Certified Tester

Foundation Level Specialist Lehrplan Performanztest

Version 2019

Zur Verfügung gestellt vom
American Software Testing Qualifications Board
und German Testing Board

Deutschsprachige Ausgabe
Herausgegeben durch das German Testing Board e.V.

Übersetzung des englischsprachigen Lehrplans des International Software Testing Qualifications Board (ISTQB®), Version 2018.

Urheberrechtsvermerk

Arbeitsgruppe Performance Testing:

Graham Bath
Rex Black
Alexander Podelko
Andrew Pollner
Randy Rice

Änderungshistorie der deutschen Version

Version	Datum	Bemerkungen
GTB V0.1	25. September 2019	Review-Kommentare eingearbeitet
GTB V0.2	28. Oktober 2019	Klärung offener Punkte
GTB V0.3	30. Oktober 2019	Tippfehler korrigiert, Urhebervermerk überarbeitet
GTB 2019	01. November 2019	Durch GTB freigegebene deutsche Version

Inhaltsverzeichnis

Dank

Das vorliegende Dokument wurde vom American Software Testing Qualifications Board (ASTQB) und vom German Testing Board (GTB) erstellt:

Graham Bath (GTB, Co-Vorsitzender der Arbeitsgruppe)
Rex Black
Alexander Podelko (CMG)
Andrew Pollner (ASTQB, Co-Vorsitzender der Arbeitsgruppe)
Randy Rice

Das Kernteam bedankt sich beim Reviewteam für die Vorschläge und Beiträge. Das ASTQB dankt der Computer Measurement Group (CMG) für ihren Beitrag zur Entwicklung dieses Lehrplans.

Folgende Personen haben an Review, Kommentierung und der Abstimmung über diesen Lehrplan und/oder dessen Vorgängerversionen mitgearbeitet (in alphabetischer Reihenfolge):

Dani Almog	Marek Majernik	Péter Sótér
Sebastian Chece	Stefan Massonet	Michael Stahl
Todd DeCapua	Judy McKay	Jan Stiller
Wim Decoutere	Gary Mogyorodi	Federico Toledo
Frans Dijkman	Joern Muenzel	Andrea Szabó
Jiangru Fu	Petr Neugebauer	Yaron Tsubery
Matthias Hamburg	Ingvar Nordström	Stephanie Ulrich
Ágota Horváth	Meile Posthuma	Mohit Verma
Mieke Jungebload	Michaël Pilaeten	Armin Wachter
Beata Karpinska	Filip Rechtoris	Huaiwen Yang
Gábor Ladányi	Adam Roman	Ting Yang
Kai Lepler	Dirk Schweier	
Ine Lutterman	Marcus Seyfert	

Dieses Dokument wurde von der Generalversammlung des ISTQB® am 9. Dezember 2018 offiziell freigegeben.

Das German Testing Board (GTB) dankt dem Reviewteam der deutschsprachigen Fassung 2019: Anke Löwer, Jörn Münzel, Dr. Klaudia Dussa-Zieger und Graham Bath (Leitung).

0. Einführung in den Syllabus

0.1 Zweck dieses Dokuments

Dieser Lehrplan bildet die Grundlage für das Softwaretest-Qualifizierungsprogramm Performanztest der Basisstufe (Foundation Level). ASTQB® und GTB® stellen diesen Lehrplan folgenden Adressaten zur Verfügung:

1. Nationalen/regionalen Boards zur Übersetzung in die jeweilige Landessprache und zur Akkreditierung von Ausbildungsanbietern. Die nationalen Boards können den Lehrplan an die eigenen sprachlichen Anforderungen anpassen sowie die Querverweise ändern und an die bei ihnen vorliegenden Veröffentlichungen angleichen.
2. Prüfungsinstitutionen zur Erarbeitung von Prüfungsfragen in der jeweiligen Landessprache, die sich an den Lernzielen der jeweiligen Lehrpläne orientieren.
3. Ausbildungsanbietern zur Erstellung ihrer Kursunterlagen und zur Bestimmung einer geeigneten Unterrichtsmethodik.
4. Prüfungskandidaten zur Vorbereitung auf die Prüfung (als Teil des Ausbildungslehrgangs oder auch kursunabhängig).
5. Allen Personen, die im Bereich Software- und Systementwicklung tätig sind und die professionelle Kompetenz beim Testen von Software verbessern möchten, sowie als Grundlage für Bücher und Fachartikel.

ASTQB® und GTB® können auch anderen Personenkreisen oder Institutionen die Nutzung dieses Lehrplans für andere Zwecke genehmigen, wenn diese vorab eine entsprechende schriftliche Genehmigung einholen.

0.2 Der Certified Tester Foundation Level Performanztest

Die Basisstufe (Foundation Level) des Certified Tester-Ausbildungsprogramms richtet sich an alle Personen im Bereich Softwaretest, die ihr Wissen über den Performanztest vertiefen wollen, bzw. an Personen, die sich in ihrer beruflichen Laufbahn auf den Performanztest spezialisieren wollen. Das Ausbildungsprogramm ist auch für Personen geeignet, die mit Performance-Engineering zu tun haben und ein grundlegendes Verständnis über das Thema Performanztest erwerben möchten.

Der Lehrplan behandelt die folgenden Hauptaspekte des Performanztests:

- Technische Aspekte
- Methodenbezogene Aspekte
- Organisatorische Aspekte

Die Inhalte zum Thema Performanztest, die im ISTQB®-Lehrplan Advanced Level Technical Test Analyst [ISTQB_ALTTA_SYL] beschrieben werden, sind konsistent mit

diesem Lehrplan und sind auf dieser Grundlage entwickelt.

0.3 Geschäftlicher Nutzen

In diesem Kapitel wird der geschäftliche Nutzen aufgezeigt, der von Kandidaten mit Zertifizierung im Foundation Level Performanztest erwartet werden kann.

PTFL-1 Verstehen der grundlegenden Konzepte von Performanz und Performanztest
PTFL-2 Definieren von Performanzrisiken, -zielen und -anforderungen, um die Erfordernisse und Erwartungen der Stakeholder zu erfüllen
PTFL-3 Verstehen der Performanzmetriken und wie diese gesammelt werden
PTFL-4 Entwickeln eines Performanztestkonzepts, um die gesetzten Ziele und Anforderungen zu erreichen
PTFL-5 Konzeptionelles Entwerfen, Realisieren und Durchführen grundlegender Performanztests
PTFL-6 Analysieren der Ergebnisse eines Performanztests und Bestimmen der Auswirkungen auf die verschiedenen Stakeholder
PTFL-7 Erläutern des Prozesses, der Begründung, der Ergebnisse und der Auswirkungen von Performanztests gegenüber verschiedenen Stakeholdern
PTFL-8 Verstehen der verschiedenen Kategorien und Verwendungen von Performanztestwerkzeugen sowie der Kriterien für deren Auswahl
PTFL-9 Bestimmen, wie die Performanztestaktivitäten im Softwarelebenszyklus eingeordnet werden

0.4 Prüfungsrelevante Lernziele

Die Lernziele unterstützen die geschäftlichen Ziele und dienen zur Ausarbeitung der Prüfung für die Zertifizierung im Foundation Level Performanztest (CTFL-PT). Den einzelnen Lernzielen ist jeweils eine kognitive Stufe des Wissens (K-Stufe) zugeordnet.

Die K-Stufe (bzw. kognitive Stufe) dient dazu, Lernziele gemäß der überarbeiteten Taxonomie von Bloom [Anderson01] zu klassifizieren. Das ISTQB® verwendet diese Taxonomie bei der Erstellung der Prüfungen zu den Lehrplänen.

Dieser Lehrplan berücksichtigt vier verschiedene kognitive Stufen (K1 bis K4):

K-Stufe	Kennwort	Beschreibung
1	Kennen	Prüfungskandidaten sollen Begriffe oder Konzepte erkennen und sich an sie erinnern können.
2	Verstehen	Prüfungskandidaten sollen Erklärungen zu Aussagen in Zusammenhang mit dem Gegenstand der Frage auswählen können.
3	Anwenden	Prüfungskandidaten sollen die korrekte Anwendung eines Konzepts oder einer Methode auswählen und in einem vorgegebenen Kontext anwenden können.
4	Analysieren	Prüfungskandidaten sollen Informationen in Zusammenhang mit einem Verfahren oder einer Vorgehensweise zum besseren Verständnis in ihre Bestandteile zerlegen und zwischen Fakten und Folgerungen unterscheiden können.

Allgemein gilt, dass der gesamte Inhalt des vorliegenden Lehrplans entsprechend der Lernziele der kognitiven Stufe K1 geprüft werden kann. Dies bedeutet, dass die Prüfungskandidaten Begriffe oder Konzepte erkennen, sich an sie erinnern und sie wiedergeben können. Die relevanten Lernziele der kognitiven Stufen K2, K3 und K4 werden immer zu Beginn des jeweiligen Kapitels angegeben.

0.5 Empfohlene Unterrichtszeiten

Für jedes Lernziel dieses Lehrplans wurde die Mindestschulungszeit festgelegt. Die gesamte Unterrichtszeit für jedes Kapitel ist in der Kapitelüberschrift angegeben.

Seminaranbieter werden darauf hingewiesen, dass in anderen Lehrplänen des ISTQB® Standardzeiten zur Anwendung kommen, die festgelegte Unterrichtszeiten je nach K-Stufe zuordnen. Der Performanztest-Lehrplan wendet dieses Schema jedoch nicht strikt an. Dies gibt den Seminaranbietern einen flexibleren und realistischeren Anhaltspunkt hinsichtlich der jeweiligen Mindestunterrichtszeiten für die einzelnen Lernziele.

0.6 Zulassung zur Prüfung

Voraussetzung für die Prüfung zum CTFL Specialist Performanztest ist das erworbene Zertifikat zum ISTQB® Certified Tester Foundation Level (CTFL®).

0.7 Informationsquellen

Die im Lehrplan verwendeten Begriffe sind im ISTQB Standardglossar der Testbegriffe definiert [ISTQB_GLOSSARY].

Kapitel 6 enthält eine Liste empfohlener Bücher und Fachartikel zum Thema Performanztest.

1. Grundkonzepte – 60 Min.

Schlüsselbegriffe

Dauertest, Kapazitätstest, Lastgenerierung, Lasttest, Nebenläufigkeitstest, Performanztest, Skalierbarkeitstest, Lastspitzentest, Stresstest

Lernziele

1.1 Grundsätze und Konzepte
PTFL-1.1.1 (K2) Die Grundsätze des Performanztests verstehen

1.2 Verschiedenen Arten von Performanztest
PTFL-1.2.1 (K2) Die verschiedenen Arten von Performanztest verstehen

1.3 Testarten im Performanztest
PTFL-1.3.1 (K1) Die Testarten im Performanztest wiedergeben können

1.4 Das Konzept der Lastgenerierung
PTFL-1.4.1 (K2) Das Konzept der Lastgenerierung verstehen

1.5 Die häufigsten Performanzfehlerarten und ihre Ursachen
PTFL-1.5.1 (K2) Beispiele für die häufigsten Performanzfehlerarten und ihre Ursachen nennen können

1.1 Grundsätze des Performanztests

Die Performanz eines Systems ist ein wesentlicher Aspekt, um Benutzern ein „gutes Gefühl“ zu bieten, wenn sie ihre Anwendungen auf einer Vielzahl nicht-mobiler und mobiler Plattformen nutzen. Performanztests spielen somit eine entscheidende Rolle bei der Ermittlung akzeptabler Qualitätsniveaus für die Endbenutzer und sind oft eng mit anderen Disziplinen wie Gebrauchstauglichkeits-Engineering und Performanz-Engineering verbunden.

Darüber hinaus kann die Bewertung der funktionalen Eignung, der Gebrauchstauglichkeit und anderer Qualitätsmerkmale unter Lastbedingungen, z.B. während der Durchführung eines Performanztests, lastspezifische Probleme aufdecken, die sich auf diese Qualitätsmerkmale auswirken.

Performanztests sind nicht auf die webbasierte Domäne beschränkt, in der der Endbenutzer im Mittelpunkt steht. Sie sind auch für verschiedene Anwendungsdomänen mit einer Vielzahl von Systemarchitekturen relevant, z.B. für klassische Client-Server-Architekturen, verteilte und eingebettete Architekturen.

Im Standard ISO 25010 [ISO25000] ist die Performanz im Produktqualitätsmodell als nicht-funktionales Qualitätsmerkmal mit den drei nachstehend beschriebenen Teilmerkmalen kategorisiert. Die richtige Fokussierung und Priorisierung hängen von den ermittelten Risiken und den Bedürfnissen der verschiedenen Stakeholder ab. Bei der Analyse der Testergebnisse können weitere Risikobereiche identifiziert werden, die zu berücksichtigen sind.

Zeitverhalten: Im Allgemeinen ist die Bewertung des Zeitverhaltens das häufigste Ziel von Performanztests. Dieser Aspekt des Performanztests untersucht die Fähigkeit einer Komponente oder eines Systems, innerhalb einer bestimmten Zeit und unter bestimmten Bedingungen auf Benutzer- oder Systemeingaben zu reagieren. Messungen des Zeitverhaltens können von der Gesamtzeit (End-to-End), die das System benötigt, um auf Benutzereingaben zu reagieren, bis zur Anzahl von CPU-Zyklen variieren, die eine Softwarekomponente zur Ausführung einer bestimmten Aufgabe benötigt.

Ressourcennutzung: Wenn die Verfügbarkeit von Systemressourcen als Risiko identifiziert wird, kann der Verbrauch dieser Ressourcen (z.B. die Zuweisung von begrenztem Hauptspeicher) mittels Durchführung bestimmter Performanztests untersucht werden**.**

Kapazität: Wenn Probleme des Systemverhaltens an den erforderlichen Kapazitätsgrenzen des Systems (z.B. hinsichtlich der Anzahl Benutzer oder der Datenmengen) als Risiko erkannt werden, können Performanztests durchgeführt werden, um die Eignung der Systemarchitektur zu bewerten.

Performanztests finden häufig in Form von Experimenten statt, wodurch eine Messung und Analyse bestimmter Systemparameter ermöglicht wird. Diese können zur Unterstützung von Systemanalyse, -entwurf und -realisierung iterativ durchgeführt werden, um Entscheidungen zur Systemarchitektur zu treffen und helfen die Erwartungen der Stakeholder zu gestalten.

Die folgenden Grundsätze des Performanztests sind besonders relevant:

- Tests müssen auf die spezifizierten Erwartungen verschiedener Stakeholder, insbesondere Benutzer, Systementwickler und Mitarbeiter im Betrieb, abgestimmt sein.
- Die Tests müssen reproduzierbar sein. Bei einer Wiederholung der Tests auf einem unveränderten System müssen statistisch identische Ergebnisse (innerhalb einer bestimmten Toleranz) erzielt werden.
- Die Tests müssen zu Ergebnissen führen, die verständlich sind und die leicht mit den Erwartungen der Stakeholder verglichen werden können.
- Die Tests können dort durchgeführt werden, wo es die Ressourcen erlauben, entweder in vollständigen Systemen, in Teilsystemen oder in Testumgebungen, die für das Produktionssystem repräsentativ sind.

- Die Tests müssen innerhalb des vom Projekt festgelegten Zeitrahmens kostenseitig erschwinglich und durchführbar sein.

Bücher von [Molyneaux09] und [Microsoft07] liefern solides Hintergrundwissen zu den Grundsätzen und praktischen Aspekten von Performanztests.

Alle drei der oben genannten Teilmerkmale der Performanz können sich auf die Skalierbarkeit des zu testenden Systems (SUT) auswirken.

1.2 Verschiedene Arten von Performanztest

Es lassen sich verschiedene Arten von Performanztests definieren. Jede dieser Arten kann je nach den Testzielen bei einem bestimmten Projekt angewendet werden.

Performanztests

Performanztest ist ein Oberbegriff für jegliche Art von Tests, die auf die Performanz (Reaktionsfähigkeit) des Systems oder einer Komponente unter unterschiedlicher Last ausgerichtet sind.

Lasttests

Lasttests konzentrieren sich auf die Fähigkeit eines Systems, mit steigender Last umzugehen, die realistisch zu erwarten ist, und die sich aus den Transaktionsanfragen ergibt, die durch eine kontrollierte Anzahl gleichzeitiger Benutzer oder Prozesse erzeugt werden.

Stresstests

Stresstests konzentrieren sich auf die Fähigkeit eines Systems oder einer Komponente, Spitzenlasten zu bewältigen, die an oder über den Grenzen der erwarteten oder spezifizierten Lastanforderungen liegen. Stresstests werden auch verwendet, um die Fähigkeit eines Systems zu bewerten, die reduzierte Verfügbarkeit von Ressourcen, wie z.B. verfügbare Rechnerkapazität, Bandbreite und Arbeitsspeicher, zu bewältigen.

Skalierbarkeitstests

Skalierbarkeitstests konzentrieren sich auf die Fähigkeit eines Systems oder einer Komponente, zukünftige Effizienzanforderungen zu erfüllen, die über den gegenwärtigen liegen. Ziel dieser Tests ist es zu beurteilen, ob das System wachsen kann (beispielsweise mit mehr Nutzern oder größeren Mengen von gespeicherten Daten), ohne abzustürzen oder gegen die aktuell festgelegten Performanzanforderungen zu verstoßen. Sind die Grenzen der Skalierbarkeit bekannt, lassen sich Schwellenwerte festlegen und in der Produktion überwachen, sodass bei bevorstehenden Problemen eine Warnung erfolgen kann. Außerdem kann die Produktionsumgebung durch entsprechende Hardware angepasst werden.

Lastspitzentest

Lastspitzentests konzentrieren sich auf die Fähigkeit eines Systems oder einer Komponente, auf plötzlich auftretende Spitzenlasten richtig zu reagieren und danach in einen stabilen Zustand zurückzukehren.

Dauertests

Dauertests konzentrieren sich auf die Stabilität des Systems über einen Zeitraum hinweg, der für den betrieblichen Kontext des Systems typisch ist. Durch diese Art von Tests wird sichergestellt, dass keine Probleme hinsichtlich der Ressourcenkapazität vorliegen (z.B. Speicherlecks, Datenbankverbindungen, Thread-Pools), die möglicherweise die Leistung irgendwann beeinträchtigen und/oder Ausfälle verursachen werden, wenn Grenzen erreicht bzw. überschritten werden.

Nebenläufigkeitstests

Nebenläufigkeitstests konzentrieren sich auf die Auswirkungen von Situationen, in denen bestimmte Aktionen gleichzeitig ausgeführt werden (beispielsweise die Situation, wenn sich eine große Anzahl von Benutzern gleichzeitig anmeldet). Nebenläufigkeitsprobleme sind äußerst schwer zu finden und zu reproduzieren, insbesondere wenn das Problem in einer Umgebung auftritt, auf die das Testen wenig oder keinen Einfluss hat, wie z.B. in der Produktion.

Kapazitätstests

Mit Kapazitätstests lässt sich bestimmen, wie viele Benutzer und/oder Transaktionen ein bestimmtes System unterstützt und dabei die erklärten Performanzziele noch erfüllt. Diese Ziele können auch im Hinblick auf die aus den Transaktionen resultierenden Datenmengen angegeben werden.

1.3 Testarten im Performanztest

Die wichtigsten Testarten, die bei Performanztests verwendet werden, sind der statische und der dynamische Test.

