Fünf wichtige Schritte zur Verbesserung der Codequalität

Alle wollen Software von höherer Qualität - und das in kürzerer Zeit. Das gilt auch für die Software für Medizinprodukte, egal ob kleine, erschwingliche Verbrauchergeräte und Devices bis hin zu großen klinischen Geräten. Die Anforderungen an moderne Softwareentwicklungsteams sind enorm - von erhöhtem Wettbewerbs- und Marktdruck über steigende Funktionalität und Komplexität bis hin zu höheren Erwartungen an Produktqualität, Sicherheit und Zuverlässigkeit. Agile Entwicklungsmethoden werden häufig bevorzugt, weil sie versprechen, schneller auf Veränderungen zu reagieren und Kundenanforderungen besser zu erfüllen.

Autor:
Ricardo Camacho
Parasoft
www.parasoft.com

Wiederholbare Prozesse schaffen

Oft werden Agile und DevOps als Möglichkeit verkauft, Software schneller und mit weniger Ressourcen zu entwickeln. Dies ist jedoch nicht die Absicht. Angesichts der Tatsache, dass 70 % der IT-Projekte scheitern oder ihre Ziele nicht erreichen, sind intelligente Entwicklungsteams bestrebt, ihre Entwicklungspraktiken zu verbessern, um nicht nur ein Projekt erfolgreich abzuschließen, sondern auch einen wiederholbaren Prozess für künftige Iterationen und Produkte zu schaffen. Dieser Beitrag erläutert, wie man die für agile und iterative Methoden erforderliche Agilität erreichen kann, um nicht nur ein Endprodukt zu erhalten, sondern ein Produkt bzw. Gerät, das die Qualitäts- und Sicherheitsziele erfüllt und übertrifft.

Kontinuierliches Testen

Es zeigt sich, dass Testen sowohl das Problem als auch die Lösung ist, um schneller eine bessere Codequalität zu erreichen. In einem agilen Prozess können viele Entwicklungsschritte reduziert werden, um sinnvolle Teile der Funktionalität zu entwerfen und zu implementieren. Die Integration der neuen Funktionalität ist jedoch riskant und der Umfang der Tests unklar. Das Testen ist einer der Hauptgründe, warum Softwareteams bei der Einführung agiler Methoden Schwierigkeiten haben. Oftmals verlieren die Teams die angestrebte Agilität, weil sie sich zu sehr oder zu wenig im Testen verzetteln.

Dabei gilt kontinuierliches Testen als Lösung für die Probleme von Softwareteams, die DevOps und agile Entwicklung einführen. Wikipedia definiert kontinuierliches Testen als „... den Prozess der Durchführung automatisierter Tests als Teil der Softwarebereitstellungspipeline, um sofortiges Feedback über die mit einem Software-Release-Kandidaten verbundenen Geschäftsrisiken zu erhalten“. Trotz dieser einfachen Definition ist die Implementierung von Continuous Testing und seine Optimierung im Laufe der Zeit eine ganz andere Sache, und darauf fokussiert dieser Beitrag.

Von der Eistüte zur Testpyramide

Die ideale Testpyramide definiert, wo Zeit und Aufwand in einem Projekt am besten investiert werden. In der idealen Pyramide investiert man wertvolle Zeit und Mühe in eine umfassende Suite von Unit-Tests an der Basis der Pyramide, unterstützt durch API- und Service-Tests, und in eine viel kleinere Anzahl von System- und GUI-basierten Tests an der Spitze der Pyramide (Bild 1).

Allerdings wird diese Pyramide oft in einen Eiskegel umgewandelt. Die Teams verbrauchen zu viel Zeit und Mühe auf fragile und komplexe GUI-Tests auf Systemebene, bei denen die gesamte Funktionalität implementiert und integriert werden muss - was dazu führt, dass sie nicht in der Lage sind, die Tests in den früheren Phasen des SDLC kontinuierlich durchzuführen. Der Schlüssel zum erfolgreichen kontinuierlichen Testen liegt darin, die Eistüte zum Schmelzen zu bringen und sich auf die Erstellung automatisierter Unit- und API-Tests zu konzentrieren, die kontinuierlich ausgeführt werden können, während die Entwickler die neue Funktionalität implementieren (Bild 2).

