CODE COVERAGE – TOOLS

GÜRKAN AYDIN UND HARTMUT POHL

Als Code Coverage (CC) wird der durch automatisierte Tests abgedeckte Quellcodeanteil eines Zielprogramms bezeichnet. CC ist nützlich, um z. B. aufzuzeigen wie vollständig der Quellcode mit Tests abgedeckt wurde. Code Coverage ist neben funktionalen Tests insbesondere relevant für Sicherheitstests.

Tool-gestützte Ermittlung der Code Coverage

Code Coverage dient damit der Bewertung der Vollständigkeit von Tests und der Effizienzsteigerung beim Testen - indem doppelte Tests vermieden und ungetestete Quellcode-Bereiche identifiziert werden. Ein Code Coverage-Tool (CC-Tool) bewertet die Abdeckung über einen ausgewählten Quellcode-Teilbereich oder auch über den gesamten Quellcode eines Zielsystems.

CC-Tools liefern die folgenden Informationen:

- Quellcode-Abdeckung in Prozent des untersuchten Quellcode-Bereichs,
- Identifizieren und markieren ungetesteter Teile des Quellcode-Bereichs,
- Ermittlung der CC nach Normen und Standards (z.B. IEC/DIN EN 61508).

So können im Falle eines Java-Programms automatisierte Tests durch ein Framework wie JUnit erstellt werden. Während das Framework JUnit für Java-Programme bei der Ausführung automatisierter Tests den Testerfolg bewertet, misst das CC-Tool, wie viel (in Prozent) von dem gesamten oder ausgewählten Quellcode-Bereich überhaupt ausgeführt wurde. JUnit kennt naturgemäß jedoch nur den Quellcode-Bereich, der ihm vorgegeben wurde. Zu 100 % erfolgreich durchlaufene Tests eines Bereichs müssen also nicht 100 % Code Coverage eines Zielsystems entsprechen.

Eine Norm wie IEC/DIN EN 61508 fordert die Ermittlung der Code Coverage bei einem Software-Projekt. Die Norm gibt vor, welche Verfahren dazu (als Metriken bezeichnet) zum Einsatz kommen. Norm und Verfahren sowie der Standard DO-178B werden im Folgenden erläutert.

IEC/DIN EN 61508

Die IEC/DIN EN 61508 ist eine Sicherheitsgrundnorm, die bei der Softwareentwicklung zum Einsatz kommen kann. Die Norm dient als Grundlage für Branchen, für die keine entsprechende eigene Norm existiert. Darüber hinaus kann sie ergänzend in Branchen wie Automotive, Maschinen und Medizingeräte genutzt werden, in denen bereits entsprechende Normen existieren. Ferner kann sie als Vorlage für neue branchenspezifische Normen dienen.

Im Abschnitt, „Anforderung an Software“ wird darauf eingegangen, dass die Code Coverage durch strukturabhängige Tests ermittelt werden sollte. Es wird eine CC von 100 % verlangt; sollte eine solche CC nicht erreicht werden, so ist dies entsprechend zu begründen und zu dokumentieren. Das einzusetzende CC-Verfahren ist einem Sicherheits-Integritätslevel (Safety Integrity Level - SIL) zugeordnet. Das SIL ist 4-stufig formuliert. Mit höherem Sicherheits-Integritätslevel steigen die Anforderungen an die Wirksamkeit einer Sicherheitsfunktion und an die einzusetzenden CC-Verfahren. Die folgenden, dem aktuellen Stand der Technik entsprechenden - von der IEC empfohlenen CC-Metriken - sind mit den jeweiligen SIL verknüpft:

- SIL 1: Eingangspunktabdeckung (Function Coverage),
- SIL 2: Anweisungsabdeckung (Statement Coverage),
- SIL 3: Verzweigungsabdeckung (Decision Coverage) und
- SIL 4: Modified Condition/Decision Coverage (MC/DC).

Als weitere (Ersatz-)Verfahren werden genannt:

- Linear Code Sequence and Jump (LCSAJ),
- Datenfluss und
- Basis-Pfad.

Diese Verfahren können durch *gleichwertige* Verfahren ersetzt werden. Jedoch muss dann der Nachweis erbracht werden, dass die gleichwertigen Verfahren auch gleichwertige Ergebnisse erzielen. Als Beispiel sei hier das Term Coverage genannt, das eine strengere Form des Modified Condition/ Decision Coverage darstellt und es ersetzen könnte (Schmidberger 2012).

Je höher das SIL desto höher ist die Anforderung an das Verfahren zur Ermittlung der CC. Tabelle 1 zeigt die Zuordnung der SIL zu dem jeweiligen CC-Verfahren. In der niedrigsten Stufe SIL 1 ist es ausreichend, die CC anhand von Function Coverage zu ermitteln. Für SIL 2 wird zusätzlich das Statement Coverage empfohlen, für SIL 3 das Decision Coverage und SIL 4 das Modified Condition/Decision Coverage. Hierbei ist zu beachten, dass ein durch ein doppeltes „+“ gekennzeichnetes Verfahren bzw. die SIL-Kombination auf alle Fälle eingesetzt werden sollte.

Tabelle 1: *Zuweisung SIL zu CC-Verfahren*

Verfahren/Maßnahme		SIL 1	SIL 2	SIL 3	SIL 4
1.	Strukturabhängige Tests mit einer Testabdeckung (Function Coverage) 100 %	++	++	++	++
2.	Strukturabhängige Tests mit einer Testabdeckung (Statement Coverage) 100 %	+	++	++	++
3.	Strukturabhängige Tests mit einer Testabdeckung (Decision Coverage) 100%	+	+	++	++
4.	Strukturabhängige Tests mit einer Testabdeckung (Modified Condition/Decision Coverage) 100%	+	+	+	++
++	Verfahren/Maßnahme wird diesem SIL besonders empfohlen. Bei Nicht-Verwendung muss der Grund für die Nicht-Verwendung im Einzelnen angegeben werden.				
+	Verfahren/Maßnahme wird diesem SIL empfohlen.				

