Code Coverage für Fortgeschrittene

Erhellendes und Erschreckendes zur Codeüberdeckungsmessung

Frank Büchner, Hitex GmbH, Karlsruhe

Im Folgenden steht nicht die Definition der Codeüberdeckungsmaße im Vordergrund. Es geht vielmehr um (vielleicht) überraschende Einsichten, das Ausräumen von (vielleicht) naiven Ansichten, den Hinweis auf unterschiedliche Interpretationsmöglichkeiten und die Vermeidung von möglichen Missverständnissen.

Welche Codeüberdeckungsmaße werden in Standards genannt?

	IEC 61508	ISO 26262	DO-178
Entry Point Coverage	X		
Statement Coverage	X	X	X
Branch Coverage	X	X	
Decision Coverage			X
Modified Condition/Decision Coverage (MC/DC)	X	X	X

Tabelle 1: Codeüberdeckungsmaße aus Standards

Wie sind die Maße definiert und wie verhalten sie sich zueinander?
Man kann die Maße nach der Testtiefe sortieren, die üblicherweise erforderlich ist, um eine Überdeckung von 100% für ein Maß zu erhalten. Dies korreliert mit der Kritikalität in Bezug auf Safety, bei der in den oben erwähnten Standards die Messung dieser Maße gefordert wird. Diese Sortierung ist in der obigen Tabelle (Tabelle 1) angegeben, wobei Entry Point Coverage (oben) das Maß mit der geringsten Testtiefe und MC/DC (unten) das Maß mit der größten Testtiefe ist. Oder anders ausgedrückt: In sicherheitskritischen Systemen, die eine hohe Risikoreduzierung erfordern, soll MC/DC gemessen werden; in sicherheitskritischen Systemen, die nur eine geringe Risikoreduzierung erfordern, reicht die Messung von Entry Point Coverage bzw. Statement Coverage.

Ein Überdeckungsmaß gibt prinzipiell an (üblicherweise in Prozent), welcher Teil des jeweiligen Messkriteriums durch die Tests der Software erreicht / durchlaufen / ausgeführt wurde. Die einzelnen Maße haben folgende Bedeutung:

Entry Point Coverage (Eingangspunkte):
Das Messkriterium sind die Eingangspunkte in die Software. Bei der Programmiersprache C sind dies die Funktionen. (Ein Label in C kann man nur aus der Funktion heraus erreichen, in der sich das Label befindet. In unserem Sinne sind Label also keine Eingangspunkte in die Software.) Besteht also die zu testende Software aus 100 Funktionen, und hat man bei den bisherigen Tests 75 dieser Funktionen zumindest einmal aufgerufen, so beträgt die Entry Point Coverage 75%. Entry Point Coverage liefert eine sehr schwache Aussage. Das hängt damit zusammen, dass Funktionen sehr unterschiedliche Größen haben können und es

zudem nicht darauf ankommt, welcher Anteil der Funktion bei dem vielleicht einzigen Aufruf ausgeführt wurde.

Statement Coverage (Anweisungsüberdeckung):
Das Messkriterium sind die Anweisungen in der Software. Die Statement Coverage gibt also den Anteil der durch die Tests ausgeführten Anweisung im Verhältnis zur Gesamtzahl der Anweisungen an. Der Nutzen der Statement Coverage zeigt sich, wenn durch Werte < 100% Anweisungen entdeckt werden, die während der Tests nie ausgeführt wurden. Abhängig von der Programmiersprache kann es schwierig sein, festzulegen, was genau als eine Anweisung gezählt wird.

Branch Coverage (Zweigüberdeckung):
Das Messkriterium sind die Zweige in der Software. Die Branch Coverage gibt also den Anteil der durch die Tests ausgeführten Zweig im Verhältnis zur Gesamtzahl der Zweige an. Eine if-Anweisung besitzt beispielsweise zwei Zweige, den then-Zweig und den else-Zweig. Für manche erstaunlich, existiert der else-Zweig auch dann, wenn er nicht explizit programmiert wurde. (Die ISO 26262 fühlt sich bemüßigt, ausdrücklich darauf hinzuweisen.) Zweige gibt es natürlich auch bei anderen Anweisungen, beispielsweise besitzt die do-while-Schleife einen Zweig „zurück“ (der bei while(0) nie durchlaufen wird). In switch-Anweisungen führt ein Zweig zu jedem case-Label, wobei sich die Frag stellt, ob das auch gelten soll, wenn mehrere Label direkt hintereinander stehen und somit gemeinsam den Beginn eines Anweisungsblocks bezeichnen. (Der Autor ist der Ansicht, dass die Antwort „ja“ ist und gibt darüber gerne weitere Auskunft.) Branch Coverage ergibt offensichtlich eine bessere Testaussage als Statement Coverage, denn Branch Coverage deckt beispielsweise nichtausgeführte nichtexistierende else-Zweige auf, was Statement Coverage nicht vermag.

