# SCORING AUF DEM PRÜFSTAND: WAS SOFTWAREPROJEKTE VON DER INDUSTRIE LERNEN KÖNNEN

Die Vermeidung von Risiken ist in komplexen Projekten ein wichtiges Thema des Projektmanagements. Anhand des in diesem Artikel vorgestellten Projekts zeigen wir einen erfolgreichen Ansatz zur Risikominimierung auf. Das Projekt – der Entwurf und die Implementierung eines Privatkunden-Rating-Systems für ein großes Finanzdienstleistungs-Rechenzentrum – hat den aus anderen Ingenieursdisziplinen bekannten Prüfstand als Metapher übernommen, um den zu entwickelnden Rechenkern bereits sechs Monate vor dem eigentlichen Integrationstest durch die Zertifizierungsstelle abnehmen zu lassen.

die autoren

Oliver Kunze
(E-Mail: Oliver.Kunze@cs-consulting.de)
ist Consultant und Senior Entwickler bei der CS Consulting AG mit den aktuellen Schwerpunkten Software-Engineering, IT-Architekturen und Teststrategien.

Sven Hohlfeld
(E-Mail: Sven.Hohlfeld@cs-consulting.de)
ist Consultant und Leiter des Fachreferats Web-Technologien im Competence Center Software Engineering bei der CS Consulting AG.

In der Automobilindustrie werden Prüfstände unter anderem während des Entwicklungsprozesses von Motoren zum Kontrollieren von Grundparametern, wie Drehzahl, Drehmoment, Schwingung, Kraftstoffverbrauch, Geräusch- und Abgasentwicklung, verwendet. Erst wenn diese Grundparameter beim Testen auf dem Prüfstand in bestimmten Beispielszenarien, wie Kaltstart, Warmstart, Volllast oder Normallast, den jeweils vorgegebenen Sollwerten entsprechen, werden die neu entwickelten Motoren in Testfahrzeuge eingebaut und abschließend unter realen Bedingungen geprüft und abgenommen. Es stellt sich die Frage, ob es in dieser Vorgehensweise der Automobilindustrie interessante Aspekte für Softwareprojekte gibt? Im Folgenden beschreiben wir das Projekt „Privatkunden-Rating", in dem mit Hilfe eines Software-Prüfstands ein agiles Akzeptanztestverfahren für eine komplexe Komponente erfolgreich eingeführt wurde. Damit wurde ein wichtiger Beitrag zum erfolgreichen Abschluss des Gesamtprojekts geleistet.

## Projektziel

Das Projekt hatte den Entwurf und die Implementierung eines Systems für das Privatkunden-Rating zum Ziel. Der Auftraggeber, die GAD eG, ist als IT-Dienstleister, Rechenzentrum und Softwarehaus für rund 450 Banken verantwortlich. Mit dem Projekt wollte die GAD ihren Privatkunden-Kreditprozess an die Basel-II-Richtlinien anpassen. Das Privatkunden-Rating-System sollte in den Privatkunden-Kreditprozess eingebettet werden und diesen im Antrags-und Verhaltens-Scoring (siehe Kasten 1) unterstützen.

## Entwurfsphase

In der Entwurfsphase des Projekts kristallisierte sich schnell heraus, dass die Scoring-Logik eine hohe Komplexität hatte: Die finanzmathematische Spezifikation dieser Berechnungslogik hatte ein Volumen von knapp 800 DIN-A4 Seiten. Es war sinnvoll, diese Scoring-Logik in einer Komponente zu bündeln. Diese Komponente bezeichnen wir im Folgenden als *Scoring-Rechenkern*.

Der Schnittstellenentwurf des Scoring-Rechenkerns wurde im Sinne von *Domain Driven Design (DDD)* (vgl. [Eva04]) stark durch die fachliche Struktur und Terminologie der finanzmathematischen Spezifikation getrieben. Dieser DDD-orientierte Entwurf war eine wichtige Voraussetzung für die später folgenden Akzeptanztests des Scoring-Rechenkerns – eine Entwurfsentscheidung, die nicht immer selbstverständlich ist. Ein übliches Anti-Pattern wäre es, im Scoring-Rechenkern die finanzmathematische Spezifikation mit der Logik von Daten aufbereitenden oder gar Daten besorgenden Schichten zu vermischen. Ein effizienter Test des Scoring-Rechenkerns gegen die finanzmathematische Spezifikation der Scoring-Logik wäre damit erschwert oder gar unmöglich gewesen.