1.3.1 Statischer Test

Statische Testaktivitäten sind bei Performanztests oft wichtiger als bei Funktionstests. Dies liegt daran, dass viele kritische Performanzfehler in der Architektur und dem Entwurf des Systems begründet sind. Diese Fehler können durch Missverständnisse oder mangelndes Wissen der Systemdesigner und -architekten verursacht werden. Zu diesen Fehlern kann es auch kommen, weil die Anforderungen bezüglich der Antwortzeit, des Systemdurchsatzes oder der Ressourcennutzung, der erwarteten

Last und Nutzung des Systems oder die Einschränkungen nicht angemessen erfasst wurden.

Statische Testaktivitäten für Performanztests können umfassen:

- Reviews der Anforderungen mit dem Schwerpunkt auf Performanzaspekte und -risiken
- Reviews von Datenbankschemata, Entity-Relationship-Diagrammen, Metadaten, 'SQL stored procedures' und Abfragen
- Reviews der System- und Netzwerkarchitektur
- Reviews kritischer Bereiche des Systemcodes (z B. komplexe Algorithmen)

1.3.2 Dynamischer Test

Im Laufe der Softwareentwicklung sollten die dynamischen Performanztests so frühzeitig wie möglich beginnen. Dynamische Performanztests können zu folgenden Gelegenheiten durchgeführt werden:

- Während der Komponententests, um potenzielle Engpässe unter Nutzung von Profilinformationen zu ermitteln, sowie um die Ressourcennutzung durch eine dynamische Analyse zu bewerten
- Während der Komponentenintegrationstests in Bezug auf alle wichtigen Anwendungsfälle und -abläufe, insbesondere bei der Integration unterschiedlicher Anwendungsfallfunktionen oder bei der Integration von Anwendungsabläufen mit den realen Systemen
- Während des Systemtests des gesamten Ende-zu-Ende-Verhaltens unter verschiedenen Lastbedingungen
- Während der Systemintegrationstests, insbesondere für Datenflüsse und Abläufe über wichtige Schnittstellen zwischen den Systemen. Bei der Systemintegration ist es nicht ungewöhnlich, dass es sich beim „Benutzer" um ein anderes System oder ein anderes Gerät handelt (z.B. Eingaben von Sensoreingängen und anderen Systemen)
- Während des Abnahmetests, um das Vertrauen der Benutzer, Kunden und Betreiber in die ordnungsgemäße Performanz des Systems aufzubauen und um eine Feinabstimmung des Systems im Betrieb unter realen Bedingungen durchzuführen (im Allgemeinen jedoch nicht, um Performanzfehler im System aufzudecken)

In den höheren Teststufen, z.B. bei den Systemtests und Systemintegrationstests, ist die Verwendung realistischer Umgebungen, Daten und Lasten von entscheidender Bedeutung, um belastbare Ergebnisse zu erhalten (siehe Kapitel 4). In der agilen Softwareentwicklung und in anderen iterativ-inkrementellen Lebenszyklen sollten Teams statische und dynamische Performanztests in frühe Iterationen integrieren, um Performanzrisiken frühzeitig zu erkennen, und damit nicht bis zu den abschließenden Iterationen warten.

Wenn spezifische oder neue Hardware Teil des Systems ist, können frühe dynamische Performanztests mit Simulatoren durchgeführt werden. Es ist jedoch ratsam, so bald wie möglich mit der eigentlichen Hardware zu testen, da Simulatoren Ressourceneinschränkungen und performanzbezogenes Verhalten oft nicht ausreichend abdecken.

1.4 Das Konzept der Lastgenerierung

Um die verschiedenen in Kapitel 1.2 beschriebenen Arten von Performanztests durchzuführen, müssen repräsentative Systemlasten modelliert, generiert und an das zu testende System übermittelt werden. Lasten sind mit den Dateneingaben vergleichbar, die für funktionale Testfälle verwendet werden, unterscheiden sich jedoch in den folgenden grundlegenden Aspekten:

- Die Last beim Performanztest muss viele Benutzereingaben repräsentieren, nicht nur die Eingaben eines Benutzers
- Die Generierung der Last für Performanztests erfordert möglicherweise spezielle Hardware und Werkzeuge
- Die Generierung der Last für Performanztests hängt davon ab, dass im getesteten System keine funktionalen Fehler mehr vorhanden sind, die die Testdurchführung beeinträchtigen könnten

Die effiziente und zuverlässige Erzeugung einer bestimmten Last ist ein wesentlicher Erfolgsfaktor bei der Durchführung von Performanztests. Es gibt verschiedene Möglichkeiten zur Lastgenerierung:

Lastgenerierung über die Benutzungsschnittstelle
Dies kann ein angemessener Ansatz sein, wenn nur eine kleine Anzahl von Benutzern repräsentiert werden soll und wenn die erforderliche Anzahl von Software-Clients zum Eingeben der erforderlichen Daten zur Verfügung steht. Dieser Ansatz kann auch in Verbindung mit Testausführungswerkzeugen für funktionale Tests verwendet werden, kann sich jedoch mit zunehmender Anzahl zu simulierender Benutzer schnell als nicht praktikabel erweisen. Die Stabilität der Benutzungsschnittstelle ist ebenfalls eine kritische Abhängigkeit: Häufige Änderungen können sich erheblich auf die Wiederholbarkeit von Performanztests und die Wartungskosten auswirken. Das Testen über die Benutzungsschnittstelle ist jedoch der repräsentativste Ansatz für End-to-End-Tests.

Lastgenerierung durch Crowds
Dieser Ansatz hängt von der Verfügbarkeit einer großen Anzahl von Testern ab, die echte Benutzer repräsentieren. Beim Crowd-Test werden so viele Tester organisiert, dass die gewünschte Last erzeugt werden kann. Dies kann eine geeignete Methode für das Testen von Anwendungen sein, die von jedem Ort der Welt aus erreichbar sind (z.B. webbasierte Apps); und es kann beinhalten, dass die Benutzer diese Last mit einer großen Vielzahl verschiedener Gerätetypen und -konfigurationen generieren.

Obwohl bei diesem Ansatz eine sehr große Anzahl von Benutzern eingesetzt werden kann, ist die erzeugte Last nicht so reproduzierbar und präzise wie bei anderen Optionen, und die Organisation ist deutlich komplexer.

Lastgenerierung über die Aufrufschnittstelle der Anwendung (API)
Dieser Ansatz ähnelt der Verwendung der Benutzungsschnittstelle für die Dateneingabe, verwendet jedoch anstelle dieser die Aufrufschnittstelle (API) der Anwendung, um die Benutzerinteraktion mit dem zu testenden System zu simulieren. Der Ansatz ist weniger anfällig für Änderungen (z.B. Verzögerungen) in der Benutzungsschnittstelle und ermöglicht, dass die Transaktionen auf dieselbe Weise verarbeitet werden, als würden sie direkt von einem Benutzer über die Benutzungsschnittstelle eingegeben. Es können dedizierte Skripte erstellt werden, die vielfach spezifische API-Routinen aufrufen und eine Simulation von mehr Benutzern ermöglichen als bei der Verwendung von Eingaben über die Benutzungsschnittstelle.

Lastgenerierung mit aufgenommenen Kommunikationsprotokollen
Dieser Ansatz basiert auf dem Aufnehmen der Benutzerinteraktionen mit dem zu testenden System auf Kommunikationsprotokollebene und die Verwendung dieser Skripte, um potenziell sehr viele Benutzer auf wiederholbare und zuverlässige Weise zu simulieren. Dieser werkzeugbasierte Ansatz wird in den Kapiteln 4.2.6 und 4.2.7 genauer beschrieben.

1.5 Häufige Performanzfehlerarten und ihre Ursachen

Es gibt zahlreiche verschiedene Performanzfehlerarten, die durch den dynamischen Test aufgedeckt werden können. Im Folgenden sind einige Beispiele für häufige Fehler/Ausfälle (einschließlich Systemabstürze) sowie deren typische Ursachen angeführt:

Langsame Reaktion in allen Laststufen
In manchen Fällen ist die Reaktion - unabhängig von der Last - nicht akzeptabel. Dies kann durch zugrunde liegende Performanzprobleme verursacht werden, unter anderem durch schlechten Datenbankentwurf oder -implementierung, Netzwerklatenz oder sonstige Hintergrundlasten. Solche Probleme können nicht nur im Performanztest, sondern auch im funktionalen Test und im Gebrauchstauglichkeitstest identifiziert werden. Testanalysten sollten daher auch diesen Aspekt beobachten und ggfs. rechtzeitig melden.

Langsame Reaktion in moderaten bis hohen Laststufen
In manchen Fällen verschlechtert sich die Reaktion bei moderater bis hoher Last auf ein inakzeptables Maß, selbst wenn diese Last komplett innerhalb des normal zu erwartenden und zulässigen Anforderungsbereichs liegt. Die oft zugrunde liegenden Fehlerzustände sind die Überlastung einer oder mehrerer Ressourcen und variierende Hintergrundlasten.

Verschlechterung der Reaktion im Laufe der Zeit
In einigen Fällen verschlechtert sich die Reaktion schrittweise oder sprunghaft im Laufe der Zeit. Zu den zugrunde liegenden Ursachen gehören Speicherlecks, Festplattenfragmentierung, zunehmende Netzwerklast, wachsendes Datei-Repository und unerwartetes Datenbankwachstum.

Unzureichende oder unpassende Fehlerbehandlung bei sehr hoher Last oder bei Überschreitung der Lastgrenze
Es gibt Fälle bei hoher Last oder bei Überschreitung der Lastgrenzen, in denen die Antwortzeit zwar akzeptabel bleibt, die Fehlerbehandlung aber schlechter wird. Zu den zugrunde liegenden Fehlern gehören unzureichende Ressourcenpools, zu kleine Warteschlangen und Stacks, sowie zu kurze Timeout-Einstellungen.

Nachfolgend einige spezifische Beispiele für die oben aufgeführten allgemeinen Arten von Fehlerwirkungen einschließlich Ausfällen:

- Eine webbasierte Anwendung, die Informationen über die Dienstleistungen eines Unternehmens bereitstellt, reagiert nicht innerhalb von sieben Sekunden auf Benutzeranfragen (dies entspricht der allgemeinen, branchenüblichen Faustregel). Die Performanz des Systems kann unter bestimmten Last-bedingungen nicht erreicht werden.
- Ein System stürzt ab oder ist nicht in der Lage, auf Benutzereingaben zu reagieren, wenn es plötzlich einer großen Anzahl von Benutzeranfragen ausgesetzt wird (z.B. Ticketverkauf für eine große Sportveranstaltung). Die Kapazität des Systems, mit dieser Anzahl von Benutzern umzugehen, ist unzu-reichend.
- Die Systemreaktion wird erheblich beeinträchtigt, wenn Benutzer große Datenmengen abrufen möchten (z.B. wenn auf einer Website ein umfangreicher Bericht zum Download bereitgestellt wird). Die Kapazität des Systems zur Verarbeitung der generierten Datenmengen reicht nicht aus.
- Die Stapelverarbeitung (batch processing) kann nicht bis zu dem Zeitpunkt abgeschlossen werden, zu dem eine Onlineverarbeitung (z.B. morgens) zur Verfügung stehen soll. Die erforderliche Ausführungszeit der Stapelverarbeitung ist nicht ausreichend innerhalb des zulässigen Zeitraums.
- Ein Echtzeitsystem hat keinen freien Arbeitsspeicher (RAM)mehr, wenn parallele Prozesse sehr viel dynamischen Speicher erfordern, der aber nicht schnell genug freigegeben wird. Der Arbeitsspeicher ist nicht ausreichend dimensioniert oder RAM Anforderungen sind nicht richtig priorisiert.
- Eine Echtzeit-Systemkomponente A, die Eingaben an die Echtzeit-Systemkomponente B liefert, kann Aktualisierungen nicht ausreichend schnell berechnen. Das Gesamtsystem reagiert nicht rechtzeitig und kann fehlschlagen. Die Codemodule in Komponente A müssen analysiert und angepasst werden („Performanzprofilanalyse“), um sicherzustellen, dass die erforderlichen Aktualisierungsraten erreicht werden.

2. Grundlagen der Performanzmessung - 55 Min.

Schlüsselbegriffe

Messung, Metrik

Lernziele:

2.1 Typische, im Performanztest erfasste Metriken
PTFL-2.1.1 (K2) Die typischen Metriken verstehen, die in Performanztests erfasst werden

2.2 Aggregieren der Ergebnisse aus Performanztests
PTFL-2.2.1 (K2) Erklären können, weshalb die Ergebnisse aus Performanztests aggregiert werden

2.3 Wichtige Quellen von Performanzmetriken
PTFL-2.3.1 (K2) Die wichtigsten Quellen von Performanzmetriken verstehen

2.4 Typische Ergebnisse eines Performanztests
PTFL-2.4.1 (K1) Die typischen Ergebnisse eines Performanztests wiedergeben können

2.1 Typische, im Performanztest erfasste Metriken

2.1.1 Warum Performanzmetriken erforderlich sind

Genaue Messungen und die aus diesen Messungen abgeleiteten Metriken sind für die Festlegung der Ziele von Performanztests und für die Bewertung der Ergebnisse der Performanztests unerlässlich. Performanztests sollten nicht durchgeführt werden, ohne zu wissen, welche Messungen und Metriken benötigt werden. Die folgenden Projektrisiken können auftreten, wenn dieser Ratschlag nicht beachtet wird:

- Es ist nicht bekannt, ob das Leistungsniveau des Systems für das Erreichen der operativen Ziele akzeptabel ist
- Die Performanzanforderungen sind nicht in messbarer Form spezifiziert
- Es ist unter Umständen nicht möglich, Trends zu erkennen, die auf ein niedrigeres Leistungsniveau hindeuten
- Die tatsächlichen Ergebnisse eines Performanztests können nicht bewertet werden, indem sie mit einer Referenz (base line) von Performanzmessungen verglichen werden, die die akzeptable und/oder inakzeptable Performanz definieren

- Performanztestergebnisse werden auf der Grundlage der subjektiven Meinung einer oder mehrerer Personen bewertet
- Die Ergebnisse eines Performanztestwerkzeugs werden nicht verstanden

2.1.2 Sammeln von Performanzmessungen und -metriken

Wie bei jeder Art von Messung ist es auch hier möglich, Metriken präzise zu ermitteln und auszudrücken. Alle in diesem Kapitel beschriebenen Metriken und Messungen können und sollten so spezifiziert werden, dass sie in einem bestimmten Kontext aussagekräftig und sinnvoll sind. Es ist somit sinnvoll aus der Durchführung erster Tests zu lernen, welche Metriken weiter verfeinert und welche hinzugefügt werden müssen.

Eine Metrik ist zum Beispiel die Antwortzeit, die wahrscheinlich in jedem Satz von Performanzmetriken enthalten ist. Um jedoch aussagekräftig und umsetzbar zu sein, sollte die Metrik der Antwortzeit im Hinblick auf die Tageszeit, die Anzahl der gleichzeitigen Benutzer, die zu verarbeitende Datenmenge usw. genauer definiert sein.

Die in einem bestimmten Performanztest gesammelten Metriken variieren je nach -

- Geschäftskontext (Geschäftsprozesse, Kunden- und Benutzerverhalten und Erwartungen der Stakeholder)
- betrieblichem Kontext (Technologie und deren Verwendung)
- Testzielen

Zum Beispiel unterscheiden sich die für die Performanztests einer internationalen E-Commerce-Website ausgewählten Metriken von denen, die für die Performanztests eines eingebetteten Systems ausgewählt werden, das für die Steuerung der Funktionalität medizinischer Geräte verwendet wird.

Eine übliche Methode für die Kategorisierung von Performanzmessungen und Performanzmetriken besteht darin, die technische Umgebung, die Geschäfts-umgebung oder die Betriebsumgebung zu betrachten, in denen die Performanz bewertet werden soll.

Nachfolgend sind für die einzelnen Umgebungen Kategorien von Messungen und Metriken aufgeführt, die üblicherweise von Performanztests erfasst werden.

Technische Umgebung
Die Performanzmetriken unterscheiden sich je nach Art der technischen Umgebung, wie aus der folgenden Auflistung ersichtlich ist:

- Webbasierte Umgebungen
- Mobile Umgebungen
- Internet-of-Things (IoT)
- Desktopgeräte als Client

- Serverseitige Verarbeitung
- Großrechner (Mainframe)
- Datenbanken
- Netzwerke
- Die Art der Software, die in der Umgebung ausgeführt wird (z.B. eingebettet)

Zu den gesammelten Metriken gehören die folgenden:

- Antwortzeit (z.B. pro Transaktion, pro gleichzeitigem Benutzer, Seitenladezeiten)
- Ressourcennutzung (z.B. CPU, Speicher, Netzwerkbandbreite, Netzwerklatenz, verfügbarer Speicherplatz, I/O-Rate, freie und genutzte Threads)
- Durchsatzrate der Haupttransaktion (d.h. die Anzahl der Transaktionen, die in einem bestimmten Zeitraum verarbeitet werden können)
- Stapelverarbeitungszeit (z.B. Wartezeiten, Durchlaufzeiten, Antwortzeiten der Datenbank, Ausführungszeiten)
- Anzahl der Fehler, die sich auf die Performanz auswirken
- Ausführungszeit (z.B. zum Erstellen, Lesen, Aktualisieren und Löschen von Daten)
- Hintergrundlast für gemeinsam genutzte Ressourcen (insbesondere in virtualisierten Umgebungen)
- Softwaremetriken (z.B. Code-Komplexität)

Geschäftsumgebung

Aus geschäftlicher oder funktionaler Sicht können die gesammelten Metriken folgende Performanzmetriken beinhalten:

- Effizienz von Geschäftsprozessen (z.B. die Geschwindigkeit der Durchführung eines gesamten Geschäftsprozesses, einschließlich normaler, alternativer und außergewöhnlicher Pfade durch Anwendungsfälle)
- Durchsatz von Daten, Transaktionen und anderen Arbeitseinheiten (z.B. verarbeitete Aufträge pro Stunde, hinzugefügte Datenzeilen pro Minute)
- Einhaltungs-/Nichteinhaltungsquote für Service Level-Vereinbarungen (z.B. Nichteinhaltungen pro Zeiteinheit)
- Nutzungsumfang (z.B. Prozentsatz der globalen oder nationalen Benutzer, die zu einem bestimmten Zeitpunkt Aufgaben ausführen)
- Nebenläufige Nutzung (z.B. Anzahl der Benutzer, die gleichzeitig eine Aufgabe ausführen)
- Verwendungszeitpunkt (z.B. die Anzahl der Bestellungen, die während Spitzenlastzeiten verarbeitet werden)

Betriebsumgebung

Der betriebliche Aspekt der Performanztests konzentriert sich auf Aufgaben, die im Allgemeinen nicht als benutzerorientiert gelten. Dazu gehören die folgenden:

- Betriebsprozesse (z.B. benötigte Zeit für den Start der Umgebung, für Datenbackup, für das Herunterfahren und für die Wiederaufnahme)

- Systemwiederherstellung (z B. die Zeit, die zum Wiederherstellen von Daten aus einer Sicherungskopie erforderlich ist)
- Alarme und Warnungen (z.B. die Zeit, die das System für die Ausgabe eines Alarms oder Warnung benötigt)

2.1.3 Auswählen von Performanzmetriken

Es ist zu beachten, dass das Sammeln von mehr Metriken als erforderlich nicht unbedingt positiv ist. Jede Metrik, die ausgewählt wird, muss auch konsistent erfasst und berichtet werden. Es ist wichtig, eine Reihe passender und verfügbarer Metriken zu definieren, die die Ziele des Performanztests unterstützen.