DO-178B

Der Standard DO-178B „Software Considerations in Airborne Systems and Equipment Certification“ ist das Ergebnis einer Zusammenarbeit der European Organisation for Civil Aviation Equipment (EUROCAE) und Radio Technical Commission for Aeronautics (RTCA); er wird in Europa unter der Bezeichnung ED-12B geführt. Darin wird der Lebenszyklus von Software für Avioniksysteme behandelt. Es handelt sich um einen weltweit anerkannten Standard, um die Sicherheit in Avioniksystemen zu gewährleisten und dient der Zertifizierung von Software für Flugzeugsysteme. Die erste Version DO-178 stammt aus dem Jahr 1980 - 1985 erschien die Version DO-178A. Seit 1992 gilt die Version DO-178B. Die DO-178C ist am 1. Mai 2012 erscheinen.

Die Relevanz ergibt aus den Abschnitten 6.4.4.2, 11.14 und Tabelle A-7 des DO-178B in denen die Anforderungen an die Code Coverage definiert werden, indem die zu verwendenden Metriken benannt werden und weiterhin bestimmt wird, wie die Ergebnisse aufbereitet werden müssen.

DO-178B teilt die Sicherheit von Flugzeugsystemen in Failure Condition Kategorien ein. Jede dieser Kategorien ist mit einem Software Level verknüpft. Software Level (SW-Level) definieren die Anforderungen zum Erhalt der Sicherheit. In Tabelle 1 sind die benannten Kategorien mit dem zugehörenden Software Level aufgelistet.

Tabelle 2: *Verknüpfung zwischen Failure Condition Kategorie und Software Level*

Nr.	Failure Condition Kategorie	Software Level
1.	Katastrophal	A
2.	Gefährlich/Schwer	B
3.	Wichtig	C
4.	Gering	D
5.	Kein Effekt	E

Tabelle 3 zeigt die ausschnittsweise Zuordnung der Code Coverage-Metriken zu den SW-Level, die einer Failure Condition Kategorie wie ‚katastrophal‘, ‚gefährlich/schwer‘ und ‚wichtig‘ zugeordnet sind aus dem Standard der RTCA. Für die Software Level D und E müssen keine Code Coverage Messungen vorgenommen werden, da diese nicht als sicherheitskritische Bereiche angesehen werden. Für Software Level C (Punkt 7) muss das Code Coverage mittels Statement Coverage erfolgen. Software Level B erfordert das Code Coverage mittels Descision Coverage und die Erfüllung der Anforderungen aus Software Level C. Die Anforderungen in Punkt 8, „data coupling“ und „control coupling“, können nicht durch ein Code Coverage-Tool bearbeitet werden und sind deshalb auch nicht relevant. Software Level A erfordert Code Coverage mittels

Tabelle 3: V*erification Process Results (RTCA/DO-178B 1992)*

Object			Applicability by SW Level				Output		Control Category by SW Level			
Description		Ref.	A	B	C	D	Description	Ref.	A	B	C	D
5.	Test Coverage of Software Structure (MC/DC) achieved	6.4.4.2	X				Software Verification Results	11.14	Y			
6.	Test Coverage of Software Structure (Decision Coverage) achieved	6.4.4.2a 6.4.4.2b	X	X			Software Verification Results	11.14	Y	Y		
7.	Test Coverage of Software Structure (Statement Coverage) achieved	6.4.4.2a 6.4.4.2b	X	X	O		Software Verification Results	11.14	Y	Y	Y	
Legend:		X = The object should be satisfied with independence O = The object should be satisfied Y = Data satisfied the objective of Control Category 2 (CC2)										

Modified Condition / Decision Coverage sowie die Erfüllung der Anforderungen aus Software Level B.

Die Dokumentation der Messergebnisse ist ein obligatorischer Schritt zur Zertifizierung nach DO-178B. So müssen alle Software Verification Results, die durch Software Verification Process Activities entstehen, lückenlos dokumentiert werden. Nach (RTCA/DO-178B 1992) hat dies folgende Bedeutung:

- Alle Überprüfungen, Analysen und Tests, die während des Software Verification Process entstehen, ob erfolgreich oder fehlgeschlagen sowie das erfolgreiche oder fehlgeschlagene Endergebnis,
- Benennung des überprüften, analysierten oder getesteten Zielobjekts (Testgegenstand oder Softwareversion),
- Enthält alle Überprüfungen, Analysen und Tests.

Funktionsweise der Metriken

Die CC-Metriken, gehören zu den kontrollflussorientierten Testverfahren (Hayhurst 2001) und entsprechen den Vorgaben der Norm IEC/DIN EN 61508. Abbildung 1 zeigt den hierarchischen Aufbau der Klasse der dynamischen Software-Testverfahren, zu denen die kontrollflussorientierten Testverfahren gehören und einige der dazugehörigen CC-Metriken. Die für das CC relevanten Punkte sind blau hervorgehoben.