> Das Scoring eines Kunden entspricht der Berechnung der Ausfallwahrscheinlichkeit, d. h. der Wahrscheinlichkeit, dass ein Kunde zahlungsunfähig wird und die offenen Forderungen seiner Kredite nicht mehr an die Bank zurückzahlen kann. Für eine Bank ist die durchschnittliche Ausfallwahrscheinlichkeit ihres Kredit-Portfolios eine wichtige Kennzahl zur Ermittlung des erforderlichen Mindesteigenkapitals.

*Kasten 1: Definition und Bedeutung des Begriffs „Scoring".*

## Entwicklungsprozess

Der beim Auftraggeber übliche Entwicklungsprozess entsprach dem traditionellen V-Modell (vgl. [Wik-c]). Die uneingeschränkte Umsetzung dieser traditionellen

Vorgehensweise im Projekt wäre riskant gewesen, da dann der Akzeptanztest des Scoring-Rechenkerns erst spät, indirekt und manuell möglich gewesen wäre:

***Spät:*** Der Akzeptanztest des Scoring-Rechenkerns wäre erst spät während des Akzeptanztests in der integrierten Testumgebung des Privatkunden-Ratings möglich gewesen. Aufgrund der hohen Komplexität des Scoring-Rechenkerns und der damit zu erwartenden hohen Fehlerwahrscheinlichkeit hätte das ein hohes Risiko in der Zeit- und Kostenplanung bedeutet.

***Indirekt:*** Im Kontext der integrierten Testumgebung des Privatkunden-Ratings hätte der Akzeptanztest des Scoring-Rechenkerns gegen die finanzmathematische Spezifikation nur indirekt durchgeführt werden können. Der Grund hierfür ist, dass der Scoring-Rechenkern im Privatkunden-Rating keine direkte Benutzungsschnittstelle hatte, sondern durch mehrere Daten aufbereitende und Daten besorgende Schichten gekapselt wurde. Der Scoring-Rechenkern erwartete als Eingabeparameter im Wesentlichen so genannte *Scoring-Merkmale*, wie beispielsweise im Verhaltens-Scoring den durchschnittlichen Habenumsatz eines Kontos in den letzten sechs Monaten. Die Spezifikation legt mathematisch fest, wie dieses Scoring-Merkmal die Ausfallwahrscheinlichkeit beeinflusst. Die Spezifikation schreibt also vor, wie die Ausfallwahrscheinlichkeit beeinflusst wird, wenn dieses Scoring-Merkmal zum Beispiel 300 Euro beträgt. Eine Prüfung in der integrierten Testumgebung, ob der Scoring-Rechenkern konform zur Spezifikation rechnet, wenn dieses Scoring-Merkmal genau 300 Euro beträgt, wäre aufwändig gewesen, denn in der integrierten Testumgebung hätte man dieses Scoring-Merkmal nicht direkt über die Benutzungsschnittstelle eingeben können. Stattdessen hätte man im Kontensystem ein Testkonto mit einer mindestens sechsmonatigen Umsatzhistorie und einem durchschnittlichen Habenumsatz der letzten sechs Monate in Höhe von 300 Euro anlegen müssen. Derartige indirekte Tests wären in der integrierten Testumgebung aufwändig und auch unsicher gewesen: unsicher, weil durch die Daten aufbereitenden Schichten hindurch getestet worden wäre, die auch Fehler enthalten konnten. Eine ausreichende Testabdeckung des Scoring-Rechenkerns hätte man unter diesen Umständen mit vertretbarem Aufwand kaum erreicht.

***Manuell:*** Insbesondere beim Testen des Scoring-Rechenkerns gegen die Spezifikation des Verhaltens-Scorings wäre eine manuelle Eingabe von Testdaten über die Benutzungsoberflächen der beteiligten Systeme in einem akzeptablen Zeitrahmen nicht möglich gewesen. Beispielsweise hätte die manuelle Eingabe eines Testkontos mit einer sechsmonatigen Umsatzhistorie und einem durchschnittlichen Habenumsatz in den letzten sechs Monaten in Höhe von 300 Euro auch sechs Monate gedauert. Um diesen Test in einem vertretbaren Zeitraum durchführen zu können, hätte der testende Fachexperte das Testkonto nebst Umsatzhistorie durch einen technischen Testhelfer an den Benutzungsoberflächen vorbei direkt in der Test-Datenbank anlegen lassen müssen. Automatische GUI-Testwerkzeuge hätte man unter diesen Umständen nicht sinnvoll einsetzen können, da die Hauptarbeit der Testfall-Ausführung – das Anlegen der Testdaten – nicht von solchen Werkzeugen unterstützt wird. Eine Automatisierung der Tests des Scoring-Rechenkerns gegen die Spezifikation wäre somit in der integrierten Testumgebung nicht möglich gewesen.

