Test nichtfunktionaler Anforderungen in der Praxis am Beispiel einer netzzentrierten JEE-Anwendung

Julia Remmert
Wincor Nixdorf International GmbH
Heinz-Nixdorf-Ring 1
33106 Paderborn
julia.remmert@wincor-nixdorf.com

Zusammenfassung

Der Test von nichtfunktionalen Anforderungen stellt einen wichtigen Bestandteil der Qualitätssicherung dar. In diesem Beitrag wird basierend auf einem praktischen Beispiel von Wincor Nixdorf erläutert, wie die Anforderungen hinsichtlich Zuverlässigkeit, Effizienz und Skalierbarkeit überprüft werden können.
Der Beitrag fokussiert dabei auf die Testarten Dauertest, Performancemessung und Failovertest. Diese Testarten stellen einen Beitrag für den sicheren Betrieb (7x24h) einer SB-Server Lösung dar. Best Practices bei der Ausführung der jeweiligen Testarten sowie praktische Erfahrungen zu deren Besonderheiten und Schwierigkeiten werden erläutert.
Dazu gehören unter anderem Erfahrungen bei dem Einsatz unterschiedlicher Werkzeuge, wie zum Beispiel Open Source Werkzeuge (z.B. JMeter), eigene Werkzeuge und kommerzielle Werkzeuge.
Es werden zudem Kennzahlen (Key Performance Indikatoren, KPIs) vorgestellt, die für die Messung und Interpretation der Testergebnis eingesetzt werden. Diese Kennzahlen werden einerseits benutzt, um die Performance zu kontrollieren und zu verbessern und andererseits um Empfehlungen bezüglich der benötigten Hardware für Kundensysteme geben zu können. Dazu ist neben regelmäßigen Lastmessungen ein Test in einem Performancecenter notwendig.

1. Einleitung

Im folgenden Text werden praktische Erfahrungen mit dem Testen von nichtfunktionalen Anforderungen am Beispiel einer netzzentrierten Anwendung wiedergegeben. Dabei liegt der Schwerpunkt auf den Anforderungen Zuverlässigkeit, Effizienz und Skalierbarkeit, welche durch die Testarten Failovertest, Ausfalltest, Stresstest, Dauertest und Performancemessungen sichergestellt werden.

Es wird zunächst ein Überblick über die Funktionalität und die Architektur der zu testenden Anwendungen gegeben. Anschließend werden die zu untersuchenden nichtfunktionalen Anforderungen und deren zugehörige Testarten vorgestellt. Erfahrungen zur Testdurchführung, Testwerkzeugen und der Möglichkeit zur Testautomatisierung werden darauf basierend erläutert.

2. Architektur und Produkte

Die PC/E Server-Architektur ist eine Client-Server Architektur. Sie ist eine offene, netzzentrierte Multikanal-Architektur, die modular und service-orientiert aufgebaut ist, wie in Abbildung 1 dargestellt wird. Der Server, der in der Regel in einem Cluster betrieben wird, ist mit einer Datenbank verbunden und bietet sowohl eine Schnittstelle zu den Clients als auch zu einem Host, der in der Regel von der Bank bereitgestellt wird. Über das Hostsystem werden von der Bank z.B. Kunden- und Kartendaten verwaltet und Anfragen der Clients beispielsweise bezüglich PIN-Autorisierung beantwortet.

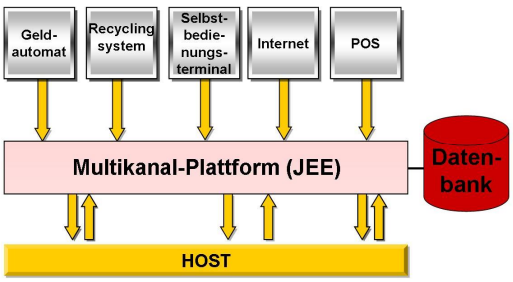

Abb. 1: Vereinfachte Darstellung der PC/E Server-Architektur

Das Produkt PC/E Self-Service basiert auf dieser Architektur und bietet selbstbedienungsbezogene Dienste für Banken wie zum Beispiel die Überweisung am Selbstbedienungsterminal oder Geldausgabe am Geldautomaten an. Ein weiteres Produkt der PC/E Payment Server basiert ebenfalls auf der PC/E Server-Architektur und bietet eine serverbasierte Lösung, die den kartengestützten, bargeldlosen Zahlungsverkehr optimiert.

