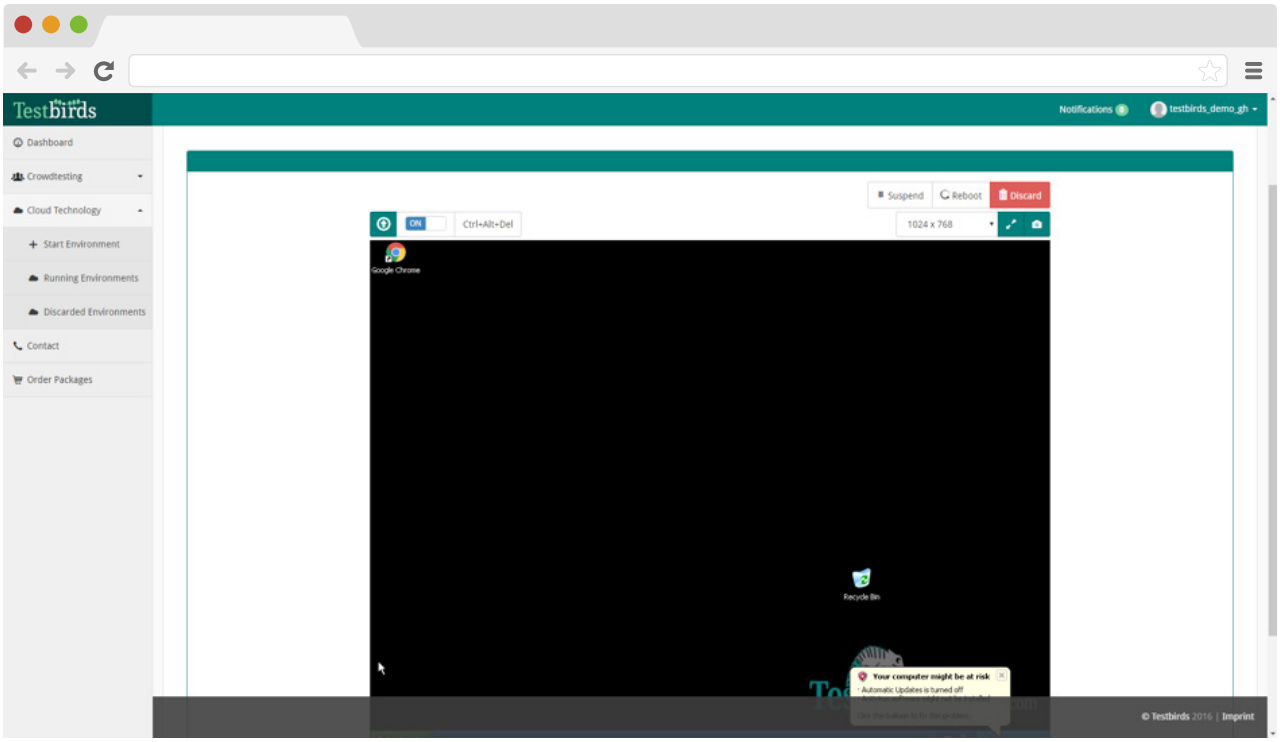

Abbildung 8: Zugriff via WebVNC auf die Testumgebung

Über eine Toolbar ist je nach Betriebssystem und Endgerätetyp eine breite Auswahl an Funktionen zugänglich.

Für Desktopumgebungen sind dies:

› Neustart und Pausieren der Umgebung
› Hochladen von lokalen Dateien
› Teilen der Zwischenablage des Browsers mit der Testumgebung
› Anpassung der Auflösung
› Vollbildansicht
› Speichern eines Screenshots

Abbildung 9: Toolbar für Desktopumgebungen

Für mobile Endgeräte stehen folgende Funktionen zur Verfügung:

› Installation von Apps
› Vollbildansicht
› Speichern eines Screenshots in Originalauflösung
› Rotieren des Endgeräts (Portrait und Landscape)
› Aktivieren einzelner Hardwarebuttons wie z.B. die „Home“-Taste