Aufgrund der genannten Risiken haben wir in Erwägung gezogen, den traditionellen Entwicklungsprozess durch die Einführung eines Software-Prüfstands (**siehe Kasten 2**) für den Scoring-Rechenkern zu erweitern. Der Einsatz des Software-Prüfstands ermöglichte einen frühen, direkten und maschinellen Akzeptanztest des Scoring-Rechenkerns:

***Früh:*** Die Entwicklung des Scoring-Rechenkerns konnte aus dem Gesamtprojekt „Privatkunden-Rating" in ein Teilprojekt ausgelagert werden, das durch den Akzeptanztest am Software-Prüfstand abgeschlossen wurde. Der Abschlusszeitpunkt dieses Teilprojekts lag deutlich vor dem des Gesamtprojekts.

***Direkt:*** Der Scoring-Rechenkern konnte am Software-Prüfstand direkt gegen die finanzmathematische Spezifikation ohne den störenden Einfluss von Daten aufbereitenden und Daten besorgenden Schichten getestet werden.

***Maschinell:*** Die am Software-Prüfstand vom Fachexperten definierten Testfälle konnten als Suite gebündelt und maschinell beliebig oft ausgeführt werden. Der komplette Akzeptanztest konnte damit bei Änderungen am Scoring-Rechenkern maschinell wiederholt werden.

Der Software-Prüfstand ermöglichte automatisierbare Testfälle, die als Kriterium für die Erfüllung der Anforderungen des Auftraggebers dienten und damit den Charakter eines agilen Akzeptanztests hatten (vgl. [Lin09]).

> Der Software-Prüfstand ist ein vom Fachexperten bedienbares Testwerkzeug. Er ist nicht mit einem entwicklungsnahen Testwerkzeug wie JUnit (vgl. [JUn]) zu verwechseln, das für einen Fachexperten eine zu technische Schnittstelle hat. Entwicklungsnahe Testwerkzeuge werden, wie der Name schon sagt, im Allgemeinen von Entwicklern verwendet und die mit ihnen erstellten Tests entsprechen Whitebox-Tests (vgl. [Wik-a]). Der Software-Prüfstand wird hingegen vom Fachexperten verwendet und die mit ihm erstellten Tests entsprechen Blackbox-Tests (siehe [Wik-b]).

***Kasten 2:*** *Definition und Bedeutung des Begriffs „Software-Prüfstand".*

## Kosten und Nutzen

Die Einführung des Software-Prüfstands verursachte Investitionskosten. Neben dem Entwurf und der Entwicklung des eigentlichen Prüfstands mussten Aufwände für die Schulung von Fachexperten zum Bedienen des Software-Prüfstands einkalkuliert werden. Diesen Investitionskosten stand der zu erwartende Nutzen gegenüber: Durch die Verwendung des Software-Prüfstands war es möglich, Entwurfs- und Implementierungsfehler des Scoring-Rechenkerns frühzeitig zu erkennen und zu korrigieren. Das sparte Fehlerbehebungskosten, denn je länger ein Fehler unentdeckt bleibt, desto aufwändiger und teurer ist seine Korrektur (vgl. [Ham08]).

Die Entscheidung für oder gegen die Einführung eines Software-Prüfstands war in diesem Sinne eine Kosten-Nutzen-Rechnung. Wegen der hohen Komplexität des Scoring-Rechenkerns und der damit zu erwartenden hohen Fehleranzahl war auch der zu erwartende Nutzen hoch. Dabei

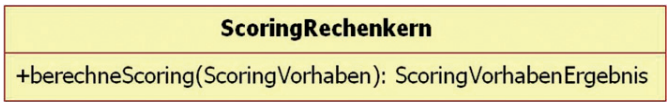


*Abb. 1: Zentrale Berechnungsmethode im Scoring-Rechenkern.*

wurde bei dieser Nutzenanalyse eine Reduzierung der Fehlerbehebungskosten nicht nur im Entwicklungs-, sondern auch im folgenden Wartungsprojekt fokussiert. Die in dieser Entscheidungsfindung durchgeführte Kosten-Nutzen-Analyse basierte auf der intuitiven und qualitativen Einschätzung, dass hier der Nutzen größer als die Kosten sein wird, und nicht auf einer quantitativen Schätzung, wie viel Geld und Zeit durch die Verwendung des Software-Prüfstands eingespart werden kann. Nachdem die Entscheidung für die Einführung des Software-Prüfstands gefallen war, sollten die Anforderungen an diesen konkretisiert werden.

