Grundlagen der Code-Coverage-Messung

Dr. Sabine Poehler
Produktmanagerin Testwell Tools
Verifysoft TECHNOLOGY GmbH
Offenburg, Deutschland
poehler@verifysoft.com

Dieser Beitrag bietet Neuanwendern einen Einstieg in das Thema „Code Coverage“. Neben einem systematischen Überblick über die verschiedenen theoretischen Ansätze werden konkrete Entscheidungshilfen für den Einsatz von Coverage Werkzeugen diskutiert.

I. EINFÜHRUNG

Der Nachweis der Code Coverage wird für den Test sicherheitskritischer Software in einer Reihe von Sicherheitsnormen gefordert oder empfohlen. Dabei werden, abhängig von der Sicherheitsklasse der Software, verschiedene starke Coverage Maße festgelegt.

Die Messung erfolgt in der Realität immer über den Einsatz geeigneter Tools, die in den bestehenden Entwicklungsprozess integriert werden müssen.

II. VERSCHIEDENE CODE-COVERAGE-MAßE

Aus theoretischen Überlegungen einerseits und für praktische Zwecke andererseits hat sich eine Reihe verschiedener Coverage Maße etabliert. Mit unterschiedlichem Schwerpunkt definieren sie alle eine Antwort auf die Kernfrage:

Wurden alle Bereiche oder Strukturen des Quellcodes vollständig von den durchgeführten Tests durchlaufen?

Sie unterscheiden sich in der Definition, was eine relevante Struktur ist.

A. Function Coverage

Die Function Coverage ist das schwächste der üblichen Coverage Maße. Vollständige Function Coverage ist erreicht, wenn jede Funktion des Programms während des Tests aufgerufen wurde.

B. Statement Coverage

Unmittelbar zugänglich ist das Maß Statement Coverage. Gemessen wird, welcher Anteil der Anweisungen in einem Programmabschnitt bzw. im gesamten Programm ausgeführt wurde.

C. Decision Coverage/ Branch Coverage

Um Decision Coverage zu erreichen, muss jede Entscheidung während des Tests als „wahr“ und als „falsch“ ausgewertet werden. Bei typischen Entscheidungen, z.B. innerhalb von if-Statements, entspricht das den gängigen Definitionen der Branch Coverage, für die jeder Zweig des Programms durchlaufen werden muss.

In der Norm DO-178C (Luftfahrt) ist eine Entscheidung (Decision) allerdings sehr allgemein als Boolescher Ausdruck definiert. Hier muss auch jede Zuweisung eines Booleschen Ausdrucks zu einer Variablen entsprechend getestet werden.

Praktisch relevanter ist die Abgrenzung zur Statement Coverage: Um Decision oder Branch Coverage zu erreichen, müssen auch Zweige ohne Anweisungen durchlaufen werden. Für ein if-Statement ohne else-Block muss dennoch der Fall getestet werden, dass die zugehörige Entscheidung falsch ist.

Aus (voller) Decision Coverage folgt volle Statement Coverage, in diesem Sinn ist sie das stärkere Maß.

D. Condition Coverage

Über die Forderungen der Decision Coverage hinaus werden nun zusammengesetzte Entscheidungen im Detail betrachtet. Besteht eine Entscheidung aus mehreren, über Boolesche Operatoren zusammengesetzten atomaren Bedingungen, dann muss jede einzelne atomare Bedingung als „wahr“ und als „falsch“ ausgewertet werden.

Die Condition Coverage ist im Wesentlichen nur relevant für das Verständnis der Multicondition Coverage und insbesondere der Modified Condition/ Decision Coverage (MC/DC).

E. Multicondition Coverage & MC/DC

Um Multicondition Coverage zu erreichen muss für eine zusammengesetzte Entscheidung jede mögliche wahr-falsch-Kombination der atomaren Bedingungen getestet werden.

Das führt in der Realität schnell zu einer nichtpraktikablen Anzahl an Testfällen, da diese Anzahl exponentiell mit der Anzahl der Bedingungen in der Entscheidung wächst.