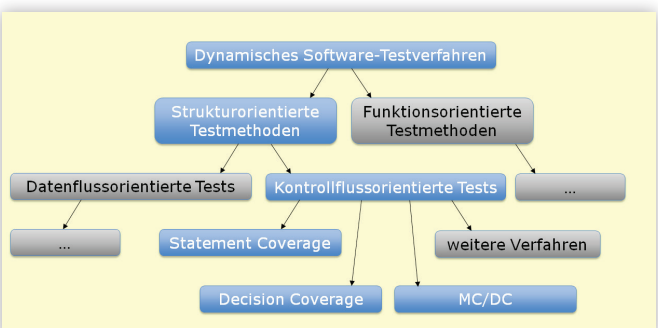

Abbildung 1: *Vereinfachte Hierarchie Dynamisches Software-Testverfahren (nach Liggesmeyer 2009)*

Die Metriken ermitteln die CC - mithilfe von Testframeworks - indem Testobjekte ausgeführt werden (Spillner 2004). Ergebnis ist der Prozentsatz der Quellcode-Abdeckung und damit die Grundlage für die Identifizierung und Markierung von getestetem bzw. ungetestetem Quellcode. Die genaue Funktionsweise der Metriken wird anhand des Statement Coverage und Decision Coverage durch Beispiele (Spillner 2004) unten erläutert.

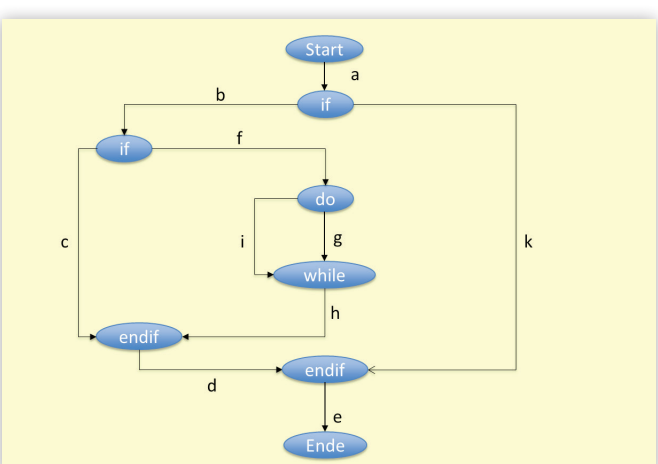

Abbildung 3: *Beispiel eines Kontrollflussgraphen (Spillner 2004)*

Statement Coverage

Beim Statement Coverage wird die CC anhand der Zahl Anweisungen gemessen. Die Abdeckung wird nach der folgenden Formel berechnet.

Anhand eines Kontrollflussgraphen eines einfachen Programmstücks wird die Funktionsweise näher erläutert. Das Programmstück besteht aus zwei „if"-Abfragen und einer „do-while"-Schleife (Abbildung 3).

Um eine hundertprozentige CC zu erreichen, muss jede Anweisung aus dem Kontrollflussgraphen durch einen oder mehrere Testfälle abgearbeitet werden. In dem Kontrollflussgraphen stellen die Knoten die Anweisungen und die Kanten den Kontrollfluss dar. Werden in Abbildung 3 die Kanten in der Reihenfolge „a, b, f, g, h, d, e" durchlaufen, so werden alle Knoten durchlaufen. Damit reicht ein Testfall aus und das Ergebnis des Statement Coverage liegt bei 100 %.

Ein Nachteil des Statement Coverage ist, dass mit einem Testfall nicht immer eine hundertprozentige Überdeckung erreicht wird, da nicht immer jede Anweisung durch Tests erreicht wird (Dead Code, Defensive Code). Ferner werden fehlende Anweisungen nicht erkannt (Spillner 2004).

Decision Coverage

Anders als beim Statement Coverage steht beim Decision Coverage der Kontrollfluss (Kanten) im Mittelpunkt. Um aufzuzeigen wie Decision Coverage funktioniert, dient auch hier der Kontrollflussgraph aus Abbildung 3 als Beispiel. Berechnet wird die Decision Coverage wie folgt.

Dem Kontrollflussgraph ist zu entnehmen, dass zehn Kanten durchlaufen werden müssen, um eine hundertprozentige Überdeckung zu erreichen. Es wird deutlich, dass durch einen Testfall nicht alle Kanten erreicht werden können, so dass weitere Tests notwendig sind, um alle Kanten zu durchlaufen. Die Testfälle sehen wie folgt aus:

- Testfall 1: a, b, f, g, h, d, e
- Testfall 2: a, b, c, d, e
- Testfall 3: a, b, f, g, i, g, h, d, e
- Testfall 4: a, k, e

Erst durch die vier Testfälle werden alle Zweige des Graphen durchlaufen und eine CC von 100 % erreicht. Die Mehrfachausführung von Kanten ist nicht vermeidbar.

Condition Coverage

Damit die Condition Coverage (CoC) eine hundertprozentige Code Coverage erreicht, muss bei den Bedingungen jeder Wert mindestens einmal true und einmal false sein. Bedingungen enthalten Relationssymbole („<", „>", „=" etc.); Operatoren (OR, AND, NOT) gehören nicht dazu. D. h., es werden die atomaren Teilbedingungen überprüft, wobei mehrere Teilbedingungen durch Operatoren zusammengesetzt sein können - z. B. in der Form: x > 3 OR y < 5.

Für die OR-Verknüpfung gilt die folgende Wahrheitstabelle:

Tabelle 4: *Wahrheitstabelle*

x	y	x OR y
true	true	true
true	false	true
false	true	true
false	false	false

Nach der Wahrheitstabelle aus Tabelle 4 und der Anforderung, dass jede atomare Teilbedingung mindestens einmal beide Wahrheitswerte annehmen muss, ergibt für x > 3 OR y < 5 für die Testdaten x = 6 und y = 8 true für die Teilbedingung x und false für die Teilbedingung y - damit ist die Gesamtauswertung für beide Bedingungen true. Durch einen weiteren Testfall mit den Testdaten x = 2 und y = 3 ist die Teilbedingung x false und die Teilbedingung y true. Die Gesamtauswertung ist für beide Bedingungen wahr. Durch die beiden Testfälle haben x und y jeweils beide Wahrheitswerte angenommen. Insgesamt ist das CoC aber eine schwache Metrik, da beide Testfälle als Gesamtbedingungen das Ergebnis wahr geliefert haben. Es ist nötig, dass die Gesamtbedingungen wahr und falsch als Ergebnis liefern.