3. Nicht-funktionale Anforderungen

Eine nicht-funktionale Anforderung beschreibt Attribute des Systemverhaltens, die sich nicht direkt auf die Funktionalität des Systems beziehen, sondern darauf, mit welcher Qualität das System diese Funktion erbringt. Beispiele sind laut ISO 9126 Zuverlässigkeit. Benutzbarkeit, Effizienz. Änderbarkeit und Übertragbarkeit [3]. In diesem Artikel soll der Schwerpunkt auf die Anforderungen Zuverlässigkeit, Skalierbarkeit und Effizienz gelegt werden.

Zuverlässigkeit ist eine Menge von Merkmalen, die sich auf die Fähigkeit einer Software/eines Systems beziehen, ihr/sein Leistungsniveau unter festgelegten Bedingungen über einen festgelegten Zeitraum oder über eine festgelegte Anzahl von Transaktionen zu bewahren [1].

Ein System, das als skalierbar bezeichnet wird, bleibt auch dann effektiv, wenn die Anzahl der Ressourcen und die Anzahl der Benutzer wesentlich steigt [2].

Effizienz ist eine Menge von Merkmalen (z.B. Verarbeitungsgeschwindigkeit, Antwortzeit), die sich auf das Verhältnis zwischen dem Leistungsniveau der Software und dem Umfang der eingesetzten Betriebsmittel (Speicher u.a.) unter festgelegten Bedingungen (in der Regel in Abhängigkeit steigender Last) beziehen [1].

Die dargestellten nicht-funktionalen Anforderungen lassen sich durch bestimmte Testarten überprüfen, die im nachfolgenden Kapitel 4 erläutert werden. Die Zuordnung wird in Tabelle 1 dargestellt.

Anforderung	**Testart**
Zuverlässigkeit	Failover Test Ausfalltest Stresstest Dauertest
Effizienz	Performancemessung
Skalierbarkeit	Performancemessung

Tabelle 1: Zuordnung der nicht-funktionalen Anforderungen zu Testarten

4. Nachweis der nicht-funktionalen Anforderung Zuverlässigkeit

Um die nicht-funktionale Anforderung Zuverlässigkeit zu überprüfen werden die Testarten Ausfalltest, Failovertest, Stresstest und Dauertest nachgewiesen.

Die Testumgebung (Abbildung 2) besteht aus einem Server, der in einem Cluster aufgebaut wird. Er ist mit der Datenbank und einem Hostsimulator verbunden. Die Clients werden über einen Simulator generiert und deren Anfragen über einen Loadbalancer auf die Clustermember verteilt.

Ein Ausfalltest testet auf Robustheit gegenüber Ausfall einzelner Komponenten. Beim Test des PC/E Servers wird der Ausfall des Servers, des Hosts, der Datenbank und des Loadbalancers simuliert.

Beim Failovertest wird der Ausfall redundanter Komponenten getestet. Im dargestellten Beispiel ist dies der Ausfall eines Clustermebers. Das Ziel ist die Übernahme der Aufgabe durch redundante Komponenten, so dass im Idealfall der Ausfall vom Benutzer überhaupt nicht bemerkt wird.

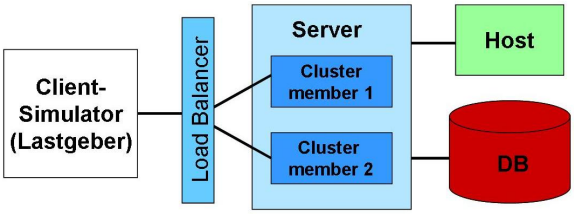

Abb. 2: Architekturbild der Testumgebung

Beim Stresstest wird das Systemverhalten bei Überlastung beobachtet [1]. Dies wird beim Test des PC/E-Servers durch die Simulation von einer ausreichenden Anzahl von Clients erreicht.

