

syngenio in der Fachpresse

Medium: Javamagazin
**Thema: Qualität in Java-Projekten, Teil 3:
Last und Performance**
Autor: Jörn Eyrich und David Gärtner
Ausgabe : 9.2005

Fuß von der Bremse!

Nachdem in den meisten Projekten sehr viel Zeit mit der Definition der funktionalen Anforderungen verbracht worden ist, werden die nichtfunktionalen Anforderungen wie Lastverhalten, Performance, Stabilität und Sicherheit meist nur am Rande erwähnt und festgehalten. Der vorliegende Artikel widmet sich diesen Themen mit dem Fokus auf Last und Performance.



Kontakt & weitere Informationen:

syngenio AG
Ivonne Machnacz
Andreas-Hermes-Straße 3
D-53175 Bonn
Fon +49 (0)2 28-6 20 95-100
Fax +49 (0)2 28-6 20 95-150
ivonne.machnacz@syngenio.de
www.syngenio.de

Qualität in Java-Projekten, Teil 3: Last und Performance

Fuß von der Bremse!

■ VON JÖRN EYRICH UND DAVID GÄRTNER

Nachdem in den meisten Projekten sehr viel Zeit mit der Definition der funktionalen Anforderungen verbracht worden ist, werden die nicht funktionalen Anforderungen wie Lastverhalten, Performance, Stabilität und Sicherheit meist nur am Rande erwähnt und festgehalten. Der vorliegende Artikel widmet sich diesen Themen mit dem Fokus auf Last und Performance.

Üblicherweise ist man froh, wenn man die funktionalen Anforderungen eines Projektes halbwegs strukturiert erfasst hat. Trotzdem sollte man die nicht funktionalen Anforderungen (Tabelle 1) nicht ganz außer Acht lassen, da diese in entsprechenden Wechselwirkungen zueinander stehen. Auch zu diesen Themen muss es ein Dokument geben, in dem die Anforderungen beschrieben und Prioritäten festgelegt werden, damit in den einzelnen Phasen darauf zurückgegriffen werden kann. Als Muster kann man sich hier an der Definition der Service Level Agreements orientieren.

Bei der Anforderung Hochverfügbarkeit/Ausfallsicherheit muss überlegt werden, ob man Clustering einsetzt oder einfach mehrere Systeme über Load Balancing verknüpft werden sollen und man bei Ausfall einer Linie ein erneutes Einloggen des Benutzers in Kauf nimmt. Hier spielen dann Themen wie Replikation der Daten zwischen den Systemen, Cache-Kohärenz, eventuelle Veränderung der Konfiguration zur Laufzeit sowie Hot Deployment eine wichtige Rolle. Diese Anforderung hat unmittelbare Auswirkungen auf die Performance und Skalierbarkeit. Die Einhaltung der Anforderungen lässt sich

verhältnismäßig einfach überprüfen – man zieht den Stecker.

Die Robustheit sollte mit Tests überprüft werden, indem falsche Eingaben gemacht werden oder Daten im Backend nicht richtig sind. Um die Robustheit des ganzen Systems abzusichern, sollte hier überprüft werden, dass alle Codestellen durchlaufen worden sind. Dies wird mithilfe von Code-Coverage-Tools erreicht.

Bei der Skalierbarkeit ist es nötig, die bestehenden und zukünftigen Anforderungen zu ermitteln. Diese Informationen sollten sich in entsprechenden Lasttests mit unterschiedlichen Szenarien (Normallast, Volllast, Überlast) wiederfinden. Andere, eher interne Aspekte, wie zum Beispiel Flexibilität, Wiederverwendbarkeit, Portabilität, und die Erweiterbarkeit des Codes oder auch Security werden in dem

