XCEPTANCE
Software Technologies

Software-Qualitätssicherung, Teil IV: Lasttests von Web-Anwendungen

Friedrich-Schiller-Universität Jena, Wintersemester 2017/2018
Ronny Vogel, Xceptance GmbH

Agenda

- Was ist ein Lasttest?
- Simulation von Web-Last
- Vorgehensweise bei Lasttests
- Interpretation der Ergebnisse – Patterns

Was ist ein Lasttest?

Definition

- Kontrollierte Nutzung des zu testenden Systems
- unter Berücksichtigung von Benutzer- und Transaktionsmengen
- mit Beobachtung des Systemverhaltens.

Ein Lasttest prüft mehrere Qualitätsmerkmale nach ISO 9126/25000

- Funktionale Eignung: Funktionale Korrektheit
- Zuverlässigkeit: Reife, Verfügbarkeit
- Leistungseffizienz: Zeitverhalten, Ressourcenverbrauch, Kapazität

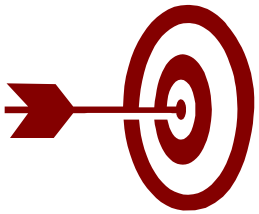
Grundsätzliche Ziele von Lasttests

1) Aufdeckung funktionaler Fehler, die bei paralleler oder intensiver Nutzung des Systems auftreten

- Deadlocks
- Race Conditions
- Crashes (unerwartetes Ende von Prozessen)
- Endlosschleifen
- Starvation (Verhungern)

2) Prüfung des Zeit- und Verbrauchsverhaltens unter Last

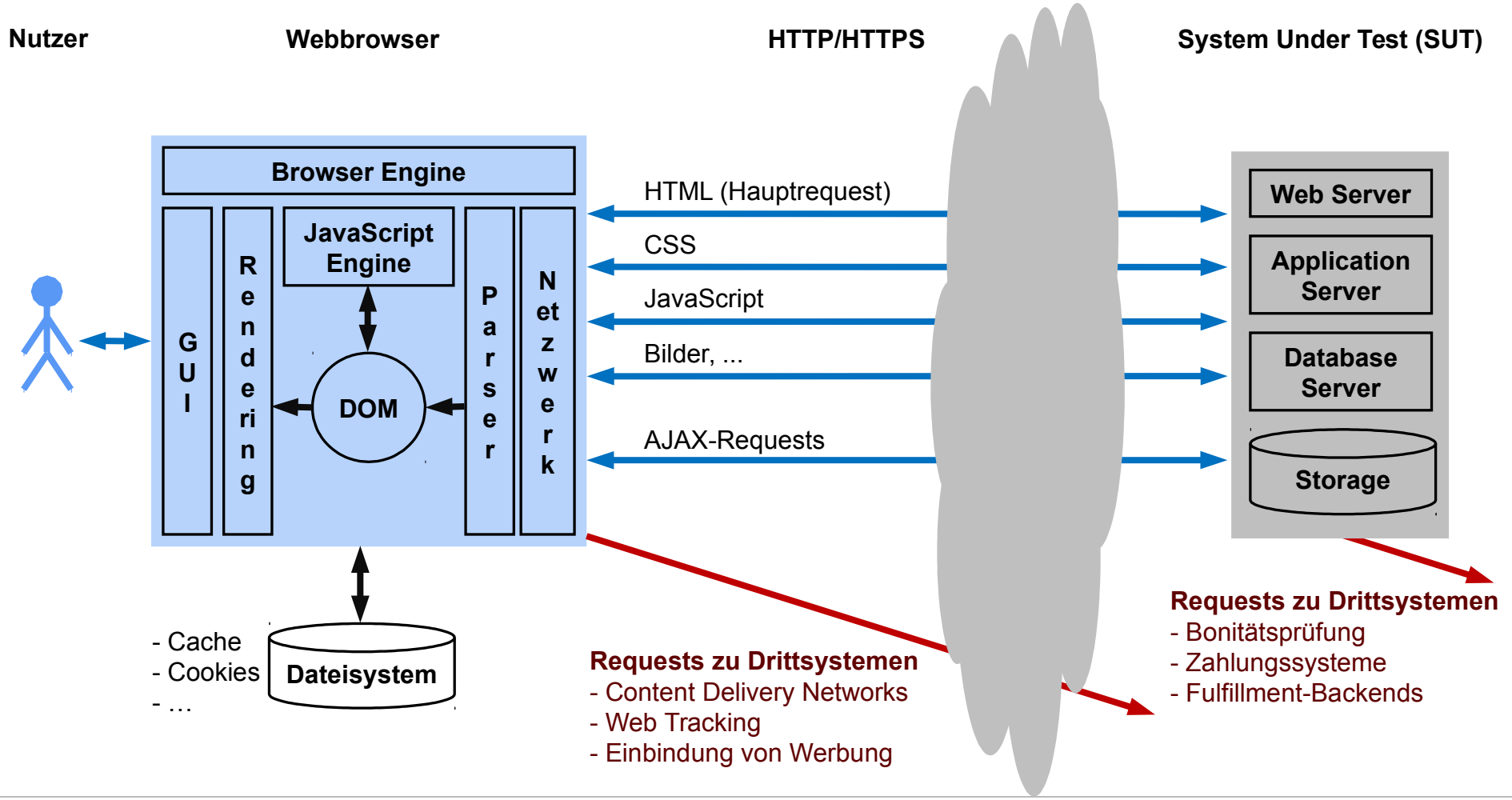
- Antwortzeiten, Durchsatz
- Verbrauch von Rechenleistung, Hauptspeicher, Bandbreite, Plattenplatz (betrieblich relevant, Kosten, Sizing-Informationen, Skalierbarkeit)