Decision Coverage (Entscheidungsüberdeckung):
Das Messkriterium sind hier die Entscheidungen in der Software. Viele setzen Decision Coverage mit Branch Coverage gleich. Für sie hat beispielsweise die if-Anweisung eine Entscheidung, und der then-Zweig der if-Anweisung wird ausgeführt, wenn die Entscheidung wahr ist und der else-Teil der if-Anweisung wird ausgeführt, wenn die Entscheidung falsch ist. Also folgern sie (fälschlicherweise), dass aus 100% Branch Coverage auch 100% Decision Coverage folgt. Wenn man allerdings die Definition zur Decision Coverage in [DO-178C] und dazu in Beziehung stehenden Papieren [CAST-10] genauer untersucht, stellt sich heraus, dass auch der Aufbau einer Entscheidung betrachtet wird. Entscheidungen bestehen demnach aus Bedingungen (conditions), die durch Boolsche Operatoren verknüpft sind. Die Decision Coverage gehört also eher zu den Bedingungsüberdeckungsmaßen. (In Bezug auf Branch Coverage sind Entscheidungen monolithisch.) Um 100% Decision Coverage zu erreichen, müssen alle Bedingungen in einer Entscheidung, die im Programm enthalten sind, mindestens einmal wahr und einmal falsch ergeben haben. Unter diesen Voraussetzungen kann man Beispiele konstruieren, bei denen 100% Branch Coverage, aber nicht 100% Decision Coverage erreicht wird. An dieser Stelle soll das nicht weiter ausgeführt werden. (Der Autor stellt auf Anfrage gerne ein Beispiel mit Erläuterungen zur Verfügung.)

Modified Condition / Decision Coverage (MC/DC), (Modifizierte Bedingungs- / Entscheidungsüberdeckung):
Dies ist ein Maß aus der Klasse der Bedingungsüberdeckungsmaße. Eine Entscheidung besteht aus Bedingungen, die durch logische Operatoren verknüpft sind. Durch MC/DC wird geprüft, ob diese Bedingungen in ausreichendem Maß zu der Gesamtentscheidung beitragen. Das ist der Fall, wenn für eine bestimmte Bedingung ein Paar von Testfällen / Eingangskombinationen (von Wahrheitswerten für die Bedingungen) existiert, für die drei Dinge zutreffen: (1) Die beiden Eingangskombinationen müssen sich im Wahrheitswert für die Bedingung, um die es geht, unterscheiden. (2) Die beiden Eingangskombinationen müssen für alle anderen Bedingungen in dieser Entscheidung denselben Wahrheitswert aufweisen. (3) Die beiden Eingangskombinationen müssen zu unterschiedlichen Wahrheitswerten für die Gesamtentscheidung führen. Wenn man für alle Bedingung aus der Entscheidung ein solches Paar von Eingangskombinationen gefunden hat und damit getestet hat, hat man 100% MC/DC erreicht. Wenn eine Entscheidung n Bedingungen besitzt, kann man mit n+1 Eingangskombinationen die benötigten Paare zusammenstellen.