Condition/Decision Coverage

Condition/Decision Coverage (C/DC) ist eine strengere Form des Condition Coverage. Durch das Berücksichtigen der Kombinationen der Teilbedingungen müssen alle Variationen untersucht werden, um eine 100%-ige Code Coverage zu erreichen. Dies wird anhand des Beispieldaten aus Kapitel 4.3 gezeigt. Durch die Kombination der Teilbedingungen ergeben sich $2^2 = 4$ Testfälle (Tabelle 5). Werden die vier Testfälle erzeugt, wird die Gesamtbedingung mindestens einmal den Wert true und einmal den Wert false annehmen.

Die Testdaten x = 6 und y = 3 für x > 3 OR y < 5 ergeben für einzelnen Teilbedingungen jeweils true – die Gesamtbedingung ist demnach ebenfalls true. In weiteren Tests werden die Testdaten so ausgewählt, dass jeweils die Teilbedingungen A und B den Wert false annehmen - dies hat nach der OR-Verknüpfung keine Auswirkung auf den Wert der Gesamtbedingung. Werden die Testdaten so ausgewählt, dass die beiden Teilbedingungen false sind, wird auch die Gesamtbedingung zu false. Damit sind für das Beispiel alle Testfälle abgedeckt.

Tabelle 5: *Testfälle für x > 3 OR y < 5 nach C/DC*

Testfall	Teilbedingung A	Teilbedingung B	Gesamtbedingung
1.	x = 6 (true)	y = 3 (true)	x > 3 OR y < 5 (true)
2.	x = 6 (true)	y = 8 (false)	x > 3 OR y < 5 (true)
3.	x = 2 (false)	y = 3 (true)	x > 3 OR y < 5 (true)
4.	x = 2 (false)	y = 8 (false)	x > 3 OR y < 5 (false)

Schnell wird aber auch der Nachteil deutlich, 2^n bei n atomaren Bedingungen beutet, dass die Zahl der Kombinationen exponentiell ansteigt. Bei zehn atomaren Bedingungen müssten $2^{10} = 1024$ Testfälle berücksichtigt werden. Wenn dies auf sicherheitskritische Systeme wie Flugzeuge, Atomkraftwerke etc. übertragen wird, ergeben sich bei einer komplexen Software schnell mehrere Millionen Testfälle.

Anzumerken ist, dass die erzeugten Testfälle die Kriterien des Statement Coverage und Decision Coverage erfüllen.

Modified Condition/Decision Coverage

Die vorangegangen Erläuterungen der Metriken haben die Voraussetzung zur Erläuterung des Modified Condition/Decision Coverage geschaffen. So ist das MC/DC eine verallgemeinerte Form des Condition/Decision Coverage. Beim MC/DC müssen nicht alle Kombinationen berücksichtigt werden, wie es bei C/DC der Fall ist. Es müssen nur die Kombinationen berücksichtigt werden, bei der die Änderung des Wahrheitswertes einer Teilbedingung den Wahrheitswert der Gesamtbedingung ändert. Für das Beispiel aus Kapitel 4.4 Tabelle 5 sieht das dann wie folgt aus:

Im ersten Fall der vier möglichen Fälle gilt bei x = 6 (true) und y = 3 (true), dann ist x > 3 OR y < 5 (true) – ein Testfall mit einem Wahrheitswert true für die Gesamtbedingung ist damit abgedeckt.

Wenn die erste oder die zweite atomare Teilbedingung von true auf false ändert, bleibt die Gesamtbedingung trotzdem noch true. Das heißt, ein Fehler hätte keine Auswirkung auf die Gesamtbedingung und es sind keine weiteren Testfälle nötig, bei der die Wahrheitswert true für die Gesamtbedingung ist. Damit entfallen die Testfälle 2 und 3 aus Tabelle 6.

Jedoch wird mindestens noch ein Testfall benötigt, durch die der Wahrheitswert der Gesamtbedingung zu false wird. Dies wird z. B. durch die Testdaten x = 2 (false) und y = 8 (false) für x > 3 OR y < 5 (false) erreicht.

Tabelle 6: *Testfälle für x > 3 OR y < 5 nach MC/DC*

Testfall	Teilbedingung A	Teilbedingung B	Gesamtbedingung
1.	x = 6 (true)	y = 3 (true)	x > 3 OR y < 5 (true)
2.	x = 6 (true)	y = 8 (false)	x > 3 OR y < 5 (true)
3.	x = 2 (false)	y = 3 (true)	x > 3 OR y < 5 (true)
4.	x = 2 (false)	y = 8 (false)	x > 3 OR y < 5 (false)

Für MC/DC sind nur die Fälle relevant, bei der eine Änderung in der Gesamtbedingung auftritt, sodass die Gesamtanzahl der Testfälle erheblich reduziert wird – es werden also nur Fehler mit Auswirkung berücksichtigt.

Nach MC/DC reichen zwei Testfälle aus, damit der Wahrheitswert des Gesamtausdrucks jeweils den Wert true und den Wert false annimmt. Die Herausforderung besteht nun darin zu erkennen, wann Fehlerzustände maskiert werden, damit ein Testfall wegfallen kann. Dies erlaubt es, die Anzahl der Testfälle auf einen Wert zwischen n + 1 und 2n, mit n = Anzahl der booleschen Operatoren der Bedingung ist, zu senken.