Der “Goal-Question-Metric”-Ansatz (GQM) ist beispielsweise eine hilfreiche Methode, Metriken an den Performanzzielen auszurichten. Die Idee ist, zunächst die Ziele festzulegen und dann Fragen zu stellen, um herauszufinden, wann die Ziele erreicht sind. Den einzelnen Fragen werden Metriken zugeordnet, um sicherzustellen, dass die Antwort auf die Frage messbar ist. (siehe Kapitel 4.3 des Syllabus Expert Level – Improving the Testing Process [ISTQB_ELTM_ITP_SYL] für eine ausführlichere Beschreibung des GQM-Ansatzes). Es ist allerdings zu beachten, dass der GQM-Ansatz nicht immer in den Performanztestprozess passt. Einige Metriken stellen beispielsweise den Zustand eines Systems dar und sind nicht direkt mit Zielen verknüpft.

Es ist wichtig zu wissen, dass nach der Definition und Erfassung anfänglicher Messungen möglicherweise weitere Messungen und Metriken benötigt werden, um das tatsächliche Leistungsniveau zu verstehen und um zu bestimmen, wo Korrekturmaßnahmen erforderlich sind.

2.2 Aggregieren der Ergebnisse aus Performanztests

Der Zweck der Aggregation von Performanzmetriken besteht darin, sie so verstehen und ausdrücken zu können, dass das Gesamtbild der Systemleistung vermittelt wird. Wenn Performanzmetriken ausschließlich auf detaillierter Ebene betrachtet werden, kann es schwierig sein, die richtigen Schlussfolgerungen zu ziehen - insbesondere für die Stakeholder des Geschäftsbereichs.

Das Hauptanliegen vieler Stakeholder besteht darin, dass die Antwortzeit eines Systems, einer Webseite oder eines anderen Testobjekts innerhalb akzeptabler Grenzen liegt.

Sobald ein tieferes Verständnis der Performanzmetriken erreicht wurde, können die Metriken aggregiert werden, damit -

- Stakeholder des Geschäftsbereichs und des Projekts den Status der Systemleistung als Ganzes sehen,
- Leistungstrends identifiziert werden können,

- Performanzmetriken auf verständliche Weise berichtet werden können.

2.3 Wichtige Quellen von Performanzmetriken

Die Systemleistung sollte nur minimal durch den Aufwand für das Sammeln von Metriken beeinflusst werden (auch als "probe effect" bezeichnet). Darüber hinaus ist aufgrund der Menge, Genauigkeit und Geschwindigkeit, mit der Performanzmetriken erfasst werden müssen, die Verwendung eines Werkzeugs erforderlich. Die kombinierte Verwendung von Werkzeugen ist zwar nicht ungewöhnlich, kann jedoch Redundanzen bei der Verwendung der Testwerkzeuge und andere Probleme verursachen (siehe Kapitel 4.4).

Es gibt drei Hauptquellen für Performanzmetriken:

Performanztestwerkzeuge
Alle Performanztestwerkzeuge liefern Messungen und Metriken als Testergebnisse. Werkzeuge können sich jedoch unterscheiden in der Anzahl der angezeigten Metriken, in der Art und Weise, wie die Metriken angezeigt werden, und hinsichtlich der Möglichkeit, die Metriken benutzerdefiniert an bestimmte Situationen anzupassen (siehe auch Abschnitt 5.1).

Einige Werkzeuge erfassen und zeigen Performanzmetriken im Textformat an, während andere Werkzeuge Performanzmetriken grafisch in einem Dashboard-Format erfassen und anzeigen. Viele Werkzeuge bieten die Möglichkeit, die erfassten Metriken zu exportieren, um weitere Testauswertungen und die Berichterstellung zu unterstützen.

Performanztestmonitore
Testmonitore werden im Performanztest häufig eingesetzt, um die Berichtsfunktionen von Performanztestwerkzeugen zu ergänzen (siehe auch Kapitel 5.1). Darüber hinaus können die Testmonitore im operativen Betrieb eingesetzt werden, um die Systemleistung laufend zu überwachen und Systemadministratoren auf eine Leistungsminderung und Zunahme von Systemfehlern und Warnungen aufmerksam zu machen. Diese Werkzeuge können auch zur Erkennung und Benachrichtigung bei verdächtigem Verhalten (z.B. bei Sicherheitsangriffen wie Dienstblockaden bzw. DOS-Angriffen und verteilte DOS-Angriffen) verwendet werden.

Protokollanalysewerkzeuge
Es gibt Werkzeuge, die Serverprotokolle durchsuchen und Metriken aus diesen zusammenstellen. Einige dieser Werkzeuge sind in der Lage, aus den erfassten Metriken Diagramme zu erstellen, um die Daten grafisch darzustellen.

In Serverprotokollen werden normalerweise Fehler, Warnungen und Alarme aufgezeichnet, u.a. die Folgenden:

- Hohe Ressourcennutzung, z.B. hohe CPU-Auslastung, hoher Verbrauch an Festplattenkapazität oder eine unzureichende Bandbreite
- Speicherfehler und -warnungen, z.B. Speicherplatzmangel
- Deadlocks und Probleme in Zusammenhang mit Multithreading, insbesondere bei der Durchführung von Datenbankoperationen
- Datenbankfehler wie SQL-Ausnahmen und SQL-Zeitüberläufe

2.4 Typische Ergebnisse eines Performanztests

Bei funktionalen Tests, insbesondere bei der Überprüfung bestimmter funktionaler Anforderungen oder funktionaler Elemente von User-Storys, können die erwarteten Ergebnisse normalerweise klar definiert und die Testergebnisse eindeutig interpretiert werden, um festzustellen, ob der Test bestanden wurde oder nicht. Beispielsweise zeigt ein monatlicher Verkaufsbericht entweder eine korrekte oder eine falsche Gesamtsumme.

Während Tests, die die funktionale Eignung nachweisen, häufig von genau definierten Testorakeln profitieren, können Performanztests häufig nicht auf eine solche Informationsquelle zurückgreifen. Dies liegt einerseits daran, dass die Stakeholder in Bezug auf die Formulierung von Performanzanforderungen oft Schwierigkeiten haben und es vernachlässigen. Und es liegt andereseits daran, dass viele Business-Analysten und Product-Owner diese Art von Anforderungen gar nicht oder unvollständig ermitteln. Tester erhalten häufig nur begrenzt Hilfestellung, um die erwarteten Testergebnisse zu definieren.

Bei der Bewertung der Ergebnisse von Performanztests ist es wichtig, die Ergebnisse genau zu betrachten. Erste Rohergebnisse können irreführend sein, wenn Performanzfehler hinter scheinbar guten Gesamtergebnissen verborgen sind. Beispielsweise kann die Ressourcennutzung für alle wichtigen potenziellen Engpassressourcen deutlich unter 75% liegen, der Durchsatz oder die Antwortzeit von Haupttransaktionen oder Anwendungsfällen kann jedoch trotzdem etwas zu langsam sein.

Die zu bewertenden Ergebnisse variieren je nach den durchgeführten Tests und umfassen häufig die in Kapitel 2.1 beschriebenen Metriken.

3. Der Performanztest im Softwarelebenszyklus – 195 Min.

Schlüsselbegriffe

Metrik, Risiko, Softwareentwicklungslebenszyklus, Testprotokoll

Lernziele

3.1 Die Hauptaktivitäten von Performanztests
PTFL-3.1.1 (K2) Die Hauptaktivitäten von Performanztests verstehen

3.2 Kategorien von Performanzrisiken bei verschiedenen Architekturen
PTFL-3.2.1 (K2) Typische Kategorien von Performanzrisiken bei verschiedenen Architekturen erklären können

3.3 Performanzrisiken über den gesamten Softwareentwicklungslebenszyklus
PTFL-3.3.1 (K4) Die Performanzrisiken für ein bestimmtes Produkt über den gesamten Softwareentwicklungslebenszyklus analysieren können

3.4 Performanztestaktivitäten
PTFL-3.4.1 (K4) Ein bestimmtes Projekt analysieren können, um die geeigneten Performanztestaktivitäten für jede Phase des Softwareentwicklungslebenszyklus zu bestimmen

3.1 Die Hauptaktivitäten von Performanztests

Performanztests werden grundsätzlich iterativ durchgeführt. Jeder Test bietet wertvolle Einblicke in die Anwendungs- und Systemleistung. Die aus einem Test gesammelten Informationen werden zur Korrektur oder Optimierung von Anwendungs- und Systemparametern verwendet. In der nächsten Testiteration werden dann die Ergebnisse der Änderungen aufgezeigt usw., bis die Testziele erreicht sind.

Die Performanztestaktivitäten orientieren sich an den Phasen gemäß des ISTQB-Testprozesses [ISTQB_FL_SYL].

Testplanung
Testplanung ist bei Performanztests besonders wichtig, da Testumgebungen, Testdaten, Testwerkzeuge und Personalressourcen festgelegt werden müssen. Darüber hinaus ist dies die Testaktivität, in der der Umfang der Performanztests festgelegt wird.

Während der Testplanung werden die Aktivitäten zur Risikoidentifizierung und Risikoanalyse durchgeführt, und die relevanten Informationen werden in allen Testplanungsdokumentationen (z.B. Testkonzept, Stufentestkonzept) aktualisiert. Genau wie die Testplanung bei Bedarf überarbeitet und geändert wird, werden auch die Risiken, die Risikostufen und der jeweilige Risikostatus aktualisiert, um Änderungen in den Risikobedingungen zu berücksichtigen.

Testüberwachung und -steuerung
Es werden Steuerungsmaßnahmen definiert, um Aktionspläne bereitzustellen, falls Probleme auftreten, die sich auf die Performanz auswirken könnten, wie z.B.

- Erhöhung der Lastgenerierungskapazität, wenn die Infrastruktur nicht die Lasten erzeugt, die für bestimmte Performanztests vorgesehen sind
- Einsatz veränderter, neuer oder ausgetauschter Hardware
- Änderungen an Netzwerkkomponenten
- Änderungen an der Software-Implementierung

Die Ziele des Performanztests werden bewertet, um das Erreichen der Endekriterien zu überprüfen.

Testanalyse
Effektive Performanztests basieren auf einer Analyse der Performanzanforderungen, Testziele, Service Level-Vereinbarungen, IT-Architektur, Prozessmodelle und sonstigen Objekten, die Teil der Testbasis sind. Diese Aktivität kann durch die Modellierung und Analyse von Systemressourcenanforderungen und / oder -verhalten mithilfe von Tabellenkalkulationsprogrammen oder Kapazitätsplanungswerkzeugen unterstützt werden.

Es werden spezifische Testbedingungen wie Laststufen, zeitliche Bedingungen und zu testende Transaktionen ermittelt. Auf dieser Grundlage wird über die erforderliche Art des/der Performanztests (z.B. Lasttest, Stresstest, Skalierbarkeitstest) entschieden.

Testentwurf
Die Performanztestfälle werden entworfen. Diese werden im Allgemeinen modular erstellt, sodass sie als Bausteine für größere, komplexere Performanztests verwendet werden können (siehe Kapitel 4.2).

Testrealisierung
In der Realisierungsphase werden Performanztestfälle in Testabläufe geordnet. Die Testabläufe der Performanztests sollten die normalerweise vom Benutzer unternommenen Schritte und andere funktionale Aktivitäten widerspiegeln, die während der Performanztests zu testen sind.

Eine Testrealisierungsaktivität ist, die Testumgebung vor jeder Testausführung einzurichten und/oder zurückzusetzen. Da Performanztests normalerweise datengetrieben sind, ist ein Prozess erforderlich, um Testdaten zu ermitteln, die für die tatsächlichen

Produktionsdaten hinsichtlich Menge und Typ repräsentativ sind, so dass eine Nutzung vergleichbar der Produktion simuliert werden kann.

Testdurchführung
Die Testdurchführung findet statt, wenn der Performanztest ausgeführt wird, häufig unter Verwendung von Performanztestwerkzeugen. Die Testergebnisse werden bewertet, um festzustellen, ob die Systemleistung den Anforderungen und anderen festgelegten Zielen entspricht. Aufgetretene Fehler werden berichtet.

Testabschluss
Die Ergebnisse der Performanztests werden den Stakeholdern (z.B. Systemarchitekten, Managern, Product-Owner) in einem Testabschlussbericht zur Verfügung gestellt. Die Ergebnisse werden durch Metriken ausgedrückt, die häufig aggregiert werden, um die Inhalte der Testergebnisse zu vereinfachen. Visuelle Berichtsmittel wie Dashboards werden häufig verwendet, um Performanztest-ergebnisse auf eine Art und Weise darzustellen, die leichter verständlich ist als textbasierte Metriken.

Performanztests werden häufig als eine fortlaufende Aktivität betrachtet, da sie zu mehreren Zeitpunkten und auf allen Teststufen (Komponenten-, Komponentenintegrations-, System-, Systemintegrations- und Abnahmetest) durchgeführt werden. Am Ende eines definierten Zeitraums der Performanztests kann der Zeitpunkt für den Testabschluss erreicht werden, um die entworfenen Tests, Skripte bzw. Daten für Testwerkzeuge (Testfälle und Testabläufe), Testdaten und andere Testmittel zu archivieren oder an andere Tester zur späteren Verwendung bei der Systemwartung weiterzugegeben.

3.2 Kategorien von Performanzrisiken bei verschiedenen Architekturen

Wie bereits erwähnt, variiert die Anwendungs- oder Systemleistung je nach Architektur, Anwendungs- und Hostumgebung erheblich. Es ist nicht möglich, eine vollständige Liste der Performanzrisiken für alle Systeme bereitzustellen; die folgende Liste enthält jedoch einige typische Arten von Risiken, die mit bestimmten Architekturen verbunden sind:

Einzel-Computersysteme
Hierbei handelt es sich um Systeme oder Anwendungen, die vollständig auf einem nicht virtualisierten Computer ausgeführt werden. Die Leistung kann beeinträchtigt werden durch

- übermäßigen Ressourcenverbrauch einschließlich Speicherlecks, Hintergrund-aktivitäten wie Sicherheitssoftware, langsame Speichersubsysteme (z.B. langsame externe Geräte oder Festplattenfragmentierung) und fehlerhaftes Management des Betriebssystems.

- ineffiziente Implementierung von Algorithmen, die verfügbare Ressourcen (z.B. Hauptspeicher) nicht nutzen und daher langsamer als erforderlich laufen.

Mehrschichtige (Multi-Tier) Systeme
Hierbei handelt es sich um Systeme von Systemen, die auf mehreren Servern ausgeführt werden, von denen jeder eine bestimmte Gruppe von Aufgaben ausführt, z.B. Datenbankserver, Anwendungsserver und Präsentationsserver. Jeder Server ist natürlich ein Computer und unterliegt den zuvor erwähnten Risiken. Darüber hinaus kann die Leistung durch schlechtes oder nicht skalierbares Datenbankdesign, Netzwerkengpässe und unzureichende Bandbreite oder Kapazität auf einem der einzelnen Server beeinträchtigt werden.

Verteilte Systeme
Dies sind Systeme von Systemen, ähnlich wie bei einer mehrschichtigen Architektur, nur dass sich die verschiedenen Server dynamisch ändern können, beispielsweise ein E-Commerce-System, das je nach geografischem Standort der bestellenden Person auf verschiedene Bestandsdatenbanken zugreift. Zusätzlich zu den Risiken in Zusammenhang mit mehrschichtigen Systemen können bei dieser Architektur auch Performanzprobleme aufgrund kritischer Abläufe oder Datenflüsse zu, von oder durch unzuverlässige oder unvorhersagbare Remote-Server Leistungen vorkommen, insbesondere wenn bei solchen Servern periodische Verbindungsprobleme oder Zeiten intensiver Last auftreten.

Virtualisierte Systeme
Hierbei handelt es sich um Systeme, auf denen die physische Hardware mehrere virtuelle Computer hostet. Diese virtuellen Maschinen können Einzelcomputersysteme und -anwendungen sowie Server enthalten, die Teil einer mehrschichtigen oder verteilten Architektur sind. Zu den Performanzrisiken, die sich speziell aus der Virtualisierung ergeben, gehören eine übermäßige Belastung der Hardware verteilt auf alle virtuellen Maschinen oder eine fehlerhafte Konfiguration des virtuellen Host-Rechners, was dann zu unzureichenden Ressourcen führt.

Dynamische/Cloud-basierte Systeme
Dies sind Systeme, die die Möglichkeit bieten, bei Bedarf zu skalieren, und die Kapazität mit zunehmender Last zu erhöhen. Bei diesen Systemen handelt es sich in der Regel um verteilte und virtualisierte mehrschichtige Systeme, die jedoch mit Funktionen zur Selbstskalierung ausgestattet sind, die speziell darauf ausgelegt sind, einige der mit diesen Architekturen verbundenen Performanzrisiken zu mindern. Es bestehen jedoch Risiken, wenn diese Funktionen während der Ersteinrichtung oder bei späteren Aktualisierungen nicht korrekt konfiguriert werden.

Client-Server-Systeme
Hierbei handelt es sich um Systeme, die auf einem Client ausgeführt werden und über eine Benutzungsschnittstelle mit einem einzelnen Server, einem mehrschichtigen Server oder einem verteilten Server kommunizieren. Da auch auf dem Client Code

ausgeführt wird, betreffen sowohl die Risiken der Einzel-Computersysteme diesen Code, als auch die oben genannten serverseitigen Probleme.

Des Weiteren bestehen Performanzrisiken aufgrund von Problemen mit der Verbindungsgeschwindigkeit und -zuverlässigkeit, Netzwerküberlastungen am Verbindungspunkt des Clients (z.B. öffentliches WLAN) und potenziellen Problemen in Zusammenhang mit Firewalls, Paketprüfung und Lastausgleich zwischen Servern.

Mobile Anwendungen
Dies sind Anwendungen, die auf einem Smartphone, Tablet oder einem anderen mobilen Gerät ausgeführt werden. Solche Anwendungen unterliegen den genannten Risiken für Client-Server- und browserbasierte Anwendungen (Web-Apps). Außerdem können Performanzprobleme aufgrund von begrenzten und variablen Ressourcen und der verfügbaren Konnektivität auf dem mobilen Gerät auftreten (die durch Standort, Akkulaufzeit, Ladezustand, verfügbaren Speicherplatz auf dem Gerät und Temperatur beeinflusst werden können). Bei Anwendungen, die Gerätesensoren oder Funkgeräte wie Beschleunigungssensoren oder Bluetooth verwenden, können langsame Datenflüsse aus diesen Quellen Probleme verursachen. Schließlich haben mobile Anwendungen häufig starke Interaktionen mit anderen lokalen mobilen Apps und Remote-Webdiensten, von denen jeder zu einem Engpass bei der Performanz führen kann.

Eingebettete Echtzeitsysteme
Dies sind Systeme, die in Alltagsgegenständen wie Autos (z.B. Unterhaltungssysteme und intelligente Bremssysteme), Aufzügen, Verkehrssignalen, Heizungs-, Lüftungs- und Klimasystemen usw. arbeiten oder diese steuern. Diese Systeme beinhalten häufig viele der Risiken mobiler Geräte, einschließlich (und in zunehmendem Maße) Konnektivitätsprobleme, da sie mit dem Internet verbunden sind. Die verminderte Performanz eines mobilen Videospiels ist zwar normalerweise kein Sicherheitsrisiko für den Benutzer, während die Verlangsamung der Performanz eines Fahrzeug-bremssystems katastrophal sein kann.

Mainframe-Anwendungen
Dies sind – nicht selten Jahrzehnte alte - Anwendungen, die häufig unternehmens-kritische Geschäftsfunktionen in einem Rechenzentrum unterstützen, manchmal auch über die Stapelverarbeitung. Die meisten sind recht vorhersehbar und schnell, wenn sie so verwendet werden, wie sie ursprünglich entwickelt wurden. Viele davon sind heute allerdings über APIs, Webdienste oder über die zugehörige Datenbank zugänglich, was zu unerwarteten Lasten führen kann, die den Durchsatz der bestehenden Anwendungen beeinträchtigen.