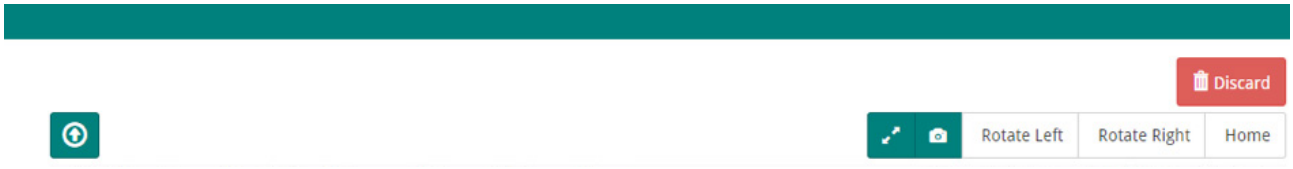

Abbildung 10: Toolbar für mobile Endgeräte

4. Praxisbeispiele

4.1. Lokales Testen mit Selenium

Im Folgenden wird ein konkretes Anwendungsbeispiel für automatisiertes Testen mit Selenium oder Appium erläutert. Nach einer generellen Einführung in automatisiertes Testen mit Selenium wird anschließend aufgezeigt, wie sich Testfälle von einem lokalen Browser auf TestChameleon™ erweitern lassen. Ohne viel an den Testskripten zu verändern, ist es dadurch möglich, die Abdeckung verschiedener Plattformen und Browser zu vervielfachen.

Alle hier dargestellten Codebeispiele finden sich auch auf Github. Dieser Code kann auf einem lokalen Browser getestet werden, ohne über einen dedizierten TestChameleon™ Zugang zu verfügen. Testobjekt ist hierbei eine für Demonstrationszwecke aufgesetzte Webanwendung.

Die folgenden Codeausschnitte zeigen den Aufbau eines Selenium-Testskripts in Java. Diese lassen sich ohne großen Aufwand auch in andere Programmiersprachen übersetzen. Zum Testen ist zunächst ein sogenannter WebDriver nötig. Beim Testen auf einem lokal installierten Chrome-Browser wird dieser wie folgt erstellt:

```
WebDriver driver = new ChromeDriver();
```

Das Objekt `driver` wird nun genutzt um den Browser fernzusteuern. Meist macht es Sinn, im zweiten Schritt eine URL im Browser aufzurufen:

```
driver.get("https://demo.testchameleon.com");
```

Das Ergebnis ist direkt sichtbar: Ein Browserfenster öffnet sich und die angegebene Seite wird geladen. Zu Beginn werden meist bestimmte Elemente auf der zu testenden Seite ausgewählt:

```
WebElement button = driver.findElement(By.id("submit"));
WebElement input = driver.findElement(By.id("username"));
List<WebElement> es = driver.findElements(By.className("user-profile"));
```

Weitere Möglichkeiten, spezifische Elemente auszuwählen, sind unter anderem `By.tagName`, `By.cssSelector` und `By.xpath`.

Mit den dadurch ausgewählten Elementen kann im nächsten Schritt interagiert werden. Beispielsweise sind das Ausführen von Mausklicks sowie Tastatureingaben möglich:

```
button.click();
```

```
input.sendKeys("username");
```

Ein vollständiger Test eines Login-Formulars könnte schließlich wie folgt aussehen:

```
driver.get(URL + "/login.html");
driver.findElement(By.id("username")).sendKeys("username");
driver.findElement(By.id("password")).sendKeys("password");
driver.findElement(By.id("submit")).click();
Assert.assertThat(driver.getCurrentURL(), endsWith("/dashboard.html"));
```

Für Appium stehen außerdem erweiterte WebDriver-Schnittstellen wie der AndroidDriver zur Verfügung.