## Anforderungen an den Prüfstand

Auf dem Software-Prüfstand sollten nur funktionale Anforderungen an den Scoring-Rechenkern getestet werden. Der einfachen Testbarkeit dieser funktionalen Anforderungen wurde bereits beim Entwurf des Scoring-Rechenkerns der Weg geebnet: Der Scoring-Rechenkern hatte im Wesentlichen eine öffentliche Methode berechneScoring, über die mit einem Aufruf für eine beliebige fachlich Konstellation (–> ScoringVorhaben) die entsprechenden Ausfallwahrscheinlichkeiten (–> ScoringVorhabenErgebnis) berechnet und zurückgegeben werden konnten (siehe Abbildung 1).

Ein Testfall für die funktionalen Anforderungen des Scoring-Rechenkerns bestand demnach aus einem Eingabeobjekt für die Berechnungsmethode, d. h. einem ScoringVorhaben, und dem gemäß Spezifikation zu erwartende Ausgabeobjekt, d. h. einem ScoringVorhabenErgebnis. Die Struktur eines solchen Testfalls ist stark vereinfacht im Klassendiagramm in Abbildung 2 dargestellt.

Der Software-Prüfstand sollte zwei Anwendungsfälle abbilden: das Bearbeiten und das Ausführen einer Test-Suite, d. h. einer bestimmten Liste von Testfällen (siehe Abbildung 3). Beim Bearbeiten sollten vom Fachexperten Testfälle in der Suite neu angelegt und vorhandene Testfälle verändert bzw. gelöscht werden können. Beim Ausführen sollten durch einen technischen Testhelfer alle zuvor in der Suite angelegten Testfälle gegen den Scoring-Rechenkern ausgeführt und ein Ergebnisprotokoll ausgegeben werden. Das Ausführen eines Testfalls bedeutet, dass der Scoring-Rechenkern mit dem ScoringVorhaben aus dem Testfall als Eingabeparameter aufgerufen wird und danach das zurückgegebene, berechnete ScoringVorhabenErgebnis mit dem im Testfall vorgegebenen erwarteten ScoringVorhabenErgebnis verglichen wird. Im Falle der Gleichheit sollte der Testfall als erfolgreich gelten, sonst als fehlerhaft. Nachdem die Anforderungen des Software-Prüfstands umrissen waren, konnte die Implementierung des Prüfstands beginnen.

## Implementierung des Prüfstands

Zuerst waren einige Implementierungsdetails zu klären. Die Wahl fiel auf eine pragmatische Eigenentwicklung des Software-Prüfstands. Der Software-Prüfstand sollte im Wesentlichen aus den folgenden drei Komponenten bestehen:

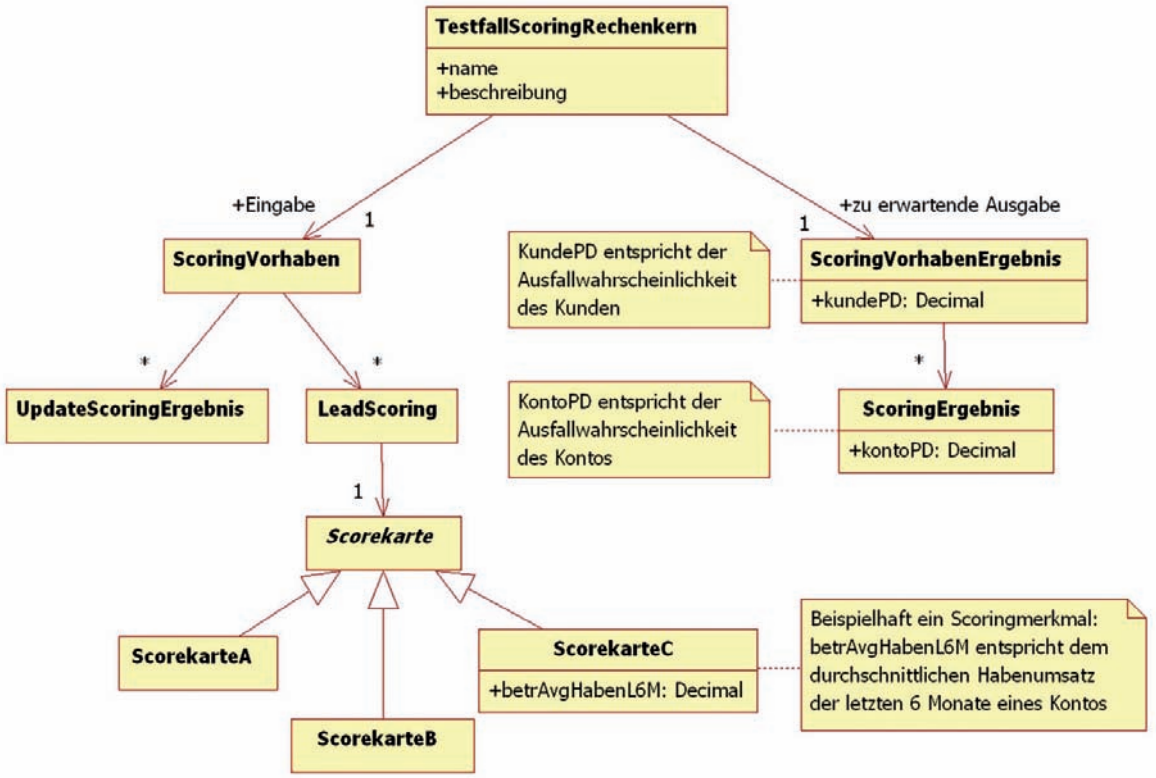


*Abb. 2: Vereinfachte Testfallstruktur der Berechnungsmethode im Scoring-Rechenkern.*

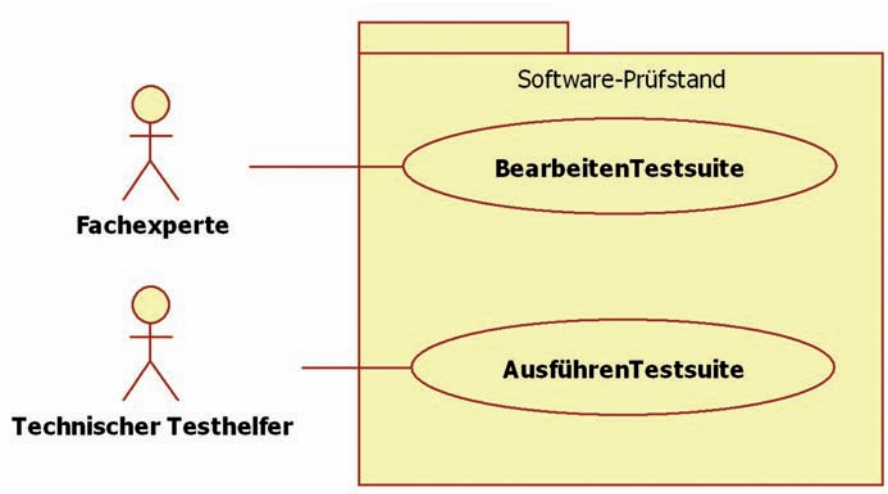


*Abb. 3: Anwendungsfälle des Software-Prüfstands.*

- *Testfall-Speicher:* Diese Komponente sollte die Testfälle verwalten und eine vom Fachexperten bedienbare Schnittstelle anbieten.
- *Testfall-Leser:* Diese Komponente sollte die im Testfall-Speicher verwalteten Testfälle als Liste von Objektnetzen lesen.
- *Testfall-Ausführer:* Diese Komponente sollte über den Testfall-Leser die Testfälle lesen und gegen den Scoring-Rechenkern testen.

beschränkte sich auf die Erstellung des Testfallmodells in Form von entsprechend strukturierten Tabellen und Integritätsregeln. Um den Aufwand für die Bereitstellung und Wartung des Testfall-Speichers minimal zu halten, wurde explizit auf eine Anwendungsentwicklung in MS-Access, beispielsweise durch die Bereitstellung benutzerfreundlicher Eingabeformulare, verzichtet. MS-Access bietet standardmäßig die Möglichkeit, Daten direkt in Tabellen zu bearbeiten. Der Fachexperte verfügte bereits über geringe MS-Access-Kenntnisse. Somit konnte er diese Tabellen-Standardschnittstelle nach geringem Schulungsaufwand benutzen.

Die Implementierung des *Testfall-Lesers* wurde in Java durchgeführt: Hierfür wurde eine JDBC/ODBC-Schnittstelle bereitgestellt, die die Test-Suite aus der MS-Access-Datenbank lesen und auf ein entsprechenden Objektnetz abbilden konnte.

Bei der Umsetzung der Komponente *Testfall-Ausführer* wurde gespart: Eine vom Fachexperten bedienbare Schnittstelle zum Ausführen der Testfälle und anschließenden Anzeigen des Berechnungsprotokolls wäre aufwändiger gewesen als eine technische, nur vom Entwickler bedienbare Schnittstelle. Außerdem haben wir den Mehrwert einer vom Fachexperten bedienbaren Schnittstelle als gering eingestuft, da im großen Akzeptanztest am Software-Prüfstand im Fehlerfall sowieso eine enge Zusammenarbeit zwischen den Fachexperten und einem technischen Testhelfer zur Auswertung und Interpretierung vom Berechnungsprotokoll vorgesehen war. Also konnte der ohnehin beteiligte technische Testhelfer für das Ausführen der Test-Suite verantwortlich gemacht werden. Die Komponente „Testfall-Ausführer" konnte jetzt einfach