Agenda

- Was ist ein Lasttest?
- Simulation von Web-Last
- Vorgehensweise bei Lasttests
- Interpretation der Ergebnisse – Patterns

Web-Anwendungen: Big Picture



Simulation von Web-Last (1)

Kontrollierte Nutzung einer Web-Anwendung…

- Verhalten von Nutzern und ihrer Webbrowser

…unter Berücksichtigung von Benutzer- und Transaktionsmengen

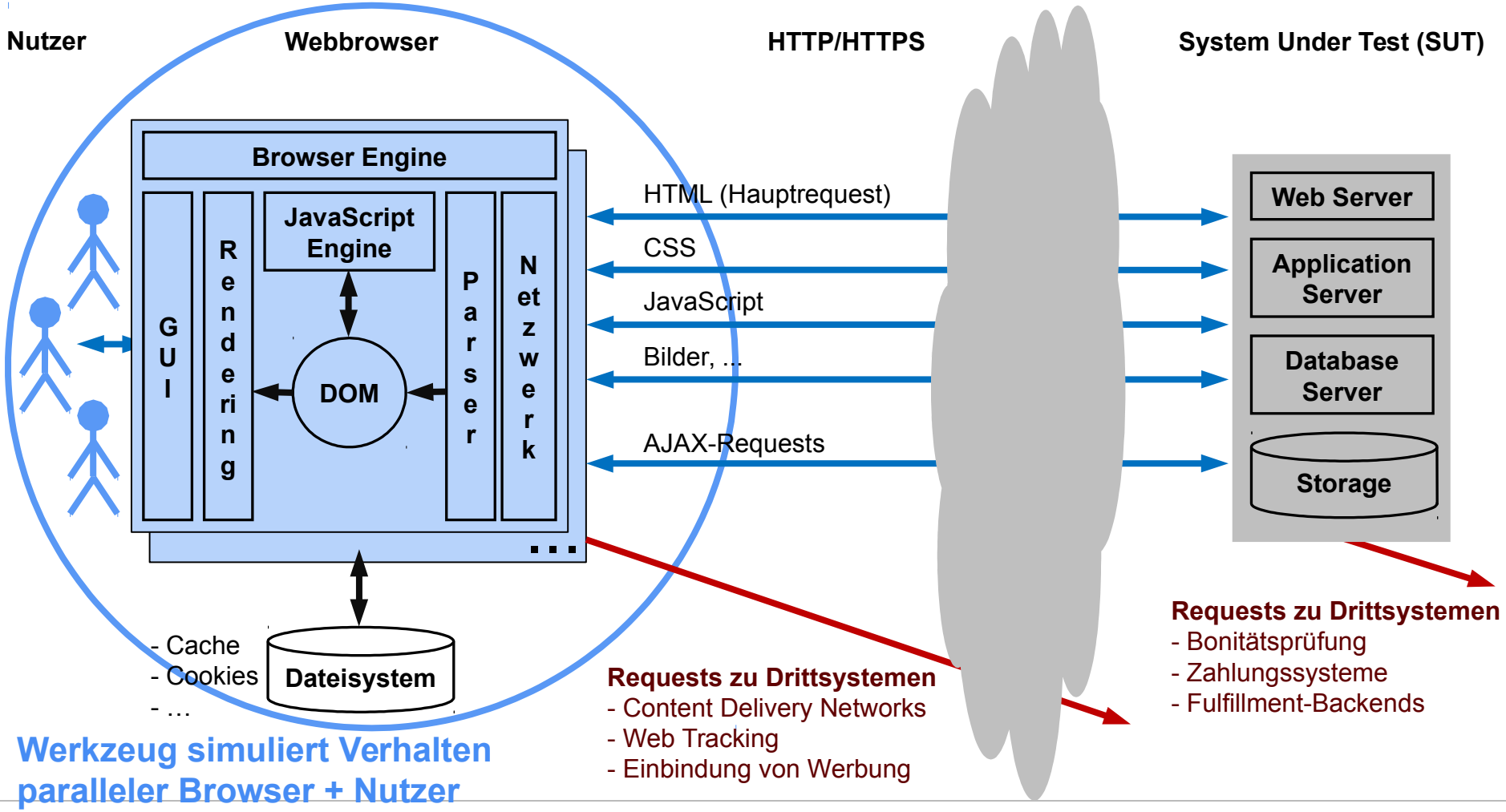
- hohe Parallelität, passende Lastmodelle und Lastparameter
- unterschiedliches Verhalten einzelner Nutzergruppen (Szenarien)
- zur Erzeugung hoher Lasten: viele Client-Rechner, Webbrowser ohne UI, …

…mit Beobachtung des Systemverhaltens

- Validierung der Antwortseiten
- Sammeln umfassender Ergebnisdaten
- Auswertung und übersichtliches Reporting der Ergebnisse

➔ spezielle Werkzeuge nötig

Simulation von Web-Last (2)

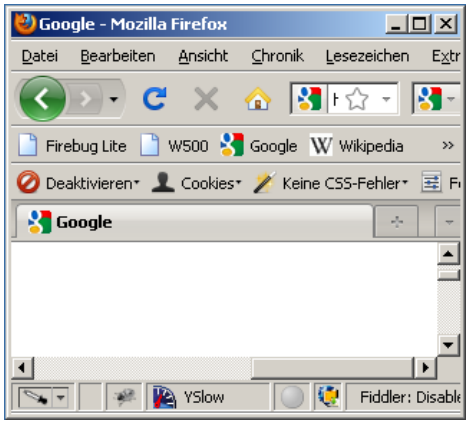


Werkzeug simuliert Verhalten paralleler Browser + Nutzer

Simulation von Browserverhalten

Zum Browserverhalten gehören zum Beispiel:

- Laden der HTML-Hauptdatei
- paralleles Nachladen und Interpretieren von CSS, JavaScript, Bildern, ...
- Cookie-Behandlung innerhalb der simulierten Nutzer-Sessions
- Caching im Browser
- AJAX-Requests
- HTTP: Header, Timeouts, Keep-Alive, Proxy-Nutzung, Basic Authentication, Retry, Content Compression, …
- HTTP-Methoden: GET, HEAD, POST, PUT, ...
- Desktop, Mobilgeräte, …
- Ausführen/Ausschließen von Anfragen an Drittsysteme



Simulation von Nutzerverhalten

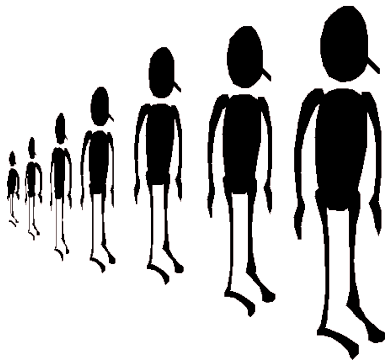
Zum Nutzerverhalten gehören zum Beispiel:

- unterschiedliche Clickstreams/Szenarien
- lesende Aktionen (reines Browsing)
- schreibende Aktionen (ändern Geschäftsobjekte)
- Scrolling, z.B. bei implementiertem „Endless Scrolling“
- Zufall (Datenvariation, zeitliche Variation)
  - Abruf unterschiedlicher Katalog- und Produktseiten
  - unterschiedliche Anzahlen von Produkten im Warenkorb
  - unterschiedliche Wartezeiten zwischen Aktionen
- Suchbegriffe mit/ohne Treffer, mit Wildcards, ...
- Paging (Blättern in mehrseitigen Listen)

Simulation des Gesamtverhaltens

Zum Gesamtverhalten aller Nutzer gehört zum Beispiel:

- hohe Parallelität
- Mix von Nutzerszenarien
- Anzahl simulierter Nutzer je Szenario
- eventuell geografische Verteilung
- Lastprofil über längere Zeiträume (Ramp-Up, Lastspitzen)
- Zufall (Datenvariation, zeitliche Variation)



Genauigkeit der Last-Simulation

Möglichst Abbildung der realen Nutzung, aber:

- schwierige Prognose von Nutzeranzahl und Nutzerverhalten, vor allem bei neuen Anwendungen
- hohe Varianz des Nutzerverhaltens, viele verschiedene Clickstreams
- viele unterschiedliche Browserversionen und Settings
- in Realität Rückwirkung des Systemverhaltens auf Nutzer möglich (lange Antwortzeiten ➔ Nutzer senden weitere Requests oder brechen ab)

➔ Simulation nur so genau wie nötig
➔ Statistische Annäherung an die Realität
➔ Gesamtheit der Requests ist relevant, nicht Abläufe in einzelnen Sessions
➔ Lastrelevante Grundfunktionen von Webbrowsern simulieren, keine Unterschiede im Detailverhalten

Agenda

- Was ist ein Lasttest?
- Simulation von Web-Last
- Vorgehensweise bei Lasttests
- Interpretation der Ergebnisse – Patterns

Vorgehensweise bei Lasttests

Phasen analog anderer Testarten

- Testplanung
- Testvorbereitung
- Testdurchführung
- Testauswertung

Besonderheiten

- Lasttests erfordern meist höhere Voraussetzungen, als andere Testarten
- technische Herausforderungen bei der Entwicklung der Testskripte
- Fehlersuche und Tuning oft langwierig
- oft viele Iterationen nötig

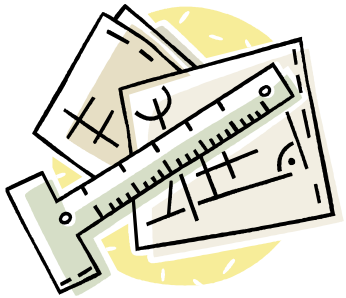
➔ Grundsätzlich so früh wie möglich beginnen!

