Test-Driven Development von Embedded-Systemen – Teil 1: Test-First-Ansatz und TDD-Cycle

Test-Driven Development (TDD) ist die Umsetzung des Test-First-Ansatzes im Komponententest und steht für das Schreiben der Unit-Testfälle vor der eigentlichen Implementierung. Die Einhaltung von nur drei Regeln und ein paar Tricks im Umgang mit dem Target-Hardware-Bottleneck ermöglicht TDD auch für Embedded-Systeme.

Viele agile Entwicklungsframeworks verweisen auf den Test-First-Ansatz, der unabhängig von der Teststufe darauf beruht, als ersten Schritt zur eigentlichen Realisierung von Funktionalität mit dem Testen zu beginnen – also Testen zu einem Zeitpunkt, an dem man noch mit dem „Was" beschäftigt ist und das „Wie" noch vor einem liegt.

Der Test-First-Ansatz in den unterschiedlichen Teststufen

Die Kosten zum Auffinden von Fehlerzuständen und deren Beseitigung nehmen mit dem Fortschritt des Projektes deutlich zu und erreichen teils enorme Höhen zur Beseitigung im Feld. Der Test-First-Ansatz hat das Ziel, Fehler so früh wie möglich aufzudecken, indem Tests erstellt und ausgeführt werden, noch bevor der jeweilige Testgegenstand existiert. Test-First lässt sich in allen Teststufen anwenden, vom Acceptance-Test bis zum Komponenten-Test und der Implementierung von Software mittels Test-Driven Development (TDD).

Test-First im Acceptance-Test

In agilen Entwicklungsmodellen, z.B. Scrum, wird üblicherweise mit User- und Systems-Stories gearbeitet; daraus werden zu entwickelnde Backlog-Items abgeleitet. Der Test-First-Ansatz auf dieser Ebene bedeutet, dass Akzeptanzkriterien ermittelt werden, noch bevor darüber nachgedacht wird, wie ein Backlog-Item realisiert werden kann. Diese Akzeptanzkriterien müssen erfüllt sein, damit ein Backlog-Item in der Sprintabnahme als „done" gewertet werden kann. Durch die Definition der Abnahmetests vor der eigentlichen Realisierung entstehen vollständigere und besser testbare User-/System-Stories, da alle in den Testfällen genannten Bedingungen, Eingaben und zu erwartenden Ausgaben in die User-/System-Stories einfließen. Die auf den Backlog-Items basierende Aufwandsschätzung ist daher leichter und genauer zu bewerkstelligen.

Test-First im Systemtest

Im Systemtest liegt der Fokus mehr auf den Last- und Stresstests in einer möglichst betriebsnahen Umgebung, um die nicht-funktionalen Qualitätsmerkmale nach ISO/IEC 9126 der Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit nachzuweisen. Auch hier bedeutet der Test-First-Ansatz die Spezifikation und Erstellung der Systemtests, bevor das Produkt selbst entwickelt ist.

©MicroConsult 12-20

Aus der Erstellung der Systemtestfälle ergeben sich oft technische Anforderungen, die zu einem möglichst frühen Zeitpunkt unter dem Begriff „Design for Testability“ in die Planung und das Design des Produktes einfließen können.

Test-First im Integrationstest

Der Integrationstest erfordert ein stufenweises Vorgehen, in dem immer mehr Komponenten im Integrationstest zusammengefügt werden. Im Test-First-Ansatz werden schrittweise Integrationstests zusammen mit den erwarteten Ergebnissen erstellt, bevor die Komponenten überhaupt existieren. Schnittstellenprobleme können dadurch frühzeitig entdeckt und im Design und der Implementierung noch behoben werden. Die Spezifikation der Schnittstellen erfolgt unter dem Testaspekt, führt dadurch vom Einfachen zum immer Komplexeren hin und ergibt besonders schlanke und gut testbare Interfaces.

Test-First im Komponententest

Die Umsetzung des Test-First-Ansatzes im Komponententest bedeutet das Schreiben der Unit-Testfälle vor der eigentlichen Implementierung. Genau dieses Ziel verfolgt der Ansatz des Test-Driven Development in kleinen Schritten. Das aus dem Extreme Programming stammende TDD ist nicht Teil des Scrum-Entwicklungsframeworks, fügt sich aber perfekt ein.

