ELEKTRONIK PRAXIS

Software-Test

Acht Irrtümer über Code Coverage

27.01.2010 | Autor / Redakteur: Frank Büchner* / Martina Hafner

*Frank Büchner (frank.buechner@hitex.de) ist Diplom-Informatiker und Senior Test Engineer bei der Firma Hitex Development Tools.

Über die Ermittlung und den Nutzen von Code-Coverage sind Halbwahrheiten und Irrtümer verbreitet, die leicht zu Verwirrung und Fehlannahmen führen können. Die häufigsten klärt Testingenieur Frank Büchner vom Toolspezialisten Hitex auf.

Irrtum 1: Man kann immer 100% Abdeckung erreichen

Natürlich nicht. Es lassen sich leicht Fälle finden, bei denen bedingt durch die Struktur des Codes oder die Zusammensetzung von Entscheidungen nicht 100% erreicht werden können.

```
int i;
for (i = 0; i < 2; i++)
{
   switch (i)
   {
      case 0:
         a = 600;
         break;
       case 1:
         a = 700;
         break;
   }
}
```

Bild 1: Ein Beispiel, bei dem 100% Zweigabdeckung nicht erreicht werden kann

Bild 1 zeigt ein Code-Schnipsel, bei dem 100% Zweigabdeckung nicht erreicht werden kann. (Dies ist ein Beispiel aus der Praxis, das hier vereinfacht dargestellt ist.) Woran liegt das? Es liegt am fehlenden Durchlauf des „default:“ case in der Switch-Anweisung. Dieser „default“-Zweig existiert immer, egal ob er explizit programmiert wurde oder nicht.

Irrtum 2: Ein Coverage-Maß hat nur einen Namen

Leider nein. Je nach Standard, Buch, Werkzeug oder Web-Seite werden für ein und dasselbe Maß unterschiedliche Namen verwendet. Beispielsweise verwendet Spillner den englischen Begriff „modified branch condition decision testing“ für Minimale Mehrfachbedingungsüberdeckung, während Liggesmeyer dieses Maß (mehr an die deutsche Bezeichnung angelehnt) „minimal multiple condition coverage“ nennt.

Irrtum 3: Ein Name bezeichnet immer dasselbe Coverage-Maß

Leider nein. Insbesondere wenn Abkürzungen wie C0, C1, C2 oder gar C1+ verwendet werden, ist Vorsicht geboten. So verwendet beispielsweise Beizer C1 zur Bezeichnung der Anweisungsüberdeckung, während Spillner C0 verwendet .

Irrtum 4: Es ist klar, wie Coverage gemessen wird

Durchaus nicht. In den Bildern 2 und 3 sind Zweifelsfälle aufgeführt, links jeweils das fragliche Code-Schnipsel, und rechts das Messergebnis des Unit-/Modul-Testwerkzeugs Tessy.

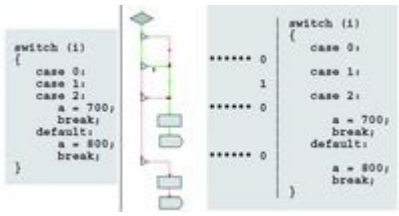

Bild 2: Aus wie vielen Zweigen besteht diese Switch-Anweisung?

Die Switch-Anweisung auf der linken Seite in Bild 2 könnte aus zwei oder aus vier Zweigen bestehen. Falls nur zwei Zweige gezählt werden, kennzeichnen die Label „case 0“, „case 1“ und „case 2“ ein und denselben Zweig, was damit begründet werden könnte, dass ja alle drei zur Ausführung derselben Anweisung führen. Falls jedes Label als eigener Zweig gezählt wird, gibt es vier Zweige.

Der Unterschied wird deutlich, wenn ein Testfall mit (i==1) ausgeführt wird. Ergeben sich dadurch 25% oder 50% Zweigüberdeckung? Auf der rechten Seite des obigen Bilds erkennt man, dass für Tessy die Switch-Anweisung aus 4 Zweigen besteht; der Testfall ergibt 25% Zweigüberdeckung (und nicht 50%).

