MICRO CONSULT

Test-Ende gut, alles gut

Blackbox-Tests mit Whitebox-Metriken ergeben keine Graybox-Tests. Doch jeder der drei erfüllt eine wichtige Aufgabe, um das Test-Ende zu definieren und die Anzahl an notwendigen Softwaretests zu minimieren.

Das Test-Ende ist immer ein Kompromiss zwischen Qualität, Kosten und Zeit. Wollen wir eine hohe Qualität, dann gehen auch Kosten und Zeit nach oben. Senken wir die Kosten, leidet die Qualität, und weniger Zeit wird investiert. Wir wollen mit einem Plan festlegen, wo innerhalb dieses Dreiecks das Test-Ende sein soll und ob die Qualität höher oder die Zeit kürzer sein soll und definieren damit die Position innerhalb dieses Dreiecks. Um die Test-Ende-Entscheidung nicht dem Zufall zu überlassen, werden Metriken als Entscheidungsgrundlage verwendet. Durch Zielwerte über ausgewählte Metriken wollen wir definieren, wo in diesem magischen Dreieck wir uns befinden.

Whitebox-Metriken sind mit entsprechenden Tools schnell zu ermitteln und die angestrebten Zielwerte zur Erreichung des Test-Endes leicht festzulegen. Ein qualitäts-, kosten- und zeitoptimiertes Vorgehensmodell besteht daher darin, mit Blackbox-Verfahren Tests zu entwerfen und auszuführen und mit diesen Blackbox-Tests Whitebox-Metriken zu ermitteln und je nach Fall noch weitere wenige Whitebox-Tests hinzuzufügen, um das gewünschte Test-Ende-Kriterium zu erreichen.

Zu jedem Zeitpunkt kann man zusätzlich erfahrungsbasierte Tests hinzufügen, um dadurch die Testabdeckung weiter zu verbessern und zusätzlich das subjektiv empfundene gute Gefühl zur Produktabnahme zu erreichen.

IEC 61508-1: Example of a documentation structure for information related to the software safety lifecycle

Abbildung 1: Manuelle statische Tests (Reviews) von fast allen Artefakten

Statische Tests vs. dynamische Tests

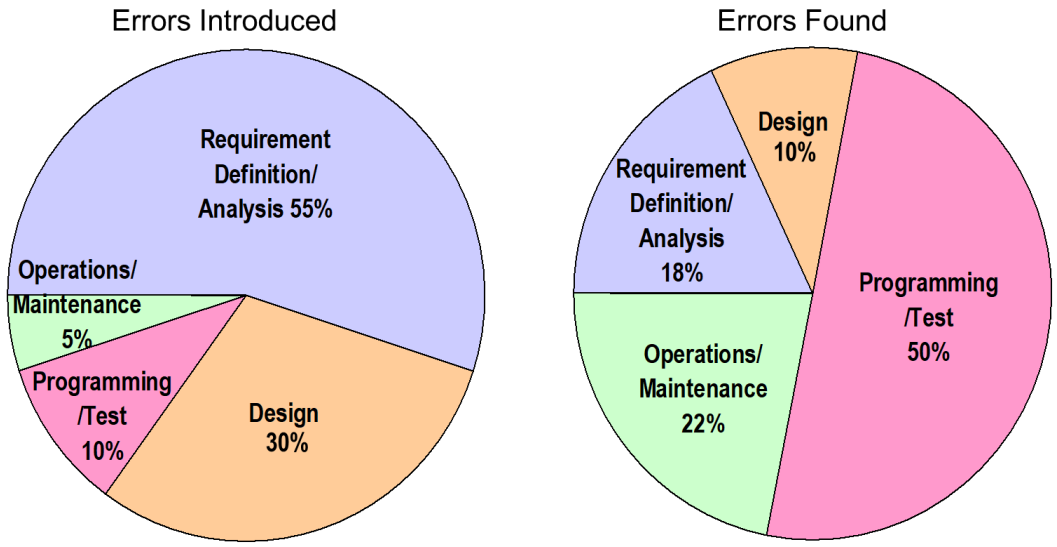
Auch wenn sich das hier vorgestellte Ermitteln der Test-Ende-Kriterien auf dynamische Tests bezieht, kann man nicht oft genug die Bedeutung statischer Tests zur Qualitätssteigerung betonen. Der große Hebel der statischen Tests hat zwei Ursachen:

1. Während sich statische Tests auf annähernd alle Artefakte eines Entwicklungsprojekts anwenden lassen, gilt das bei dynamischen Tests nur für Software-Komponenten (Abbildung 1).