Anforderung	Beschreibung	Wechselwirkung
Hochverfügbarkeit/Ausfallsicherheit	Definiert, wie oft in einem bestimmten Zeitraum die Software ausfallen darf.	Performance/Skalierbarkeit
Robustheit	Gibt die Fehlertoleranz und das Verhalten im Fehlerfall an.	Komplexität
Wartbarkeit	Definiert den Aufwand für die Wartung des Systems nach Abschluss der Implementierung.	Komplexität
Skalierbarkeit	Die Fähigkeit des Systems, seine Dienste bei zunehmender Anzahl von Anfragen in der geforderten Antwortzeit anzubieten.	Performance
Erweiterbarkeit	Je höher die Erweiterbarkeit, desto effizienter lässt sich neue Funktionalität nachträglich bereitstellen.	Wartbarkeit/Komplexität/Performance
Flexibilität	Die Fähigkeit, auf Hard- und Softwareänderungen effizient zu reagieren.	Komplexität/Performance
Wiederverwendbarkeit	Die Fähigkeit, bestehende (Teil-)Funktionalität eines Systems wiederzuverwenden.	Wartbarkeit/Erweiterbarkeit/Komplexität
Portabilität	Bestimmt, wie effizient eine Anwendung auf eine andere Softwareplattform migriert werden kann.	Komplexität
Performance	Die Fähigkeit, die Dienste des Systems im Rahmen der Vorgaben zu erfüllen.	Skalierbarkeit/Wartbarkeit/Erweiterbarkeit
Security	Bestimmt, wie sicher das System ist.	Komplexität/Performance/Skalierbarkeit

Tabelle 1: Nicht funktionale Anforderungen

Ihr Feedback

Wir freuen uns auf Ihr Feedback zu unserer „Qualität in Java-Projekten“-Reihe. Bitte schreiben Sie uns: qualitaet@javamagazin.de.

nächsten Artikel der Reihe genauer behandelt.

Need for Speed

Hat man nun also eine Applikation gezaubert, die durch vorbildliches Projektmanagement [1] in time und im Rahmen des Budget genau den Vorstellungen der Benutzer entspricht [2], heißt das noch lange nicht, dass das Ganze noch befriedigend läuft, wenn erst mal die Datenmengen und/oder Benutzerzahlen des Produktivbetriebs auftreten. Um dies sicherzustellen und die Qualität auch bei nicht funktionalen Anforderungen abzusichern, ist es notwendig, sich aktiv in jeder Projektphase darum zu kümmern. In den einzelnen Projektphasen müssen unterschiedliche Punkte betrachtet werden. Dies fängt schon bei der Erfassung der Anforderungen an. Man darf jedoch nicht die Kosten außer Acht lassen, denn nicht nur Probleme kosten Geld, sondern auch deren Vermeidung. Hier gilt es, die richtige Balance zu finden (Abb. 1). Dabei kann Performance für verschiedene Betroffene unterschiedliche Dinge bedeuten.

Ein cooler grafischer Übergangseffekt zwischen zwei Seiten einer Datenerfassung ist für den Entwickler performant, weil es nicht ruckelt, auch wenn im Hintergrund das neueste Musikalbum MP3-kodiert wird. Der Sachbearbeiter, der pro Stunde 80 Anträge einhacken soll, wird allerdings von den zwei Sekunden Übergangszeit eher genervt sein. Achten Sie darauf, die richtigen Dinge zu optimieren. Speziell im Business-Umfeld kommt es oft vor, dass große Datenmengen schnell ma-