Bedeutet ein Name immer dasselbe Überdeckungsmaß bzw. gibt es immer nur einen Namen für ein Überdeckungsmaß?
Leider nein. Für die Überdeckungsmaße sind auch Abkürzungen im Gebrauch, beispielsweise C0, C1, C2, und deren Bedeutung ist nicht immer einheitlich. So wird beispielsweise im Buch „Software Testing Techniques“ von Boris Beizer [BEIZER] mit der Abkürzung C1 die Statement Coverage bezeichnet, während im Buch „Basiswissen Softwaretest“ [SPILLNER] die Statement Coverage mit C0 abgekürzt wird. In diesem Buch ist dann C1 die Abkürzung für Branch Coverage, was bei Beizer mit C2 abgekürzt wird. Aber auch wenn man „richtige“ Namen verwendet, gibt es Überraschungen: So wird beispielsweise die Bezeichnung „Mehrfachbedingungsüberdeckung“ im oben erwähnten Buch „Basiswissen Softwaretest“ mit dem englischen Begriff „branch condition combination testing“ bezeichnet, im Buch „Software-Qualität“ [LIGGESMEYER] jedoch mit der wörtlichen Übersetzung „Multiple Condition Coverage“. Man braucht also nicht gleich die Flinte ins Korn zu werfen, wenn man Multiple Condition Coverge messen soll, das zur Verfügung stehende Werkzeug aber nur branch condition combination testing messen kann. Man sollte sich also immer über die Begrifflichkeiten mit dem Gegenüber auseinandersetzen.

In welchem Prozessschritt sollte Codeüberdeckung gemessen werden und warum?
Wenn man als Ziel hat, 100% eines Maßes zu erreichen, kann es sein, das man dies nur im Unit- bzw. Modul-Test erreichen kann, aber nicht im Integrationstest und nicht im Systemtest. Das liegt daran, dass man das Testobjekt im Unit-Test mit beliebigen Testeingangsdaten testen kann, was im Systemtest unter Umständen nicht möglich ist.

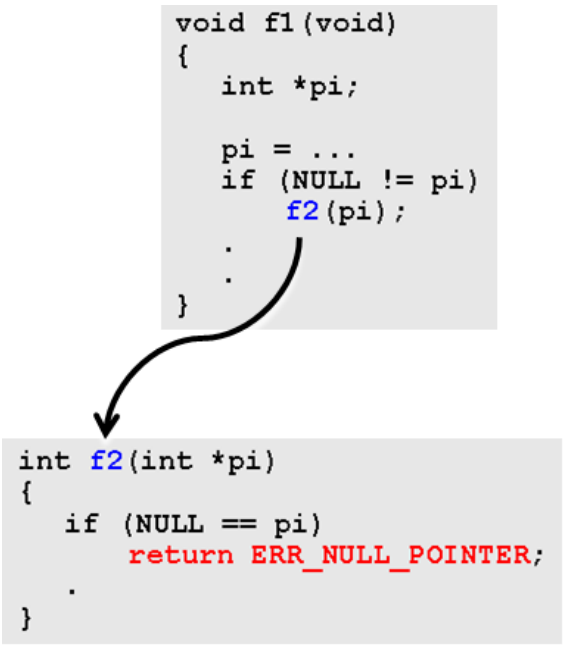

Bild 1 Im Systemtest sind 100% Überdeckung für f2() nicht erreichbar

Im obigen Bild ist eine Situation dargestellt, bei der im Systemtest nicht 100% Codeüberdeckung erreicht werden kann. Die Funktion f1() ruft die Funktion f2() mit einem Pointer als Parameter auf, aber nur nachdem sichergestellt ist, dass dieser Pointer kein NULL-Pointer ist. Die Funktion f2() prüft jedoch vor der Verwendung des übergebenen Pointers, ob der Pointer ein NULL-Pointer ist und gibt in diesem Fall eine Fehlerkennung zurück (defensive Programmierung). Wenn man unterstellt, dass f2() nicht noch von anderen Funktionen aufgerufen wird, wird man im Systemtest nicht erreichen können, dass ein NULL-Pointer an f2() übergeben wird und somit der then-Zweig der if-Anweisung bzw. die return-Anweisung ausgeführt wird. Im Unit-Test kann man beliebige Werte an das Testobjekt übergeben, also auch einen NULL-Pointer an f2(). Dadurch kann man 100% Überdeckung für f2() erreichen und zudem den return-Wert auf Korrektheit prüfen.