Daher wird die Multicondition Coverage in keinem Sicherheitsstandard gefordert, sondern die Modified Condition/ Decision Coverage (MC/DC). Mit ihr steigt die Anzahl der Testfälle nur noch linear mit der Anzahl der Bedingungen, gleichzeitig ist sie aber schwieriger zu verstehen und auch schwieriger zu ermitteln:

Für jede atomare Bedingung muss nachgewiesen werden, dass sie das Ergebnis der Entscheidung unabhängig von den anderen Bedingungen beeinflussen kann.

Dazu muss unter den Testfällen ein Paar von wahr-falsch-Kombinationen vorkommen mit drei Eigenschaften:

- Die betrachtete Bedingung ist einmal wahr und einmal falsch,
- alle anderen Bedingungen haben den gleichen Wert,
- das Ergebnis der gesamten Entscheidung ist einmal wahr und einmal falsch.

Beispiel:

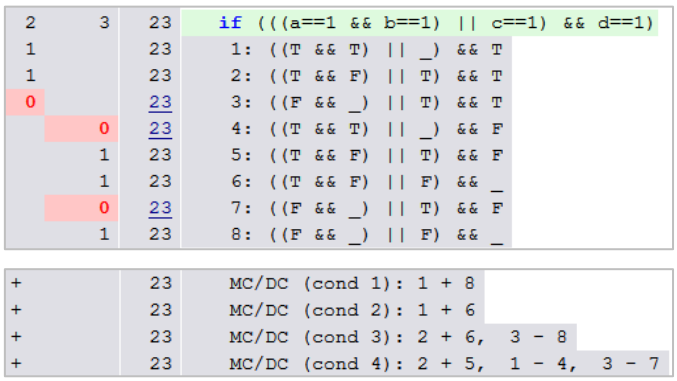

Fig. 1. Testwell CTC++: Multicondition & MC/DC

Hier wurde keine Multicondition Coverage erreicht, denn dazu fehlen drei wahr-falsch-Kombinationen.

MC/DC ist allerdings für diese if-Statement voll gegeben, da für jede der vier atomaren Bedingungen ein passendes Paar getestet wurde (z.B. für die Bedingung c==1 das Paar 2 & 6, aber nicht das mögliche Paar 3 & 8).

F. Path Coverage

Im Unterschied zu den (Multi-)Condition Coverage Maßen wird bei verschiedenen Konzepten der Path Coverage nicht die einzelne Entscheidung im Detail, sondern der gesamte Programmablauf betrachtet. Im Kern wird hier gefordert, dass jeder mögliche Pfad durch das Programm während des Tests durchlaufen wurde.

Das ergibt praktisch und auch theoretisch eine zu hohe Anzahl von Testfällen, entsprechend existiert eine Reihe von abgeschwächten Path-Coverage-Maßen. Allerdings wird in keiner gängigen Sicherheitsnorm der Nachweis von Path Coverage gefordert, sie ist also von eher theoretischem Interesse.

III. Anforderung in den verschiedenen Sicherheitsnormen

Die gängigen Normen definieren Sicherheitsklassen der zu testenden Software, abhängig vom möglichen Schaden, den die Software im Fehlerfall verursachen (oder nicht verhindern) kann.

Diese Sicherheitsklassen beeinflussen das Code-Coverage-Maß, das verlangt wird:

- In der ISO 26262-6 (Automobil) wird für die niedrigste Klasse, ASIL A, Statement Coverage gefordert, für die folgenden Klassen B und C Branch Coverage und für ASIL D, die höchste Klasse, MC/DC.
- In der DO-178C (Luftfahrt) ist E bzw. D die niedrigste Sicherheitsklasse, für beide Klassen wird keine Code Coverage Messung verlangt. Danach folgt die Statement Coverage für Klasse C, Decision Coverage für Klasse B und MC/DC für die höchste Klasse A.
- In der Medizintechnik (IEC 6203) wird Code Coverage zwar empfohlen, aber nicht explizit gefordert.