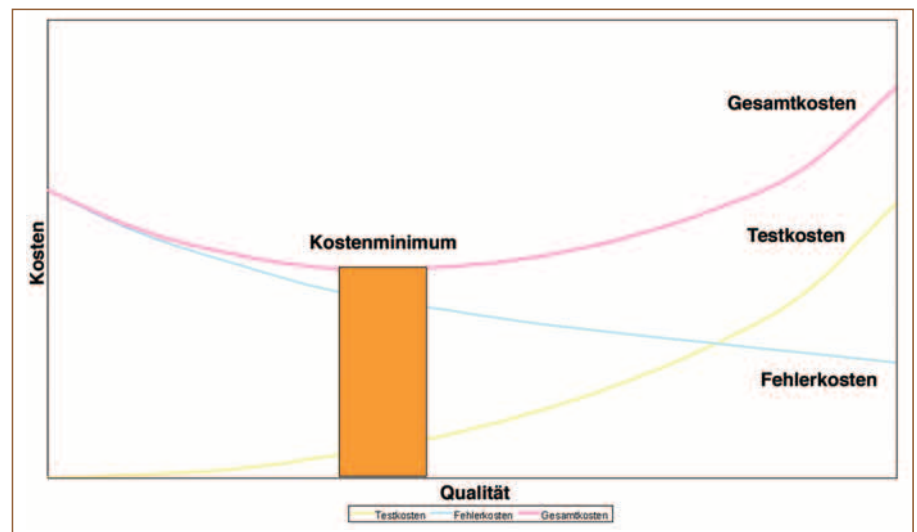


Abb. 1: Beziehung zwischen Fehler- und Qualitätskosten

nuell erfasst werden müssen. Menschen, die nichts anderes als das Web kennen, machen sich vielleicht nichts aus sekundelangen Pausen zwischen Seiten, aber manchen Datenerfassern will nicht recht einleuchten, warum Sie mit der neuen tolen bunten Software, die alles besser machen soll, langsamer sind als mit dem alten 3270-Terminal.

Beliebte Produktivitätshemmer sind auch Seiten, die zu groß sind, um auf den Bildschirm zu passen, sodass man immer hin und her scrollen muss. Auch Seiten, die nicht komplett tastaturbedienbar sind und zum ständigen Wechsel zwischen Tastatur (Eingabe) und Maus (Navigation) zwingen, können Benutzer ausbremsen. Da nützt es auch nichts mehr, wenn man den Request zum Server noch um ein paar Millisekunden beschleunigt. Überlegen

Sie also genau, wo Sie ansetzen, wenn Sie die Performance optimieren wollen. Letztendlich zählen nicht die Zahlen, die in den Logfiles stehen, sondern wie schnell die Benutzer dann ihre Aufgaben erledigen können.

Projektstart

Um die Qualität in einem Projekt messen zu können, ist es wichtig, zunächst die Anforderungen an das zu bauende System zu definieren. Hierzu gehört nicht nur die Definition des Mengengerüsts und der dazugehörigen Antwortzeiten, sondern auch, unter welchen Bedingungen die Anwendung arbeitet. Folgende Fragen sind vorab zu klären:

- Welche Anwendungsfälle sind bezüglich der Performanz besonders kritisch?

- Welche Infrastrukturkomponenten sind nicht skalierbar?
- Gibt es Spitzenzeiten, in denen die Anwendung besonders frequentiert ist?
- Mit wie vielen gleichzeitigen Sitzungen ist in den Spitzenzeiten zu rechnen?
- Mit wie vielen gleichzeitigen Anfragen/Transaktionen ist in Spitzenzeiten zu rechnen?
- Auf welchen Datenbeständen soll eine Anwendung operieren (Menge der Testdaten!)?
- Welche Datenmengen sind zwischen den einzelnen Schichten der Anwendung zu übertragen?
- Mit welchen Wachstumsraten ist in der Zukunft zu rechnen?

Hat man sich erst mal zu diesem Aspekt Gedanken gemacht, ist der nächste Schritt das Aufsetzen eines vernünftigen Software-Entwicklungsprozesses (SEP). Im Laufe eines Projektes ergeben sich Änderungen, die unmittelbar Auswirkungen auf den SEP und somit auf die Entwicklungs-Performance haben. Die Entwicklungs-Performance, also die Zeit, die für das Erstellen des Projektes benötigt wird, wird meistens bei den Betrachtungen vergessen. So kann es passieren, dass die Round-Trip-Zeit (Kodierung, Generierung, Deployment) bei Änderungen durch lange Build-Zeiten unerträglich wird. Dies ist dann meist Zeit, die einfach damit verbracht wird, darauf zu warten, dass man endlich testen kann. So ist es z.B. sinnvoll, sehr große Projekte geeignet zu modularisieren, damit das Testen von Teilen der Anwendung nicht Build und Deployment der ganzen Anwendung erfordert.