Was wird bei der Codeüberdeckungsmessung nicht berücksichtigt?

```
long double sinus(long double x_deg)
{
      int i;
      long double temp, x_rad;
      int sign = -1;

      x_rad = x_deg / 180 * pi ;
      temp = x_rad;

      for(i=3; i<=(MAX_FAC-2); i+=2)
      {
        temp += sign * pot(x_rad,i) / fac(i);
        sign *= -1;
      }
      return(temp);
}
```

Bild 2 Dieses rechenintensive Testobjekt erreicht mit einem Testfall 100% Überdeckung

Die Funktion im obigen Bild führt im Wesentlichen Berechnungen durch. Ausgehend vom Namen kann man vermuten, dass die Funktion den Sinus-Wert zu ihrer Eingabe berechnet. Tatsächlich: Ruft man diese Funktion mit dem Wert 0 für den Eingabewert x_deg auf (vermutlich eine Gradzahl), liefert die Funktion das Ergebnis 0. Das wäre der korrekte Wert. Die Codeüberdeckungsmessung für diesen einen Testfall (Eingabe 0 ergibt Rückgabewert 0) liefert 100% für alle Maße, die bisher erwähnt wurden. Sind wir also fertig mit Testen? Wir haben schließlich einen bestandenen Testfall und 100% Coverage! Ich hoffe, die Antwort ist nein. Es sollte klar sein, dass es noch etlicher zusätzlicher Testfälle bedarf, um unsere Vermutung zu stärken, dass die Funktion den Sinus-Wert ihrer Eingabe berechnet. Und dies liegt daran, dass Codeüberdeckungsmessung Berechnungen nicht berücksichtigt. Als Konsequenz aus dem obigen Beispiel muss man einsehen, dass man kein großes Vertrauen in die Qualität der Software setzen kann, nur weil man 100% Überdeckung erreicht hat.

Welches Problem kann man durch Codeüberdeckungsmessung nicht finden?
Ganz einfach: Auslassungen im Code werden durch Überdeckungsmessungen nicht aufgedeckt. Wurde beispielsweise der Test, ob der übergebene Parameter ein NULL-Pointer ist, in der Funktion f2() in Bild 1 vergessen zu implementieren, kann dies durch Überdeckungsmessung nicht aufgedeckt werden. Auch in der Funktion sinus() aus Bild 2 fehlt eine wichtige Code-Zeile, durch welche die Genauigkeit der Berechnung erhöht wird. Auch diese Auslassung wird durch die Überdeckungsmessung nicht aufgedeckt. Dies sollte eigentlich selbstverständlich sein, trotzdem ist es nicht immer bewusst.

Wieso sollte man nicht die Testfälle zur Erreichung von 100% Überdeckung aus dem Code herleiten?
Der Hauptgrund ist wie oben erwähnt, dass dadurch Auslassungen im Code nicht gefunden werden können. Deshalb sollten Testfälle immer auf Basis der Anforderungen erstellt werden. Ein zusätzlicher Grund ist, dass man bei dieser Vorgehensweise den Code eigentlich als korrekt voraussetzt. Natürlich kann man die Ergebnisse der aus dem Code abgeleiteten Testfälle gegen die Anforderungen prüfen (und dabei Fehler und vielleicht sogar Anforderungen finden, für die es keine Tests gibt), aber es ist m.E. trotzdem der falsche Ansatz.

Welche Berechnungsmöglichkeiten für MC/DC gibt es?