Es ist zu beachten, dass eine bestimmte Anwendung oder ein System zwei oder mehr der oben aufgeführten Architekturen enthalten kann. Dies bedeutet, dass alle relevanten Risiken für diese Anwendung oder dieses System zutreffen. Angesichts des Internets der Dinge und der explosionsartigen Zunahme mobiler Anwendungen -

beides Bereiche, in denen ein extrem hohes Maß an Interaktion und Konnektivität die Regel ist - ist es möglich, dass in einer Anwendung alle Architekturen in irgendeiner Form vorhanden sind und somit auch alle Risiken zutreffen können.

Während die Architektur eindeutig eine wichtige technische Entscheidung ist, die tiefgreifende Auswirkungen auf die Performanzrisiken hat, werden diese Risiken auch durch andere technische Entscheidungen beeinflusst und geschaffen. Beispielsweise treten Speicherlecks häufiger bei Sprachen auf, die eine direkte Heap-Speicherverwaltung ermöglichen, wie beispielsweise C und C++, und Performanzprobleme sind bei relationalen und nichtrelationalen Datenbanken unterschiedlich. Solche technischen Entscheidungen reichen bis hin zum Entwurf einzelner Funktionen oder Methoden (z.B. die Wahl eines rekursiven anstatt eines iterativen Algorithmus). Abhängig von den Rollen und Verantwortlichkeiten der Tester innerhalb des Unternehmens und des Softwareentwicklungslebenszyklus kann es variieren, ob der Tester von solchen Entscheidungen Kenntnis erlangt oder die Fähigkeit bekommt, diese gar zu beeinflussen.

3.3 Performanzrisiken über den gesamten Softwareentwicklungslebenszyklus

Das Verfahren zum Analysieren von Risiken für die Qualität eines Softwareprodukts im Allgemeinen wird in verschiedenen ISTQB-Lehrplänen behandelt (siehe z.B. [ISTQB_FL_SYL] und [ISTQB_ALTM_SYL]). Inhalte zu spezifischen Risiken und Überlegungen bezüglich bestimmter Qualitätsmerkmale (z.B. in Bezug auf die Gebrauchstauglichkeit) sind in [ISTQB_UT_SYL]) enthalten; und die Risiken aus geschäftlicher oder technischer Sicht in [ISTQB_ALTA_SYL] bzw. [ISTQB_ALTTA_SYL]. In diesem Kapitel stehen die performanzbezogenen Risiken für die Produktqualität im Mittelpunkt, einschließlich der Art und Weise, wie sich der Prozess, die Teilnehmer und die Überlegungen verändern.

Der Prozess für performanzbezogene Risiken für die Produktqualität ist wie folgt:

1. Identifikation von Risiken für die Produktqualität mit Fokussierung auf Merkmale wie Zeitverhalten, Ressourcennutzung und Kapazität.
2. Bewerten der identifizierten Risiken und sicherstellen, dass die relevanten Architekturkategorien (siehe Kapitel 3.2) hierbei berücksichtigt werden. Festlegung der Risikostufe für jedes ermittelte Risiko nach Eintrittswahrscheinlichkeit und Schadensausmaß anhand klar definierter Kriterien.
3. Ergreifen geeigneter Maßnahmen zur Risikobeherrschung für jedes ermittelte Risiko, basierend auf der Art des Risikos und der Risikostufe.
4. Fortlaufende Steuerung der Risiken, um sicherzustellen, dass die Risiken vor der Freigabe angemessen gemindert werden.

Wie bei der Analyse von Qualitätsrisiken im Allgemeinen sollten an diesem Prozess sowohl Stakeholder des Geschäfts- als auch des technischen Bereichs mitwirken. Für die performanzbezogene Risikoanalyse müssen bei den Stakeholdern des Geschäftsbereichs diejenigen einbezogen sein, die ein besonderes Bewusstsein dafür haben, wie sich Performanzprobleme in der Produktion tatsächlich auf Kunden, Benutzer, das Geschäft und andere nachgeordnete Stakeholder auswirken. Die Stakeholder des Geschäftsbereichs müssen sich darüber im Klaren sein, dass der beabsichtigte Gebrauch, die Geschäfts-, Gesellschafts- oder Sicherheitskritikalität, potenzielle finanzielle Einbußen und/oder Reputationsschäden, die zivil- oder strafrechtliche Haftung und ähnliche Faktoren das Risiko aus geschäftlicher Sicht beeinflussen. Das kann zu Risiken führen und zur Beeinflussung der Auswirkungen von Ausfällen.

Darüber hinaus müssen bei den technischen Stakeholdern diejenigen einbezogen werden, die ein tiefes Verständnis dafür haben, welche Auswirkungen bestimmte relevante Anforderungen sowie Architektur-, Entwurfs- und Realisierungsentscheidungen auf die Performanz haben. Technische Stakeholder müssen sich bewusst sein, dass Architektur-, Entwurfs- und Realisierungsentscheidungen die Performanzrisiken aus technischer Sicht beeinflussen, somit Risiken erzeugen und Einfluss auf die Wahrscheinlichkeit von Fehlern haben.

Der Risikoanalyseprozess, der ausgewählt wird, sollte über ein angemessenes Maß an Formalität und Gründlichkeit verfügen. Für performanzbezogene Risiken ist es besonders wichtig, dass der Risikoanalyseprozess frühzeitig startet und regelmäßig wiederholt durchgeführt wird. Mit anderen Worten, der Tester sollte sich nicht vollständig auf Performanztests verlassen, die erst gegen Ende der System- und Systemintegrationsteststufen durchgeführt werden. Bei vielen Projekten, insbesondere bei Projekten für größere und komplexe Systeme von Systemen, kam es aufgrund der späten Entdeckung von Performanzfehlern, die sich aus Anforderungen, Entwurfs-, Architektur- und Realisierungsentscheidungen ergaben, zu unliebsamen Überraschungen. Der Schwerpunkt sollte daher auf einem iterativen Ansatz für die Identifikation, Bewertung, Reduzierung und Steuerung von Performanzrisiken während des gesamten Softwareentwicklungszyklus liegen.

Wenn beispielsweise große Datenmengen über eine relationale Datenbank verarbeitet werden, zeigt sich die langsame Performanz von „many-to-many“ Verknüpfungen aufgrund eines schlechten Datenbankentwurfs möglicherweise nur bei dynamischen Tests mit umfangreichen Testdatensätzen, wie sie im Systemtest verwendet werden. Durch ein sorgfältiges technisches Review, bei dem auch erfahrene Datenbank-spezialisten mitwirken, lassen sich derartige Probleme schon vor der Datenbank-implementierung vorhersagen. Nach einem solchen Review werden in einem iterativen Ansatz Risiken identifiziert und erneut bewertet.

Darüber hinaus müssen die Risikobeherrschung und das Risikomanagement den gesamten Softwareentwicklungsprozess umfassen und beeinflussen, nicht nur das dynamische Testen. Wenn zum Beispiel kritische, performanzbezogene Entscheidungen wie die erwartete Anzahl von Transaktionen oder gleichzeitiger Benutzer nicht früh im Projekt festgelegt werden können, ist es wichtig, dass Entwurfs- und Architekturentscheidungen eine sehr variable Skalierbarkeit ermöglichen (z.B. Ressourcen durch Cloud-Computing auf Abruf). Dies ermöglicht frühzeitige Entscheidungen zur Risikobeherrschung.

Gutes Performanz-Engineering kann Projektteams dabei helfen, die späte Aufdeckung kritischer Performanzdefekte in höheren Teststufen zu vermeiden, beispielsweise in den Systemintegrationstests oder Benutzerabnahmetests. Performanzfehler, die zu einem späten Zeitpunkt des Projekts festgestellt werden, können äußerst kostspielig sein und möglicherweise sogar zum Abbruch ganzer Projekte führen.

Wie alle Qualitätsrisiken können auch die performanzbezogenen Risiken niemals vollständig vermieden werden, d.h. es besteht immer ein gewisses Risiko eines performanzbezogenen Produktionsausfalls. Daher muss der Risikomanagementprozess eine realistische und spezifische Bewertung des verbleibenden Risikos für die am Prozess beteiligten Stakeholder des Geschäfts- und technischen Bereichs beinhalten. So ist beispielsweise der Hinweis, dass es immer noch möglich ist, dass Kunden beim Check-out lange Verzögerungen haben werden, wenig hilfreich, denn dieser Hinweis gibt keinerlei Aufschluss darüber, wieviel Risikobeherrschung erreicht ist oder wie hoch das noch verbleibende Risiko ist. Stakeholder können den Status besser verstehen, wenn der Prozentsatz der Kunden angegeben wird, bei denen wahrscheinlich Verzögerungen auftreten werden, wenn bestimmte Schwellenwerte erreicht oder überschritten werden.

3.4 Performanztestaktivitäten

Die Performanztestaktivitäten werden je nach Art des verwendeten Softwareentwicklungslebenszyklus unterschiedlich organisiert und durchgeführt.

Sequentielle Entwicklungsmodelle
Die ideale Praxis beim Performanztest in sequentiellen Entwicklungsmodellen besteht darin, die Performanzkriterien als Teil der zu Beginn eines Projekts definierten Abnahmekriterien zu behandeln. Entsprechend der Lebenszyklusperspektive des Testens sollten die Performanztestaktivitäten während des gesamten Softwareentwicklungszyklus durchgeführt werden. Im Verlauf des Projekts sollte jede nachfolgende Performanztestaktivität auf Elementen basieren, die in den vorherigen Aktivitäten definiert wurden (siehe unten):

- Konzept - Sicherstellen, dass die Performanzziele als Abnahmekriterien für das Projekt definiert sind.

- Anforderungen – Sicherstellen, dass die Performanzanforderungen definiert sind und die Bedürfnisse der Stakeholder korrekt wiedergeben.
- Analyse und Entwurf - Sicherstellen, dass der Systementwurf die Performanz-anforderungen widerspiegelt.
- Kodierung/Realisierung – Sicherstellen, dass der Programmcode effizient ist und die Anforderungen und den Entwurf in Bezug auf die Performanz widerspiegelt.
- Komponententest – Durchführung von Performanztests auf Komponenten-teststufe.
- Komponentenintegrationstest – Durchführung von Performanztests auf Komponentenintegrationsteststufe.
- Systemtest – Performanztests auf Systemteststufe durchführen, einschließlich von für die Produktionsumgebung repräsentativer Hardware, Software, Verfahren und Daten. Systemschnittstellen können simuliert werden, vorausgesetzt, sie bilden die reale Performanz repräsentativ nach.
- Systemintegrationstest – Durchführung von Performanztests mit dem gesamten System, das für die Produktionsumgebung repräsentativ ist.
- Abnahmetests – Validieren, dass die Performanz des Systems die ursprünglich spezifizierten Benutzerbedürfnisse und Abnahmekriterien erfüllt.

Iterative und inkrementelle Entwicklungsmodelle
Bei diesen Entwicklungsmodellen (z.B. bei der agilen Softwareentwicklung) werden Performanztests auch als iterative und inkrementelle Testaktivität betrachtet (siehe [ISTQB_FL_AT]). Performanztests können als Teil der ersten Iteration oder als eigener Sprint für den Performanztest durchgeführt werden. Bei diesen Lebenszyklusmodellen kann die Durchführung von Performanztests jedoch von einem separaten Team durchgeführt werden, das für Performanztests zuständig ist.

Die kontinuierliche Integration wird im Allgemeinen bei iterativen und inkrementellen Softwareentwicklungszyklen durchgeführt, was eine hochautomatisierte Ausführung von Tests erfordert. Die häufigste Zielsetzung der kontinuierlichen Integration besteht darin, Regressionstests durchzuführen und sicherzustellen, dass jeder Build stabil ist. Performanztests können Teil der automatisierten Tests sein, die in Zusammenhang mit kontinuierlicher Integration durchgeführt werden, vorausgesetzt die Tests sind so ausgelegt, dass sie auf Build-Ebene ausgeführt werden können. Anders als bei funktionalen automatisierten Tests sind jedoch einige zusätzliche Faktoren zu bedenken:

- Einrichtung der Performanztestumgebung - Häufig ist eine Testumgebung erforderlich, die bei Bedarf verfügbar ist, beispielsweise eine Cloud-basierte Testumgebung für den Performanztest.
- Entscheiden, welche Performanztests in der kontinuierlichen Integration automatisiert werden sollen – Aufgrund des kurzen Zeitrahmens, der für kontinuierliche Integrationstests zur Verfügung steht, können die für die kontinuierliche Integration ausgewählten Performanztests eine Teilmenge

umfangreicherer Performanztests sein, die von einem Spezialistenteam zu einem anderen Zeitpunkt während einer Iteration durchgeführt werden.
- Erstellen von Performanztests für die kontinuierliche Integration – Das Hauptziel von Performanztests im Rahmen der kontinuierlichen Integration ist es sicherzustellen, dass sich Änderungen nicht negativ auf die Performanz auswirken. Abhängig von den für einen bestimmten Build vorgenommenen Änderungen sind möglicherweise neue Performanztests erforderlich.
- Ausführen von Performanztests für Teile einer Anwendung oder eines Systems – Dies erfordert häufig, dass die Werkzeuge und Testumgebungen in der Lage sind, schnelle Performanztests durchzuführen, und hierfür Teilmengen geeigneter Tests auszuwählen.

Performanztests in iterativen und inkrementellen Softwareentwicklungslebenszyklen können auch ihre eigenen Lebenszyklusaktivitäten haben:
- Releaseplanung – Bei dieser Aktivität werden Performanztests aus der Sicht aller Iterationen in einem Release betrachtet, von der ersten bis zur letzten Iteration. Performanzrisiken werden identifiziert, bewertet und Maßnahmen zur Risikobeherrschung geplant. Dies beinhaltet häufig die Planung aller abschließenden Performanztests vor Freigabe der Anwendung.
- Iterationsplanung – In Zusammenhang mit jeder Iteration können Performanztests innerhalb der Iteration und nach Abschluss der Iteration durchgeführt werden. Die Performanzrisiken werden für die einzelnen User-Storys detaillierter bewertet.
- Erstellung von User-Storys – User-Storys bilden häufig die Grundlage für die Performanzanforderungen in der agilen Entwicklung, wobei die spezifischen Performanzkriterien in den jeweiligen Abnahmekriterien beschrieben werden. Diese werden als "nicht-funktionale" User-Storys bezeichnet.
- Entwurf der Performanztests – Performanzanforderungen und -kriterien, die in bestimmten User-Storys beschrieben sind, werden für den Entwurf von Tests verwendet (siehe Kapitel 4.2)
- Kodierung/Realisierung – Während der Kodierung können Performanztests in der Komponententeststufe durchgeführt werden. Ein Beispiel dafür ist die Feinabstimmung von Algorithmen für eine optimale Performanz.
- Testen/Evaluieren – Während das Testen normalerweise sehr nahe an den Entwicklungsaktivitäten durchgeführt wird, können Performanztests als eine separate Aktivität durchgeführt werden, je nach Umfang und Zielen der Performanztests während der Iteration. Wenn beispielsweise das Ziel des Performanztests darin besteht, die Performanz der Iteration als Gruppe abgeschlossener User-Storys zu testen, dann ist dafür ein erweiterter Umfang von Performanztests erforderlich als für den Performanztest einer einzelnen User-Story. Dies kann als eine eigene Iteration speziell für den Performanztest eingeplant werden.
- Bereitstellung – Mit der Bereitstellung der Anwendung in der Produktionsumgebung muss die Performanz überwacht werden, um zu bestimmen, ob die

Anwendung bei der tatsächlichen Verwendung die gewünschte Performanz erreicht.

Kommerzielle Standardsoftware (COTS) und sonstige Lieferanten-/Kaufmodelle
Viele Organisationen entwickeln Anwendungen und Systeme nicht selbst, sondern erwerben Software von Anbietern oder aus Open-Source Projekten. Bei solchen Softwarebeschaffungsmodellen ist die Performanz ein wichtiger Aspekt, der sowohl aus der Perspektive des Anbieters (Lieferant bzw. Entwickler) als auch der des Erwerbers (bzw. des Kunden) getestet werden muss.

Unabhängig vom Beschaffungsmodell liegt es häufig in der Verantwortung des Kunden zu überprüfen, ob die Performanz der Anwendung den Anforderungen entspricht. Im Falle kundenspezifischer Software, die vom Anbieter entwickelt wird, sind die Performanzanforderungen und die entsprechenden Abnahmekriterien im Vertrag zwischen dem Verkäufer und dem Kunden spezifiziert. Im Falle kommerzieller Standardsoftware ist der Kunde allein dafür verantwortlich, vor der Bereitstellung des Produkts die Performanz in einer realistischen Testumgebung zu testen.

4. Performanztestaufgaben – 475 Min.

Schlüsselbegriffe

Bedenkzeit, Lastgenerierung, Lastprofil, Nutzungsprofil, Nebenläufigkeit, Last herunterfahren, Last hochfahren, System von Systemen, Systemdurchsatz, Testkonzept, virtueller Benutzer

Lernziele

4.1 Planung

PTFL-4.1.1 (K4) Performanztestziele aus relevanten Informationen ableiten können
PTFL-4.1.2 (K4) Ein Performanztestkonzept skizzieren können, in dem die Performanzziele für ein bestimmtes Projekt berücksichtigt sind
PTFL-4.1.3 (K4) Eine Präsentation erstellen können, die es verschiedenen Stakeholdern ermöglicht, die Begründung für die geplanten Performanztests zu verstehen

4.2 Analyse, Entwurf und Realisierung

PTFL-4.2.1 (K2) Beispiele für typische Protokolle nennen können, die bei Performanztests vorkommen
PTFL-4.2.2 (K2) Das Konzept von Transaktionen beim Performanztest verstehen
PTFL-4.2.3 (K4) Nutzungsprofile für die Systemnutzung analysieren können
PTFL-4.2.4 (K4) Lastprofile erstellen können, die aus den Nutzungsprofilen zu bestimmten Performanzzielen abgeleitet sind
PTFL-4.2.5 (K4) Durchsatz und Parallelität bei der Entwicklung von Performanztests analysieren können
PTFL-4.2.6 (K2) Die grundlegende Struktur eines Performanztestskripts verstehen
PTFL-4.2.7 (K3) Performanztestskripte implementieren können, die dem Testkonzept und den Lastprofilen entsprechen
PTFL-4.2.8 (K2) Die Aktivitäten verstehen, die zur Vorbereitung der Ausführung von Performanztests erforderlich sind

4.3 Durchführung

PTFL-4.3.1 (K2) Die Hauptaktivitäten beim Durchführen von Performanztestskripten verstehen

4.4 Analyse der Ergebnisse und Berichterstattung

PTFL-4.4.1 (K4) Die Ergebnisse und Auswirkungen von Performanztests analysieren und berichten können

4.1 Planung

4.1.1 Performanztestziele ableiten

Stakeholder können Benutzer oder Personen aus Geschäftsbereichen oder mit technischem Hintergrund sein. Sie können unterschiedliche Ziele in Bezug auf Performanztests haben. Die Stakeholder legen die Ziele, die zu verwendende Terminologie und die Kriterien fest, nach denen am Ende entschieden wird, ob das Ziel erreicht wurde.