Implementierung

Die Qualität der Implementierung hängt natürlich vom Know-how der einzelnen Entwickler ab. Hier sollte der Projektleiter die Teammitglieder und deren Leistungsfähigkeit einschätzen können. Um sich dabei jedoch nicht auf sein Bauchgefühl verlassen zu müssen, gibt es verschiedene Mittel und Wege, möglichst früh Einfluss zu nehmen, um die Qualität zu sichern. Auch hier gilt das Prinzip: Je früher die Probleme erkannt werden, desto einfacher und günstiger ist es, diese zu beheben. So ist es möglich, durch eine frühe Integration von

automatisierten Tests Probleme zu erkennen und ihnen entgegenzuwirken. So kann man sogar bei JUnit-Tests und Nightly Builds feststellen, ob sich die Zeiten der Tests bei gleich bleibender Funktionalität von Tag zu Tag verschlechtern. Um Schwächen einzelner Teammitglieder auszugleichen, kann man Pair Programming einsetzen. Dies fördert gleichzeitig das Knowledge Sharing und dient als implizites Review.

Bei der Implementierung muss man mit dem Einbau typischer performancehemmender Konstrukte rechnen. Solche Probleme können unter anderem synchronisierte Blöcke, die Erzeugung vieler Objekte (unmittelbarer Zusammenhang zum Garbage Collection) und die Verwendung von synchronisierten Collection/Maps an Stellen, wo auch die unsynchronisierten benutzt werden können. Für die Auffindung solcher Probleme müssen Performance- und Lasttests sowie Codereviews durchgeführt werden. Auch kann man hier Bug-Finder-Tools wie zum Beispiel FindBugs [3], JLint [4] und PMD [5] zum Einsatz bringen.

Performance-Analyse

Bei Analysen bestehender Projekte, die als Backend eine Datenbank benutzen, stellt man oft fest, dass diese Anbindung Performance-Probleme verursacht. So werden z.B. falsche oder gar keine Indizes auf Tabellen gesetzt, keine Prepared Statements benutzt oder die Statements an sich sind nicht performant. Nur selten ist das Datenbankschema ungeeignet. Hier sollte man wirklich jedes Statement sammeln und entsprechend von einem Datenbankexperten prüfen und optimieren lassen. Hier kann man auch Tools benutzen, die den Execution Plan und Ausführungszeiten aufzeigen.

Um diese Probleme erkennen zu können, müssen demnach entsprechende Tests durchgeführt werden. Es ist jedoch wichtig, dass die entsprechenden relevanten Messpunkte in der Applikation vorhanden sind. Für diese Tests ist es essenziell, entsprechende Profile wie für

- Normallast,
- Volllast und
- Überlast

zu erstellen und mit den erforderlichen Testdaten auszustatten. Dabei ist darauf zu achten, dass der Umfang der Testdaten und das Sizing des Testsystems dem des produktiven Umfeldes gleichen.

Für die Analyse können auch Profiling-Tools eingesetzt werden. Diese Tools (zum Beispiel EclipseProfiler, JProbe [6], der JavaScript Profiler Venkman [7]) ermitteln die Ausführungszeiten von Aufrufen, erstellen Ausführungsgraphen, machen das Speicherverhalten sichtbar und ermöglichen es, Threads und deren Ausführung gezielt zu verfolgen.

Oft werden nur die Businesslogik und Backend-Schicht auf deren Qualität hinsichtlich Performanz geprüft. Man sollte jedoch auf keinen Fall die Client-Seite vernachlässigen. So haben wir in einem Fall festgestellt, dass eine Webanwendung massiven Gebrauch von JavaScript machte und es durch die entsprechende Interpretation zu massiven Performance-Problemen kam. In der Entwicklung wurden keine Probleme festgestellt, da die entsprechende technische Ausstattung der Entwicklerarbeitsplätze gegeben war. Die älteren, kleineren Rechner der Anwender waren dagegen nicht leistungsfähig genug (Bildaufbau im Minutenbereich). An dieser Stelle stand man dann vor der Wahl, die Logik der Präsentationsschicht zu ändern oder 150 Arbeitsplätze mit neuen Rechnern auszustatten.