In Scrum werden Sprint-Backlog-Items zur Abarbeitung in Tasks heruntergebrochen. Die kleinste übliche Granularität des Aufwands eines Tasks liegt bei einem Entwicklungstag. Weiter herunter reicht das Scrum-Framework nicht. Genau hier setzt TDD an und bricht Tasks in kleine Schritte herunter, die man iterativ abarbeitet – natürlich wieder nach dem Test-First-Ansatz: erst Testen, dann Implementieren.

TDD-Cycle

Im Test-Driven Development entsteht der Code zusammen mit den Unit-Testfällen im TDD-Cycle in kleinen, sich wiederholenden Mikroschritten. Im Fokus eines TDD-Cycles liegt jeweils eine abgeschlossene Funktionalität, z.B. eine Funktion in C, ein C-Modul, eine C++ Klasse oder eine Methode eines Objektes. Eine Iteration im TDD-Cycle besteht aus drei Hauptbestandteilen und dauert nur wenige Minuten – je kürzer, desto besser.

Die drei Hauptteile nennt man Red, Green und Refactor:

- **Red:** Schreibe einen Test, der eine noch zu erstellende Funktionalität prüft. Da die Funktionalität noch nicht existiert, muss der Test bei Ausführung noch fehlschlagen (Red).
- **Green:** Implementiere die fehlende Funktionalität, bis der Code den neuen Test und alle zuvor im Fokus dieses TDD-Cycles erstellten Tests besteht. (Green)
- **Refactor:** Nur wenn alle zuvor erstellten Tests bestanden sind, darf optional ein Refactoring des Codes und/oder der Tests durchgeführt werden. Durch das Refactoring darf nur die innere Struktur verändert werden. Es darf keine von außen sichtbare Funktionalität hinzugefügt oder entfernt werden. Am Ende des Refactorings sind alle Testfälle erneut auszuführen und zu bestehen.

©MicroConsult 12-20

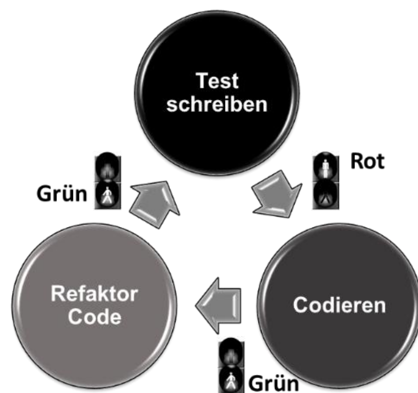

Abbildung: Der Test-Driven Development Cycle

Die drei Hauptschritte des TDD-Cycles werden so lange wiederholt, bis die geplante Funktionalität entwickelt und getestet wurde. Die konsequente Umsetzung dieses Vorgehensmodells fördert die schrittweise Verfeinerung vom einfachen Fall hin zu immer komplexeren Fällen. Um den sprichwörtlichen Wald vor lauter Bäumen nicht aus den Augen zu verlieren, ist es dringend erforderlich, eine Testliste mit den geplanten Iterationen des TDD-Cycles zu erstellen. Diese Liste ist Ihr persönlicher Abarbeitungsplan; er wird nicht dauerhaft gespeichert und darf jederzeit verändert werden. Sie dürfen jederzeit einzelne Schritte hinzufügen, streichen, umbenennen oder in der Reihenfolge verändern.

Unabhängig davon, ob eine neue Funktionalität oder nur eine Erweiterung entwickelt und getestet werden soll, ist zu Beginn die Erstellung einer persönlichen Testliste dringend angeraten.

WICHTIG: Erstellen Sie unbedingt auch eine Testliste für vermeintlich kleine Change Requests!

Im zweiten Teil des Beitrags werden u.a. die drei TDD-Regeln der kleinen Schritte beleuchtet.

Weiterführende Informationen

MicroConsult Training & Coaching zum Thema Test & Debug

MicroConsult Fachwissen zum Thema Test & Debug

MicroConsult Training & Coaching zum Thema Qualität, Safety & Security

MicroConsult Fachwissen zum Thema Qualität, Safety & Security

Autor

Remo Markgraf ist Senior Management Consultant bei der MicroConsult GmbH. Neben Begeisterung für Innovation und Leidenschaft für Embedded-Systeme verfügt er über langjährige Projekt- und internationale Führungserfahrung in Softwareentwicklung, Systems Engineering, Projekt-, Produkt-, Innovations- und Business Development Management sowie dem technischen Vertrieb.

©MicroConsult 12-20