Die Ziele für Performanztests beziehen sich auf diese unterschiedlichen Arten von Stakeholdern. Es empfiehlt sich, zwischen benutzerbasierten und technischen Zielen zu unterscheiden. Benutzerbasierte Ziele konzentrieren sich in erster Linie auf die Zufriedenheit der Endbenutzer und auf die Geschäftsziele. Im Allgemeinen geht es Benutzern weniger um unterschiedliche Funktionen oder darum, wie ein Produkt bereitgestellt wird. Sie wollen einfach nur in der Lage sein ihre erforderlichen Aufgaben zu erledigen.

Die technischen Ziele konzentrieren sich hingegen auf betriebliche Aspekte und geben Antworten auf Fragen der Skalierbarkeit eines Systems oder unter welchen Bedingungen eingeschränkte Performanz offensichtlich werden könnte.

Zu den wichtigsten Zielen von Performanztests gehören das Identifizieren potenzieller Risiken, das Finden von Verbesserungsmöglichkeiten und das Ermitteln notwendiger Änderungen.

Beim Sammeln von Informationen von den verschiedenen Stakeholdern sollten die folgenden Fragen beantwortet werden:

- Welche Transaktionen werden im Performanztest ausgeführt und welche durchschnittliche Antwortzeit wird erwartet?
- Welche Systemmetriken müssen erfasst werden (z. B. Speichernutzung, Netzwerkdurchsatz) und welche Werte werden erwartet?
- Welche Performanzverbesserungen werden von diesen Tests im Vergleich zu früheren Testzyklen erwartet?

4.1.2 Das Performanztestkonzept

Das Performanztestkonzept ist ein Dokument, das vor dem Beginn der Performanztests erstellt wird. Auf das Performanztestkonzept sollte im Testkonzept (siehe [ISTQB_FL_SYL]) verwiesen werden, welches auch relevante Informationen über die Terminplanung enthält. Das Performanztestkonzept wird nach Beginn der Performanztests fortlaufend aktualisiert.

In einem Performanztestkonzept sollten die folgenden Informationen enthalten sein:

Zielsetzung
Die im Performanztestkonzept beschriebene Zielsetzung umfasst Ziele, Strategien und Methoden für den Performanztest. Sie ermöglicht eine quantifizierbare Antwort auf die zentrale Frage der Angemessenheit und der Einsatzbereitschaft des Systems, unter Last leistungsgerecht zu funktionieren.

Testziele
Die allgemeinen Testziele hinsichtlich der Performanz, die das System unter Test (SUT) erzielen soll, sind für die einzelnen Stakeholdergruppen aufgelistet (siehe Kapitel 4.1.1).

Systemübersicht
Eine kurze Beschreibung des SUT liefert den Kontext für die Messung der Performanz-testparameter. Die Übersicht sollte eine grundsätzliche Beschreibung der Funktionalität enthalten, die unter Last getestet wird.

Vorgesehene Performanztestarten
Die verschiedenen Arten der vorgesehenen Performanztests werden zusammen mit einer Beschreibung des jeweiligen Zwecks aufgelistet (Arten von Performanztests, siehe Kapitel 1.2).

Abnahmekriterien
Performanztests sollen die Reaktionsfähigkeit, den Durchsatz, die Zuverlässigkeit und/oder die Skalierbarkeit des Systems unter einer bestimmten Arbeitslast ermitteln. Im Allgemeinen ist die Antwortzeit ein Anliegen der Benutzer, der Durchsatz ist ein Geschäftsanliegen und die Ressourcennutzung ist ein technisches Anliegen. Die Abnahmekriterien sollten für alle relevanten Maßnahmen festgelegt und gegebenenfalls mit Bezug auf die folgenden Punkte angegeben werden:

- Allgemeine Performanztestziele
- Service Level-Vereinbarungen
- Referenzwert - Die Referenzwerte sind eine Reihe von Metriken, die zum Vergleich von aktuellen und zuvor erzielten Performanzmessungen verwendet werden. Dadurch können bestimmte Performanzverbesserungen nachgewiesen und/ oder das Erreichen von Testabnahmekriterien bestätigt werden. Möglicherweise ist es erforderlich, die Referenzwerte im Vorfeld ggf. mit bereinigten Daten aus einer Datenbank zu erstellen.

Testdaten
Die Testdaten umfassen eine Vielzahl von Daten, die für einen Performanztest spezifiziert werden müssen. Diese Daten können Folgendes umfassen:

- Benutzerkonton (z.B. Benutzerkonten, die für die gleichzeitige Anmeldung verfügbar sind)
- Benutzereingabedaten (z.B. die Daten, die ein Benutzer in die Anwendung eingeben würde, um einen Geschäftsprozess auszuführen)

- Datenbank (z.B. die vorab befüllte Datenbank, die mit Daten zur Verwendung beim Testen gefüllt ist)

Bei der Erstellung der Testdaten sollten die folgenden Aspekte berücksichtigt werden:

- Datenextraktion aus Produktionsdaten
- Importieren von Daten in das SUT
- Erstellung neuer Daten
- Erstellung von Backups, die zur Wiederherstellung der Daten verwendet werden können, wenn neue Testzyklen durchgeführt werden
- Datenmaskierung oder -anonymisierung. Diese Vorgehensweise wird bei Produktionsdaten angewendet, die personenbezogene Daten enthalten, und ist gemäß den Allgemeinen Datenschutzverordnung (EU-DSGVO) verpflichtend vorgeschrieben. Beim Performanztest erhöht die Datenmaskierung jedoch das Risiko für die Performanztests, da die Daten möglicherweise nicht die gleichen Datenmerkmale wie in der Praxis aufweisen.

Systemkonfiguration

Das Kapitel des Performanztestkonzepts zur Systemkonfiguration enthält die folgenden technischen Informationen:

- Eine Beschreibung der spezifischen Systemarchitektur, einschließlich der Server (z.B. Web-, Datenbank-, Lastausgleichsserver).
- Definition mehrerer Schichten
- Spezifische Informationen zur Computerhardware (z.B. CPU-Kerne, RAM, Solid State Disks (SSD), Festplattenlaufwerke (HDD)), einschließlich der jeweiligen Versionen
- Spezifische Informationen zur Software (z.B. Anwendungen, Betriebssysteme, Datenbanken, Support-Services des Unternehmens) einschließlich der jeweiligen Versionen
- Externe Systeme, die mit dem SUT arbeiten, sowie deren Konfiguration und Version (z.B. eCommerce-Systeme mit Integration in NetSuite)
- Build / Versionskennung des SUT

Testumgebung

Die Testumgebung ist häufig eine separate Umgebung, die die Produktion nachahmt, jedoch in einem kleineren Maßstab. Dieses Kapitel des Performanztestkonzepts sollte beschreiben, wie die Ergebnisse der Performanztests hochgerechnet werden, um auf die größere Produktionsumgebung angewendet zu werden. Bei einigen Systemen ist die Produktionsumgebung die einzig mögliche Option für das Testen. In diesem Fall müssen jedoch die spezifischen Risiken, die mit dieser Art des Testens verbunden sind, erörtert werden.

Testwerkzeuge befinden sich manchmal auch außerhalb der eigentlichen Testumgebung und erfordern möglicherweise spezielle Zugriffsrechte, um mit den Systemkomponenten zu interagieren. Dies ist in Zusammenhang mit der Testumgebung und -konfiguration zu berücksichtigen.

Performanztests können auch mit einem Teil des Systems durchgeführt werden, falls der ohne die anderen Komponenten lauffähig ist. Dies ist häufig günstiger als das Testen mit dem gesamten System und kann bereits durchgeführt werden, sobald die Komponente entwickelt ist.

Testwerkzeuge
In diesem Kapitel des Performanztestkonzepts wird beschrieben, welche Testwerkzeuge (und -versionen) bei der Skripterstellung, Ausführung und Überwachung der Performanztests verwendet werden (siehe Kapitel 5). Diese Liste enthält normalerweise:

- Werkzeug(e) zum Simulieren von Benutzertransaktionen
- Werkzeuge zur Bereitstellung von Last von mehreren Knotenpunkten innerhalb der Systemarchitektur (Points of Presence)
- Werkzeuge zur Überwachung der Systemleistung, einschließlich der oben unter Systemkonfiguration beschriebenen Werkzeuge

Profile
Nutzungsprofile ermöglichen einen wiederholbaren, schrittweisen Ablauf durch die Anwendung für eine bestimmte Verwendung des Systems. Das Aggregieren dieser Nutzungsprofile führt zu einem Lastprofil (im Allgemeinen als Szenario bezeichnet). Für weitere Informationen zu Profilen, siehe Kapitel 4.2.3.

Relevante Metriken
Bei der Ausführung von Performanztests kann eine große Anzahl von Messungen und Metriken erfasst werden (siehe Kapitel 2). Zu viele Messungen können jedoch die Analyse erschweren und die tatsächliche Leistung der Anwendung negativ beeinflussen. Aus diesen Gründen ist es wichtig, die Messungen und Metriken zu ermitteln, die für die Erreichung der Ziele des Performanztests am relevantesten sind.

Die folgende Tabelle, die in Kapitel 4.4 ausführlicher erläutert wird, zeigt einen typischen Satz von Metriken für Performanztests und deren Überwachung. Für diese Metriken sollten Testziele bezüglich der Performanz festgelegt werden, sofern dies für das Projekt erforderlich ist:

Performanzmetriken	
Art	**Metrik**
Status des virtuellen Benutzers	Bestanden Nicht bestanden

Performanzmetriken	
Art	**Metrik**
Transaktionsantwortzeit	Minimum Maximum Durchschnitt 90% Perzentil
Transaktionen pro Sekunde	Bestanden/Sekunde Nicht bestanden/Sekunde Gesamtzahl/Sekunde
Treffer (z.B. Anfragen an eine Datenbank oder an einen Webserver)	Treffer/Sekunde ▪ Minimum ▪ Maximum ▪ Durchschnitt ▪ Gesamtzahl
Durchsatz	Bits/Sekunde (BPS) ▪ Minimum ▪ Maximum ▪ Durchschnitt ▪ Gesamt
HTTP-Antworten pro Sekunde	Antworten/Sekunde ▪ Minimum ▪ Maximum ▪ Durchschnitt ▪ Gesamt Antwort durch HTTP-Antwortcodes

Performanzüberwachung	
Art	**Metrik**
CPU-Nutzung	% der verfügbaren CPU genutzt
Speichernutzung	% des verfügbaren Haupt-speichers genutzt

Risiken
Risiken können Bereiche betreffen, die nicht als Teil des Performanztests durch Messungen erfasst werden, sowie Einschränkungen beim Performanztest unterliegen (z.B. externe Schnittstellen, die nicht simuliert werden können, unzureichende Last, keine Möglichkeit, Server zu überwachen). Einschränkungen der Testumgebung

können auch zu Risiken führen (z.B. unzureichende Daten, verkleinerte Umgebung). Weitere Risikoarten sind in den Kapiteln 3.2 und 3.3 aufgeführt.

4.1.3 Kommunikation über den Performanztest

Tester müssen mit allen Beteiligten über die Gründe für den gewählten Ansatz des Performanztests und die durchzuführenden Aktivitäten, die im Performanztestkonzept beschrieben sind, kommunizieren können. Die Themen, über die kommuniziert wird, können zwischen den einzelnen Stakeholdern erheblich variieren, je nachdem, ob für diese die „geschäftlichen/benutzerbezogenen Interessen“ oder eher die „technologie-/betriebsbezogenen Interessen“ im Vordergrund stehen.

Stakeholder mit Geschäftsfokussierung
Die folgenden Faktoren sollten bei der Kommunikation mit Stakeholdern mit geschäftlichem Fokus berücksichtigt werden:

- Stakeholder mit geschäftlichem Fokus sind weniger an der Unterscheidung zwischen funktionalen und nicht funktionalen Qualitätsmerkmalen interessiert.
- Technische Fragen zu Werkzeugen, Skripterstellung und Lastgenerierung sind im Allgemeinen von untergeordnetem Interesse.
- Der Zusammenhang zwischen Produktrisiken und Performanztestzielen muss klar kommuniziert werden.
- Die Stakeholder müssen auf das Verhältnis zwischen den Kosten der geplanten Performanztests und der Aussagequalität der Performanztestergebnisse im Vergleich zur Performanz unter Produktionsbedingungen hingewiesen werden.
- Über die Wiederholbarkeit der geplanten Performanztests muss informiert werden. Ist der Test schwer zu wiederholen oder kann er mit minimalem Aufwand wiederholt werden?
- Projektrisiken müssen kommuniziert werden. Dazu gehören Einschränkungen und Abhängigkeiten bezüglich der Einrichtung der Tests, Infrastruktur-anforderungen (z.B. Hardware, Werkzeuge, Daten, Bandbreite, Testumgebung, Ressourcen) und Abhängigkeiten von Schlüsselpersonal.
- Die übergeordneten Aktivitäten müssen kommuniziert werden (siehe Kapitel 4.2 und 4.3), zusammen mit einem umfassenden Plan mit Kosten, Zeitplan und Meilensteinen.

Stakeholder mit Technologiefokussierung
Die folgenden Faktoren sollten bei der Kommunikation mit Stakeholdern mit Technologiefokus berücksichtigt werden:

- Der geplante Ansatz zur Erstellung der erforderlichen Lastprofile muss erläutert und die erwartete Einbindung der technischen Stakeholder geklärt werden.
- Die einzelnen Schritte beim Einrichten und Ausführen der Performanztests müssen detailliert erläutert und die Relation der Tests zu den architektur-bezogenen Risiken aufgezeigt werden.
- Es müssen die Schritte kommuniziert werden, die erforderlich sind, um die Performanztests wiederholbar zu machen. Dazu können organisatorische

Aspekte gehören (z B. Mitwirkung von Schlüsselpersonal) sowie technische Fragen.
- Wenn Testumgebungen gemeinsam genutzt werden, müssen die geplanten Termine für die Performanztests mitgeteilt werden, um sicherzustellen, dass die Testergebnisse dieser Tests nicht beeinträchtigt werden.
- Falls Performanztests in der Produktionsumgebung ausgeführt werden sollen, müssen potenzielle Auswirkungen auf die Benutzer und Maßnahmen zur Verringerung dieser kommuniziert und akzeptiert werden.
- Technische Stakeholder müssen sich über ihre Aufgaben und den geplanten Zeitpunkt im Klaren sein.

4.2 Analyse, Entwurf und Realisierung

4.2.1 Typische Kommunikationsprotokolle

Kommunikationsprotokolle definieren eine Reihe von Kommunikationsregeln zwischen Computern und Systemen. Um Tests zielgerichtet für bestimmte Teile des Systems zu entwerfen, ist das Verstehen der Protokolle erforderlich.

Kommunikationsprotokolle werden häufig auf Basis der Schichten des OSI-Modells (Open Systems Interconnection Model) beschrieben (siehe ISO/IEC 7498-1), obwohl einige Protokolle außerhalb dieses Modells liegen können. Für Performanztests werden am häufigsten die Protokolle von Schicht 5 (Sitzungsschicht) bis Schicht 7 (Anwendungsschicht) verwendet. Übliche Protokolle umfassen:
- Datenbank – ODBC-, JDBC-, sonstige herstellerspezifische Protokolle
- Web – HTTP-, HTTPS-, HTML-Protokolle
- Webdienste – SOAP-, REST-Protokolle

Im Allgemeinen bezieht sich die Ebene der OSI-Schicht, auf die sich die Performanztests konzentrieren, auf die Schicht der getesteten Architektur. Wenn beispielsweise eine eingebettete Low-Level-Architektur getestet wird, dann werden die Schichten des OSI-Modells mit niedrigerer Nummerierung besonders im Fokus stehen.

Zusätzliche Protokolle, die bei Performanztests verwendet werden, sind u.a.:
- Netzwerkprotokolle – DNS, FTP, IMAP, LDAP, POP3, SMTP, Windows Sockets, CORBA
- Mobile Protokolle – TruClient, SMP, MMS
- Protokolle für den Fernzugriff – Citrix ICA, RTE
- SOA-Protokolle – MQSeries, JSON, WSCL

Es ist wichtig, die Gesamtsystemarchitektur zu verstehen, da Performanztests auf einer einzelnen Systemkomponente (z.B. Webserver, Datenbankserver) oder auf einem gesamten System mit End-to-End-Tests ausgeführt werden können. Herkömmliche zweischichtige Anwendungen, die mit einem Client-Server-Modell erstellt

wurden, geben den "Client" als die GUI und primäre Benutzungsschnittstelle und den "Server" als Backend-Datenbank an. Diese Anwendungen erfordern die Verwendung eines Protokolls wie ODBC, um auf die Datenbank zuzugreifen. Mit der Entwicklung von webbasierten Anwendungen und mehrschichtigen Architekturen sind viele Server an der Verarbeitung von Informationen beteiligt, die letztendlich für den Browser des Benutzers bereitgestellt werden.

Je nachdem, auf welchen Teil des Systems die Tests fokussiert sind, ist ein Verständnis des entsprechenden Protokolls erforderlich. Wenn zum Beispiel End-to-End-Tests durchgeführt werden sollen, bei denen die Benutzeraktivität im Browser emuliert wird, kann ein Webprotokoll wie HTTP bzw. HTTPS verwendet werden. Auf diese Weise lässt sich die Interaktion mit der GUI umgehen, und die Tests können sich ganz auf die Kommunikation und Aktivitäten der Backend-Server konzentrieren.

4.2.2 Transaktionen

Transaktionen beschreiben die Menge von Aktivitäten, die ein System von der Initiierung bis zum Abschluss eines oder mehrerer Prozesse (Anfragen, Operationen oder Betriebsprozesse) ausführt. Die Antwortzeit von Transaktionen kann zur Bewertung der Systemleistung gemessen werden. Innerhalb eines Performanztests werden diese Messungen verwendet, um Komponenten zu identifizieren, die korrigiert oder optimiert werden müssen.

Simulierte Transaktionen können eine Bedenkzeit einschließen, um den Zeitpunkt, wann ein realer Benutzer eine Aktion ausführt (z.B. Drücken der Taste SENDEN), besser widerzuspiegeln. Die Transaktionsantwortzeit plus der Bedenkzeit entspricht der für diese Transaktion verbrauchten Zeit.

Die Transaktionsantwortzeiten, die während des Performanztests erfasst werden, zeigen, ob und wie sich diese Messungen bei unterschiedlichen Systemlasten ändern. Die Analyse zeigt möglicherweise keine Verschlechterung unter steigender Last, während andere Messungen eine starke Verschlechterung zeigen können. Durch das Hochfahren der Last und Messen der zugrunde liegenden Transaktionszeiten ist es möglich, die Ursache der Verschlechterung mit den Antwortzeiten einer oder mehrerer Transaktionen in Beziehung zu setzen.

Transaktionen können auch geschachtelt werden, sodass einzelne und aggregierte Aktivitäten gemessen werden können. Dies kann beispielsweise hilfreich sein, um die Performanz eines Online-Bestellsystems zu verstehen. Der Tester kann die einzelnen Schritte im Bestellprozess messen (z.B. Artikel suchen, Artikel in den Warenkorb legen, Artikel bezahlen, Bestellung bestätigen) sowie den Bestellprozess insgesamt. Durch Verschachteln von Transaktionen kann beides in einem Test erfasst werden.

4.2.3 Nutzungsprofile identifizieren

Nutzungsprofile definieren unterschiedliche Interaktionsmuster mit einer Anwendung, z.B. von Benutzern oder anderen Systemkomponenten. Für eine bestimmte Anwendung können mehrere Nutzungsprofile spezifiziert werden. Sie können kombiniert werden, um ein gewünschtes Lastprofil zum Erreichen bestimmter Performanztestziele zu erstellen (siehe Kapitel 4.2.4).