Auch vor unorthodoxen Methoden sollte man nicht zurückschrecken, wie das folgende Beispiel zeigt: Die Aufgabenstellung war eine Performanzanalyse eines bestehenden Systems, das sich jedoch nicht nur aus Java-Komponenten zusammensetzte, sondern auch noch Host- und Cobol-Komponenten enthielt. Diese waren auf bis zu drei unterschiedlichen Systemen verteilt. Zunächst war unklar, wo genau die Performance-Probleme verursacht wurden. Schließlich gelang es, eine Analyse über die Netzwerkkommunikation durchzuführen. Hierfür wurden auf den einzelnen Maschinen Netzwerk-Traces erstellt und dann über ein selbst geschriebenes Java-Programm ausgewertet. Dabei wurden die verschiedenen Systemübergänge sichtbar gemacht und bei Auffälligkeiten konnte dann jeweils genauer im Netzwerkprotokoll nachgesehen werden,

Anzeige

um das Problem zu erkennen. Dies ergab, dass verschiedene Datenbank-Statements optimiert werden müssen.

Konfiguri ...

Hatte man als Programmierer vor 50 Jahren noch die ganze Maschine für sich, haben sich mittlerweile unter unseren Applikationen diverse Softwareschichten angesammelt, die uns vieles erleichtern. Da aber ihre Erschaffer nicht vorausahnen konnten, was wir alles Tolles mit unserer Applikation vorhaben, sind diverse Parameter so mit Kompromisswerten belegt, dass sie in den vielen verschiedenen möglichen Anwendungsfällen nicht allzu viel kaputt machen. Zum Glück können wir daran aber oft drehen, um für unseren konkreten Fall ein bisschen mehr Power zu bekommen. Ein bunter Strauß an mehr oder minder dokumentierten (und mehr oder weniger voneinander unabhängigen) Einstellungen erlauben uns, Stunde um Stunde an den richtigen Einstellungen zu feilen. Als Ansatzpunkte bieten sich das Betriebssystem, die VM und, so vorhanden, der Application Server an.

Die JVM von Sun z.B. bietet zwei verschiedene JIT-Implementierungen an, eine Client-Version und eine Server-Version. Die Client-Version ist für kurze Latenzzeiten optimiert, was dann auf Kosten des Durchsatzes gehen kann. Das ist für Benutzeroberflächen interessant, wo es von dem User gern gesehen wird, dass das Programm schnell startet und beim Klick auf Buttons auch schnell eine Rückmeldung kommt. Die Server-Version favorisiert dagegen Durchsatz gegenüber Latenz. Wenn der JIT in diesem Fall ein paar Sekunden mehr zum Optimieren des Codes braucht, macht sich das beim späteren Aufruf des Codes allemal wieder bezahlt. Hier kann man also durch entsprechende Wahl einiges bewegen.

Ein weiteres Thema bei der JVM-Konfiguration, das Speichermanagement, ist da deutlich komplizierter. Eine Vielzahl von Parametern kann gesetzt werden, die sich natürlich gegenseitig beeinflussen. Zu diesem Thema gibt es eine Reihe von Tuning-Guides, die man zu Rate ziehen sollte. Dabei ist darauf zu achten, den zur verwendeten JVM-Version passenden zu verwenden, denn gerade in diesem Bereich

ändert sich bei einem Versionswechsel oft einiges.

Zur Erlangung einer günstigen Konfiguration hilft meist nur, durch Parametervariation und einer dem erwarteten Benutzungprofil möglichst nahe kommenden Simulation der Profile die günstigsten Werte zu ermitteln. Dabei ist gerade bei Serveranwendungen darauf zu achten, dass nicht nur die Performance stimmt, sondern auch kein Memory Leak vorliegt. Wenn bei konstanter Last nach jeder Full GC immer weniger freier Speicher übrig bleibt, hat man ein Problem. Es dauert vielleicht drei Wochen, aber irgendwann ist der Speicher doch zu Ende.

