Behavior Driven Development (BDD) und Funktionstests

Unit-Tests sind eine Methode, bei der Codeeinheiten isoliert vom Rest der Anwendung getestet werden. Ein Komponententest kann eine bestimmte Funktion, ein Objekt, eine Klasse oder ein Modul testen. Unit-Tests eignen sich hervorragend, um herauszufinden, ob einzelne Teile einer Anwendung funktionieren oder nicht.

Unit-Tests testen jedoch nicht, ob Units zusammenarbeiten, wenn sie zu einer vollständigen Anwendung zusammengesetzt werden. Dazu benötigt man Integrationstests, bei denen es sich um Kollaborationstests zwischen zwei oder mehr Einheiten oder um vollständige End-to-End-Funktionstests der gesamten laufenden Anwendung (auch Systemtests genannt) handeln kann.

Es gibt mehrere Denkschulen, wenn es um Systemtests geht, einschließlich Behavior Driven Development (BDD) und Funktionstests.

## Was ist Behavior Driven Development?

Behavior Driven Development (BDD) ist ein Zweig des Test Driven Developments (TDD). BDD verwendet für Menschen lesbare Beschreibungen von Softwarebenutzeranforderungen als Grundlage für Softwaretests. Wie beim Domain Driven Design (DDD) ist ein früher Schritt bei BDD die Definition eines gemeinsamen Vokabulars zwischen Interessenvertretern, Domänenexperten und Ingenieuren. Dieser Prozess umfasst die Definition von Entitäten, Ereignissen und Ausgaben, die den Benutzern wichtig sind, und die Vergabe von Namen, auf die sich alle einigen können.

BDD-Praktiker verwenden dieses Vokabular dann, um eine domänenspezifische Sprache zu erstellen, die sie verwenden können, um Systemtests wie Benutzerakzeptanztests (UAT) zu codieren.

Jeder Test basiert auf einer User Story, die in der formal festgelegten, allgegenwärtigen, auf Englisch basierenden Sprache geschrieben ist. (Eine allgegenwärtige Sprache ist ein Vokabular, das von allen Beteiligten geteilt wird.)

Ein Test für eine Überweisung in einem Kryptowährungs-Wallet könnte so aussehen:

Story: Transfers change balancesAs a wallet user

In order to send money  
I want wallet balances to updateGiven that I have $40 in my balance  
And my friend has $10 is their balance  
When I transfer $20 to my friend  
Then I should have $20 in my balance  
And my friend should have $30 in their balance.

Beachten Sie, dass sich diese Sprache ausschließlich auf den Geschäftswert konzentriert, den ein Kunde aus der Software ziehen sollte, anstatt die Benutzeroberfläche der Software zu beschreiben oder wie die Software die Ziele erreichen soll. Dies ist die Art von Sprache, die Sie als Eingabe für den UX-Designprozess verwenden könnten. Das Entwerfen dieser Art von Benutzeranforderungen im Voraus kann später im Prozess viel Nacharbeit ersparen, indem es dem Team und den Kunden hilft, sich darüber einig zu werden, welches Produkt Sie entwickeln.

Von dieser Phase aus gibt es zwei Wege, die Sie hinuntergehen können:

Geben Sie dem Test eine konkrete technische Bedeutung, indem Sie die Beschreibung in eine domänenspezifische Sprache (DSL) umwandeln, sodass die menschenlesbare Beschreibung gleichzeitig maschinenlesbarer Code ist (weiter auf dem BDD-Pfad) oder

Übersetzen Sie die User Stories in automatisierte Tests in einer Allzwecksprache wie JavaScript, Rust oder Haskell.

In jedem Fall ist es im Allgemeinen eine gute Idee, Ihre Tests als Black-Box-Tests zu behandeln, was bedeutet, dass sich der Testcode nicht um die Implementierungsdetails der getesteten Funktion kümmern sollte. Black-Box-Tests sind weniger spröde als White-Box-Tests, da Black-Box-Tests im Gegensatz zu White-Box-Tests nicht an die Implementierungsdetails gekoppelt sind, die sich wahrscheinlich ändern, wenn Anforderungen hinzugefügt oder angepasst oder Code umgestaltet wird.