➔ Aber: Das SUT muss in einem funktional stabilen Zustand sein.

Testplanung – Allgemeines

Festlegung folgender Aspekte

- Testziele
- Testszenarien (zu simulierende Geschäftsfälle)
- zu simulierende Last
- Testdaten (Art, Menge, Zusammensetzung, Erzeugung...)
- Zielwerte, z.B. Grenzen für Antwortzeiten
- Testumgebung (Hard- und Software, welche Drittsysteme, Simulatoren)
- Lasterzeugung (Hard- und Software)
- Scripting
- Organisatorisches (Termine, Verantwortlichkeiten, Zugänge, Dokumente, Prozesse)
- benötigte Reports und Metriken



Testplanung – Detaillierte Ziele

Detaillierte Lasttest-Ziele sind zum Beispiel:

- Finden lastbedingter Fehler, Prüfen der Stabilität
- Prüfung des Systemverhaltens unter gegebener Last
- Ermitteln der maximalen Last unter Einhaltung gegebener Antwortzeiten
- Finden der am höchsten ausgelasteten Ressourcen/Bottlenecks
- Optimierung des SUT
- Ermittlung des Ressourcenbedarfs zur Erreichung bestimmter Performance
- Gewinnen von Betriebs- und Wartungserfahrungen unter Last

Die Ziele beeinflussen:

- Lasttest-Arten
- Testszenarien
- Lastmodelle, Lastprofile

Testplanung – Lasttest-Arten

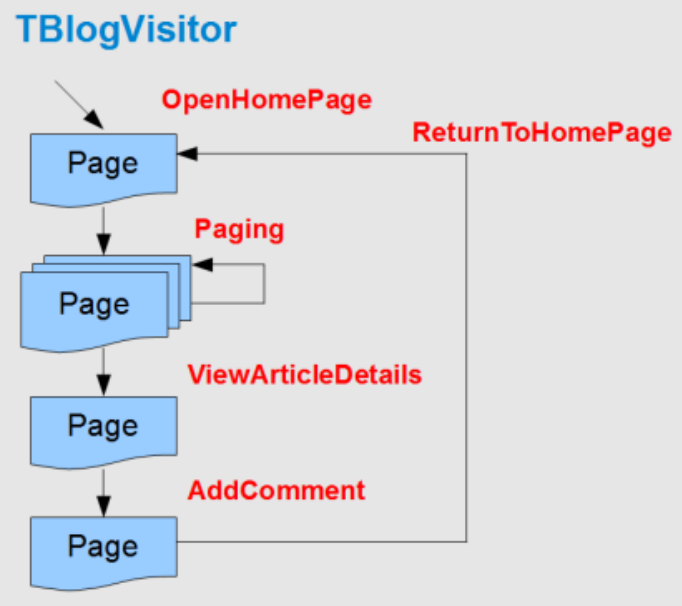
Lasttest-Arten je nach Ziel

- *Stresstest:* Last erhöhen, bis Fehler auftreten oder das Antwortverhalten bestimmte Grenzen überschreitet
- *Dauerlasttest:* Lasttest kontinuierlich über einen längeren Zeitraum
- *Fail-Over-Test:* Ausfall und Wiederanlauf redundanter Systemkomponenten unter Last testen
- *Performance-Test:* Test mit festgelegter Last auf Einhaltung vorgegebener Lastanforderungen und Finden von Bottlenecks
- *Sizing-Test/Kapazitätsplanung:* Ermitteln von Grenzen diverser HW-/SW-Konfigurationen; Abschätzung benötigter HW und SW für gegebene Lasten
- *Skalierbarkeitstest:* prüft
  - Antwort- und Ressourcenverhalten bei steigender Last
  - verarbeitbare Last bei Hinzunahme von HW- und SW-Komponenten