Reporting

Die Erzeugung eines Reports ist – neben der Ermittlung des Prozentsatzes der Code Coverage - ein wesentlicher Bestandteil eines CC-Tools. Die Ergebnisse müssen nach jedem Messdurchgang zusammengefasst und gespeichert werden. Damit wird ein Vergleich mit nachfolgenden Messungen möglich und der Fortschritt der CC kann aufgezeigt werden; ein Prüfprotokoll wird geführt und dient als ein Nachweis gegenüber Dritten wie Auftraggebern (Liggesmeyer 2009). Dabei sollen die Informationen knapp, genau und verständlich sein (Börcsök 2006). Des Weiteren müssen für unterschiedliche Personenkreise unterschiedliche Reports erstellt werden, z. B. Überblick verschaffende Reports für das Management oder detailliertere Reports für die Entwickler.

Beispiel CodeCover

Im Folgenden werden Beispiele anhand des Open-Source-Tools CodeCover erläutert. CodeCover liegt als Standalone- und Eclipse Plug-in-Lösung vor - für das Beispiel wird das Eclipse Plug-in verwendet.

Das Tool bietet insgesamt sechs Metriken an. Für das Beispiel werden die beiden Metriken Statement Coverage und Decision/Branch Coverage betrachtet. Die Klassen Fakultaet (Abbildung 5) und Fibonacci (Abbildung 6) beinhalten die zu testenden Funktionen – jeweils in iterativer und rekursiver Form. Getestet wird anhand von JUnit-Tests, wobei die Tests die Grundlage der Code Coverage-Ermittlung bildet.

Abbildung 7: *Testklasse zu den Funktionen Fibonacci und Fakultät*

```
public class Fakultaet {
	public int iterativ(int zahl){
		int erg = 1;
		for(int i = zahl;i>0;i--){
			erg = erg*i;
		}
		return erg;
	}
	public int rekursiv(int zahl){
		if(zahl>1){
			return zahl*this.rekursiv(zahl-1);
		}else{
			return 1;
		}
	}
}
Abbildung 5: Klasse Fakultät
public class Fibonacci {
	public int[] rekursiv(int anz){
		int[] ergebnis = new int[anz];
		this.rekursiv_sub(ergebnis,anz,0);
		return ergebnis;
	}
	private void rekursiv_sub(int[] erg, int anz, int pos){
		if(anz>0){
			if((pos < 1)){
				erg[0] = 0;
			}else if((pos == 1)){
				erg[1] = 1;
			}else{
				erg[pos] = erg[pos-1]+erg[pos-2];
			}
			this.rekursiv_sub(erg, anz-1, pos+1);
		}
	}

	public int[] iterativ(int anz){
		int[] ergebnis = new int[anz];
		if(anz>=0){
			ergebnis[0]=0;
		}
		if(anz>=1){
			ergebnis[1]=1;
		}
		if(anz>1){
			for(int i = anz-2;i>0;i--){
				ergebnis[anz-(i)]=ergebnis[anz-i-1]+ergebnis[anz-i-2];
			}
		}
		return ergebnis;
	}
}
Abbildung 6: Klasse Fibonacci
Die letzte Java-Klasse enthält eine Reihe von Tests zu den  zuvor genannten Funktionen.
import junit.framework.TestCase;
public class TestJunit extends TestCase{
	Fakultaet objFak;
	Fibonacci objFib;
	protected void setUp() throws Exception {
		super.setUp();
		objFak = new Fakultaet();
		objFib = new Fibonacci();
	}
	protected void tearDown() throws Exception {
		super.tearDown();
	}
	public void testFak(){
		int expected = 120;
		int actual = 0;

		try{
			actual = objFak.rekursiv(5);
		}catch (Exception e) {
			fail(„Falsches Ergebnis“);
			e.printStackTrace();
		}
		assertEquals(expected, actual);
	}
	public void testFib(){
		int[] expected = {0,1,1,2,3,5};
		int[] actual = {0,0,0,0,0,0};
		try {
			actual = objFib.iterativ(6);
		} catch (Exception e) {
			fail();
			e.printStackTrace();
		}
		for(int i=0;i<=5;i++){
			assertEquals(expected[i], actual[i]);
		}
	}
}
```

Sind die genannten Klassen bzw. Tests in einem Projekt angelegt, müssen die Voraussetzungen für die Ermittlung der Code Coverage geschaffen werden. Dazu muss in den Projekteinstellungen unter dem Menüpunkt „CodeCover“ die „Coverage Criteria“ aktiviert werden (Abbildung 8).

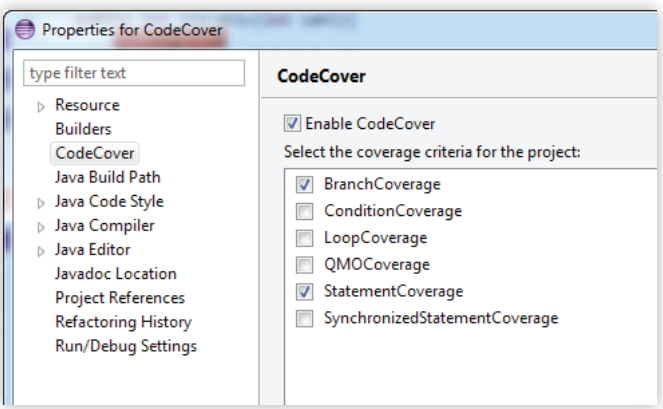

Abbildung 8: *Aktivieren der „Coverage Criteria“ für das Projekt*

Die Java-Klassen, deren Code Coverage ermittelt werden soll, werden durch einen Klick auf „Use For Coverage Measurement“ im Kontextmenü der jeweiligen Datei zur CodeCover hinzugefügt (Abbildung 9) - in diesem Fall sind es die beiden Java-Klassen Fakultaet.java und Fibonacci.java. Ein grün/rotes-Symbol markiert diese Dateien im Package Explorer. Damit sind alle Vorbereitungen getroffen, um die Code Coverage anhand der JUnit-Testklasse zu ermitteln.