Im nachfolgenden Kapitel werden die grundlegenden Schritte zum Identifizieren von Nutzungsprofilen beschrieben:

1. Identifizieren der zu erfassenden Daten
2. Sammeln der Daten aus einer oder mehreren Quellen
3. Auswerten der Daten zur Erstellung der Nutzungsprofile

Identifizieren der Daten

Wenn Benutzer mit dem zu testenden System interagieren, werden die folgenden Daten erfasst oder geschätzt, um ihre Nutzungsprofile zu modellieren (d.h. wie sie mit dem System interagieren):

- Verschiedene Arten von Benutzer-Personas und ihre Rollen (z.B. Standardbenutzer, registriertes Mitglied, Administrator, Benutzergruppen mit bestimmten Berechtigungen).
- Verschiedene generische Aufgaben, die von diesen Benutzern/Rollen ausgeführt werden (z.B. Durchsuchen einer Webseite nach Informationen, Durchsuchen einer Webseite nach einem bestimmten Produkt, Durchführen rollenspezifischer Aktivitäten). Es wird empfohlen diese Aufgaben am besten auf einer hohen Abstraktionsebene zu modellieren (z.B. auf Ebene der Geschäftsprozesse oder der wichtigsten User-Storys).
- Geschätzte Anzahl von Benutzern für die jeweiligen Rollen/Aufgaben pro Zeiteinheit für einen bestimmten Zeitraum. Diese Informationen sind auch für das spätere Erstellen von Lastprofilen hilfreich (siehe Kapitel 4.2.4).

Sammeln von Daten

Die oben genannten Daten können aus verschiedenen Quellen gesammelt werden:

- Durchführung von Interviews oder Workshops mit Stakeholdern wie Product-Ownern, Vertriebsmanagern und (potenziellen) Endbenutzern. Die geführten Diskussionen zeigen oft die wichtigsten Nutzungsprofile der Benutzer auf und geben Antworten auf die grundlegende Frage „Für wen ist diese Anwendung gedacht?”.
- Funktionale Spezifikationen und Anforderungen (sofern verfügbar) sind eine wertvolle Informationsquelle über beabsichtigte Nutzungsmuster, die auch dazu beitragen können, Nutzertypen und deren Nutzungsprofile zu identifizieren. Wenn funktionale Spezifikationen als User-Storys vorliegen, ermöglicht das Standardformat eine direkte Identifizierung der Benutzertypen (d.h. „als <Benutzertyp/Rolle/Persona> möchte ich <Funktion/Fähigkeit>, um <Nutzen>).

In ähnlicher Weise identifizieren UML-Anwendungsfalldiagramme und -beschreibungen den "Akteur" für den Anwendungsfall.

- Die Auswertung von Nutzungsdaten und Metriken aus ähnlichen Anwendungen kann die Identifizierung von Benutzertypen unterstützen und einige erste Hinweise auf die erwartete Anzahl von Benutzern liefern. Der Zugang zu automatisch erfassten Überwachungsdaten (z.B. über ein Webverwaltungswerkzeug) wird empfohlen. Dies umfasst zum Beispiel die Überwachungsprotokolle (Logs) und Nutzungsdaten des aktuellen, im Betrieb befindlichen Systems, das aktualisiert werden soll.
- Die Überwachung des Verhaltens von Benutzern bei der Ausführung vordefinierter Aufgaben mit der Anwendung kann einen Einblick in die Arten von Nutzungsprofilen geben, die für die Performanztests modelliert werden sollten. Es wird empfohlen, diese Aufgabe ggf. mit den geplanten Gebrauchstauglichkeitstests zu koordinieren (insbesondere dann, wenn ein Gebrauchstauglichkeitstest-Labor verfügbar ist).

Erstellen von Nutzungsprofilen

Für die Identifizierung und Erstellung von Nutzungsprofilen für Benutzer werden die folgenden Schritte durchgeführt:

- Es wird vom Generellen zum Speziellen vorgegangen (Top-Down-Ansatz). Zunächst werden relativ einfache, allgemein gehaltene Nutzungsprofile erstellt und nur dann weiter untergliedert, wenn dies zur Erreichung der Performanztestziele erforderlich ist (siehe Kapitel 4.1.1).
- Bestimmte Nutzungsprofile können als relevant für die Performanztests ausgewählt werden, wenn es sich um Aufgaben handelt, die häufig ausgeführt werden, oder die kritische (mit einem hohen Risiko verbundene) oder häufige Transaktionen zwischen verschiedenen Systemkomponenten erfordern, oder für die die Übertragung großer Datenmengen notwendig sein kann.
- Die Nutzungsprofile werden mit den wichtigsten Stakeholdern überprüft und verfeinert, bevor sie für die Erstellung von Lastprofilen verwendet werden (siehe Kapitel 4.2.4).

Das zu testende System ist nicht immer nur Lasten ausgesetzt, die vom Benutzer ausgehen. Nutzungsprofile können auch für den Performanztest der folgenden Systemarten benötigt werden (es ist zu beachten, dass die nachfolgende Auflistung nicht vollständig ist):

Offline-Stapelverarbeitungssysteme (batch processing)

Der Fokus liegt hierbei hauptsächlich auf dem Durchsatz des Stapelverarbeitungssystems (siehe Kapitel 4.2.5) und dessen Fähigkeit, die Verarbeitung innerhalb eines bestimmten Zeitraums abzuschließen. Die Nutzungsprofile konzentrieren sich auf die Arten der Verarbeitung, die von den Batch-Prozessen gefordert werden. Beispielsweise können die Nutzungsprofile für ein Aktienhandelssystem (zu dem normalerweise Online- und Batch-basierte Transaktionsverarbeitung gehört) solche umfassen, die sich auf Zahlungstransaktionen, Verifizierung von Anmeldedaten oder

auf die Überprüfung der Einhaltung der rechtlichen Bedingungen für bestimmte Arten von Aktiengeschäften beziehen. Jedes dieser Nutzungsprofile würde zu unterschiedlichen Pfaden durch den gesamten Batch-Prozess für eine Aktie führen. Die oben beschriebenen Schritte zur Identifizierung der Nutzungsprofile benutzerbasierter Online-Systeme können auch in Zusammenhang mit der Stapelverarbeitung angewendet werden.

Systeme von Systemen
Komponenten in einer Umgebung mit mehreren Systemen (die auch eingebettet sein können) reagieren auf unterschiedliche Eingaben von anderen Systemen oder Komponenten. Je nach der Art des zu testenden Systems kann dies die Modellierung mehrerer unterschiedlicher Nutzungsprofile erfordern, um die von diesen vorgelagerten Systemen bereitgestellten unterschiedlichen Eingaben effektiv darzustellen. Dies kann eine detaillierte Analyse (z.B. von Puffern und Warteschlangen) beinhalten, die zusammen mit den Systemarchitekten durchgeführt wird und auf den System- und Schnittstellenspezifikationen basiert.

4.2.4 Lastprofile erstellen

Ein Lastprofil spezifiziert die Arbeitslast, die eine zu testende Komponente oder ein zu testendes System in der Produktion erfahren könnte. Ein Lastprofil besteht aus einer bestimmten Anzahl von Instanzen, die die Aktionen vordefinierter Nutzungsprofile über einen bestimmten Zeitraum ausführen. Wenn es sich bei den Instanzen um Benutzer handelt, wird üblicherweise der Begriff „virtuelle Benutzer“ verwendet.

Die wichtigsten Informationen, die zur Erstellung eines realistischen und wiederholbaren Lastprofils erforderlich sind, beinhalten:

- Das Performanztestziel (z.B. Bewertung des Systemverhaltens unter Stresslasten)
- Nutzungsprofile , die individuelle Nutzungsmuster genau darstellen (siehe Kapitel 4.2.3)
- Bekannte Probleme mit Durchsatz und Nebenläufigkeit (siehe Kapitel 4.2.5)
- Die Mengen- und Zeitverteilung, mit der die Nutzungsprofile so ausgeführt werden sollen, dass das SUT die gewünschte Last erfährt. Typische Beispiele sind:
    - Ramp-up: Stetig zunehmende Last (z.B. einen virtuellen Benutzer pro Minute hinzufügen)
    - Ramp-down: Stetig abnehmende Last
    - Stufen: Augenblickliche Änderungen der Last (z.B. alle fünf Minuten 100 virtuelle Benutzer hinzufügen)
    - Vordefinierte Verteilungen (z.B. nachgeahmtes Volumen täglicher oder saisonaler Geschäftszyklen)

Das folgende Beispiel zeigt den Aufbau eines Lastprofils mit dem Ziel, für das zu testende System Stresszustände (an der oder über der erwarteten Maximallastgrenze des Systems) zu erzeugen.

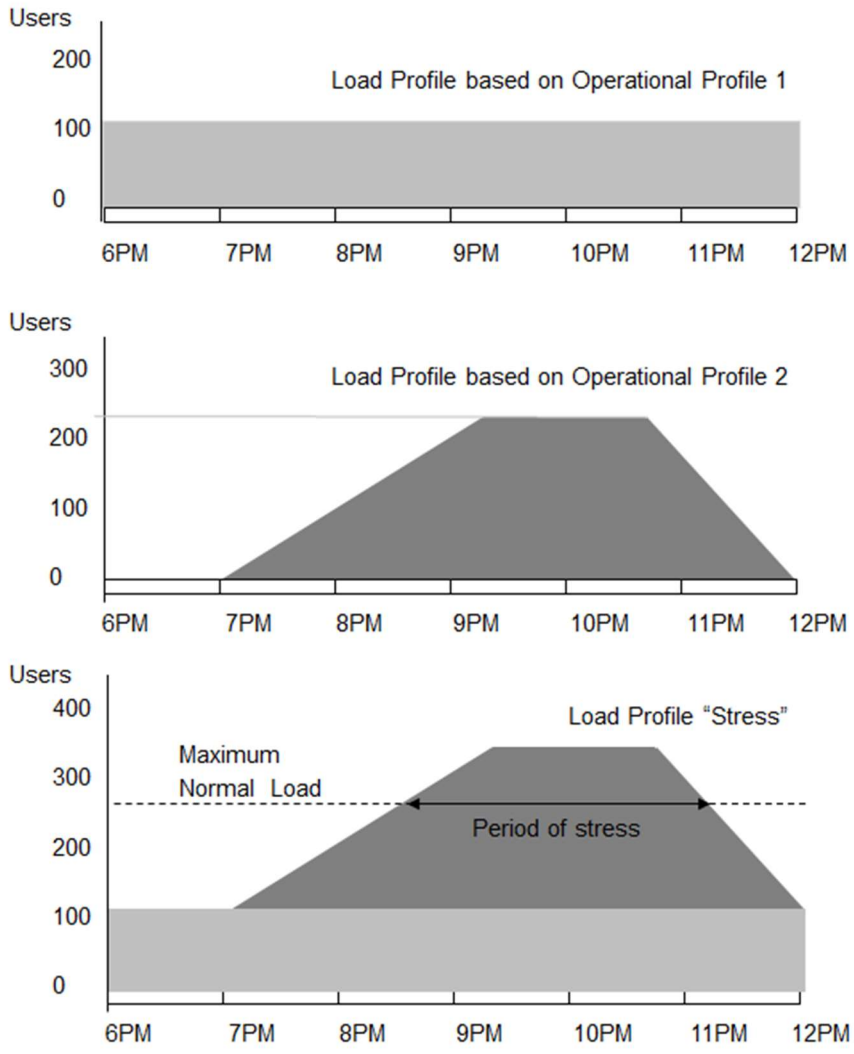

Diagramm 1: Beispiel für den Aufbau eines Stresslastprofils

Oben im Diagramm ist ein Lastprofil dargestellt, das aus einer Stufeneingabe von 100 virtuellen Benutzern besteht. Diese Benutzer führen die durch das Nutzungsprofil 1 definierten Aktivitäten während der gesamten Testdauer aus. Dies ist typisch für viele Performanzlastprofile, die eine Hintergrundlast darstellen.

Das mittlere Diagramm zeigt ein Lastprofil, das aus einem Hochfahren der Last auf bis zu 220 virtuellen Benutzern besteht; diese Last wird zwei Stunden beibehalten und dann heruntergefahren. Jeder virtuelle Benutzer führt die in Nutzungsprofil 2 definierten Aktivitäten aus.

Das untere Diagramm zeigt das Lastprofil, das sich aus der Kombination der beiden oben beschriebenen Lastprofile ergibt. Das getestete System wird drei Stunden lang einer Stressbelastung ausgesetzt. Weitere Beispiele, siehe [Bath14].

4.2.5 Durchsatz und Nebenläufigkeit analysieren

Es ist wichtig, die verschiedenen Aspekte der Arbeitslast zu verstehen: Durchsatz und Nebenläufigkeit. Um die Nutzungs- und Lastprofile richtig zu modellieren, müssen beide Aspekte berücksichtigt werden.

Systemdurchsatz
Der Systemdurchsatz ist ein Maß für die Anzahl der Transaktionen eines bestimmten Typs, die das System in einer Zeiteinheit verarbeitet; zum Beispiel die Anzahl der Bestellungen pro Stunde oder die Anzahl der HTTP-Anfragen pro Sekunde. Der Systemdurchsatz sollte vom Netzwerkdurchsatz unterschieden werden, der die Datenmenge, die über das Netzwerk übertragen wird, betrifft (Kapitel 2.1).

Der Systemdurchsatz definiert die Belastbarkeit des Systems. Leider wird häufig die Anzahl der gleichzeitigen Benutzer verwendet, um die Belastung für interaktive Systeme zu definieren, und nicht der Durchsatz. Der Ansatz ist jedoch nur zum Teil korrekt, auch wenn diese Anzahl oft einfacher zu finden ist und auch Lasttestwerkzeuge oft die Last so definieren. Ohne das Definieren von Nutzungsprofilen, d.h. was jeder Benutzer macht und wie intensiv (was auch dem Durchsatz für einen Benutzer entspricht), ist die Anzahl der Benutzer kein gutes Maß für die Last. Wenn beispielsweise 500 Benutzer jede Minute kurze Abfragen ausführen, ergibt dies eine Last und Durchsatz von 30.000 Abfragen pro Stunde. Wenn dieselben 500 Benutzer dieselben Abfragen nur einmal pro Stunde ausführen, beträgt die Last und der Durchsatz 500 Abfragen pro Stunde. Dies ergibt bei der gleichen Zahl von 500 Benutzern eine 60 mal höhere Last und einen mindestens 60fachen Unterschied hinsichtlich der Hardwareanforderungen für das System.

Die Modellierung der Arbeitslast erfolgt normalerweise unter Berücksichtigung der Anzahl virtueller Benutzer (Ausführungs-Threads) und der Bedenkzeit (Verzögerungen zwischen Benutzeraktionen). Der Systemdurchsatz wird jedoch auch durch die Verarbeitungszeit definiert, und diese Zeit kann sich mit zunehmender Last erhöhen.

Systemdurchsatz = [Anzahl virtueller Benutzer] / ([Verarbeitungszeit] + [Bedenkzeit])

Wenn sich also die Verarbeitungszeit erhöht, kann sich der Durchsatz erheblich verringern, selbst wenn alles andere gleichbleibt.

Der Systemdurchsatz ist ein wichtiger Aspekt beim Testen von Stapelverarbeitungssystemen. In diesem Fall wird der Durchsatz normalerweise anhand der Anzahl

Transaktionen gemessen, die innerhalb eines gegebenen Zeitrahmens (z.B. eines nächtlichen Stapelverarbeitungsfensters) ausgeführt werden können.

Nebenläufigkeit
Nebenläufigkeit ist ein Maß für die Anzahl der gleichzeitigen / parallelen Ausführungs-Threads. Bei interaktiven Systemen kann es eine Anzahl gleichzeitiger / paralleler Benutzer sein. Nebenläufigkeit wird in Lasttestwerkzeugen normalerweise modelliert, indem die Anzahl der virtuellen Benutzer festgelegt wird.

Nebenläufigkeit ist ein wichtiges Maß und gibt die Anzahl der parallelen Sitzungen an, von denen jede ihre eigenen Ressourcen verwenden kann. Selbst wenn der Durchsatz gleich ist, kann die Menge der verwendeten Ressourcen je nach Nebenläufigkeit variieren. Typische Testaufbauten sind geschlossene Systeme (aus Sicht der Warteschlangentheorie), bei denen die Anzahl der Benutzer im System festgelegt ist (feste Population). Wenn alle Benutzer in einem geschlossenen System auf die Antwort des Systems warten, können keine neuen Benutzer hinzukommen. Bei vielen öffentlichen Systemen handelt es sich um offene Systeme - neue Benutzer kommen ständig hinzu, auch wenn alle aktuellen Benutzer auf die Antwort des Systems warten.

4.2.6 Grundstruktur von Performanztestskripten

Ein Performanztestskript sollte eine Benutzer- oder Komponentenaktivität simulieren, die zur Last des getesteten Systems beiträgt (dies kann das gesamte System oder eine seiner Komponenten betreffen). Es initiiert Anfragen an den Server in der richtigen Reihenfolge und in einem bestimmten Tempo.

Die beste Methode zum Erstellen von Performanztestskripten hängt von der für die Lastgenerierung verwendeten Methode ab (Kapitel 4.1).

- Traditionell wird die Kommunikation zwischen dem Client und dem System oder der Komponente auf Protokollebene aufgezeichnet und dann wiedergegeben, nachdem das Skript parametriert und dokumentiert wurde. Die Parametrisierung führt zu einem skalierbaren und wartbaren Skript; allerdings kann die Parametrisierung zeitaufwändig sein.
- Die Aufzeichnung auf GUI-Ebene beinhaltet normalerweise das Erfassen von GUI-Aktionen eines einzelnen Clients mit einem Testausführungswerkzeug und das Ausführen dieses Skripts mit einem Lastgenerierungswerkzeug, um mehrere Clients darzustellen.
- Eine Programmierung kann auf Basis von Protokollanfragen (z.B. HTTP-Anfragen), GUI-Aktionen oder API-Aufrufen erfolgen. Im Falle von Programmierskripten muss die genaue Reihenfolge der Anfragen und Antworten festgelegt werden, die an das reale System gesendet bzw. von diesem zurückgesendet werden. Das kann möglicherweise eine komplexe Aufgabe sein.

Normalerweise handelt es sich bei einem Skript um einen oder mehrere Codeabschnitte (in einer generischen Programmiersprache mit einigen Erweiterungen oder in einer speziellen Sprache geschrieben) oder um ein Objekt, das einem Benutzer vom Werkzeug in einer GUI angezeigt wird. In beiden Fällen enthält das Skript Serveranfragen, die Last erzeugen (z.B. HTTP-Anfragen), und etwas Programmierlogik, die festlegt, wie genau diese Anfragen aufgerufen werden sollen (z.B. in welcher Reihenfolge, zu welchem Zeitpunkt, mit welchen Parametern und was geprüft werden soll). Je ausgefeilter die Logik ist, desto wichtiger wird es, leistungsfähige Programmiersprachen zu verwenden.

Allgemeine Struktur
Häufig enthält das Skript einen Initialisierungsabschnitt (in dem alles für den Hauptteil vorbereitet wird), sowie Hauptabschnitte, die eventuell mehrmals ausgeführt werden, und einen Bereinigungsabschnitt (in dem die notwendigen Schritte unternommen werden, um den Test ordnungsgemäß abzuschließen).

Datensammlung
Um Antwortzeiten zu sammeln, sollten dem Skript Zeitmesser (timer) hinzugefügt werden, um zu messen, wie lange eine Anfrage oder eine Kombination von Anfragen dauert. Die Anfragen, für die die Zeitdauer gemessen wird, sollten einer sinnvollen logischen Arbeitseinheit entsprechen, z.B. einer Geschäftstransaktion, um einen Artikel zu einer Bestellung hinzuzufügen oder eine Bestellung zu senden.