Befürworter von BDD verwenden benutzerdefinierte Tools wie Cucumber, um ihre benutzerdefinierten DSLs zu erstellen und zu warten.

Im Gegensatz dazu testen Befürworter von Funktionstests im Allgemeinen die Funktionalität, indem sie Benutzerinteraktionen mit der Schnittstelle simulieren und die tatsächliche Ausgabe mit der erwarteten Ausgabe vergleichen.

Bei Websoftware bedeutet dies in der Regel die Verwendung eines Testframeworks, das mit dem Webbrowser verbunden ist, um das Tippen, Drücken von Tasten, Scrollen, Zoomen, Ziehen usw. zu simulieren und dann die Ausgabe aus der Ansicht auszuwählen.

Normalerweise übersetze ich Benutzeranforderungen in Funktionstests, anstatt BDD-Tests fortzusetzen, hauptsächlich wegen der Komplexität der Integration von BDD-Frameworks in moderne Anwendungen und der Kosten für die Wartung von benutzerdefiniertem, englischähnlichem DSL, dessen Definitionen am Ende mehrere Systeme umfassen können, und sogar mehrere Implementierungssprachen.

Ich finde die für Laien lesbare DSL nützlich für Spezifikationen auf sehr hoher Ebene als Kommunikationswerkzeug zwischen den Beteiligten, aber ein typisches Softwaresystem erfordert um Größenordnungen mehr Low-Level-Tests, um eine angemessene Code- und Fallabdeckung zu erzeugen und Show-Stopp zu verhindern Fehler davon ab, die Produktion zu erreichen.

In der Praxis müssen Sie „Ich überweise meinem Freund 20 Dollar“ so übersetzen:

Brieftasche öffnen

Klicken Sie auf Übertragung

Geben Sie den Betrag ein

Geben Sie die Wallet-Adresse des Empfängers ein

Klicken Sie auf [Geld senden]

Warten Sie auf einen Bestätigungsdialog

Klicken Sie auf „Transaktion bestätigen“

Eine Schicht darunter pflegen Sie den Status für den Workflow „Geld überweisen“, und Sie möchten Komponententests, die sicherstellen, dass der richtige Betrag an die richtige Wallet-Adresse überwiesen wird, und eine Schicht darunter, die Sie benötigen um die Blockchain-APIs zu treffen, um sicherzustellen, dass die Wallet-Guthaben tatsächlich angemessen angepasst wurden (etwas, wofür der Kunde möglicherweise nicht einmal eine Ansicht hat).

Diese unterschiedlichen Testanforderungen werden am besten durch verschiedene Testebenen erfüllt:

Komponententests können testen, ob der lokale Clientstatus korrekt aktualisiert und in der Clientansicht richtig angezeigt wird.

Funktionstests können UI-Interaktionen testen und sicherstellen, dass Benutzeranforderungen auf der UI-Ebene erfüllt werden. Dadurch wird auch sichergestellt, dass UI-Elemente richtig verdrahtet sind.

Integrationstests können testen, ob die API-Kommunikation ordnungsgemäß erfolgt und ob die Beträge der Benutzer-Wallet tatsächlich korrekt in der Blockchain aktualisiert wurden.

Ich habe noch nie einen Laien-Stakeholder getroffen, der alle funktionalen Tests, die selbst das Verhalten der Benutzeroberfläche auf oberster Ebene überprüfen, auch nur annähernd kannte, geschweige denn jemanden, der sich um alle Verhaltensweisen auf niedrigerer Ebene kümmert. Da Laien kein Interesse haben, warum sollten Sie die Kosten für die Aufrechterhaltung eines DSL zahlen, um für sie zu übersetzen?

Unabhängig davon, ob Sie den vollständigen BDD-Prozess praktizieren oder nicht, enthält er viele großartige Ideen und Praktiken, die wir nicht aus den Augen verlieren sollten. Speziell:

* Die Bildung eines gemeinsamen Vokabulars, das Ingenieure und Interessengruppen verwenden können, um effektiv über die Benutzeranforderungen und Softwarelösungen zu kommunizieren.
* Die Erstellung von User Stories und Szenarien, die dabei helfen, Akzeptanzkriterien und eine Definition of Done für ein bestimmtes Feature der Software zu formulieren.
* Die Praxis der Zusammenarbeit zwischen Benutzern, dem Qualitätsteam, dem Produktteam und den Ingenieuren, um einen Konsens darüber zu erzielen, was das Team entwickelt.