Mit Dauertest wird der Test der Stabilität/Zuverlässigkeit im Dauerbetrieb bezeichnet. Dieser wurde von uns so durchgeführt, dass der Server ausgelastet ist, aber nicht an der Lastgrenze läuft. Das Hauptziel des Tests ist das Aufdecken von Memory Leaks. Das heißt, es soll nachgewiesen werden, dass allokierter Speicher wieder freigegeben wird und auch bei lang andauernden Tests es nicht zu Speicherengpässen kommt. Um mögliche Fehler frühzeitig zu erkennen, wird sowohl die CPU-Auslastung als auch der Speicherverbrauch während des gesamten Tests beobachtet. Ein erfolgreicher Dauertest ist die Basis für die anschließend folgende Performancemessung. Dabei ist wichtig, dass keine Fehler auftreten, da zum Beispiel abgebrochene Transaktionen das Messergebnis verfälschen können und man so auch einen zu guten, aber falschen Wert erhalten kann.

In der Regel sollte der Dauertest sowie der darauf basierende Performancetest in einer möglichst realistischen Kundenumgebung durchgeführt werden. Bei stabilen Umgebungen ist dieses Verfahren unserer Erfahrung nach auch empfehlenswert. Das Produkt PC/E Self-Service wurde in einer Kombination aus Einzahltransaktionen, Auszahltransaktionen, Transaktionen am Kontoauszugsdrucker, dem Bedienen der Benutzerkonsole sowie einem Datenimport parallel getestet. Der Vorteil besteht hier in dem realistischen Szenario, dessen Komplexität aber im Fehlerfall die Analyse schwierig macht, da die Ursache des Fehlers in allen parallel durchgeführten Aktivitäten liegen kann.

Daher wurde das neu entwickelte Produkt PC/E Payment Server beim Test der Version 1.0 zunächst mit

einem vereinfachten Szenario getestet, das nur eine Transaktionsart beinhaltet, um die Analyse und Optimierung möglichst effizient gestalten zu können. Nachdem das Produkt einen gewissen Stabilitätsgrad erreicht hatte, wurde das Szenario sukzessive komplexer gestaltet.

5. Nachweis der nicht-funktionalen Anforderungen Effizienz und Skalierbarkeit

Die nicht-funktionalen Anforderungen Effizienz und Skalierbarkeit können durch eine Performancemessung nachgewiesen werden.

Eine Performancemessung ist die Prüfung der Performance für bestimmte Anwendungsfälle, in der Regel in Abhängigkeit von steigender Systemlast [1]. Dabei können sowohl die Antwortzeit als auch die Transaktionen pro Sekunde gemessen werden. Zusätzlich wird die CPU- und Speicherauslastung beobachtet.
Die Testdurchführung besteht aus der Vorbereitung und Optimierung der Testumgebung sowie der eigentlichen Messung. Optional ist anschließend eine Messung in einem großen Szenario z.B. in einem speziellen Performancelab möglich.

5.1. Vorbereitung der Testumgebung

Ein in der Regel zeitaufwändiger Part der Performancemessung ist die Vorbereitung der Testumgebung. Zunächst muss das zu vermessende Testobjekt festgelegt werden, in diesem Beispiel ist dies der PC/E Server. Da die langsamste Komponente der Testumgebung das Testergebnis bestimmt, muss man dafür sorgen, dass alle übrigen Komponenten “unendlich” schnell sind und nicht zum Engpass werden.

Ist die Hardware der Datenbank nicht entsprechend leistungsfähig, muss sie optimiert werden. Dies erfordert in der Regel Expertenwissen. Die ausreichende Verfügbarkeit von Lastgebern, Loadbalancern und Hostsimulatoren kann in unserem Fall einfach durch die Vervielfältigung der entsprechenden Werkzeuge sichergestellt werden.