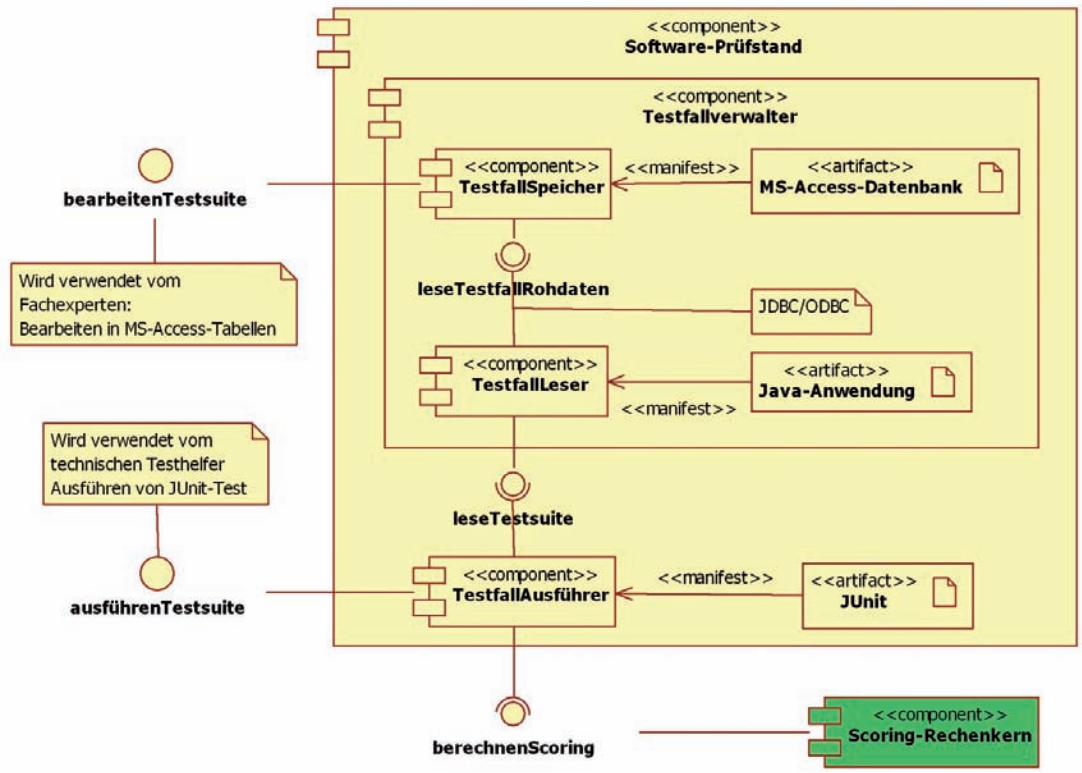


*Abb. 4: Komponentendiagramm des großen Software-Prüfstands.*

als JUnit-Test (vgl. [JUn]) implementiert werden. Diese JUnit-Testklasse besitzt eine Testmethode, in der die Test-Suite aus dem Testfall-Speicher gegen den Scoring-Rechenkern ausgeführt wird. Diese Testmethode sollte genau dann fehlschlagen, wenn mindestens einer der Testfälle in der Suite fehlschlug. In **Abbildung 4** ist die Komponentenstruktur dieses Software-Prüfstands dargestellt.

Der Scoring-Rechenkern ist nicht Bestandteil des Software-Prüfstands. Er wird nur in den Prüfstand „angedockt" und dann beim Ausführen der Testfälle aufgerufen. Man beachte, dass der im Prüfstand getestete Scoring-Rechenkern identisch mit demjenigen ist, der im System Privatkunden-Rating verwendet wird. Nach der Beendigung der Implementierung musste der Fachexperte in der Anwendung des Software-Prüfstands geschult werden.

## Schulung des Fachexperten

Ziel der Schulung war es, dass der Fachexperte das Prinzip des Software-Prüfstands versteht und die Testfälle selbst bearbeiten kann. Zum Erreichen dieses Ziels wurde neben der Erstellung eines Benutzerhandbuchs für den Software-Prüfstand auch ein kleiner Workshop veranstaltet, in dem der Fachexperte und der technischen Testhelfer bzw. Entwickler einige beispielhafte Testfälle im Testfall-Speicher des Software-Prüfstands gemeinsam angelegt, ausgeführt und die Ergebnisse analysiert haben. Diese Schulung war nebenbei auch eine gute Gelegenheit, um den Entwurf des Scoring-Rechenkerns zu überprüfen: Zum Erstellen von Testfällen musste der Fachexperte sich mit der Testfallstruktur beschäftigen, insbesondere mit der Ein- und Ausgabestruktur des Scoring-Rechenkerns (–> ScoringVorhaben und ScoringVorhabenErgebnis). Wenn der Fachexperte hier nicht in der Lage gewesen wäre, seine Testfälle strukturell darzustellen, hätte das ein Hinweis auf eine Schwäche im Entwurf und damit auch in der Berechnungslogik des Scoring-Rechenkerns sein können.