IV. Grundlegende Ansätze zur Messung der Code Coverage

Werkzeuge zur Code Coverage Messung unterscheiden sich wesentlich im Konzept, wie die Messung ermöglicht wird:

- Instrumentierung des Quellcodes: Relevante Strukturen des Quellcodes werden mit Zählern angereichert. So entsteht eine instrumentierte Version des zu testenden Programms, bei dessen Ausführung „mitgezählt“ wird.
- Instrumentierung des Binärcodes: Auch hier wird eine andere, instrumentierte Version des Programms erstellt, allerdings dient der Binärcode als Grundlage und nicht der Quellcode.

Beide Varianten verändern das zu testende Programm, entsprechend muss nachgewiesen werden, dass das für die Ausführung nicht relevant ist. Dazu werden üblicherweise die gleichen Tests gegen beide Versionen ausgeführt und kontrolliert, dass sich die Testresultate nicht verändern.

Darüber hinaus existieren auch Ansätze, Code Coverage ohne Instrumentierung zu messen, z.B. über das Auslesen und Analysieren von Trace-Daten.

V. Einsatz von Tools in der Praxis

Wird die Code-Coverage-Messung neu eingeführt, dann stehen zunächst die inhaltlichen Kernthemen im Vordergrund: Welches Coverage-Maß wird eingesetzt? Wie müssen die Berichte interpretiert werden? Deckt sich das eigene Verständnis der Coverage Maße mit dem der relevanten Norm und mit dem des Toolherstellers?

Während in den Sicherheitsnormen immer implizit vollständige Coverage gemeint ist, wird die Coverage durch die Werkzeuge normalerweise als prozentuales Verhältnis ausgegeben.

Die genaue Definition, welche Größen dabei ins Verhältnis gesetzt werden, ist nicht standardisiert: Bei Statement Coverage ist es recht offensichtlich die Anzahl der ausgeführten Statements im Verhältnis zu allen Statements. Die Tools unterscheiden sich höchstens in Detailfragen, was *ein* Statement ist. Doch bei den höheren Abdeckungsmaßen gibt es viele gleichberechtigte Zählweisen. Damit sind Coverage-Kennzahlen unterhalb von 100% zwischen verschiedenen Tools schwer vergleichbar.

Einen weit größeren Raum nimmt aber die konkrete Integration des Werkzeugs in die bestehende Organisation ein:

- Wer ist verantwortlich für die Coverage Messung?
- Wie wird das Tool in den Build-Prozess integriert?
- Welche Anwender müssen damit arbeiten?
- Findet die Messung der Code Coverage im Rahmen manuell ausgeführter Tests statt? Oder hochautomatisiert, z.B. bei jedem Build + Testlauf im Rahmen eines Continuous Integration Prozesses?

Diese Themen können bereits bei der Auswahl eines Code-Coverage-Tools als kritische Entscheidungspunkte berücksichtigt werden. Sie sind einerseits für die langfristige Zufriedenheit der Anwender relevant, und andererseits unterscheiden sich verfügbare Tools in diesen Punkten fundamental:

A. Relevante Coverage Maße

Die einfachste Frage ist, ob das Werkzeug die relevanten Coverage Maße ermitteln kann. Dabei sind nicht nur die aktuellen Sicherheitsanforderungen zu berücksichtigen, sondern auch mögliche zukünftige. Ein Ausschlusskriterium ist häufig die MC/DC Coverage bei kostenlosen oder sehr preiswerten Tools.

B. Unterstützung von Compilern und Sprachen

Vordergründig genauso einfach ist die Frage, welche Compiler und welche Programmiersprachen unterstützt werden. Doch gerade bei den Compilern versteht jeder Hersteller unter Unterstützung etwas anderes: Unterstützung durch einfache Konfiguration? Wird eine Nachlizensierung notwendig? Werden ausgewählte Compiler unterstützt oder grundsätzlich alle? Wie wird mit compiler-spezifischen Sprachkonstrukten umgegangen?

