# *Refactoring und (J)Unit-Test im Projekt oskar*

Michael Brunner
Dresdner Bank AG, CC IT oskar
Jürgen-Ponto-Platz 1
60301 Frankfurt
Michael.Brunner@Dresdner-Bank.com

oskar (online strategic position keeping and reporting) wird das zentrale System für Disposition, Buchung und geldumsatzbezogene Informationen in der Dresdner Bank Gruppe. Mit oskar [5] werden zentrale Softwareteile der Bank abgelöst.

## Das Projekt oskar

Information steht bei oskar im Mittelpunkt, der bei Zahlungsvorgängen ausgelöste Geldumsatz wird als Träger für Buchungs- und Zusatzinformationen genutzt. Für die Bank ergeben sich Kostenvorteile durch Massenverarbeitung in Kombination mit individuellen Informationsservices. Als hocheffiziente Buchungsplattform dient oskar der Systementflechtung und ermöglicht den Einheiten der Bank höchste Flexibilität durch das Andocken geschäftsspezifischer Systeme. Echtzeitverarbeitung sowie zeitnahe und flexible Anpassungen durch komponentenbasierte Technologien und regelwerkbasierte Steuerung runden das Bild ab.

Ein besonderer Reiz des Projektes liegt in der Verbindung hochmoderner Technologien wie Java und XML mit der Systemumgebung der IBM-Großrechnersysteme (S/390, MQSeries, CICS, DB2). Die hier vorliegende Systemumgebung stellt aus Gründen der Betriebssicherheit noch geraume Zeit den technischen Standard für Kernsysteme von Banken dar [3].

Das UML-basierte Design bildet die Grundlage für das modulare, komponentenbasierte System, von dem insbesondere erhöhte Wartbarkeit und geringere Weiterentwicklungsaufwände zu erwarten sind.

Bei der Java-Entwicklung wurde von Beginn an großer Wert auf die Entwicklung aussagekräftiger Komponententests gelegt. Hierbei kommt das JUnit Framework[1] zum Einsatz, welches für einige spezielle Projektanforderungen angepasst wurde.

Der modulare, komponentenorientierte Aufbau des Systems, die Trennung der Präsentation von der Geschäftslogik sowie insbesondere die Parametrisierbarkeit der Geschäftslogik sind Voraussetzungen für bessere Wart- und Erweiterbarkeit, die oskar von monolithischen Altsystemen unterscheidet.

## Komponententests

Nach einer Analyse der zu testenden Klassen wurde die ursprüngliche Idee, das Erstellen von Komponententest von der Kritikalität und Komplexität der Klassen abhängig zu machen, verworfen und der Beschluss gefasst, für alle Klassen Komponententests zu erstellen. Eine Einschränkung wurde lediglich bzgl. der zu testenden Methoden getroffen. Insgesamt finden sich aktuell ca. 3000 Testmethoden im oskar-System, die gut 1000 produktive Klassen testen.

Eine wichtige Erkenntnis bei der Erstellung der Test ist die Notwendigkeit, zwischen Klassen- und Clustertests zu unterscheiden. Klassentests prüfen in der Regel nur Features der betrachteten Klasse, Clustertests prüfen das Zusammenspiel mehrerer Klassen, die potentiell aus verschiedenen Paketen und somit auch von verschiedenen Verantwortlichen erstellt wurden. Insbesondere bei letzteren besteht die Gefahr, durch zu starke Abhängigkeiten der Tests von änderungsanfälligen Klassen (speziell Entitäten) den Anpassungsaufwand der Test über den Aufwand für die Änderung am produktiven Code anwachsen zu lassen. Abhilfe kann hier u.a. durch den konsequenten Einsatz von Interfaces für die Entitäten geschaffen werden, die die Nutzung von Mock-Objekten in den Tests erlauben. Die Mock-Objekte garantieren zwar die im Interface verlangte Schnittstelle der entsprechenden Klasse, jedoch müssen Änderungen an der Klasse nicht zwangsläufig zu

Änderungen am Test führen. Ebenfalls sehr hilfreich ist in diesem Zusammenhang die Verwendung von Object Factories, da man mit ihrer Hilfe die Objektkonstruktion zentralisieren und somit den oben beschriebenen Nutzen weiter erhöhen kann.

## Refactoring

Die jederzeitige Lauffähigkeit der Komponententests ist zwingende Voraussetzung, um in der Lage zu sein, effektives Code-Refactoring [4] durchführen zu können. Unter Verwendung von Metriken für objektorientierte Software [2] (u.a. Kopplung, Klassengröße, etc.) können aus der Menge der produktiven Klassen diejenigen identifiziert werden, die einer genaueren Analyse im Hinblick auf die Einhaltung der Projektrichtlinien und auf die Vermeidung der in Fowlers Refactoring-Buch beschriebenen Stinks untersucht werden sollten. Basierend auf diesen Erkenntnissen werden in den oskar-Entwicklungsteams nunmehr selbständig Refactoringmaßnahmen durchgeführt. Als Werkzeug unterstützt JFactor den Refactoringprozess, welches als Plugin für Visual Age for Java eingesetzt wurde. In der inzwischen zum Einsatz kommenden Eclipse-Umgebung sind ebenfalls einige Refactoring-Funktionen eingebaut.

Der beispielsweise durch das konsistente Umbenennen von Klassen oder Methoden im gesamten Code zu erreichende Zeitgewinn ist enorm. Rechnet man dem noch die Zeitersparnis aus Fehlern hinzu, die durch automatisiertes Ändern vermieden werden, erhöht sich die Produktivität weiter.

Zusammenfassend lässt sich sagen, dass die Erfahrungen mit dem Einsatz des JUnit-Frameworks sehr positiv sind, wenn man einige Maßnahmen zur Begrenzung des Änderungsaufwands bei den Testklassen unternimmt. Der Einsatz von Refactoringwerkzeugen setzt das Vorhandensein einer hinreichend große Funktionalität abdeckenden Testsuite voraus. Der durch den Einsatz von Refactoringmaßnahmen zu erzielende Qualitätsgewinn bei gleichzeitiger Reduzierung des Aufwands ist erheblich.

## Referenzen


[1] Kent Beck, Erich Gamma: TestInfected ; www.junit.org/

[2] S.R. Chidamber, C.F. Kemerer Towards a metric suite for object-oriented Design, in: Proc. OOPSLA ’91, Sigplan Notices, 26(11), 1991, pp.197-211

[3] Sanjay Dewal, Ludger Schnichels: Bank 2010: Eine fachliche und technische Vision, Softwaretechniktrends 20(3), 2000, S. 9ff

[4] Martin Fowler: Refactoring - Wie Sie das Design vorhandener Software verbessern, Addison-Wesley, 2000, ISBN 3827316308

[5] Markus Warg, Gero Scholz : Großprojekte erfolgreich managen, FAZ Verlag, 2000, ISBN 3898430332