Begründet werden kann dies damit, dass anstelle der Zuweisung „a = 700“ ja auch z.B. „a = 1/(2-i)“ stehen könnte. Dann würde ein Testfall mit (i==2) zur Division durch 0 führen. Ein solcher Testfall kann aber leicht unterbleiben, wenn schon mit (i==1) die vollständige Coverage erreicht wird.

Wie viele Zweige enthält der Fragezeichenoperator? Eigentlich wählt der Fragezeichenoperator nur zwischen Werten aus, die dann zugewiesen werden. Muss man hierfür einen oder zwei Zweige zählen?

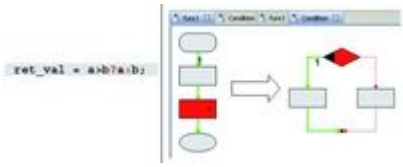

Bild 3: Wie viele Zweige enthält der Fragezeichenoperator?

In Bild 3 ist das Ergebnis der Ausführung eines Testfalls dargestellt, bei dem (a > b) war:Man erkennt, dass für Tessy der Fragezeichenoperator zwei Zweige besitzt, und durch den einen Testfall ein Zweig ausgeführt wurde, der andere jedoch nicht. Als Begründung für diese Zählweise kann man sich leicht ein Beispiel analog dem vorhergehenden Beispiel überlegen, z.B. indem man eine Zuweisung „ret_val = 1/(ret_val-b)“ nach der Zuweisung mit dem Fragezeichenoperator annimmt.

Irrtum 5: Für die Coverage ist es egal, wie der Code formuliert ist

Nein. Auch zwei funktional gleichwertige Code-Schnipsel (d.h. die zu einer bestimmten Eingabe immer das gleiche Ergebnis liefern) in derselben Programmiersprache können zur Erreichung eines bestimmten Coverage-Maßes eine unterschiedliche Anzahl von Testfällen erfordern.

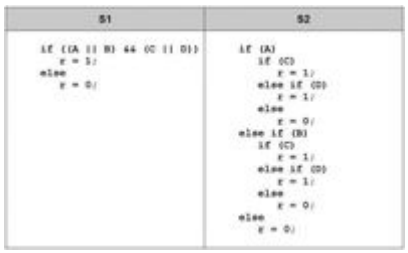

Bild 4: Zwei funktional äquivalente Code-Schnipsel

Die beiden Code-Schnipsel S1 und S2 in Bild 4 sind funktional äquivalent. Trotzdem braucht man beispielsweise zur Erreichung von 100% Zweigabdeckung bei S1 nur zwei Testfälle, während bei S2 sieben Testfälle notwendig sind. Durch geeignete Programmierung kann man also sogar vermeiden, dass ein bestimmtes Coverage-Maß überhaupt geprüft werden muss.

Wenn beispielsweise der (gesamte) Code wie in S2 formuliert ist (d.h. ohne Verwendung von logischem UND und ODER), kann man aus 100% Zweigabdeckung beispielsweise 100% Bedingungsabdeckung folgern, ohne letztere explizit zu messen.

Irrtum 6: Durch geschickte Programmierung kann man sich das Leben

erleichtern

Dies scheint eine Schlussfolgerung aus dem letzten behandelten Irrtum zu sein. Leider ist diese Schlussfolgerung nicht unbedingt richtig:

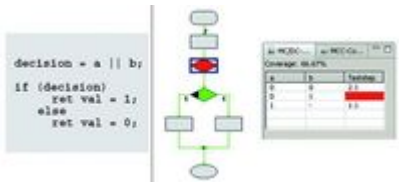

Bild 5: Reichen zwei Testfälle für 100% MC/DC?

Auf den ersten Blick sieht es so aus, als seien zur Erreichung von 100% MC/DC für den Code-Schnipsel auf der linken Seite in Bild 5 nur zwei Testfälle notwendig, einer, bei dem decision zu wahr und einer, bei dem decision zu falsch evaluiert, weil der Programmierer die zusammengesetzte Bedingung geschickt „ausgelagert“ hat.