	A	B	C	A && (B || C)
1	f	-	-	F
2	t	f	f	F
3	t	f	t	T
4	t	t	-	T

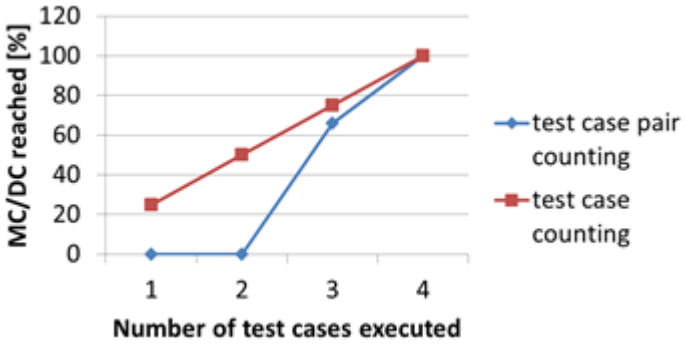

Bild 3 Testfälle und MC/DC-Werte

Im linken Teil von Bild 3 sind die vier Testfälle dargestellt, die man ausführen muss, um 100% MC/DC für die Entscheidung (A && (B || C) zu erhalten. [Aufgrund der unvollständigen Evaluierung in C sind trotz der drei Bedingungen nicht acht Testfälle möglich, sondern nur vier. Und weil bei drei Bedingungen vier Testfälle zur Erreichung von 100% MC/DC notwendig sind, müssen alle vier Testfälle ausgeführt werden.] Erstaunlicherweise erhält man unterschiedliche MC/DC-Werte solange noch nicht alle vier Testfälle ausgeführt sind, abhängig von der Berechnungsmethode, die angewendet wird. Die eine Methode (test case counting) geht von der Anzahl der notwendigen Testfälle aus, in unserem Beispiel vier. Und wenn einer dieser vier Testfälle (egal welcher) ausgeführt worden ist, ergeben sich 25% MC/DC. Zwei Testfälle ergeben 50%, drei Testfälle 75% und vier Testfälle schließlich 100% MC/DC. Die andere Methode (test pair counting) geht von der Anzahl der notwendigen Paare von Testfällen aus, in unserem Beispiel sind dies drei, weil es drei Bedingungen gibt. Und wenn ein Testfall, beispielsweise Testfall Nr. 1, ausgeführt wurde, ist noch kein Paar ausgeführt, denn dazu gehören zwei Testfalle. Deshalb ermittelt diese Methode für Testfall Nr. 1 alleine 0% MC/DC. Wird dann ein weiterer Testfall, beispielsweise Testfall Nr. 2, ausgeführt, bleibt der MC/DC-Wert bei 0%, denn Testfall Nr. 1 und Testfall Nr. 2 haben beide das gleiche Gesamtergebnis und können deshalb für keine der Bedingungen ein Paar ergeben. Mit Testfall Nr. 3 ergeben sich dann zwei Paare (das für Bedingung A und das für Bedingung C), und damit sind auf einen Schlag 66% MC/DC erreicht. Diese Werteverläufe sind auf der rechten Seite von Bild 3 dargestellt. Bemerkenswert ist,

dass beispielsweise um die Vorgabe 70% MC/DC zu erfüllen, nach der einen Methode 3 Testfälle ausreichen, nach der anderen jedoch 4 Testfälle notwendig sind.

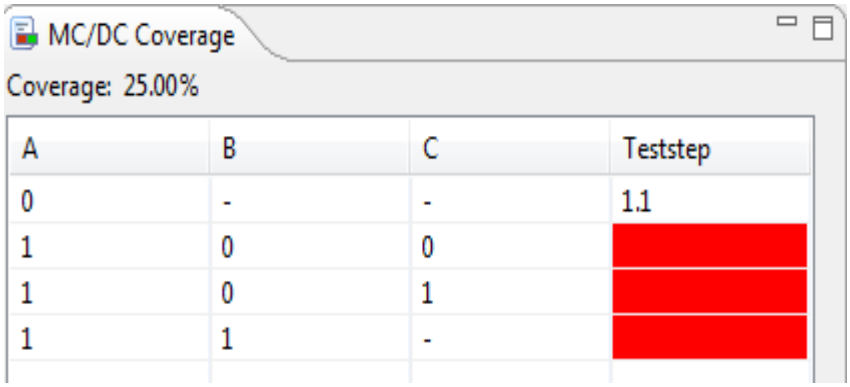

Bild 4 Mit einem Testfall ermittelt TESSY 25% MC/DC → test case counting

Darf man Überdeckungswerte von Codevarianten ergänzen?

```
short func(short input)
{
    short a;
    if (input >= 3)
        a = 3;
    else
        a = 0;
#ifdef VARIANT_1
    a = 12/(a+1);
#endif
#ifdef VARIANT_2
    a = 12/a;
#endif
    return a;
}
```

	Constant	Input	Branch	Output
TC1	none	3	then	3
TC2	none	0	else	0
TC3	VARIANT_1	3	then	3
TC4	VARIANT_1	0	else	12
TC5	VARIANT_2	3	then	4