Es ist wichtig zu verstehen, was genau gemessen wird: Bei Protokollen auf Protokollebene ist dies nur die Antwortzeit des Servers und des Netzwerks, während GUI-Skripte die Gesamtzeit (End-to-End) messen (wobei die genaue Messung von der verwendeten Technologie abhängt).

Verifizierung der Ergebnisse und Fehlerbehandlung
Ein wichtiger Teil des Skripts sind Ergebnisprüfung und Fehlerbehandlung. Selbst bei den besten Lasttestwerkzeugen ist die Standardfehlerbehandlung in der Regel minimal (z.B. Return-Code der HTTP-Anfrage). Es wird daher empfohlen, zusätzliche Verifizierungsschritte hinzuzufügen, um zu überprüfen, was die Anfragen tatsächlich zurückgeben. Wenn z.B. im Fehlerfall eine Bereinigung erforderlich ist, muss sie wahrscheinlich manuell implementiert werden. Es macht grundsätzlich Sinn zu überprüfen, ob das Skript das tut, was es tun soll, und hierfür auch indirekte Methoden zu verwenden, z.B. Überprüfen der Datenbank, ob die richtigen Informationen hinzugefügt wurden.

Skripte können auch noch andere Logik enthalten, in der Regeln festgelegt werden, wann und wie die Serveranfragen durchgeführt werden. Ein Beispiel ist das Setzen von Synchronisationspunkten, die dafür sorgen, dass das Skript zu diesem Zeitpunkt auf ein Ereignis warten soll, bevor es fortfährt. Die Synchronisationspunkte können verwendet werden, um sicherzustellen, dass eine bestimmte Aktion gleichzeitig aufgerufen wird, oder um die Arbeit zwischen mehreren Skripten zu koordinieren.

Performanztestskripte sind Software. Das Erstellen von Testskripten für Performanztests ist somit eine Softwareentwicklungsaktivität, die Qualitätssicherung und Tests beinhalten sollte, um zu überprüfen, ob das Skript wie erwartet abläuft und mit den gesamten Eingabedatenbereich arbeiten kann.

4.2.7 Performanztestskripte implementieren

Performanztestskripte werden basierend auf dem Performanztestkonzept und den Lastprofilen implementiert. Zwar unterscheiden sich die technischen Details der Implementierung je nach Ansatz und verwendeten Werkzeugen, der Gesamtprozess bleibt jedoch derselbe. Das Performanztestskript wird mithilfe einer integrierten Entwicklungsumgebung (IDE) oder eines Skripteditors erstellt, um das Verhalten eines Benutzers oder einer Komponente zu simulieren. Normalerweise wird das Skript erstellt, um ein bestimmtes Nutzungsprofil zu simulieren (häufig ist es jedoch möglich, mehrere Nutzungsprofile in einem Skript mit bedingten Anweisungen zu kombinieren).

Da die Reihenfolge der Anfragen festgelegt ist, kann das Skript je nach Ansatz aufgezeichnet oder programmiert werden. Die Aufzeichnung stellt normalerweise sicher, dass das reale System exakt simuliert wird, während die Programmierung auf die Kenntnis der richtigen Anfragenreihenfolge angewiesen ist.

Bei Aufzeichnungen auf Protokollebene müssen in den meisten Fällen nach der Aufzeichnung alle aufgezeichneten internen Bezeichner, die den Kontext definieren, ersetzt werden. Diese Bezeichner müssen in Variablen umgewandelt werden, die zwischen den Durchläufen mit geeigneten Werten verändert werden können, die aus den Anfrageantworten extrahiert werden (z.B. eine Benutzerkennung, die bei der Anmeldung abgerufen wird und für alle nachfolgenden Transaktionen bereitgestellt werden muss). Dies ist ein Teil der Parametrisierung der Skripte, manchmal auch als "Korrelation" bezeichnet. In diesem Zusammenhang hat der Begriff "Korrelation" eine andere Bedeutung als in der Statistik (wo er eine Beziehung zwischen zwei oder mehreren Dingen bedeutet). Fortgeschrittene Lasttestwerkzeuge führen meist eine gewisse Korrelation automatisch durch, die in manchen Fällen für eine Transparenz sorgt. In komplexeren Fällen kann jedoch eine manuelle Korrelation oder das Hinzufügen neuer Korrelationsregeln erforderlich sein. Eine falsche Korrelation oder eine fehlende Korrelation ist der Hauptgrund, warum aufgezeichnete Skripte nicht abgespielt werden können.

Wenn Tests mit mehreren virtuellen Benutzern mit demselben Benutzernamen ausgeführt werden, die auf denselben Datensatz zugreifen (was normalerweise während der Wiedergabe eines aufgezeichneten Skripts geschieht, wenn keine über die erforderliche Korrelation hinausgehende Änderungen gemacht werden), dann kann dies sehr leicht zu irreführenden Ergebnissen führen. Die Daten können im SUT vollständig zwischengespeichert werden (für einen schnelleren Zugriff von der Festplatte in den Speicher kopiert: caching) und die Ergebnisse wären dann viel besser als in der Produktion (wo solche Daten möglicherweise von einer Festplatte

gelesen werden). Die Verwendung derselben Benutzer und/oder Daten kann auch Nebenläufigkeitsprobleme verursachen (z.B. wenn Daten gesperrt sind, weil ein Benutzer sie gerade aktualisiert). Die Ergebnisse wären dann viel schlechter als in der Produktion, da die Software darauf warten würde, dass die Sperre wieder freigegeben wird, bevor der nächste Benutzer die Daten für eine Aktualisierung sperren könnte.

Folglich sollten Skripte und Testrahmen parametrisiert werden (d.h. dass feste oder aufgezeichnete Daten durch Werte aus einer Liste möglicher Auswahloptionen ersetzt werden), so dass jeder virtuelle Benutzer einen geeigneten Datensatz verwendet. Der Begriff "geeignet" bedeutet hier, dass die Daten verschieden genug sind, um Probleme mit der Zwischenspeicherung und Nebenläufigkeit zu vermeiden, die für das System, die Daten und die Testanforderungen spezifisch sind. Diese weitergehende Parametrisierung hängt von den Daten im System und von der Art und Weise ab, wie das System mit diesen Daten arbeitet. Daher erfolgt dies in der Regel manuell, aber es gibt Werkzeuge, die dies unterstützen.

Es gibt Fälle, in denen einige Daten parametrisiert werden müssen, damit der Test mehr als einmal funktioniert, beispielsweise wenn eine Bestellung erstellt wird und der Auftragsname eindeutig sein muss. Wenn der Name der Bestellung nicht parametrisiert wird, schlägt der Test fehl, sobald versucht wird, eine Bestellung mit einem vorhandenen (aufgezeichneten) Namen zu erstellen.

Um die Nutzungsprofile an die Realität anzugleichen, sollten Bedenkzeiten eingefügt und/oder (falls aufgezeichnet) angepasst werden, um eine angemessene Anzahl von Anfragen bzw. ausreichend Systemdurchsatz zu erzeugen, wie in Kapitel 4.2.5 beschrieben.

Wenn Skripte für separate Nutzungsprofile erstellt werden, werden diese zu einem Szenario kombiniert, das das gesamte Lastprofil implementiert. Das Lastprofil steuert, wie viele virtuelle Benutzer mit den jeweiligen Skripten wann und mit welchen Parametern gestartet werden. Die genauen Implementierungsdetails hängen vom jeweiligen Lasttestwerkzeug oder vom Testrahmen ab.

4.2.8 Die Durchführung des Performanztests vorbereiten

Die Hauptaktivitäten bei der Vorbereitung der Durchführung von Performanztests umfassen:

- Einrichten des zu testenden Systems
- Aufsetzen der Umgebung
- Einrichten der Lastgenerierungswerkzeuge und Testmonitore und dafür sorgen, dass alle erforderlichen Daten gesammelt werden können

Es ist wichtig sicherzustellen, dass die Testumgebung der Produktionsumgebung möglichst nahe kommt. Wenn dies nicht möglich ist, muss ein klares Verständnis der Unterschiede zwischen den Umgebungen vorhanden sein, sowie darüber, wie die

Testergebnisse auf die Produktionsumgebung projiziert werden. Im Idealfall werden die echte Produktionsumgebung und die echten Daten verwendet. Das Testen in einer maßstäblich verkleinerten Umgebung kann jedoch bereits dazu beitragen, eine Reihe von Performanzrisiken zu mindern.

Wichtig ist, im Auge zu behalten, dass die Performanz eine nichtlineare Funktion der Umgebung ist. Je weiter die Testumgebung von der normalen Produktionsumgebung weg ist, desto schwieriger wird es, genaue Prognosen über die Performanz in der Produktion zu machen. Die Unzuverlässigkeit der Prognosen und das steigende Risikoniveau wachsen, je mehr das Testsystem von der Produktion abweicht.

Die wichtigsten Teile der Testumgebung sind die Daten-, die Hardware- und Softwarekonfiguration, sowie die Netzwerkkonfiguration. Die Menge und Struktur der Daten könnten die Ergebnisse des Lasttests entscheidend beeinflussen. Wird für die Performanztests ein kleines Beispieldatenset oder ein Datenset mit anderer Datenkomplexität als in der Produktionsumgebung verwendet, kann dies zu irreführenden Ergebnissen führen, insbesondere wenn das Produktionssystem mit großen Datenmengen arbeitet. Es ist schwer vorherzusagen, wie stark die Datenmenge die Performanz beeinflusst, bevor reale Tests durchgeführt werden. Je näher die Testdaten in Größe und Struktur an die Produktionsdaten herankommen, desto zuverlässiger sind die Testergebnisse.

Wenn während des Tests Daten erzeugt oder geändert werden, müssen die ursprünglichen Daten möglicherweise vor dem nächsten Testzyklus wiederhergestellt werden, um sicherzustellen, dass sich das System im richtigen Zustand befindet.

Falls Teile des Systems oder einige Daten aus irgendeinem Grund nicht für die Performanztests verfügbar sind, sollten Maßnahmen implementiert werden, um das Problem zu umgehen. Zum Beispiel kann ein Platzhalter implementiert werden, um die Komponente eines Drittanbieters zur Kreditkartenverarbeitung zu ersetzen und zu simulieren. Dieser Prozess wird häufig als „Service-Virtualisierung“ bezeichnet und es gibt spezielle Werkzeuge, die diesen Prozess unterstützen. Die Verwendung solcher Werkzeuge wird dringend empfohlen, um das zu testende System zu isolieren.

Es gibt viele Möglichkeiten, Umgebungen bereitzustellen. Hierfür können beispielsweise die folgenden Optionen verwendet werden:

- Herkömmliche interne (und externe) Testlabore
- Cloud als Umgebung mit Infrastructure as a Service (IaaS), wenn einige Teile des Systems oder das gesamte System in der Cloud bereitgestellt werden
- Cloud als Umgebung, in der Software as a Service (SaaS) verwendet wird, wenn Drittanbieter den Lasttest als Service bereitstellen

Je nach den spezifischen Zielen und den zu testenden Systemen kann eine Testumgebung einer anderen vorgezogen werden, zum Beispiel:

- Um die Wirkung einer Leistungsverbesserung bzw. -optimierung zu testen, kann die Verwendung einer isolierten Testlaborumgebung die bessere Option sein, um selbst kleine Abweichungen zu erkennen, die aus der Änderung resultieren.
- Um einen vollständigen Lasttest der gesamten Produktionsumgebung durchzuführen, der sicherzustellen soll, dass das System die Last ohne größere Probleme bewältigen kann, könnten Tests aus der Cloud oder Tests als ein Service geeigneter sein. (Es ist zu beachten, dass dies nur für SUT funktioniert, die von einer Cloud aus erreichbar sind.)
- Um die Kosten zu senken, wenn die Performanztests zeitlich begrenzt sind, kann das Erstellen einer Testumgebung in der Cloud die wirtschaftlichere Lösung sein.

Unabhängig von dem für die Bereitstellung verwendeten Ansatz sollten sowohl Hardware als auch Software so konfiguriert werden, dass sie das Testziel und das Testkonzept erfüllen. Wenn die Umgebung der Produktion entspricht, sollte sie auf dieselbe Weise konfiguriert werden. Wenn es jedoch Unterschiede gibt, muss die Konfiguration möglicherweise angepasst werden, um diese Unterschiede zu berücksichtigen. Wenn beispielsweise Testrechner über weniger physischen Speicher verfügen als die Produktionsrechner, müssen möglicherweise die Software-Speicherparameter (z.B. die Java-Heap-Größe) angepasst werden, um das Auslagern von Speicher (memory paging) zu vermeiden.

Die richtige Konfiguration/Emulation des Netzwerks ist für globale und mobile Systeme wichtig. Für globale Systeme (d.h. wenn die Benutzer oder die Verarbeitung weltweit verteilt sind) kann ein möglicher Ansatz sein, Lastgeneratoren an den Orten bereitzustellen, an denen sich die Benutzer befinden. Für mobile Systeme ist die Netzwerkemulation aufgrund der unterschiedlichen verwendbaren Netzwerktypen die praktikabelste Option. Einige Lasttestwerkzeuge verfügen über integrierte Netzwerkemulationswerkzeuge und es gibt auch eigenständige Werkzeuge für die Netzwerkemulation.

Die Werkzeuge zur Lastgenerierung sollten ordnungsgemäß bereitgestellt werden, und die Testmonitore müssen so konfiguriert sein, dass alle für den Test erforderlichen Metriken erfasst werden. Die Liste der Metriken hängt von den Testzielen ab; es wird jedoch empfohlen, für alle Tests mindestens grundlegende Metriken zu erfassen (siehe Kapitel 2.1.2).

Abhängig von der Last, dem spezifischen Werkzeug-/Lastgenerierungsansatz und der Maschinenkonfiguration kann mehr als ein Lastgenerator erforderlich sein. Zur Überprüfung des Setups sollten auch die an der Lastgenerierung beteiligten Rechner überwacht werden. So wird eine Situation vermieden, in der die Last nicht ordnungsgemäß aufrechterhalten wird, weil einer der Lastgeneratoren langsam läuft.

Je nach Setup und den verwendeten Werkzeugen müssen die Lasttestwerkzeuge konfiguriert werden, um die entsprechende Last zu erzeugen. Beispielsweise können bestimmte Browser-Emulationsparameter eingestellt werden oder es kann ein IP-Spoofing verwendet werden (das simuliert, dass jeder virtuelle Benutzer eine andere IP-Adresse hat).

Vor Durchführung der Tests müssen die Umgebung und das Setup validiert werden. Dies geschieht in der Regel, indem ein kontrollierter Satz von Tests durchgeführt und das Ergebnis der Tests verifiziert wird. Außerdem wird verifiziert, ob die Testmonitore die wichtigen Informationen aufzeichnen.

Um zu überprüfen, ob ein Test wie geplant funktioniert, können verschiedene Techniken verwendet werden, einschließlich Protokollanalyse/Protokollauswertung und Verifizieren des Datenbankinhalts. Bei der Vorbereitung des Tests wird auch überprüft, ob die erforderlichen Informationen protokolliert werden, ob sich das System im ordnungsgemäßen Zustand befindet usw. Wenn der Test beispielsweise den Zustand des Systems erheblich ändert (weil Informationen der Datenbank hinzugefügt/geändert werden), muss das System möglicherweise vor dem Wiederholen des Tests in den ursprünglichen Zustand zurückversetzt werden.

4.3 Testdurchführung

Die Durchführung des Performanztests umfasst die Generierung einer Last gegen das SUT gemäß eines Lastprofils (normalerweise implementiert durch Performanztest-skripte, die anhand eines bestimmten Szenario aufgerufen werden), das Überwachen aller Teile der Umgebung sowie das Sammeln und Sichern aller mit dem Test verbundenen Ergebnisse und Informationen. In der Regel führen fortgeschrittene Lasttestwerkzeuge/Testrahmen diese Aufgaben automatisch aus (nachdem sie entsprechend konfiguriert wurden). Sie verfügen normalerweise über eine Konsole, um die Leistungsdaten während des Tests zu überwachen und bei Bedarf notwendige Anpassungen vornehmen zu können (siehe Kapitel 5.1). Je nach verwendetem Werkzeug, dem SUT und den spezifischen Tests, die durchgeführt werden, sind jedoch möglicherweise einige manuelle Schritte erforderlich.

Performanztests konzentrieren sich meist auf einen stabilen Zustand des Systems, in dem das Verhalten eingeschwungen ist.; Zum Beispiel der Zustand, wenn alle simulierten Benutzer/Threads initiiert sind und die Arbeit wie geplant ausführen. Wenn sich die Last ändert (z.B. weil neue Benutzer hinzugefügt werden), ändert sich das Verhalten des Systems und die Überwachung und Analyse der Testergebnisse wird schwieriger. Die Phase des in den stabilen Zustand Kommens wird üblicherweise als „Ramp-up“ und die Phase des Testabschlusses oft als „Ramp-down“ Phase bezeichnet..

Manchmal ist es wichtig, Übergangszustände zu testen, in denen sich das Verhalten des Systems ändert. Dies kann zum Beispiel für eine sehr große Anzahl von Benutzern oder für Lastspitzentests und ihre gleichzeitige Protokollierung gelten. Beim Testen von Übergangszuständen ist es wichtig zu verstehen, dass eine sorgfältige Überwachung und Analyse der Ergebnisse unbedingt erforderlich sind, da einige Standardansätze - wie beispielsweise die Überwachung von Durchschnitten - sehr irreführend sein können.

Während des “Ramp-ups” ist es ratsam, inkrementelle Lastzustände zu implementieren, um die Auswirkungen der stetig zunehmenden Last auf die Reaktionsfähigkeit des Systems zu überwachen. Dadurch wird sichergestellt, dass ausreichend Zeit für den “Ramp-up” zur Verfügung steht und das System die Last bewältigen kann. Sobald der stabile Zustand erreicht ist, ist es ratsam, die Stabilität der Last und der Systemreaktionen zu überwachen sowie zu kontrollieren, das zufällige Schwankungen (die immer vorhanden sind) nicht erheblich sind.

Es ist wichtig zu spezifizieren, wie Fehlerwirkungen und Ausfälle behandelt werden sollen, um sicherzustellen, dass keine Systemprobleme verursacht werden. Beispielsweise kann es wichtig sein, dass Benutzer bei einer Fehlerwirkung abgemeldet werden, um sicherzustellen, dass alle diesem Benutzer zugeordneten Ressourcen freigegeben werden.

Wenn die Überwachung in das Lasttestwerkzeug integriert ist und richtig konfiguriert wurde, wird die Funktion normalerweise zur gleichen Zeit gestartet wie die Testdurchführung. Wenn jedoch eigenständige Testmonitore verwendet werden, sollte die Überwachung separat gestartet und die erforderlichen Informationen gesammelt werden, damit eine anschließende Analyse der gesammelten Informationen zusammen mit den Testergebnissen durchgeführt werden kann. Gleiches gilt für die Protokollanalyse. Alle verwendeten Werkzeuge müssen von der Systemzeit her synchronisiert werden, damit die Informationen, die sich auf einen bestimmten Testausführungszyklus beziehen, gefunden werden können.