Diese Schulung implizierte somit schon früh eine enge Zusammenarbeit zwischen Fachexperten und Entwickler, ein wichtiger Baustein einer agilen Vorgehensweise. Der Fachexperte war bald in der Lage, Testfälle auch ohne Unterstützung des technischen Testhelfers zu erzeugen und zu bearbeiten. Dem eigentlichen Ziel, dem Akzeptanztest des Scoring-Rechenkerns, stand nun nichts mehr im Weg.

## Großer Akzeptanztest mit dem Prüfstand

Für den Akzeptanztest des Scoring-Rechenkerns mit Hilfe des Software-Prüfstands hatte der Fachexperte eine Suite von 70 Testfällen vorbereitet. Zuerst liefen nicht alle Testfälle fehlerfrei, aber innerhalb von etwa einem Arbeitstag konnten der technische Testhelfer und der Fachexperte alle Fehler im Scoring-Rechenkern und auch teilweise in den Testfällen selbst finden und beseitigen. Die Test-Suite lief damit fehlerfrei durch und der Scoring-Rechenkern war offiziell vom Fachexperten abgenommen.

Eine wichtige Voraussetzung für die schnelle Durchführbarkeit des Akzeptanztests war ein detailliertes und vom Fachexperten gut lesbares Berechnungsprotokoll, was vom Scoring-Rechenkern selbst erzeugt wurde. Mit Hilfe dieses Berechnungsprotokolls waren der testende Fachexperte und der technische Testhelfer schnell in der Lage, die Ursache für das Fehlschlagen eines Tests zu analysieren.

Nach erfolgreicher Durchführung des Akzeptanztests stellte sich die Frage, wie die erreichte Qualität im Scoring-Rechenkern auch über zukünftige Wartungs- und Build-Zyklen hinweg beibehalten werden konnte. Die Lösung war eine maschinelle Wiederholung des Akzeptanztests bei jedem Build.

## Kleiner Akzeptanztests mit dem Prüfstand

Der vom Auftraggeber vorgegebene Integrationsprozess umfasste selbsttestende Builds mit *Continuous Integration* (vgl. [Fow06]). Gemäß dieser Vorgabe sollten automatisierte Tests in den Build-Prozess einer Komponente aufgenommen werden. Das Fehlschlagen dieser Tests sollte zum Abbruch des Builds führen.

Es war naheliegend, in dem selbsttestenden Build des Scoring-Rechenkerns auch den Abnahmetest des Fachexperten maschinell zu wiederholen, um eine Steigerung in der Qualitätssicherung zu erzielen. Dabei kristallisierten sich zwei Arten von Akzeptanztests heraus:

- *Großer Akzeptanztest:* Auslösendes Ereignis für den großen Akzeptanztest ist die Erstellung bzw. Veränderung der Spezifikation. Der Fachexperte erstellt bzw. verändert daraufhin die Test-Suite im Testfall-Speicher des Software-Prüfstands. Die Test-Suite wird dann vom technischen Testhelfer über den Software-Prüfstand ausgeführt.
- *Kleiner Akzeptanztest:* Auslösendes Ereignis für den kleinen Akzeptanztest ist ein Re-Build des Scoring-Rechenkerns, das zum Beispiel wegen technischer Wartungsmaßnahmen erforderlich ist. Die Test-Suite des letzten großen und erfolgreichen Akzeptanztests wird daraufhin vom Build-Prozess über den Software-Prüfstand ausgeführt.

Beim kleinen Akzeptanztest musste noch ein technisches Problem gelöst werden: Dieser Test war auf dem beim Auftraggeber eingesetzten Unix-Build-Server nicht lauffähig, da der Zugriff auf den MS-Access-Testfall-Speicher von der Unix-Umgebung aus nicht möglich war. Die Lösung dieses Problems bestand darin, im kleinen Akzeptanztest auf die Verwendung von MS-Access als Testfall-Speicher zu verzichten. Stattdessen wurde als Testfall-Speicher eine Datei mit der serialisierten Test-Suite des letzten erfolgreichen großen Akzeptanztests verwendet. Die Test-Suite in diesem Testfall-Speicher kann nicht mehr vom Fachexperten bearbeitet werden, was im kleinen Akzeptanztest aber auch nicht erforderlich ist. Der Testfall-Leser musste ebenfalls ausgewechselt werden: Die Test-Suite wurde hier statt über JDBC/ODBC über den Deserialisierungs-Mechanismus der Java-Standard-Programmierschnittstelle ausgelesen.