Die optimale Konfiguration des Anwendungsserver (in unserem Fall Websphere), auf dem der PC/E-Server läuft hat entscheidenden Einfluss auf die Testergebnisse. Hier werden beispielsweise die Anzahl der Datenbankverbindungen und der internen Threads konfiguriert. Da es zwischen den Parametern Abhängigkeiten gibt, muss die optimale Konfiguration durch genaue Analyse der Ressourcen und oft auch "Try and Error" festgelegt werden. Die Optimierung der Parameter ist sehr zeitaufwändig. Ist die optimale Konfiguration einmal gefunden, sollte sie möglichst nicht mehr verändert werden, um eine Vergleichbarkeit der Ergebnisse sicherzustellen. Ist dennoch eine Änderung erforderlich, müssen alle festgelegten Key Performance Indikatoren vor und nach der Änderung gemessen werden. So wird sichergestellt, dass im Fall einer Änderung der ermittelten Werte auf die geänderten Parameter zurückzuführen sind.

Um die Reproduzierbarkeit der Ergebnisse zu gewährleisten, ist zudem sicherzustellen, dass die Tests in einem separaten Netz durchgeführt werden.

5.2. Performancemessung

Das Ziel der Performancemessung ist zum einen eine Vergleichbarkeit zwischen zwei Versionen herzustellen. Dazu ist es notwendig unter gleichen Testbedingungen identische Messungen mit verschiedenen Softwareversionen durchzuführen. Als Key Performance Indikatoren haben sich unseren Erfahrungen nach die Antwortzeiten, die Transaktionen pro Sekunde (TPS) sowie die Speicher- und CPU-Auslastung bewährt.

Zum anderen sollen basierend auf der Performancemessung Empfehlungen für Kundenumfelder gegeben werden. Um eine Empfehlung für die Datenbank zu geben, wird der Zuwachs an Datenvolumen pro Transaktion gemessen. Dieser lässt sich für ein Kundenszenario nach unseren Testergebnissen linear hochrechnen. Um die Zugriffe auf die Datenbank möglichst gering zu halten, wird deren Anzahl gezählt. Das Ergebnis dieses Tests ist einfach zu analysieren und außerdem gibt es Kunden, die an ihren Datenbankbetreiber pro Datenbankzugriff bezahlen und folglich an einem möglichst optimierten Wert interessiert sind.

Das Ergebnis des Servertests ist in der Regel nicht linear hochzurechnen. Im Beispiel von PC/E Self-Service sollte folgende daher Frage durch eine Performancemessung beantwortet werden: *Wie viele Clients können parallel betrieben werden, um die Antwortzeiten (z.B. 2 Sekunden) einzuhalten?*

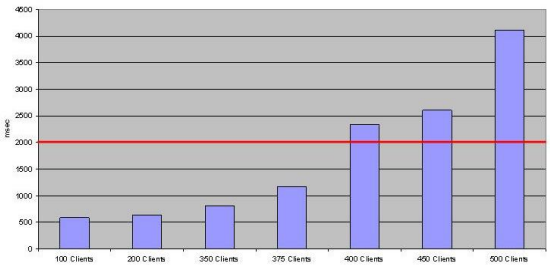

Abb. 3: Beispielhaftes Ergebnis einer Performancemessung

Abbildung 3 stellt ein beispielhaftes Ergebnis einer Performancemessung dar. An diesem ist deutlich sichtbar, dass sich die Leistung im Bezug auf die Last nicht linear verhält.

5.3. Messung im großen Szenario

Da die verfügbare Hardware in der Regel leistungsschwächer als die Hardware des größten Kunden ist und sich die Ergebnisse in der Regel nicht linear hochrechnen lassen, ist ein Test im großen Szenario erforderlich. Dieser wird in einem Performancelab zum Beispiel von IBM oder Oracle vorgenommen.

Das Ziel des Tests ist, zum einen die Skalierbarkeit der Software zu untersuchen und zum anderen eine Hardwareempfehlungen für Kundenprojekte (Sizing) geben zu können.

Beim Test des PC/E Payment Servers zum Beispiel wird die Hardware und Anzahl der Clustermember sukzessive erhöht bis eine bestimmte Anzahl an TPS (Transaktionen pro Sekunde) erreicht wird. Messungen in großen Szenarien finden in der Regel einmalig statt und anschließend werden Veränderungen zur Vorgängerversion überwacht.

6. Werkzeuge

Prinzipiell gibt es drei verschieden Arten von Werkzeugen: Kommerzielle Werkzeuge, Open Source und selbstgeschriebene Werkzeuge.