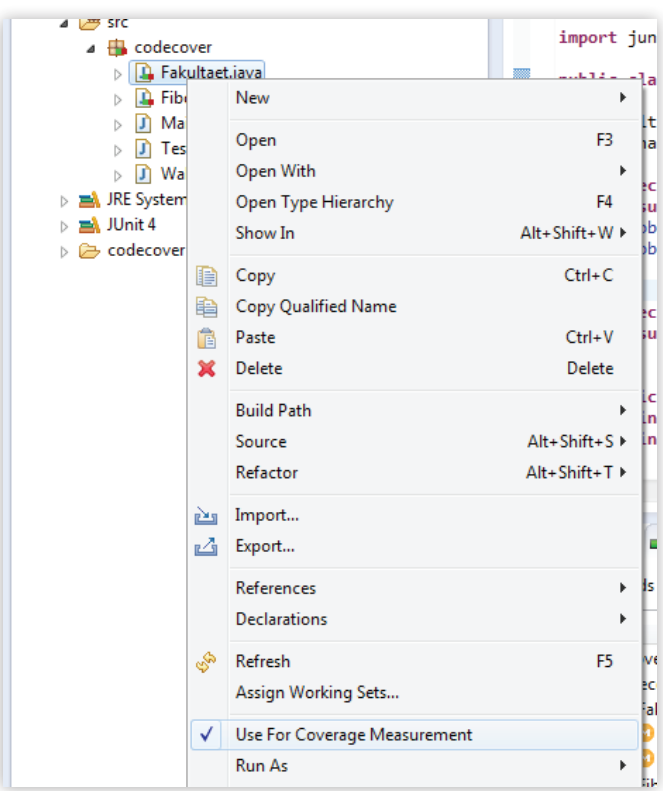

Abbildung 9: *Klasse zur Messung freigeben*

Für die Messung wird die Testklasse ausgewählt und mit „Run -> Run As -> CodeCover Measurement For JUnit“ gestartet. Nach dem Testdurchlauf wird ein „Test Session Container“ (Abbildung 10), der die ermittelten Code Coverage-Daten enthält, erstellt.

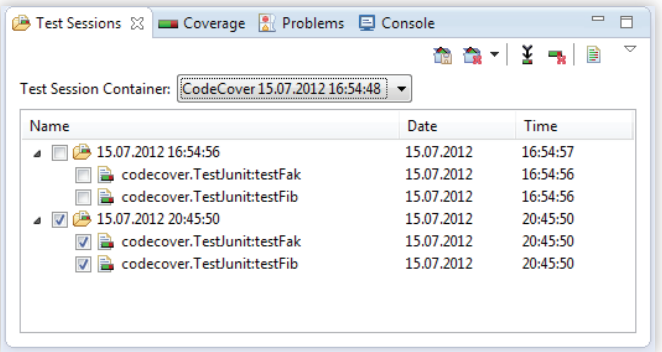

Abbildung 10: *Test Session Container*

Nach Auswahl von mindestens einem Test Session Container werden in der Coverage Konsole (Abbildung 11) die detaillierten Daten des Testdurchlaufs angezeigt. Die Coverage Konsole zeigt das Package, die Klasse und die jeweilige Funktion an für die eine Messung erfolgte. Die Metriken sind übersichtlich spaltenweise angeordnet. Durch den hierarchischen Aufbau vom Package zur Funktion hin, sind die einzelnen Messergebnisse für das gesamte Projekt oder aber auch für eine einzelne Funktion sehr gut ablesbar.

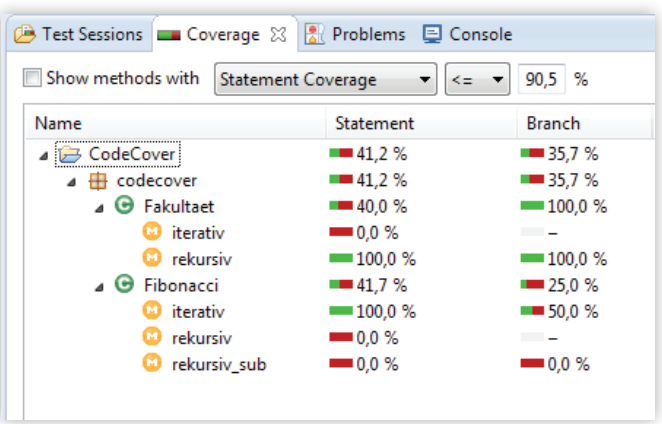

Abbildung 11: *Coverage Konsole*

Wenn noch detailliertere Informationen erforderlich sind, wie z. B. die exakte Code Coverage für bestimmte Funktionen oder aber auch für Zeilen, kann dies mit einem Klick auf die entsprechende Klasse oder Funktion in der Coverage Console aufgerufen werden. Die Klasse wird im Code-Fenster geöffnet und die entsprechenden Zeilen werden farblich hervorgehoben. Eine Überdeckung ist erfolgt, wenn die Zeile grün hinterlegt ist. Bei einer roten Hinterlegung, ist dies noch nicht erfolgt, da z.B. kein Testfall existiert oder der Testfall nicht ausreicht um alle Verzweigungen zu durchlaufen.