2. Erfahrungsgemäß werden durch fehlerhafte, unvollständige, missverständliche und schwer testbare Anforderungen mehr als 50 % und in der Designphase bis zu 30 % der Fehlerursachen beigetragen (Abbildung 2).

Praxis-Tipp:
Wenn Sie in qualitätssteigernde Maßnahmen investieren wollen oder müssen, starten Sie mit Reviews.

Praxis-Tipp:
Um die hohen Kosten für Reviews zu senken, sollten vor dem Review werkzeuggestützte statische Testverfahren eingesetzt werden. Damit erhöhen Sie die Eingangsqualität des Review-Gegenstandes und verringern die Review-Dauer.



Source: Beltracchi, Leo: Notes on a Means-Ends Requirements Hierarchy. In: Halden Reactor Project HPR-348, Volume I; Loen 1996;
For the data reference is given to: Koss, E.: Developing Reliable Space Flight Software, Las Vegas, Nevada, January 28-30, 1986.

Abbildung 2: Verteilung von Fehlerursachen auf die Phasen des SW-Lifecycles

Blackbox-Tests vs. Whitebox-Tests

Einem Test sieht man am Ende nicht unbedingt an, ob er durch ein Blackbox- oder Whitebox-Testentwurfsverfahren entstanden ist. Das angestrebte Entwurfsziel ist zwar sehr unterschiedlich, ein entworfener Test besteht aber immer aus den gleichen Bestandteilen und vergleicht nach der Ausführung das erwartete Soll- mit dem erzielten Ist-Ergebnis. Eine Abweichung wird dabei als Fehler gewertet.

Das Entwurfsziel eines Blackbox-Tests bezieht sich auf die **Spezifikation**. Der Test verfolgt also das Ziel, die Übereinstimmung eines funktionalen oder nicht-funktionalen erwarteten Ergebnisses, das sich aus der Spezifikation ergibt, mit dem reellen Ergebnis zu vergleichen.

Das Entwurfsziel eines Whitebox-Tests besteht hingegen aus der Absicht, eine bestimmte Codemetrik zu erfüllen, also beispielsweise in einer bestimmten Anweisung/Codezeile gewesen zu sein, oder eine If- oder Else-Bedingung einer Entscheidung zu durchlaufen oder einen bestimmten Teil einer Bedingung auf einen booleschen Wert zu setzen. Dabei werden die anderen Bedingungen auf einen Wert eingestellt, der es erlaubt, den Wechsel des True/False-Wertes der untersuchten Bedingung durch eine abweichende Ausgabe zu beobachten.

Kombination aus Blackbox-Tests und Whitebox-Tests

Abbildung 3 verdeutlicht einen weiteren Unterschied der Blackbox- und Whitebox-Entwurfsverfahren: Während bei den Blackbox-Verfahren die Testabdeckung zunächst schnell anwächst, flacht die dargestellte Kurve im weiteren Verlauf stark ab. Es müssen also sehr viele Blackbox-Tests entworfen werden, um das angestrebte Ziel einer hohen Testabdeckung zu erreichen.

Beim Whitebox-Testentwurfsverfahren verhält es sich genau umgekehrt. Man muss viele Whitebox-Tests entwerfen, bis die Abdeckung – dafür aber dann sehr schnell – ansteigt.

Es liegt also auf der Hand, die beiden Entwurfsverfahren miteinander zu kombinieren. Starten Sie am besten mit Blackbox-Entwurfsverfahren und schalten Sie dann auf Whitebox-Entwurfsverfahren um. Das ergibt eine deutlich schneller ansteigende Testabdeckung, und es werden weniger Tests benötigt, um das Ziel zu erreichen.

In diesem Zusammenhang drängen sich zwei Fragen geradezu auf:

1. Wie finde ich den Umschaltzeitpunkt?
2. Wie wird das angestrebte Testabdeckungsziel bestimmt?

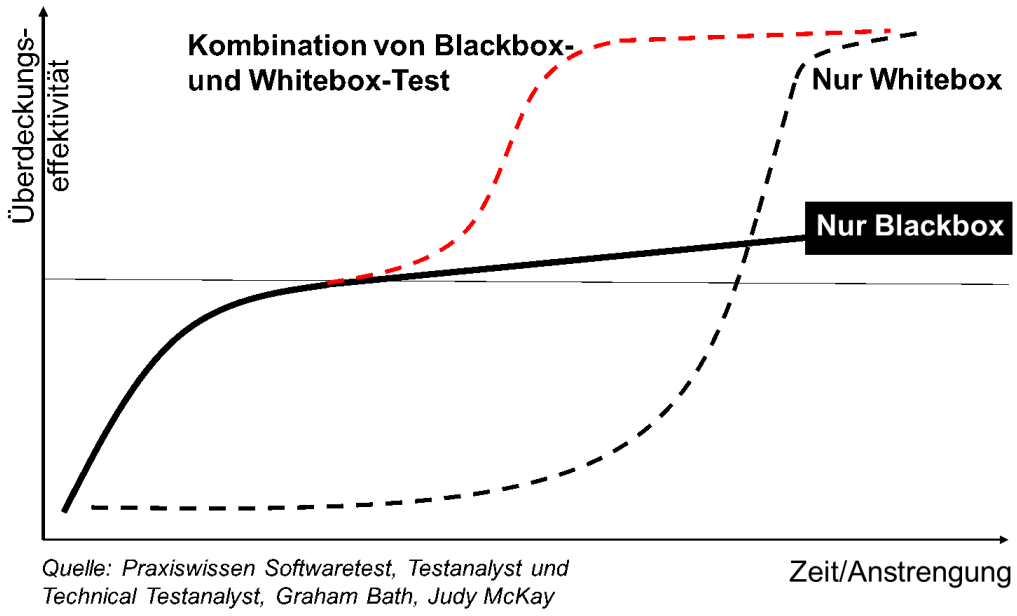


Abbildung 3: Kombination aus Blackbox- und Whitebox-Tests

Umschalten zwischen Blackbox- und Whitebox-Tests

Die gute Nachricht zuerst: Der Umschaltzeitpunkt muss nicht exakt bestimmt werden.

Sollten Sie etwas zu früh umschalten, dann befinden Sie sich immer noch auf dem schneller ansteigenden Teil der Blackbox-Testkurve und ersetzen einige Blackbox-Testfälle durch mühsamer zu erstellende Whitebox-Tests.

Sollten Sie etwas zu spät umschalten, so befinden Sie sich bereits auf dem abgeflachten Teil der Blackbox-Testkurve und haben vermutlich bereits einige Blackbox-Testfälle erstellt, die nur wenig zum Erreichen der angestrebten Test-Ende-Kriterien beitragen.

Der schlechte Teil der Nachricht ist, dass der vermeintlich ideale Umschaltzeitpunkt nur durch Erfahrung annäherungsweise getroffen werden kann. Wenn Sie nur noch eine oder zwei Handvoll Tests nach dem Whitebox-Testentwurfsverfahren erstellen mussten, um die Test-Ende-Kriterien zu erreichen, dann war der Umschaltzeitpunkt gut gewählt.

Test-Ende-Kriterium festlegen

Wie wird nun eigentlich das angestrebte Testabdeckungsziel bestimmt?

Einige Whitebox-Metriken sind zwingend mit 100 % zu erfüllen. Wirklich? Nein, es gibt keine einzige Whitebox-Metrik, die unter allen Umständen auf 100 % Erfüllungsgrad steigen kann.

Selbst die so wichtige und grundlegende Forderung nach einer Anweisungsüberdeckung von 100 % ist nicht immer erfüllbar; so zum Beispiel wenn Sie Codeteile generieren und gleichzeitig Input-Parameterwerte sauber überprüfen, damit keine Division durch 0 entstehen kann. Der generierte Code enthält unter Umständen ebenfalls solche Überprüfungen, und nur die jeweils erste lässt sich testen. Das leuchtet ein.

Sie wollen nun einzelne Parameterüberprüfungen streichen, um die Anweisungsüberdeckung zu erhöhen? Tun Sie das bitte nicht; es verbessert keineswegs die Qualität Ihres Codes. Leben Sie lieber mit einer dadurch begründeten, schlechteren Anweisungsüberdeckung.

Praxis-Tipp:
Sollte Ihre Anweisungsüberdeckung für bestimmte Codeteile keine 100 % erreichen können, weil Sie Codegenerierung oder Bibliotheken (z.B. C++ Class-Libraries) und defensive Programmierung kombiniert haben, dann unterziehen Sie die betroffenen Codeteile einem Review und sichern dadurch Ihre Behauptung ab, dass das Nichterreichen der 100 % Anweisungsüberdeckung durch doppelte Parameterprüfungen entsteht.

Ähnliches gilt analog für die Entscheidungs-, Bedingungs-, Mehrfachbedingungs- und die minimalbestimmte Mehrfachbedingungsüberdeckung (MC/DC). Gerade bei den Mehrfachbedingungsüberdeckungen ist es leicht möglich, dass durch den Daten- und Kontrollfluss nicht alle Kombinationen an Bedingungen einstellbar sind. Die 100 % sind damit nicht erreichbar. Starten Sie daher lieber mit 90 % oder 95 % und erlauben Sie ein Feintuning während des Projektfortschritts.

