oder andere schwere Verletzungen der Spezifikation der Komponente. Dementsprechend viel Kritik musste das Originalverfahren einstecken. Eine gute Testabdeckung ist also kein Garant für gute Tests, wie auch der folgende Erfahrungsbericht zeigt.

Erfahrungsbericht Testabdeckung

Bei einer Zulieferfirma für die europäische Raumfahrt werden rigoros Unit-Tests eingesetzt. Als man bei einem Projekt zeitlich in Bedrängnis geriet, stellte man einen neuen Mitarbeiter zur Verstärkung ein. Seine Aufgabe war es, Unit-Tests zu machen. Das Projekt war nicht missionskritisch, die geforderte Testabdeckung war 100% Decision Coverage. Unit-Tests wurden im Projekt meist durch den Programmierer selbst gemacht. Um einen erfahrenen Programmierer für Implementierungsaufgaben freizubekommen, überließ man seinen Code dem neuen Mitarbeiter zum Unit-Test. Dieser hatte große Mühe, sich in den Code einzulesen. Also beschränkte er sich darauf, Tests zu schreiben, die zwar 100% Decision Coverage erreichten, die Aufgabe der zu testenden Funktionen hinterfragte er aber nicht weiter. Zum Zahlungsmeilenstein »Unit Test Completion« schien noch alles in bester Ordnung zu sein. Man hatte durch die Personalverstärkung Zeit aufgeholt, 100% Testabdeckung wurde erreicht, man war bereit, die Systemtests zu starten.

Bei den Systemtests erkannte das Team allerdings, dass gar nichts in bester Ordnung war. Die umfangreichen Tests fanden viele Fehler. Die Ursachenfindung im Zielsystem war aber sehr mühsam und erst nach einiger Zeit erkannte man, dass schlechte Unit-Tests für die ungewöhnlich hohe Fehlerquote verantwortlich waren. Eine Inspektion der Unit-Tests zeigte, dass der neue Mitarbeiter zwar 100% Testabdeckung erreicht hatte, aber keine Tests gegen das Design gemacht hatte. Seine Tests durchliefen die Software nur, aber testeten sie nicht. Man war über die Fahrlässigkeit des Mitarbeiters so verärgert, dass man sich von ihm nach kurzer Zeit wieder trennte.

Dieses Beispiel zeigt eine Stärke der Test-First-Strategie (siehe Abschnitt 1.5.13): Der Tester wird gezwungen, gegen das Komponentendesign zu testen. Die Testfälle beim TDD können per se den Code nicht bloß durchlaufen. Allerdings hätte man bei TDD auch nicht so einfach Zeit durch die Arbeitsteilung in Implementierung/Unit-Test aufholen können.

6.13.2 Organisation von Unit-Tests

Wenn das Projekt klein genug ist, dass man sich Bottom Up Unit Testing erlauben kann, dann ist das die ideale Form der Testorganisation. Man beginnt bei den Low-Level-Routinen und arbeitet sich Hierarchiestufe für Hierarchiestufe im Baum der Abhängigkeiten der Module hinauf bis an die Spitze. Die Integration der Module an den neu hinzugekommenen Schnittstellen wird gleichzeitig mit den Modulen selbst getestet. Die fixe Test- und Integrationsreihenfolge macht diese Testmethode allerdings in großen Projekten nicht umsetzbar.

Egal ob bei Isolationstests oder im Bottom-up-Verfahren, egal ob mit Werkzeug oder ohne: Die Testfälle sollten am besten so entworfen werden, dass sie