Qualität durch kontinuierliches Testen – in fünf Schritten

1. Aufbau einer Basis für Unit-Tests

Eine Basis für Unit-Tests schaffen, indem man die Erstellung, Ausführung und Wartung von Tests automatisiert. Nur wenn man die Erstel-

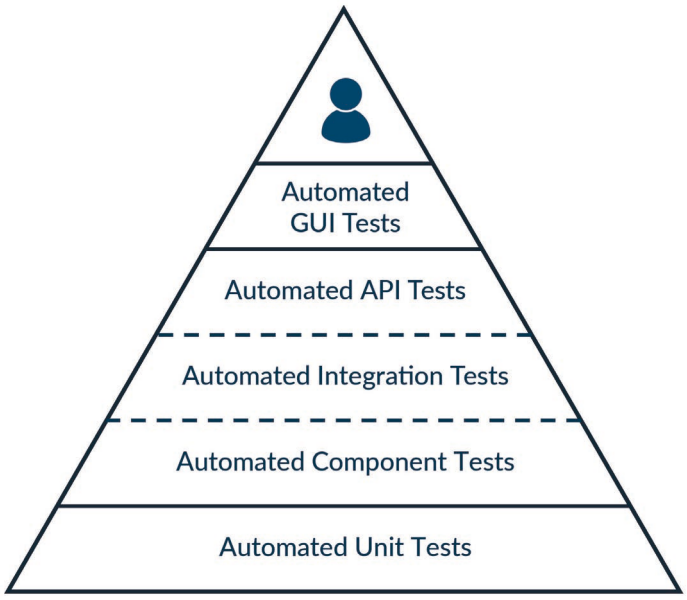

Bild 1: Die ideale Testpyramide zeigt auf, in welche Aufgaben mehr Zeit und Aufwand investiert werden sollte. Alle Bilder © Parasoft

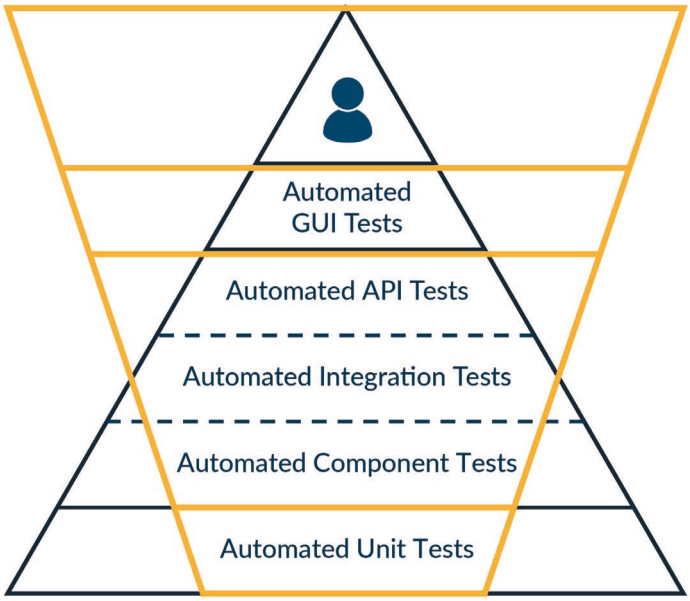

Bild 2: Die Testpyramide in der Praxis ist in vielen Fällen ein ‚Eiskegel'.

lung und Pflege von Unit-Tests vereinfacht, werden Entwicklungsteams projektweite Unit-Tests für alle Komponenten einführen.

Man sollte die Testautomatisierung sowohl für die Testerstellung als auch für die Testausführung und das Testmanagement einführen und die aktuelle Unit-Test-Suite erweitern, um so viel Code des Produkts wie sinnvoll einzubeziehen.

2. UI-zentrierte Tests am Ende des Zyklus vermeiden

Man sollte sich nicht auf späte, fragile UI-zentrierte Tests verlassen, die nur dazu führen, dass die Diagnose und Behebung von Problemen am zeitaufwändigsten und teuersten wird. Anstatt sich auf die Automatisierung aller manuellen Testszenarien zu konzentrieren, ist es ratsam, in eine solide Basis von Unit- und API-Tests zu investieren, um sicherzustellen, dass die Architektur, die mit der Benutzeroberfläche kommuniziert, von Anfang an solide ist.

Obwohl Tests auf Systemebene weiterhin wichtig und notwendig sind, sollten sie nicht an erster Stelle stehen. Es ist auch nicht der richtige Zeitpunkt, um kritische Architektur-, Leistungs- und Sicherheitsprobleme zu entdecken. Softwareteams können ihre Abhängigkeit von diesen UI- und Systemtests verringern, indem sie eine solide Basis von Unit- und API-Tests aufbauen. Wenn man die folgenden Empfehlungen befolgt, sollte ein Großteil der Systeme bereits gut getestet sein, bevor mit dem Testen auf Systemebene begonnen wird.

Die statische Analyse sollte auch dazu verwendet werden, die gesamte Codebasis, einschließlich Legacy-Code und Code von Drittanbietern, zu analysieren, um Fehler und Sicherheitslücken aufzudecken, die man beim Testen übersehen könnte. Darüber hinaus ist die statische Analyse wichtig für die Durchsetzung von Programmierstandards im Projekt.