Hüten sollte man sich vor der Methode „viel hilft viel“. Zum einen wiegt man sich dadurch leicht in falscher Sicherheit, was Memory Leaks angeht, zum anderen muss der ganze Müll, der sich da ansammelt, irgendwann weggeräumt werden. Wenn man den Heap viel größer als den typischen Working Set der Anwendung macht, kann es passieren, dass, wenn die Zeit für eine Full GC reif ist, der Großteil auf die Platte ausgelagert ist. Wenn dann nicht genug physikalisches RAM für das Working Set der GC da ist und man ins Disk Thrashing gerät, sieht man echt alt aus (leider kein konstruierter Fall ...).

... Konfigura

Die Application Server erfreuen uns ebenfalls mit einer Plethora von möglichen Einstellungen. Hier kommt es leicht zu Performance-Engpässen, wenn man diverse Pools zu klein wählt. Typischerweise gibt es eine Reihe von Thread Pools, Connection Pools und diversen Caches. Die Admin-Tools der Server sollten darüber Auskunft geben, wie gut die Auslastung ist bzw. welche dieser Ressourcen an ihre Grenzen stoßen. Ein Anzeichen für das Ausbremsen der Anwendung durch diese Phänomene ist auch, dass die Performance in den Keller geht, die CPU-Aktivität aber niedrig bleibt.

Um den Application Server zum Fliegen zu bringen, sollte man auch nachschauen, welche nicht benötigten Komponenten man abschalten kann. Das spart zur Laufzeit Speicher (was bei GCs auch Zeit bedeutet) und Zeit bei Start-up und Shutdown. Die Standardinstallation von Ap-

plication-Servern bringt auch oft Beispielapplikationen mit, die manchmal vergessen werden, auch die für sie Default-mäßig angelegten Ressourcen kann man gewinnbringend löschen, wie etwa Data Sources, an denen evtl. wieder Connection Pools hängen etc.

Jetzt aber

So, nun haben wir unser System zu unserer Zufriedenheit getunt. Können wir uns jetzt endlich dem nächsten spannenden Projekt zuwenden? Fast. Wie mehrfach erwähnt, haben wir auf ein bestimmtes Benutzungsszenario hin optimiert und konfiguriert. Wenn unsere Anwendung aber wirklich so toll ist, wird das nicht lange so bleiben. Sie wird neue User anziehen und diese werden ihr immer mehr Daten anvertrauen und damit immer mehr Last erzeugen. Wir tun also gut daran, das System auch im Betrieb zu beobachten und die Konfiguration entsprechend anzupassen. Manchmal wird sogar das Redesign einer Komponente nötig sein, um unter den veränderten Bedingungen weiter bestehen zu können. Davon abgesehen verlangen die User natürlich auch nach neuen Features, die sie noch glücklicher machen. Dazu sollten natürlich das Applikations-Design und der Code entsprechend änderungsfreundlich sein, ein Thema, das der nächste Artikel dieser Reihe beleuchten wird.

Jörn Eyrich ist Senior Consultant Technology bei syngenio. Er konzipiert, realisiert und testet seit mehreren Jahren Enterprise-Java-Applikationen.

David Gärtner ist Senior Consultant Technology bei syngenio. In seiner Projektarbeit beschäftigt er sich mit Konzeption, Realisierung, Testen und Reviews von Enterprise-Java-Applikationen.

■ Links & Literatur

- [1] Elmar Borgmeier: Intelligenz statt Konformität. Qualität in Java-Projekten, Teil 1: Mit agilem Qualitätsmanagement zum Projekterfolg, in *Java Magazin* 7.2005
- [2] Lars Bachert, Stefan Reisner: Das können wir nicht gebrauchen! Qualität in Java-Projekten, Teil 2: der anwenderbezogene Qualitätsaspekt, in *Java Magazin* 8.2005
- [3] findbugs.sourceforge.net
- [4] artho.com/jlint/
- [5] pmd.sourceforge.net
- [6] www.quest.com/jprobe/
- [7] www.mozilla.org/projects/venkman/