Es kristallisieren sich zwei Ausprägungen von Software-Prüfständen heraus: der große und der kleine Software-Prüfstand zur Verwendung im großen bzw. kleinen Akzeptanztest. Die Struktur des großen Software-Prüfstands ist in **Abbildung 4** dargestellt. Der kleine und der große Softwareprüfstand unterschieden sich nur durch die unterschiedlichen Ausprägungen des Testfall-Verwalters. Beim Entwurf des Software-Prüfstands wurden diese Ausprägungen im Sinne des Strategie-Patterns (vgl. [Gam94]) abstrahiert. Dadurch konnte im großen und im kleinen Software-Prüfstand der gleiche Testfall-Ausführer verwendet werden. ▸

Mit Hilfe des kleinen Software-Prüfstands konnte der kleine Akzeptanztest erfolgreich im selbsttestenden Build des Scoring-Rechenkerns integriert werden, wodurch im Build eine deutliche Steigerung der Qualitätskontrolle erreicht wurde.

## Fazit

Die Entwicklung und die Abnahme des Scoring-Rechenkerns nach dem beschriebenen Vorgehen waren erfolgreich: Der Scoring-Rechenkern konnte sechs Monate vor dem Abnahmetest des Gesamtsystems abgenommen werden. Wesentliche Fehler sind in dieser komplexen Komponente danach nicht mehr entdeckt worden.

Obwohl das Projekt bereits vor einigen Jahren durchgeführt wurde, ist die Idee hinter der hier beschriebenen Vorgehensweise immer noch aktuell: In einem Softwareprojekt lohnt sich unter bestimmten Bedingungen die Verwendung eines speziellen Akzeptanztestwerkzeugs, vergleichbar mit einem Prüfstand im Motorentwicklungsprozess in der Automobilindustrie. Ein solcher Software-Prüfstand ermöglicht einen vom Fachexperten definierten, direkten, automatisierbaren und damit agilen Akzeptanztest einer Komponente. Die Einführung eines Prüfstands für eine Komponente ist besonders dann sinnvoll, wenn die Fachspezifikation dieser Komponente eine hohe Komplexität hat und ein Abnahmetest ohne Prüfstand nur indirekt, manuell und erst spät bei der Abnahme des gesamten Systems möglich wäre. Sind diese Bedingungen erfüllt, sollte für die Einführung des Prüfstands die Schnittstelle der zu testenden Komponente im Sinne von *Domain Driven Design* eng an die Fachspezifikation angelehnt sein, damit der Fachexperte sich in den Testfall-Strukturen des Prüfstands zurechtfinden kann. Weiterhin sollte diese Schnittstelle nur die Spezifikation abbilden und nicht vermischt werden mit Daten aufbereitenden Schichten, damit ein direkter Test der Spezifikation möglich ist.

Die Entwickler und Fachexperten sollten die Bereitschaft für mehr Kommunikation mitbringen, als das bei einer traditionellen Vorgehensweise üblich ist. Der Prüfstand ermöglicht nicht nur einen agilen Akzeptanztest, er erzwingt somit auch Entscheidungen, die die Qualität der Software und des Entwicklungsprozesses im Allgemeinen steigern: ein strukturierter und fachlich getriebener Komponentenentwurf und eine frühe Kommunikation zwischen Entwickler und Fachexperten. ■

## Literatur & Links


**[Eva04]** E. Evans, Domain Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley 2004

**[Fow06]** M. Fowler, Continuous Integration, 2006, siehe: www.martinfowler.com/articles/continuousIntegration.html

**[Gam94]** E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Resuable Object-Oriented Software, Addison-Wesley 1994

**[Ham08]** T. Hampp, M. Knauß, Eine Untersuchung über Korrekturkosten von Software-Fehlern, in: Softwaretechnik-Trends, Band 28 Heft 2, siehe: pi.informatik.uni-siegen.de/stt/28_2/03_Technische_Beitraege/Sopra.pdf

**[JUn]** Junit.org, siehe: www.junit.org

**[Lin09]** J. Link, Agile Akzeptanztests, in: OBJEKTspektrum 05/2009

**[Wik-a]** Wikipedia, Whiteboxtest, siehe: www.en.wikipedia.org/wiki/White_box_testing

**[Wik-b]** Wikipedia, Blackboxtest, siehe: www.en.wikipedia.org/wiki/Black_box_testing

**[Wik-c]** Wikipedia, V-Model, siehe: www.en.wikipedia.org/wiki/V-Model