4.2. Automatisiertes Testen mit TestChameleon™

TestChameleon™ stellt ein Selenium Grid bereit, mit dem in derselben Art und Weise interagiert werden kann wie mit einem selbst aufgesetzten Grid. Der Unterschied besteht lediglich darin, dass die vorhandene Infrastruktur im Hintergrund automatisch virtuelle Maschinen für die angeforderten Systeme bereitstellt, auf denen der Test ausgeführt wird.

WebDriver werden im Grid, sowie im Hintergrund in TestChameleon™, mithilfe sogenannter DesiredCapabilities angefordert. Dadurch ist es möglich, das System auf dem getestet werden soll, beliebig genau zu definieren. DesiredCapabilities sehen wie folgt aus:

```
DesiredCapabilities dc1 = DesiredCapabilities.chrome();
DesiredCapabilities dc2 = DesiredCapabilities.firefox();
dc2.setPlatform(Platform.WIN);
dc2.setJavaScriptEnabled(true);
...
```

Eine vollständige Liste solcher DesiredCapabilities findet sich unter
https://github.com/SeleniumHQ/selenium/wiki/DesiredCapabilities

Nachdem die DesiredCapabilities definiert sind, wird durch den Hub ein RemoteWebDriver angefordert, der diesen Anforderungen entspricht:

```
RemoteWebDriver driver = new RemoteWebDriver(HUB_URL, dc2);
```

Im Hintergrund wird nun den definierten Anforderungen entsprechend eine virtuelle Maschine über TestChameleon™ gestartet. In der Regel ist hierbei innerhalb weniger Minuten ein RemoteWebDriver verfügbar, mit dem genauso agiert werden kann.

Weitere Informationen und Codebeispiele sind in der TestChameleon™ Dokumentation zu finden:
https://confluence.testbirds.com/display/TED

5. Zusammenfassung

Eine schnelle, effiziente und trotzdem umfangreiche Qualitätssicherung wird in zunehmend agilen Entwicklungsprozessen zum essentiellen Faktor, der die Softwarequalität beeinflusst. Hierbei müssen bestehende Komponenten, aber auch neue Bestandteile, stetig getestet werden, was die Kosten schnell in die Höhe treiben kann. Eine Mischung aus manuellen sowie automatisierten Tests ist daher oft empfehlenswert. Während neue Features einfacher und schneller manuell getestet werden können, verspricht die Automatisierung eine ressourcenschonende und damit langfristig kostengünstigere Alternative.

Obwohl das Einrichten eines Prozesses für automatisierte Tests sowie das Erstellen der entsprechenden Skripte vor allem zu Beginn mit hohem Aufwand verbunden ist, lohnt sich dieser Ansatz vor allem für Regressionstests der Kernfunktionalitäten einer Anwendung. Selenium für Desktopanwendungen sowie Appium für mobile Browser und native Applikationen bieten hierfür die entsprechenden Möglichkeiten, sämtliche Komponenten sowie Funktionalitäten entsprechend zu testen.

Genau wie im manuellen Testumfeld, stellt auch bei automatisierten Tests die enorme Vielfalt an Geräte-, Software- und Betriebssystemkombinationen eine ernstzunehmende Herausforderung dar. Die SaaS-Lösung TestChameleon™ bietet vor diesem Hintergrund die Möglichkeit, das zu testende System über bestehende Frameworks wie Selenium oder Appium gleichzeitig auf einer Vielzahl unterschiedlicher virtueller Maschinen zu testen.

Während viele praktische Fälle für den Einstieg mit Selenium auf Github verfügbar sind, kann der zusätzliche Einsatz virtueller Maschinen durch TestChameleon™ nach Anmeldung auf der Testbirds Plattform 30 Tage kostenlos getestet werden.

Für Rückfragen sowie Anmerkungen zum Inhalt dieses Whitepapers freuen wir uns über Ihre Kontaktaufnahme per Email (kontakt@testbirds.de) oder telefonisch unter der +49 (0) 89 856 33 350.

6. Glossar

1. Agile Software-Entwicklung
- Das klassische Wasserfall-Modell mit seinen starren Entwicklungs- und Testphasen wird zunehmend durch agile Methoden ersetzt. Kurze Iterationen, oft Sprints genannt, sorgen für häufige und ebenso kurze Testphasen.