Bild 5 Testobjekt mit Varianten und einige Testfälle

Im obigen Bild (Bild 5) ist links eine Funktion dargestellt, die unterschiedliches Verhalten zeigt, je nachdem, ob eine Präprozessorkonstante definiert ist und wenn ja, welche. Im rechten Teil von Bild 5 sind fünf Testfälle für diese Funktion angegeben: Bei Testfall 1 ist keine Präprozessorkonstante definiert, die Eingabe ist 3, damit wird der then-Zweig der if-Anweisung durchlaufen und der Return-Wert, die Ausgabe, ist 3. Bei Testfall 4 ist die Präprozessorkonstante VARIANT_1 definiert, der Input ist 0, damit wird der else-Zweig durchlaufen und die Ausgabe ist 12 (=12/(0+1)). Man erkennt, dass der else-Zweig der if-Anweisung nicht durchlaufen wird, wenn die Präprozessorkonstante VARIANT_2 definiert ist. Somit ist die Branch Coverage für diese Variante nicht 100%. Da der else-Zweig für alle Varianten identisch ist, könnte man auf die Idee kommen, dass der else-Zeig der if-Anweisung ja bereits in den anderen Varianten durchlaufen wurde. Somit könnte man die zu geringe Coverage in einer Variante durch die Coverage der anderen Varianten auf 100% „ergänzen“. Dass

dies keine gute Idee ist, zeigt sich, wenn man den noch fehlenden Testfall für Variante 2 durchspielt: Die Eingabe 3 bei definierter Präprozessorkonstante VARIANT_2 führt zu einer Division durch null. Die kumulierte Coverage der drei Varianten darf also nicht 100% betragen.

No.	Name	C1	Test Cases	Result
	Hitex-E1	83.33 %	5 of 5 passed	✖
	Hitex-Examples	83.33 %	5 of 5 passed	✖
	Miscellaneous	83.33 %	5 of 5 passed	✖
	Variants	83.33 %	5 of 5 passed	✖
	func	83.33 %	5 of 5 passed	✖
	Cumulation-of-coverage	83.33 %	5 of 5 passed	✖
	Variant_0	100 %	2 of 2 passed	✔
1	func	100 %	2 of 2 passed	✔
	Variant_1	100 %	2 of 2 passed	✔
2	func	100 %	2 of 2 passed	✔
	Variant_2	50 %	1 of 1 passed	✖
3	func	50 %	1 of 1 passed	✖

Bild 6 Die kumulierte Coverage für die drei Varianten ist 83,33% (Auszug aus einem Testreport von TESSY)

Misst Coverage die Güte der Testfälle?

Weil die Überdeckungsmessung Berechnungen nicht ausreichend berücksichtig, weil Überdeckungsmessungen Auslassungen nicht finden kann und weil auch schlechte Testfälle oft zu 100% Coverage führen, kann man Coverage nicht zur Messung der Güte der Testfälle heranziehen. Eine Methode zur Beurteilung der Güte wäre beispielsweise der Mutationstest (oder Error Seeding, wie dieses Verfahren in ISO 61508 genannt wird).

Literatur- und Quellenverzeichnis

[TESSY] http://www.hitex.de/tessy: More about the unit testing tool TESSY.
[BEIZER] Beizer, Boris: Software Testing Techniques, 2nd edition, New York, 1990.
[LIGGESMEYER] Liggesmeyer, Peter: Software-Qualität: Testen, Analysieren und Verifizieren von Software. Heidelberg, Berlin, 2002. Spektrum Akademischer Verlag.
[SPILLNER] Spillner, A., Linz, T.: Basiswissen Softwaretest, Heidelberg, 2003. dpunkt-Verlag.
[61508] IEC 61508, Functional Safety of electrical/electronic/programmable electronic safety-related systems, 2010.
[26262] ISO 26262, Road Vehicles – Functional Safety, 2011.
[DO-178C] Software Considerations In Airborne Systems And Equipment Certification, RTCA, 2011.
[CAST-10] Certification Authorities Software Team, Position Paper 10, 2002

Autor
Frank Büchner hat ein Diplom in Informatik von der Technischen Hochschule Karlsruhe, heute KIT. Seit mehreren Jahren widmet er sich dem Thema Testen und Software-Qualität. Seine Kenntnisse vermittelt er regelmäßig durch Vorträge und Fachartikel. Momentan arbeitet er als „Principal Engineer Software Quality“ bei der Fa. Hitex GmbH in Karlsruhe.

Kontakt
Internet: www.hitex.de
Email: frank.buechner@hitex.de

Tel.: +49 / 721 / 9628 – 125