3. Verstehen der Codeabdeckung für die gesamte Testpyramide

Es ist wichtig, die Codeabdeckung über die gesamte Pyramide und die Rückverfolgbarkeit zu den Anforderungen/User Stories zu begreifen, ansonsten wissen die Entwicklungsteams nicht, was getestet wurde und was nicht. Versteht man die Testabdeckung nicht, weiß man auch nicht, was auf jeder Ebene der Pyramide getestet werden muss - das bedeutet, dass selbst kleine Änderungen so viele Tests erfordern, dass der gesamte Prozess ins Stocken gerät.

4. Vorverlagerung (shift-left) mit Service-Virtualisierung

Die Service-Virtualisierung von Anwendungsabhängigkeiten ermöglicht automatisierte API-Tests zu einem viel früheren Zeitpunkt im Entwicklungszyklus. Ein höherer Automatisierungsgrad und eine frühere Fehlererkennung sind entscheidend für den Erfolg. Die frühere Durchführung von API-Tests hilft, kritische Aspekte des Systems, wie z. B. Performance und architektonische Stabilität, zu erkennen. Dies ist auch eine wichtige Phase für Sicherheitstests.

5. Einsatz von Change Impact Analyse zur Beschleunigung der agilen Entwicklung

Die Änderungsauswirkungsanalyse kann die agile Entwicklung auf der Basis von Builds beschleunigen, um die Risiken zu verstehen, die mit jeder neuen Iteration verbunden sind. Die durch die Change Impact Analyse gewonnenen Erkenntnisse sind der Schlüssel, um sicherzustellen, dass sich das Testen nur darauf konzentriert, was getestet werden muss, und nicht auf den üblichen Schrotflinten-Ansatz.

Ein wirklich kontinuierliches Testen ist nur durch eine intelligente, datengestützte Entscheidungsfindung möglich. Die Fokussierung des Entwicklungsteams auf ein Minimum an Tests, um eine angemessene Abdeckung in jeder Iteration zu gewährleisten, ist der Schlüssel, um wieder Agilität in die agilen Entwicklungsmethoden zu bringen.

Den Weg zur kontinuierlichen Verbesserung aufzeichnen

Es überrascht nicht, dass der beste Anfang darin besteht, die Testpyramide zu betrachten und dann zu bewerten, wo ein Projekt derzeit steht (Bild 3). Fragen sollten sein:

- Gibt es eine solide Grundlage an automatisierten Unit-Tests, die pro Build ausgeführt werden?
- Werden so viele Produkt-APIs wie möglich automatisiert getestet?
- Kommt Virtualisierung zum Einsatz?
- Basiert das Testen auf einer komplexen Reihe von manuellen UI-Tests, die erst ausgeführt werden können, wenn das System fast fertig ist?

Höhere Code-Qualität

Wichtige Testmethoden und -werkzeuge: Aufgrund der hohen Arbeitsbelastung von innovativen Softwareentwicklungsteams ist es schwierig, Produkte fristgerecht und spezifikationsgerecht zu entwickeln. Obwohl Teams mit Entwicklungsmethoden wie Agile sich darauf zu konzentrieren können, das Richtige für den Kunden zu entwickeln, sind die Projekte immer noch verspätet und fehleranfällig. Um signifikante Verbesserungen zu erzielen, sollte eine solide Basis an automatisierten Unit-Tests eingeführt und API-Tests frühzeitig und häufig mit Hilfe von Service-Virtualisierungslösungen (wie z. B. Parasoft Virtualize) durchgeführt werden. Nicht zu vergessen ist, dass sich die Testergebnisse deutlich optimieren lassen, wenn Daten aus modernen, hochentwickelten Software-Testanalysen zur Steuerung des Testmanagements zur Anwendung kommen. ◄

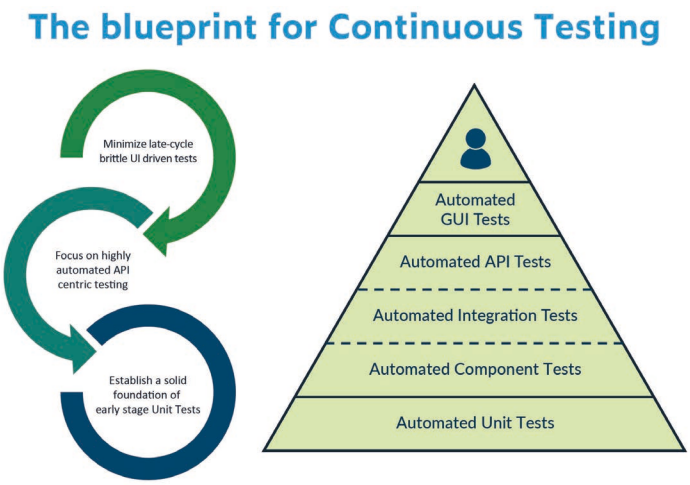

Bild 3: Der Weg zur Verbesserung basiert auf dem Aufbau einer geeigneten Testpyramide, Automatisierung sowie Datenerfassung und -analyse.