Ein weiterer Ansatz für Systemtests ist der Funktionstest.

Der Begriff „Funktionstest“ kann verwirrend sein, da er in der Softwareliteratur mehrere Bedeutungen hatte.

IEEE 24765 gibt zwei Definitionen:

1. Tests, die den internen Mechanismus eines Systems oder einer Komponente ignorieren und sich ausschließlich auf die Ergebnisse konzentrieren, die als Reaktion auf ausgewählte Eingaben und Ausführungsbedingungen erzeugt werden [d. h. Black-Box-Tests]
2. Tests, die durchgeführt werden, um die Übereinstimmung eines Systems oder einer Komponente mit festgelegten funktionalen Anforderungen zu bewerten

Die erste Definition ist allgemein genug, um auf fast alle gängigen Formen des Testens anwendbar zu sein, und hat bereits einen perfekt passenden Namen, der von Softwaretestern gut verstanden wird: „Black Box Testing“. Wenn ich über Black-Box-Tests spreche, verwende ich stattdessen diesen Begriff.

Die zweite Definition wird normalerweise im Gegensatz zu Tests verwendet, die sich nicht direkt auf die Funktionen und Funktionen der App beziehen, sondern sich stattdessen auf andere Eigenschaften der App konzentrieren, z. B. Ladezeiten, UI-Antwortzeiten, Serverlasttests, Sicherheitspenetrationstests , usw. Auch diese Definition ist zu vage, um für sich genommen sehr nützlich zu sein. Normalerweise möchten wir genauer wissen, welche Art von Tests wir durchführen, z. B. Unit-Tests, Smoke-Tests, Benutzerakzeptanztests?

Aus diesen Gründen bevorzuge ich eine andere Definition, die in letzter Zeit populär geworden ist.

IBMs Developer Works sagt:

* Funktionstests werden aus der Sicht des Benutzers geschrieben und konzentrieren sich auf das Systemverhalten, an dem Benutzer interessiert sind.

Das ist viel näher am Ziel, aber wenn wir Tests automatisieren und diese Tests aus der Perspektive des Benutzers testen, bedeutet das, dass wir Tests schreiben müssen, die mit der Benutzeroberfläche interagieren.

Solche Tests können auch unter den Namen „UI-Tests“ oder „E2E-Tests“ laufen, aber diese Namen ersetzen nicht die Notwendigkeit des Begriffs „Funktionstests“, da es eine Klasse von UI-Tests gibt, die Dinge wie Stile und Farben testen. die nicht direkt mit Benutzeranforderungen wie „Ich sollte meinem Freund Geld überweisen können“ zusammenhängen.

Die Verwendung von „funktionalem Testen“, um sich auf das Testen der Benutzeroberfläche zu beziehen, um sicherzustellen, dass sie die angegebenen Benutzeranforderungen erfüllt, wird normalerweise im Gegensatz zu Komponententests verwendet, die wie folgt definiert sind:

* das Testen einzelner Codeeinheiten (wie Funktionen oder Module) isoliert vom Rest der Anwendung

Mit anderen Worten, während ein Einheitentest dazu dient, einzelne Codeeinheiten (Funktionen, Objekte, Klassen, Module) isoliert von der Anwendung zu testen, dient ein Funktionstest dem Testen der Einheiten in Integration mit dem Rest der App, aus der Perspektive des Benutzers, der mit der UI interagiert.

Mir gefällt die Klassifizierung von „Unit-Tests“ für Code-Units aus Entwicklerperspektive und „Funktionstests“ für UI-Tests aus Benutzerperspektive.

## Unit-Tests vs. Funktionstests

Unit-Tests werden in der Regel vom implementierenden Programmierer geschrieben und testen aus der Perspektive des Programmierers.