Abbildung 12: *Code Highlight*

```
public class Fakultaet {
	public int iterativ(int zahl){
		int erg = 1;
		for(int i = zahl;i>0;i--){
			erg = erg*i;
		}
		return erg;
	}
	public int rekursiv(int zahl){
		if(zahl>1){
		return zahl*this.rekursiv(zahl-1);
		}else{
			return 1;
		}
	}
}
```

Zusätzlich kann für jede Testsession und in jedem Testfall, wie in Abbildung 10 dargestellt Kommentare oder Informationen (Abbildung 13) hinterlegt werden. Hier kann dann z. B. festgehalten werden, warum die gewünschte Code Coverage mit den aktuellen Tests nicht erreicht werden konnte und Hinweise darauf gegeben werden, wie der nächste Test aussehen sollte, damit der gewünschte Überdeckungsgrad erreicht wird. Die Kommentare sind auf den ausgedruckten Reports sichtbar.

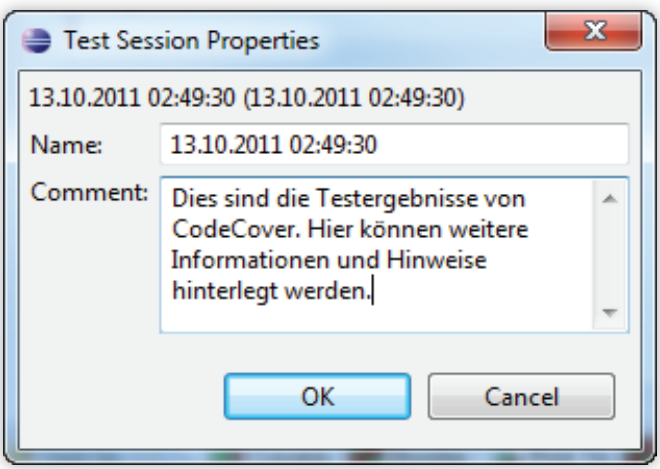

Abbildung 13: *Kommentarfunktion für die Test-Sessions*

Taxonomie

Einsatzzweck

CC-Tools lassen sich nach ihrem Einsatzzweck klassifizieren. Es lassen sich zwei Arten ableiten - die spezialisierten und die universellen CC-Tools.

Da die universellen CC-Tools nicht an Normen und Standards gebunden sind, sind sie auch nicht an die zuvor vorgestellten CC-Metriken gebunden. Viele Tools bieten hier Lösungen mit eigenen Metriken an.

Die Anforderungen spezialisierter CC-Tools ergeben sich aus den Normen und Standards. Diese CC-Tools richten sich an die strengen Vorgaben, welche die Metriken vorgeben, die das CC-Tool unterstützen muss. Das Ergebnis der Messungen und Auswertungen mit diesen CC-Tools muss ausreichen, um beispielweise den Zertifizierungsverfahren der Federal Aviation Administration (FAA) gerecht zu werden (RTCA/DO-178B 1992). Abbildung 5 stellt die erläuterte Taxonomie in hierarchischer Form dar.

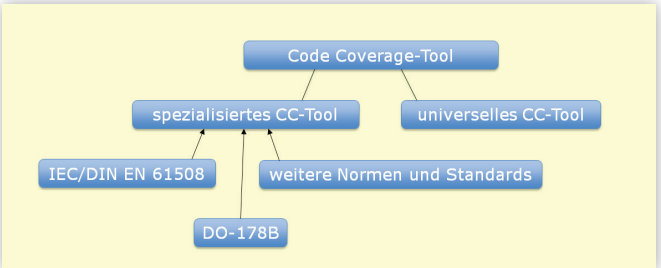

Abbildung 14: *CC-Tool-Taxonomie nach Einsatzzweck*

Programmiersprachen-Abhängigkeit der Tools

Kontrollflussorientiertes Testen zur Ermittlung der CC erfordert den unmittelbaren Zugriff auf den Quellcode. Zur Testfallkonstruktion werden Teile des Quellcodes herangezogen, daher muss die interne Struktur bekannt sein (Fenton 1997). Daraus folgt, dass kontrollflussorientiertes Testen zu den White-Box-Testverfahren zählt (vgl. Hoffmann 2008; Liggesmeyer 2009). Hieraus ergeben sich nach unterstützten Programmiersprachen unterscheidbare, unterschiedliche Arten von CC-Tools. Abbildung 6 zeigt eine Auswahl von Programmiersprachen, die von mehreren CC-Tools unterstützt werden. Bei der Auswahl eines CC-Tools muss die für die Softwareentwicklung benutzte Programmiersprache bekannt sein - unabhängig davon, ob ein spezialisiertes oder universelles CC-Tool benötigt wird.

Unterstützte Metriken

Ein markantes Merkmal von CC-Tools sind die unterstützten Metriken. Sie bilden die Grundlage der das Resultat der Messungen wiedergebenden Reports. Welche Metriken ein CC-Tool unterstützt, ist bei den universellen CC-Tools abhängig vom Hersteller. Häufig handelt es sich um identische Metriken, die sich nur in der Bezeichnung unterscheiden.

Die spezialisierten CC-Tools, die der Norm IEC/DIN EN 61508 entsprechen und damit auf dem aktuellen Stand der Technik sind, unterstützen die zuvor in Kapitel 4 vorgestellten Metriken (vgl. Spillner 2004; Liggesmeyer 2009).