Eigene Entwicklungen werden speziell für ein bestimmtes Projekt entwickelt und werden vor allem dann eingesetzt, wenn für ein bestimmtes Protokoll keine Standardwerkzeuge vorhanden sind. Hier besteht die Möglichkeit, dass Werkzeug individuell auf seine Bedürfnisse anzupassen, aber auf der anderen Seite muss jede Standardfunktion zum Logging oder zur Auswertung eigens implementiert werden.

Im Bereich Open Source konnten gute Erfahrungen mit dem Werkzeug JMeter gemacht werden. Dieses ermöglicht eine einfache Handhabung, da Transaktionen über das http-Protokoll einfach aufgenommen werden können. So können die Aufnahmen ohne große Einarbeitung durchgeführt werden und für jede Softwareversion aktualisiert werden. Es unterstützt allerdings nur eine beschränkte Anzahl an Protokollen.

Kommerzielle Werkzeuge bieten zusätzlich eine Reihe von nützlichen Funktionen wie die Generierung von Testdaten, z.B. Karten und Terminals, die Konfigurationsmöglichkeiten zur Kombination von Transaktionsarten sowie Funktionen zum Logging und zur Auswertung der Tests. Kommerzielle Werkzeuge erfordern einen gewissen Einarbeitungs- und Konfigurationsaufwand. Zudem sind sie in der Regel nur für Standardprotokolle anwendbar.

7. Testautomatisierung

Es stellt sich die Frage, inwieweit Testautomatisierung im Bereich der Performancetest möglich ist. Die Erzeugung des Transaktionsvolumens erfordert immer Automatisierung. Aber vor allem im Bereich der Testvorbereitung und Testauswertung kann durch Automatisierung viel Zeit eingespart werden. Automatisierung macht es einfacher gleiche Startbedingungen herzustellen und vermeidet so einerseits Fehler und stellt andererseits die Reproduzierbarkeit der Testfälle sicher.

Da Tests bei einer Lastmessung oft in verschiedenen Variationen wiederholt werden müssen, bei denen aber die Testauswertung gleich bleibt, kann hier auch durch Automatisierung Zeit eingespart werden. Besonders beim Sammeln und Aufbereiten der Ergebnisse gibt es hier Potential.

Trotzdem bleiben als manuelle Tätigkeiten die Optimierung der Umgebung, die Problem- und Ergebnisanalyse sowie der Testentwurf, deren Zeitaufwand nicht zu unterschätzen ist.

8. Zusammenfassung

Das Testen der nicht-funktionalen Anforderungen Zuverlässigkeit, Effizienz und Skalierbarkeit ist komplexer und aufwändiger als oft erwartet wird. Es werden eine ganze Reihe von Testarten benötigt, um diese zu überprüfen: Failovertest, Ausfalltest, Stresstest, Dauertest und Performancemessung. Besonders bei der letztgenannten Testart ist die Testvorbereitung der Testumgebung sehr aufwändig. Um Hardwareempfehlungen für Kunden geben zu können, ist zudem oft eine Messung in einem Performancelab notwendig.

Der Einsatz von Testwerkzeugen wird oft schon durch deren Unterstützung für das zu testenden Objektes eingeschränkt. Mit dem Open-Source Produkt JMeter konnten gute Erfahrungen gemacht werden. Individuelle Software ist genau auf die Anforderungen zugeschnitten, aber kommerzielle Werkzeuge bieten mehr Standardfunktionen.

Testautomatisierung ist nicht nur bei der Erzeugung der Last sondern auch bei der Testvorbereitung und -auswertung sehr zu empfehlen, da sie nicht nur Zeit spart, sondern auch Fehler vermeidet.

Referenzen

[1] Spillner, Andreas, Linz, Tilo: *Basiswissen Softwaretest*, dpunkt.verlag, ISBN 978-3-89864-642-0, 2010

[2] Coulouris, George, Dollimore, Jean, Kindberg, Tim: Verteilte Systeme: Konzept und Design, Addison-Wesley, ISBN 3-8273-7022-1, 2002

[3] ISO/IEC 9126-1:2001 Software engineering - Product quality - Part 1: Quality model, ISO/IEC TR 9126-2:2003 Software engineering - Product quality