Funktionstests basieren auf den Benutzerakzeptanzkriterien und sollten die Anwendung aus der Perspektive des Benutzers testen, um sicherzustellen, dass die Anforderungen des Benutzers erfüllt werden. In vielen Teams können Funktionstests von Qualitätsingenieuren geschrieben oder erweitert werden, aber jeder Softwareentwickler sollte wissen, wie Funktionstests für das Projekt geschrieben werden und welche Funktionstests erforderlich sind, um die „Definition of Done“ für ein bestimmtes Projekt zu vervollständigen Funktionsumfang.

Unit-Tests werden geschrieben, um einzelne Units isoliert vom Rest des Codes zu testen. Dieser Ansatz hat zwei große Vorteile:

* Komponententests werden sehr schnell ausgeführt, da sie nicht von anderen Teilen des Systems abhängig sind und daher normalerweise keine asynchronen E/A-Vorgänge haben, auf die sie warten müssen. Es ist viel schneller und kostengünstiger, einen Fehler mit Komponententests zu finden und zu beheben, als darauf zu warten, dass eine vollständige Integrationssuite ausgeführt wird. Komponententests werden normalerweise in Millisekunden abgeschlossen, im Gegensatz zu Minuten oder Stunden.
* Die Einheiten müssen modular aufgebaut sein, damit sie auf einfache Weise isoliert von anderen Einheiten getestet werden können. Dies hat den zusätzlichen Vorteil, dass es sehr gut für die Architektur der Anwendung ist. Modularer Code ist einfacher zu erweitern, zu warten oder zu ersetzen, da die Auswirkungen einer Änderung im Allgemeinen auf die zu testende Moduleinheit beschränkt sind. Modulare Anwendungen sind flexibler und einfacher für Entwickler, mit der Zeit zu arbeiten.

Funktionsprüfungen dagegen:

* Die Ausführung dauert länger, weil sie das System End-to-End testen müssen, indem sie es mit all den verschiedenen Teilen und Subsystemen integrieren müssen, auf die sich die Anwendung stützt, um den zu testenden Benutzer-Workflow zu ermöglichen. Die Ausführung großer Integrationssuiten dauert manchmal Stunden. Ich habe Geschichten von Integrationssuiten gehört, deren Ausführung Tage gedauert hat. Ich empfehle, Ihre Integrationspipeline so zu optimieren, dass sie parallel läuft, damit sie in weniger als 10 Minuten abgeschlossen werden kann – aber das ist immer noch zu lange für Entwickler, um auf jede Änderung zu warten.
* Stellen Sie sicher, dass die Einheiten als Gesamtsystem zusammenarbeiten. Selbst wenn Sie über eine hervorragende Codeabdeckung für Unit-Tests verfügen, müssen Sie dennoch Ihre Units testen, die in den Rest der Anwendung integriert sind. Es spielt keine Rolle, ob die Hitzeschilde der NASA funktionieren, wenn sie beim Wiedereintritt nicht an der Rakete haften bleiben. Funktionstests sind eine Form von Systemtests, die sicherstellen, dass sich das Gesamtsystem bei vollständiger Integration wie erwartet verhält.

Funktionale Tests ohne Unit-Tests können niemals eine Codeabdeckung bieten, die tief genug ist, um sicher sein zu können, dass Sie über ein angemessenes Regressions-Sicherheitsnetz für Continuous Delivery verfügen. Einheitentests bieten Codeabdeckungstiefe. Funktionstests bieten eine breite Abdeckung von Testfällen für Benutzeranforderungen.

Funktionstests helfen uns, das richtige Produkt zu bauen. (Validierung)

Komponententests helfen uns, das Produkt richtig zu entwickeln. (Überprüfung)

Sie braucht beides.

## Wie man Funktionstests für Webanwendungen schreibt

Es gibt viele Frameworks, mit denen Sie Funktionstests für Webanwendungen erstellen können. Viele von ihnen verwenden eine Schnittstelle namens Selenium.

Selenium ist eine plattformübergreifende, browserübergreifende Automatisierungslösung, die 2004 entwickelt wurde und es ermöglicht, Interaktionen mit dem Webbrowser zu automatisieren. Das Problem mit Selenium ist, dass es sich um eine browserexterne Engine handelt, die auf Java angewiesen ist, und es kann daher schwieriger sein, sie mit Ihren Browsern zum Laufen zu bringen, als es sein muss.