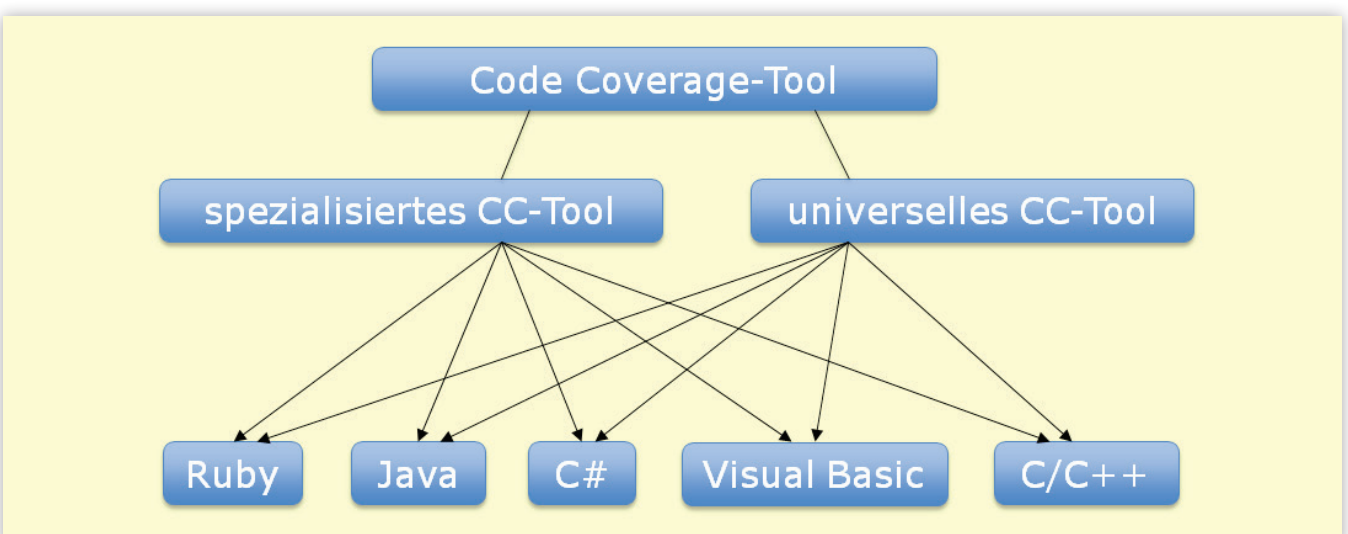

Abbildung 15: *CC-Tool-Taxonomie nach Programmiersprache*

Messmethoden

Es existieren mehrere Messmethoden der CC (vgl. Liggesmeyer 2009; Thaller 2002). Die wichtigste Methode ist die non-invasive Messung mit CC-Tools, mit nicht-instrumentierend arbeitenden CC-Tools: Bei der Messung werden keine Änderungen am Quellcode durchgeführt.

Im Gegensatz dazu steht die invasive, instrumentierende Messung. Bei der Instrumentierung werden im Quellcode oder (besser) in einer speziellen Kopie davon, zusätzliche Zähler eingefügt und initialisiert. Beim Durchlaufen der entsprechenden Stellen werden diese Zähler inkrementiert und im Anschluss ausgewertet.

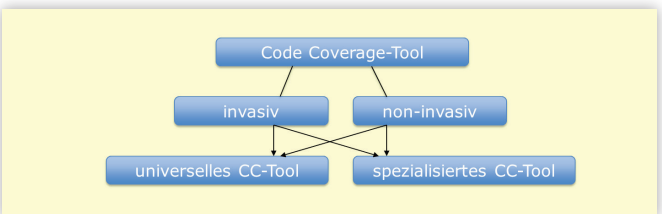

Abbildung 16: *CC-Tool-Taxonomie nach Messmethode*

Code Coverage-Tool-Auswahl

Die eingesetzte Programmiersprache schränkt die zur Auswahl stehenden Tools ein. Daher kann eine pauschale Empfehlung für ein CC-Tool nicht ausgesprochen werden.

Zusätzlich spielen die benötigten Metriken eine wichtige Rolle. Die Auswahl muss nach den individuellen Bedürfnissen des Nutzers oder des Kunden vorgenommen werden: Der gewissenhafte Entwickler, für den die Vollständigkeit seiner Tests wichtig ist, braucht ein anderes Tool als jemand der den Nachweis erbringen muss, dass die CC nach den Kriterien des IEC/DIN EN 61508 oder einer anderen Norm oder Standards ermittelt wurde.

Daher sollte anhand eines flexiblen Bewertungskatalogs das zu einem Projekt jeweils wirkungsvollste CC-Tool ausgewählt werden. Entscheidungsparameter können sein:

- Unterstütze Metriken,
- Unterstütze Normen und Standards,
- Eingesetzte Programmiersprache, Framework,
- Dokumentation,
- Anpassbarkeit,
- Schnittstellen,
- Kosten und Lizenzform sowie
- Usability.

Weiterführende Literatur

- Börcsök, J.: Funktionale Sicherheit, Heidelberg 2006
- Schmidberger, R.: CodeCover, Stuttgart 2012 *http://codecover.org*
- Hayhurst, K. J. et al.: A Practical Tutorial on Modified Condition/ Decision Coverage, Hampton 2001 *http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20010057789_2001090482.pdf*
- Hoffmann, D. W.: Software-Qualität, Berlin 2008
- IEC/DIN EN 61508: Funktionale Sicherheit sicherheitsbezogener elektrischer / elektronischer / programmierbarer elektronischer Systeme, Berlin 2011
- Liggesmeyer, P.: Software-Qualität, Heidelberg 2009
- RTCA/DO-178B: Software Considerations in Airborne Systems and Equipment Certification, Washington D.C. 1992
- Spillner, A.; Linz, T.: Basiswissen Softwaretest, Heidelberg 2004
- Thaller, G. E.: Software-Test – Verifikation und Validation, Hannover 2002