2. Appium
- Appium ist eine Erweiterung von Selenium für das Testen von mobilen Apps und Webseiten auf Android und iOS. Es ermöglicht, Selenium-Tests in mobilen Browsern auszuführen sowie native Apps zu testen.

3. CI-Systeme
- Dies sind Programme zur Unterstützung von Continuous Integration. Dazu zählen etwa Jenkins, Cruise Control, TeamCity, Bamboo oder Gitlab CI.

4. Continuous Integration
- Durch diesen Ansatz wird Software ständig neu gebaut und automatisch getestet. Parallel zur Entwicklung findet auch das Testen regelmäßig statt (sog. Continuous Testing) und die Ergebnisse sind für alle Beteiligten einsehbar.

5. DevOps
- Das Kofferwort DevOps beschreibt die Zusammenarbeit von Software-Entwicklern (Development) und IT-Betrieb (Operations).

6. Jubula
- Mit dieser Testsuite können Tests ohne Programmierkenntnisse erstellt werden. Jubula ist eine Alternative zu Tools wie Selenium und Appium.

7. Regressionstest
- Nach einem Bugtest prüfen QA-Manager mit Regressionstests, ob durch das Entfernen von Bugs an anderer Stelle neue Fehler in vormals funktionsfähigen Komponenten entstanden sind.

8. Selenium
- Selenium ist ein Framework um Tests von Webseiten und mobilen Apps zu automatisieren. Es simuliert dabei Nutzeraktionen und steuert somit den Browser fern.

9. Software-as-a-Service (SaaS)
- Mit Software-as-a-Service werden Anwendungen in einer Cloud bereitgestellt. Der Kunde greift online auf die Software zu und muss diese nicht installieren.

10. Testumgebung
- Hiermit ist die Umgebung gemeint, auf der eine Software getestet wird. Bei TestChameleon™ befindet sich die Testumgebung in der Cloud.

11. UI-Test
- In einem UI-Test wird die Benutzeroberfläche getestet. Da dies zeitaufwendig ist, wird der Vorgang nach Möglichkeit automatisiert.

12. VM
- Eine virtuelle Maschine, kurz VM, ermöglicht es Geräte, Betriebssysteme, Browser und weitere Software in einer virtuellen Umgebung anstelle eines physikalischen Computers zu betreiben.

Testbirds

Testbirds ist spezialisiert auf das Testen von Software wie Apps, Webseiten oder Internet of Things-Anwendungen mithilfe innovativer Technologien und Lösungen. Unter dem Motto „Testing 4.0 –The Next Generation of Quality. Powered by Crowd and Cloud Technologies“ bietet das Unternehmen seinen Kunden verschiedenste Testarten für die Optimierung von Benutzerfreundlichkeit und Funktionalität. Mit über 150.000 registrierten Testern in 193 Ländern zählt Testbirds zu den weltweit führenden Crowdtesting-Anbietern. Darüber hinaus setzt der IT-Dienstleister auf cloudbasierte Technologien, um Kunden bei der Optimierung ihrer digitalen Produkte zu helfen. Im Zusammenspiel der beiden Bereiche entsteht so ein einzigartiges Portfolio mit zahlreichen Synergien, um die Qualität von Software auf eine neue Eben zu heben.

Testbirds wurde 2011 von Philipp Benkler, Georg Hansbauer und Markus Steinhauser gegründet und zählt mittlerweile über 65 Mitarbeiter. Neben dem Hauptsitz in München verfügt das Unternehmen über weitere Büros in Amsterdam, London und Stockholm, Franchisenehmer in Ungarn, Russland und der Slowakei sowie Vertriebspartner in Italien und Nordamerika.

Kontaktieren Sie uns:
kontakt@testbirds.de

Testbirds GmbH
Radlkoferstr. 2
81373 München
E-Mail: kontakt@testbirds.de
Telefon: +49 (0) 89 856 33 350
www.testbirds.de