Für die Überwachung der Testdurchführung werden häufig die Konsole des Performanztestwerkzeugs und die Echtzeit-Protokollanalyse verwendet, um sowohl den Test als auch das SUT auf Probleme und Fehler zu prüfen. So lässt sich vermeiden, unnötig lange mit dem Durchführen umfangreicher Tests fortzufahren, die eventuell sogar andere Systeme beeinflussen können, wenn Probleme auftreten (z.B. wenn eine Fehlerwirkung auftritt, wenn Komponenten ausfallen oder wenn die erzeugten Lasten zu niedrig oder zu hoch sind). Da die Durchführung dieser Tests oft kostspielig ist, kann es erforderlich sein wenn Tests vom erwarteten Verhalten abweichen, diese abzubrechen oder direkt einige Anpassungen des Performanztests oder der Systemkonfiguration vorzunehmen.

Eine Methode zum Verifizieren von Lasttests, die direkt auf Protokollebene kommunizieren, besteht darin, mehrere (funktionale) Skripte auf GUI-Ebene

auszuführen oder ähnliche Nutzungsprofile parallel zum laufenden Lasttest manuell auszuführen. Dadurch wird geprüft, dass sich die Antwortzeiten während des Tests auf Protokollebene nur um die auf der Clientseite verbrauchte Zeit von den manuell auf GUI-Ebene gemessenen Antwortzeiten unterscheiden.

In einigen Fällen von Performanztests, die automatisiert ausgeführt werden (z.B. als Teil der kontinuierlichen Integration, wie in Kapitel 3.4 beschrieben), müssen die Kontrollen automatisch durchgeführt werden, da eine manuelle Überwachung und Eingreifen möglicherweise nicht möglich sind. In diesem Fall sollte der Testaufbau in der Lage sein, Abweichungen oder Probleme zu erkennen und eine Warnung auszulösen (normalerweise während der Test ordnungsgemäß abgeschlossen wird). Dieser Ansatz ist bei regressiven Performanztests einfacher zu implementieren, da das Verhalten des Systems allgemein bekannt ist, als bei explorativen oder bei umfangreichen teuren Performanztests, die möglicherweise dynamische Anpassungen während des Tests erfordern.

4.4 Analyse der Ergebnisse und Berichterstattung

In Kapitel 4.1.2 wurden die verschiedenen Metriken in einem Performanztestkonzept erläutert. Wenn diese im Voraus definiert werden, wird festgelegt, was für jeden Testlauf gemessen werden muss. Nach Abschluss eines Testzyklus sollten die Daten für die definierten Metriken gesammelt werden.

Bei der Analyse der Daten werden diese zunächst mit den Zielen des Performanztests verglichen. Sobald das Verhalten verstanden wurde, können Schlussfolgerungen gezogen werden, die einen aussagekräftigen zusammenfassenden Bericht einschließlich der empfohlenen Maßnahmen ergeben. Diese Maßnahmen können das Ändern physischer Komponenten (z.B. Hardware, Router), das Ändern von Software (z.B. das Optimieren von Anwendungen und Datenbankaufrufen) und das Ändern des Netzwerks (z.B. Lastausgleich, Routing) umfassen.

Typischerweise werden die folgenden Daten analysiert:

- **Status der simulierten (z.B. virtuellen) Benutzer**. Dies muss zuerst untersucht werden. Es wird normalerweise erwartet, dass alle simulierten Benutzer die im Nutzungsprofil spezifizierten Aufgaben ausführen konnten. Jede Unterbrechung dieser Aktivitäten würde das aufzeigen, was einem tatsächlicher Benutzer möglicherweise widerfährt. Daher ist es sehr wichtig, zuerst zu prüfen, ob alle Benutzeraktivitäten abgeschlossen wurden, da die aufgetretenen Fehler die anderen Performanzdaten beeinflussen können.
- **Transaktionsantwortzeit**. Diese kann auf mehrere Arten gemessen werden, einschließlich Minimum, Maximum, Durchschnitt und Perzentil (z.B. 90% Perzentil). Die minimalen und maximalen Messwerte zeigen die Extremwerte der Systemperformanz. Die durchschnittliche Performanz weist erst einmal nur auf den mathematischen Durchschnitt hin, der durch Ausreißer häufig verzerrt

sein kann. Das 90% Perzentil wird häufig als Ziel verwendet, da es die Mehrheit der Benutzer darstellt, die eine bestimmte Performanzschwelle unterschreiten. Es wird nicht empfohlen, eine 100%ige Übereinstimmung mit den Performanzzielen zu fordern, da die dafür erforderlichen Testressourcen möglicherweise zu groß sind und der zu erreichende Nettoeffekt für die Benutzer oft gering ist.

- **Transaktionen pro Sekunde**. Dies gibt Auskunft darüber, wie viel Arbeit das System geleistet hat (Systemdurchsatz).
- **Transaktionsfehler** Diese Daten werden für die Analyse der Transaktionen pro Sekunde verwendet. Fehler zeigen an, dass das erwartete Ereignis oder der erwartete Prozess nicht abgeschlossen oder nicht ausgeführt wurde. Jeder aufgetretene Fehler kann bedenklich sein und die Grundursache muss untersucht werden. Fehlgeschlagene Transaktionen können auch zu ungültigen Messdaten für die Transaktionen pro Sekunde führen, da eine fehlgeschlagene Transaktion viel weniger Zeit in Anspruch nimmt als eine abgeschlossene Transaktion.
- **Anfragen pro Sekunde**. Dieser Wert vermittelt eine Aussage über die Anzahl der Treffer bzw. Anfragen der simulierten Benutzer pro Sekunde an einen Server während des Tests.
- **Netzwerkdurchsatz**. Dies wird normalerweise in Bits je Zeiteinheit gemessen, wie z.B. in Bits pro Sekunde. Dies repräsentiert die Datenmenge, die die simulierten Benutzer pro Sekunde vom Server erhalten (siehe Kapitel 4.2.5).
- **HTTP-Antworten**. Diese werden pro Sekunde gemessen und enthalten mögliche Antwortcodes wie: 200, 302, 304, 404, wobei letzterer angibt, dass eine Seite nicht gefunden wurde.

Obwohl viele dieser Informationen in Tabellen dargestellt werden können, erleichtern grafische Darstellungen das Sichten der Daten und das Erkennen von Trends.

Für die Analyse von Daten können folgende Techniken verwendet werden:

- Vergleich der Ergebnisse mit den festgelegten Anforderungen
- Beobachten von Trends bei den Ergebnissen
- Statistische Qualitätskontrollen
- Identifizieren von Fehlern
- Vergleich der erwarteten und tatsächlichen Ergebnisse
- Vergleichen der Ergebnisse mit früheren Testergebnissen
- Verifizieren der ordnungsgemäßen Funktion von Komponenten (z.B. Server, Netzwerke)

Durch das Identifizieren der Korrelation zwischen Metriken lässt sich oft besser verstehen, wann genau die Performanz nachlässt. Wie viele Transaktionen pro Sekunde wurden beispielsweise verarbeitet, als die CPU eine Kapazität von 90% erreichte und das System langsamer wurde?

Die Analyse kann dabei helfen, die Grundursache für Leistungsabfälle oder -ausfälle zu ermitteln, was wiederum die Korrektur erleichtert. Fehlernachtests helfen zu ermitteln, ob die Korrekturmaßnahme die Grundursache des Fehlers behoben hat.

Berichterstattung
Die Analyseergebnisse werden konsolidiert und mit den im Performanztestkonzept festgelegten Zielen verglichen. Diese werden zusammen mit anderen Testergebnissen im allgemeinen Teststatusbericht oder in einem speziellen Bericht für den Performanztest berichtet. Der Detaillierungsgrad, mit dem berichtet wird, sollte den Erfordernissen der Stakeholder entsprechen. Die auf diesen Ergebnissen basierenden Empfehlungen beziehen sich normalerweise auf Kriterien für das Software-Release (einschließlich der Zielumgebung) oder erforderliche Leistungsverbesserungen.

Ein typischer Performanztestbericht kann Folgendes enthalten:

Zusammenfassung
Dieser Abschnitt wird erstellt, wenn alle Performanztests durchgeführt und alle Ergebnisse analysiert und verstanden wurden. Ziel ist es, kurze und verständliche Schlussfolgerungen, Befunde und Empfehlungen für das Management mit dem Ziel eines umsetzbaren Ergebnisses zu präsentieren.

Testergebnisse
Die Testergebnisse können einige oder alle der folgenden Informationen enthalten:

- Eine Zusammenfassung, die die Ergebnisse erläutert und näher ausführt.
- Ergebnisse eines Basistests, der als „Momentaufnahme“ der Systemleistung zu einem bestimmten Zeitpunkt dient und die Grundlage für den Vergleich mit nachfolgenden Tests bildet. Die Ergebnisse sollten das Datum und die Uhrzeit des Testbeginns, die Höchstzahl gleichzeitiger Benutzer, den gemessenen Durchsatz und die wichtigsten Befunde umfassen. Zu den wichtigsten Befunden zählen die gemessene Gesamtfehlerrate, die Antwortzeiten und der durchschnittliche Durchsatz.
- Ein Diagramm auf hoher Ebene, das alle Komponenten der Systemarchitektur zeigt, die Testziele beeinflussen konnten (oder tatsächlich beeinflusst haben).
- Eine detaillierte Analyse (Tabellen und Diagramme) der Testergebnisse mit Antwortzeiten, Transaktionsraten, Fehlerraten und Leistungsanalyse. Die Analyse enthält auch eine Beschreibung dessen, was beobachtet wurde, z.B. wann eine stabile Anwendung instabil wurde und die Fehlerquelle (z.B. Webserver, Datenbankserver).

Testprotokolle/Aufgezeichnete Informationen
Ein Protokoll jedes Testlaufs sollte aufgezeichnet werden. Das Protokoll enthält normalerweise Folgendes:

- Datum und Uhrzeit des Testbeginns
- Testdauer

- Die für den Test verwendeten Testskripte (einschließlich des Zusammenspiels, wenn mehrere Skripte verwendet wurden) sowie relevante Skriptkonfigurationsdaten
- Testdatendatei bzw. -dateien, die beim Test verwendet wurden
- Name und Speicherort der während des Tests erstellten Daten-/ Protokoll-dateien
- Getestete Hardware-/Softwarekonfiguration (insbesondere Änderungen zwischen den Testläufen)
- Durchschnittliche und Spitzenauslastung von CPU und RAM auf Web- und Datenbankservern
- Hinweise zur erreichten Leistung
- Gefundene Fehler

Empfehlungen
Empfehlungen, die sich aus den Tests ergeben, können Folgendes umfassen:

- Empfohlene technische Änderungen, z.B. Neukonfiguration von Hardware oder Software oder Netzwerkinfrastruktur
- Bereiche, die zur weiteren Analyse identifiziert wurden (z.B. Analyse der Web-server-Protokolle, um die Grundursachen von Problemen und/oder Fehlern zu ermitteln)
- Empfehlungen hinsichtlich einer zusätzlichen Überwachung von Gateways, Servern und Netzwerken, die erforderlich ist, um detailliertere Daten zur Messung von Leistungsmerkmalen und Trends (z B. Leistungsverschlechte-rung) zu erhalten.

5. Werkzeuge – 90 Min.

Schlüsselbegriffe

Lastgenerator, Lastmanagement, Performanztestwerkzeug, Testmonitor

Lernziele

5.1 Werkzeugunterstützung
PTFL-5.1.1 (K2) Verstehen, wie Werkzeuge den Performanztest unterstützen

5.2 Eignung von Werkzeugen
PTFL-5.2.1 (K4) Die Eignung von Performanztestwerkzeugen in einem bestimmten Projektszenario bewerten können

5.1 Werkzeugunterstützung

Zu den Testwerkzeugen, die den Performanztest unterstützen, zählen die folgenden Werkzeugtypen:

Lastgeneratoren
Der Lastgenerator kann über eine integrierte Entwicklungsumgebung (IDE), einen Skripteditor oder eine Toolsuite mehrere Client-Instanzen erstellen und ausführen, die das Benutzerverhalten gemäß eines definierten Nutzungsprofils simulieren. Das Erzeugen mehrerer Instanzen in kurzer Zeit führt zu einer Belastung des zu testenden Systems. Der Generator erzeugt die Last und erfasst auch Metriken für die später zu erstellenden Berichte.

Beim Ausführen der Performanztests ist es das Ziel von Lastgeneratoren, die reale Welt so gut wie möglich nachzuahmen. Dies bedeutet oft, dass Benutzeranfragen von verschiedenen Standorten aus benötigt werden, und nicht nur vom Testort. Umgebungen, die mit mehreren Präsenzpunkten (Points of Presence) eingerichtet sind, sorgen dafür, dass die Last verteilt wird, sodass nicht alle Last aus einem einzigen Netzwerk stammt. Dies gibt dem Test einen realistischen Charakter, obwohl es auch vorkommen kann, dass die Ergebnisse manchmal verzerrt werden, wenn Netzwerk-Hops Verzögerungen verursachen.

Lastmanagement-Konsole
Die Lastmanagement-Konsole bietet die Steuerung zum Starten und Stoppen des Lastgenerators (bzw. der Lastgeneratoren). Die Konsole sammelt auch Metriken aus den verschiedenen Transaktionen, die in den vom Generator verwendeten Lastinstanzen definiert sind. Die Konsole ermöglicht die Anzeige von Berichten und Diagrammen aus den Testdurchführungen und unterstützt die Ergebnisanalyse.

Testmonitore
Die Testmonitore (Performanzmonitore) laufen gleichzeitig mit der zu testenden Komponente oder dem zu testenden System und überwachen, zeichnen auf und/oder analysieren das Verhalten der Komponente oder des Systems. Zu den typischen Komponenten, die von Testmonitoren überwacht werden, zählen Webserver-Warteschlangen, Hauptspeicher und Festplattenspeicher. Testmonitore können die Grundursachenanalyse bei einer Leistungsverschlechterung in einem zu testenden System effektiv unterstützen. Auch können sie zur Überwachung einer Produktionsumgebung verwendet werden, nachdem das Produkt freigegeben wurde. Während der Testdurchführung von Performanztests können Testmonitore auch für die Überwachung des Lastgenerators selbst verwendet werden.

Zu den Lizenzmodellen für Performanztestwerkzeuge gehören die klassischen Standort-/Arbeitsplatz-Lizenzen mit vollständiger Eigentümerschaft, sowie Cloud-basierte, nutzungsbasierte Lizenzmodelle („Pay-As-You-Go“) und Open-Source-Lizenzen, die in einer definierten Umgebung oder über Cloud-Angebote kostenlos genutzt werden können.

Jedes dieser Modelle bringt eine andere Kostenstruktur mit sich und kann eine laufende Wartung beinhalten. Grundsätzlich gilt, dass für jedes ausgewählte Werkzeug Zeit und Budget (für Schulungen und/oder Selbststudium) erforderlich sind, um zu verstehen, wie das Werkzeug arbeitet.

5.2 Eignung von Werkzeugen

Die folgenden Faktoren sollten bei der Auswahl eines Performanztestwerkzeugs berücksichtigt werden:

Kompatibilität
Im Allgemeinen wird ein Werkzeug für die Organisation und nicht nur für ein Projekt ausgewählt. Dies bedeutet, dass die folgenden Faktoren in der Organisation berücksichtigt werden:

- Protokolle: Wie in Kapitel 4.2.1 beschrieben, sind Protokolle ein sehr wichtiger Aspekt bei der Auswahl des Performanztestwerkzeugs. Wer weiß, welche Protokolle ein System verwendet und welche davon getestet werden, verfügt über notwendige Informationen, um ein geeignetes Testwerkzeug zu bewerten.
- Schnittstellen zu externen Komponenten: Schnittstellen zu Softwarekomponenten oder anderen Werkzeugen müssen gegebenenfalls als Teil der vollständigen Integrationsanforderungen betrachtet werden, um Prozess- oder andere Interoperabilitätsanforderungen zu erfüllen (z.B. Integration in den kontinuierlichen Integrationsprozess).
- Plattformen: Die Kompatibilität mit den Plattformen (und deren Versionen) innerhalb einer Organisation ist unerlässlich. Dies gilt für die Plattformen, auf

denen die Werkzeuge gehostet werden, und für die Plattformen, mit denen die Werkzeuge zur Überwachung und/oder Lasterzeugung interagieren.

Skalierbarkeit
Ein weiterer Faktor, der zu berücksichtigen ist, ist die Gesamtzahl der gleichzeitigen Benutzersimulationen, die das Werkzeug bearbeiten kann. Dies umfasst mehrere Faktoren:

- Maximale Anzahl der erforderlichen Lizenzen
- Konfigurationsanforderungen für die Workstation/Server zur Lasterzeugung
- Möglichkeit, Last von mehreren Präsenzpunkten bzw. Points of Presence (z.B. von verteilten Servern) zu generieren

Verständlichkeit
Ein weiterer, zu berücksichtigender Faktor ist, wieviel technisches Wissen für die Verwendung des Werkzeugs erforderlich ist. Dies wird oft übersehen und kann dazu führen, dass unerfahrene Tester die Tests nicht passend konfigurieren, was dann zu nicht nutzbaren Ergebnisse führt. Für Tests, die komplexe Szenarien sowie ein hohes Maß an Programmierung und Anpassungsfähigkeit erfordern, sollten die Teams sicherstellen, dass die Tester über die Fähigkeiten, das Wissen und das Training verfügen, die dafür erforderlich sind.

Überwachung
Ist die Überwachung, die das Werkzeug liefert, ausreichend? Gibt es andere Test-monitore in der Umgebung, mit denen die Überwachung durch das Werkzeug ergänzt werden kann? Kann die Überwachung mit den definierten Transaktionen korreliert werden? Alle diese Fragen müssen beantwortet werden, um festzustellen, ob das Werkzeug die für das Projekt erforderliche Überwachung bereitstellt.

Wenn die Überwachung durch ein separates Programm, einzelne Werkzeuge oder Werkzeuge im Verbund bereitgestellt wird, dann kann dies auch zur Überwachung der Produktionsumgebung verwendet werden, nachdem das Produkt freigegeben ist.

6. Referenzen

6.1 Standards

- [ISO25000] ISO/IEC 25000:2005, Software-Engineering – Qualitätskriterien und Bewertung von Softwareprodukten (SQuaRE)

6.2 ISTQB-Dokumente

- [ISTQB_UT_SYL] ISTQB® Foundation Level Usability Testing Lehrplan, Version 2018
- [ISTQB_ALTA_SYL] ISTQB® Advanced Level Test Analyst Lehrplan, Version 2012
- [ISTQB_ALTTA_SYL] ISTQB® Advanced Level Technical Test Analyst Lehrplan, Version 2012
- [ISTQB_ALTM_SYL] ISTQB® Advanced Level Test Manager Lehrplan, Version 2012
- [ISTQB_FL_SYL] ISTQB® Foundation Level (Core) Lehrplan, Version 2018
- [ISTQB_FL_AT] ISTQB® Foundation Level Agile Tester Lehrplan, Version 2014
- [ISTQB_GLOSSARY] ISTQB® GTB Standardglossar der Testbegriffe, http://glossary.istqb.org

6.3 Fachliteratur

[Anderson01] Lorin W. Anderson, David R. Krathwohl (Hrsg.) “A Taxonomy for Learning, Teaching and Assessing: A Revision of Bloom’s Taxonomy of Educational Objectives”, Allyn & Bacon, 2001, ISBN 978-0801319037

[Bath14] Graham Bath, Judy McKay, “Praxiswissen Softwaretest – Test Analyst und Technical Test Analyst”, dpunkt.verlag, 2015, ISBN 978-3-86490-137-9

[Molyneaux09] Ian Molyneaux, “The Art of Application Performance Testing: From Strategy to Tools”, O’Reilly, 2009, ISBN: 9780596520663

[Microsoft07] Microsoft Corporation, “Performance Testing Guidance for Web Applications”, Microsoft, 2007, ISBN: 9780735625709

7. Index