Praxis-Tipp:
Wenn Sie den aktuellen Zielwert für eine Test-Ende-Metrik nicht erreichen können, unterziehen Sie die betroffenen Codeteile einem Review und sichern dadurch die Qualität auch ohne das Erreichen des Zielwertes ab.

Praxis-Tipp:
Wo Sie sehr viele Blackbox-Testfälle bräuchten, kürzen Sie mittels Graybox-Tests ab. Sie handeln sich dadurch zwar eine Abhängigkeit Ihrer Tests von der gewählten Codierung ein; das ist aber immer noch deutlich besser, als Tests nicht zu erstellen, weil die Zeit nicht reicht und der Aufwand zu groß würde.

Graybox-Tests zur Reduzierung der Anzahl nötiger Tests

Stellen Sie sich bitte das einfache Beispiel einer isLeapYear() Funktion vor. Als Parameter wird ein Integer übergeben, und die Funktion gibt einen booleschen Wert zurück, je nachdem, ob die im Parameter übergebene Jahreszahl ein Schaltjahr ist oder nicht.

Bei der Äquivalenzklassen- und Grenzwertbildung zur Ermittlung von Blackbox-Tests wird schnell klar, dass man für 75 % der möglichen Jahre einen eigenen Test braucht. Bei einem angenommenen Wertebereich von 0 bis 9999 wären das satte 7500 Tests. Und dabei ist der Übergang vom julianischen auf den gregorianischen Kalender durch das Konzil im Jahre 1582 noch gar nicht berücksichtigt. Zur Erläuterung: Lassen wir die 100er- und 400er-Schaltjahresregel der Einfachheit halber mal kurz beiseite, dann wäre jedes vierte Jahr ein Schaltjahr. Die drei dazwischenliegenden Jahre bilden jeweils eine Äquivalenzklasse, und mit Hilfe der Grenzwertanalyse müsste der untere und obere Wert getestet werden. Das macht bei 2500 Schaltjahren in Summe 5000 Tests für die Nicht-Schaltjahre und 2500 für die Schaltjahre. Mit der Blackbox-Methode wird man hier nicht glücklich.

Spickt man aber in den Code und stellt fest, der Entwickler hat sich auf die Modulo-Funktion verlassen, die bei vielen Programmiersprachen eine erlaubte Operation ist, dann kann man sich auf die fehlerlose Funktion der Modulo-Funktion verlassen: Sie wurde bereits durch den Compiler-Hersteller getestet. So lassen sich die nötigen Testfälle auf eine bis zwei Handvoll reduzieren.

Würde in einem späteren Refactoring die Modulo-Funktion beispielsweise durch eine Tabelle ersetzt, blieben Fehler in der Tabelle vermutlich unentdeckt. Das ist der Preis, den man für die Reduktion des Testaufwands bezahlt.

Praxis-Tipp:
Wo Sie Graybox-Tests anstelle von Blackbox-Tests einsetzen, um die Anzahl der nötigen Tests deutlich zu reduzieren, fügen Sie weitere stark codierungsabhängige Tests oder innerhalb der Tests zusätzliche Assertions ein, um in späteren Veränderungen durch Refactorings auf die Codierungsabhängigkeit durch fehlschlagende Tests hinzuweisen.

Erfahrungsbasierte Tests

Zuletzt sollte man in jedem Fall noch erfahrungsbasierte Tests hinzufügen, um an fehlerträchtigen Stellen die Testtiefe über die Blackbox- und Whitebox-Testentwurfsverfahren hinaus zu erhöhen.

MicroConsult ist auf Ausbildung, Weiterbildung und Beratung für Hersteller von Embedded-Systemen spezialisiert. Sehr gerne unterstützen wir Sie mit Rat und Tat auf Ihrem Weg zur Einführung neuer Testmethoden.

Quellen

[1] Andreas Spillner, Theo Linz, Basiswissen Softwaretest, dpunkt.verlag
[2] Graham Bath, Judy McKay, Praxiswissen Softwaretest – Test Analyst und Technical Test Analyst – Advanced Level nach ISTQB-Standard, dpunkt.verlag
[3] Embedded-Test
[4] Agile-TDD
[5] Remo Markgraf, Test-First = Erst testen, dann denken?, ESE Kongress 2019

Weiterführende Informationen

MicroConsult Training: Embedded-SW-Test

MicroConsult Training: Agiles Testen und TDD

MicroConsult Training & Coaching zum Thema Test

MicroConsult Fachwissen zum Thema Test

Alle MicroConsult Trainings & Termine auf einen Blick

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.