Aber auf der rechten Seite des Bildes erkennt man, dass Tessy die zusammengesetzte Bedingung in der Zuweisung erkennt und deshalb 3 Testfälle zur Erreichung von 100% MC/DC notwendig sind. Das ist auch sinnvoll, denn sonst könnte man durch Umformulierungen im Code die Coverage-Messung unterlaufen.

Irrtum 7: Es reicht, die Testfälle zur vollständigen Code-Abdeckung aus dem Code abzuleiten

Man kann natürlich aus dem Code Testfälle ableiten, mit denen man eine 100% Code-Coverage erreicht. Es gibt sogar Tools, die dies automatisch für einen erledigen. Allerdings muss man sich dabei zweier Dinge bewusst sein:

- Durch aus dem Code abgeleitete Testfälle kann man keine Auslassungen im Code entdecken! Wurde z.B. vergessen, eine Abfrage zu implementieren, kann natürlich aus dem Code auch kein Testfall abgeleitet werden, der diese Abfrage prüft. Auch wenn man 100% Coverage erreicht, ist die Testaussage zweifelhaft, denn der Code wurde als vollständig vorausgesetzt.
- Das Ergebnis von automatisch abgeleiteten Testfällen sollte explizit überprüft werden. Unterlässt man dies, werden bei ihrer Ausführung nur schwerwiegende Probleme des Testobjekts, beispielsweise Abstürze, erkannt. Die Ableitung der Testergebnisse aus dem Code, bzw. die Hinnahme der Ergebnisse reduziert die Testaussage erheblich, denn dabei wird der Code als korrekt vorausgesetzt.

Irrtum 8: Code-Coverage misst die Qualität des Codes

Nicht absolut. Das einzige, was man sagen kann: Je höher die Prozentzahl der

Code-Coverage für ein bestimmtes Coverage-Maß ist, umso „besser“ ist es im Hinblick auf dieses Coverage-Maß getestet. Das ist ein kleiner aber feiner Unterschied. Wenn 100% für ein bestimmtes Maß erreicht sind, heißt das nicht, dass man ausreichend getestet hat.

Vier Ratschläge zum Thema Code Coverage

Wenn es um Code-Coverage geht, sollten Sie folgendes beherzigen:

- Vergewissern Sie sich immer, dass Ihr Gesprächspartner unter einer Bezeichnung auch dasselbe Maß versteht wie Sie selbst.
- Wenn Sie ein Werkzeug zur Messung der Code-Coverage einsetzen: Prüfen Sie, wie dieses die geschilderten Zweifelsfälle behandelt.
- Prüfen Sie, ob der Aufwand, den Sie zur Erreichung von z.B. 100% MC/DC erbringen müssen, im richtigen Verhältnis zu der dadurch erreichbaren verbesserten Testaussage steht, oder ob der Aufwand besser für andere Testaktivitäten verwendet werden sollte.
- Lehnen Sie sich nicht entspannt zurück, nur weil Sie 100% Code-Coverage erreicht haben!

Weitere häufige Irrtümer lesen Sie in der Märzausgabe von ESE Report sowie online.

Literaturhinweise

[Tessy] http://www.hitex.de/perm/tessy.htm: More about Tessy and the Classification Tree Method.

[Spillner] Spillner, A., Linz, T.: Basiswissen Softwaretest, Heidelberg, 2003. dpunkt-Verlag.

[Liggesmeyer] Liggesmeyer, Peter: Software-Qualität: Testen, Analysieren und Verifizieren von Software. Heidelberg, Berlin, 2002. Spektrum Akademischer Verlag.

[Beizer] Beizer, Boris: Software Testing Techniques, 2nd edition, New York, 1990.

[DO-178B] Software Considerations In Airborne Systems And Equipment Certification, RTCA, 1992.

[IEC 61508] Functional safety of electrical/electronic/programmable electronic safety-

related systems, IEC, www.iec.ch.

[ED-94B] Final report for clarification of ED-12B DO-178B “Software considerations in airborne system and equipment certification”, Eurocae, 2001