C. Nachvollziehbarkeit der Berichte

Weit weniger konkret, und nur durch Evaluierung der Tools zu beantworten: Lassen sich die Berichte für den eigenen Code und die eigenen Tests nachvollziehen? Vor allem, wenn keine vollständige Coverage erreicht wurde? Kann der Tester auf Quellcodeebene einfach erkennen, welche Testfälle noch fehlen?

D. Instrumentation Overhead

Kommt ein instrumentierendes Tool zum Einsatz, ist der Instrumentation Overhead für Systeme mit wenig Speicher relevant: Wird das instrumentierte Programm zu groß? Wieviel Arbeitsspeicher wird zusätzlich verbraucht?

Pauschalaussagen („20% – 30%“) sind hier bei Anwendern und Herstellern gleichermaßen sehr beliebt – und wenig hilfreich. Die Spannweite ist riesig und hängt von so vielen Parametern ab, dass nur ein Test am eigenen Code wirklich Auskunft gibt.

E. Integration & Integrierbarkeit

Einige Werkzeuge sind voll integriert in ein Testsystem, andere sind Stand-Alone-Tools. Ausgehend von dieser grundlegend verschiedenen Ausrichtung ist relevant, welche Integrationen, z.B. in IDEs, von vorneherein zur Verfügung stehen – und noch entscheidender, wie gut sich das Tool in automatisierte Abläufe *integrieren lässt.*

Vollintegrierte Tools werden naturgemäß bei der Integration in das entsprechende System ihre Vorteile ausspielen, Stand-Alone-Tools dafür bei der Integrierbarkeit in Software anderer Hersteller.

F. Einfachheit!

Kaum ein Anwender misst die Code Coverage völlig freiwillig. Das Tool, das zur Messung zum Einsatz kommt, sollte ihm die vielleicht ungeliebte Aufgabe so einfach und leichtgängig wie irgendwie möglich machen.

Als weiches, schwer zu prüfendes Kriterium gerät das bei der Auswahl gerne aus dem Blick – spielt aber langfristig für die Akzeptanz bei den Anwendern eine entscheidende Rolle.

VI. Inhaltliche Bedeutung im Entwicklungs- und Testprozess

Neben dem eigentlichen Zweck der Code Coverage Messung gibt es durch ihren Einsatz einige positive Effekte in der Softwareentwicklung.

Obwohl das Konzept auf die Vollständigkeit von Tests abzielt, schleichen sich spätestens mit der MC/DC auch Anforderungen an den Code ein: Überflüssiger Code, konkret: redundante Bedingungen, machen beim Testen das Erreichen vollständiger Coverage unmöglich. Hier entsteht also im Idealfall durch eine Rückkoppelung vom Testen in die Entwicklung einfacherer Code.

Auf einer höheren Ebene wird das häufig schwierige inhaltliche (und organisatorische!) Zusammenwirken aus Anforderungen, Entwicklung und Testen enger verzahnt – allerdings nur, wenn aus fehlender Code Coverage keine voreiligen Schlüsse gezogen werden, sondern alle drei Möglichkeiten geprüft werden:

- Sind die Tests nicht vollständig? Das ist die naheliegende Vermutung, die allerdings häufig nicht stimmt.
- Gibt es tatsächlich überflüssigen oder falschen Code, der nichts mit den Anforderungen zu tun hat?
- Gibt es *sinnvollerweise* zusätzlichen Code, der von den Anforderungen und daher auch von den Tests nicht abgedeckt wird, sind also die Anforderungen unvollständig?

Die beiden Aspekte, implizite Verbesserung der Code-Qualität und engere Verzahnung von Anforderungen, Entwicklung und Tests, lassen sich als Kollateralnutzen der Code-Coverage-Messung betrachten. Tatsächlich wird sie nicht nur klassisch in der sicherheitskritischen Softwareentwicklung eingesetzt, sondern auch gezielt für derartige interne Qualitäts- und Prozessverbesserungen.