Testplanung – Szenarien (1)

Testszenario

- Abfolge von Schritten in der zu testenden Anwendung
- entspricht einer Sitzung eines Nutzers
- Begriffe: Testszenario = Nutzerszenario ≈ Geschäftsfall ≈ Nutzertyp



Testplanung – Szenarien (2)

Hinweise

- meist Beschränkung auf 5..10 Testszenarien
- sollten in Summe eine möglichst reale Nutzung des Systems widerspiegeln
- Szenarien durch Aufteilung in isolierte Features/Qualitäten flexibel halten
  - Feature nicht verfügbar ➔ Nutzertyp bei Ausführung weglassen
  - Probleme unter Last ➔ Eingrenzen durch Test von Einzelszenarien
  - Lastprofile einfacher justierbar ➔ „weniger Suche, mehr Bestellungen“
- eventuell auch Backoffice-Nutzer und Batchprozesse berücksichtigen

Testplanung – Szenarien, Beispiel

Beispiel für einfachen Online-Shop, 5 Nutzerszenarien

- Browsing von Katalog- und Produktseiten: 50% der Sessions
- Warenkorb (Produkte zufügen, entfernen): 10% der Sessions
- Bestellung: 5% der Sessions
- Suche mit Anzeige von Suchergebnissen: 30% der Sessions
- Nutzer-Registrierung: 5% der Sessions

Testplanung – Validierung

Reales Erlebnis:

- „Als wir die Last weiter erhöht haben, wurde das System wieder schneller und stabiler!“
- Die Fehlerseiten wurden tatsächlich sehr schnell ausgeliefert!

➔ Vom System Under Test (SUT) gelieferte Antworten auf Korrektheit prüfen!

In der Praxis richtiges Maß finden!

- Zu viel Validierung
  - Scripte brechen bei Änderungen der Anwendung oder der Testdaten
  - hoher Ressourcenbedarf durch Validierung
- Zu wenig Validierung
  - Fehler werden nicht erkannt

Testplanung – Lastmodelle (1)

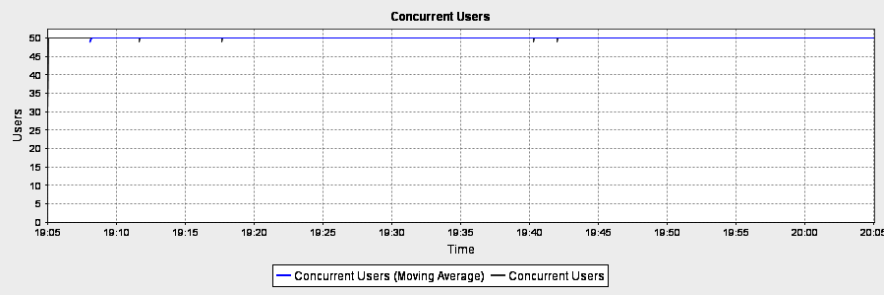
Lastmodell: Vorgegebene Nutzerzahl

- definierte parallele Last durch gegebene Anzahl virtueller Nutzer je Szenario, z.B. 60 Browsing-Nutzer, 5 Bestell-Nutzer, 20 Such-Nutzer
- einfacher vergleichbar, aber weniger realistisch
- kein Aufschaukeln der Last bei schlechtem Antwortverhalten des SUT
- Nachteil: Anzahl der ausgeführten Testszenarien je Zeiteinheit durch Antwortzeiten des SUT beeinflusst

➔ ein vorgegebenes Verhältnis der auszuführenden Testszenarien erfordert Probeläufe und Anpassung der Nutzeranzahlen je Szenario

Einsatzbeispiele

- Simulation einzelner Testszenarien, z.B. für Durchsatztests
- Vergleiche bei Tuning-Maßnahmen
- Fehlersuche



Testplanung – Lastmodelle (2)

Lastmodell: Vorgegebene Ankunftsrate

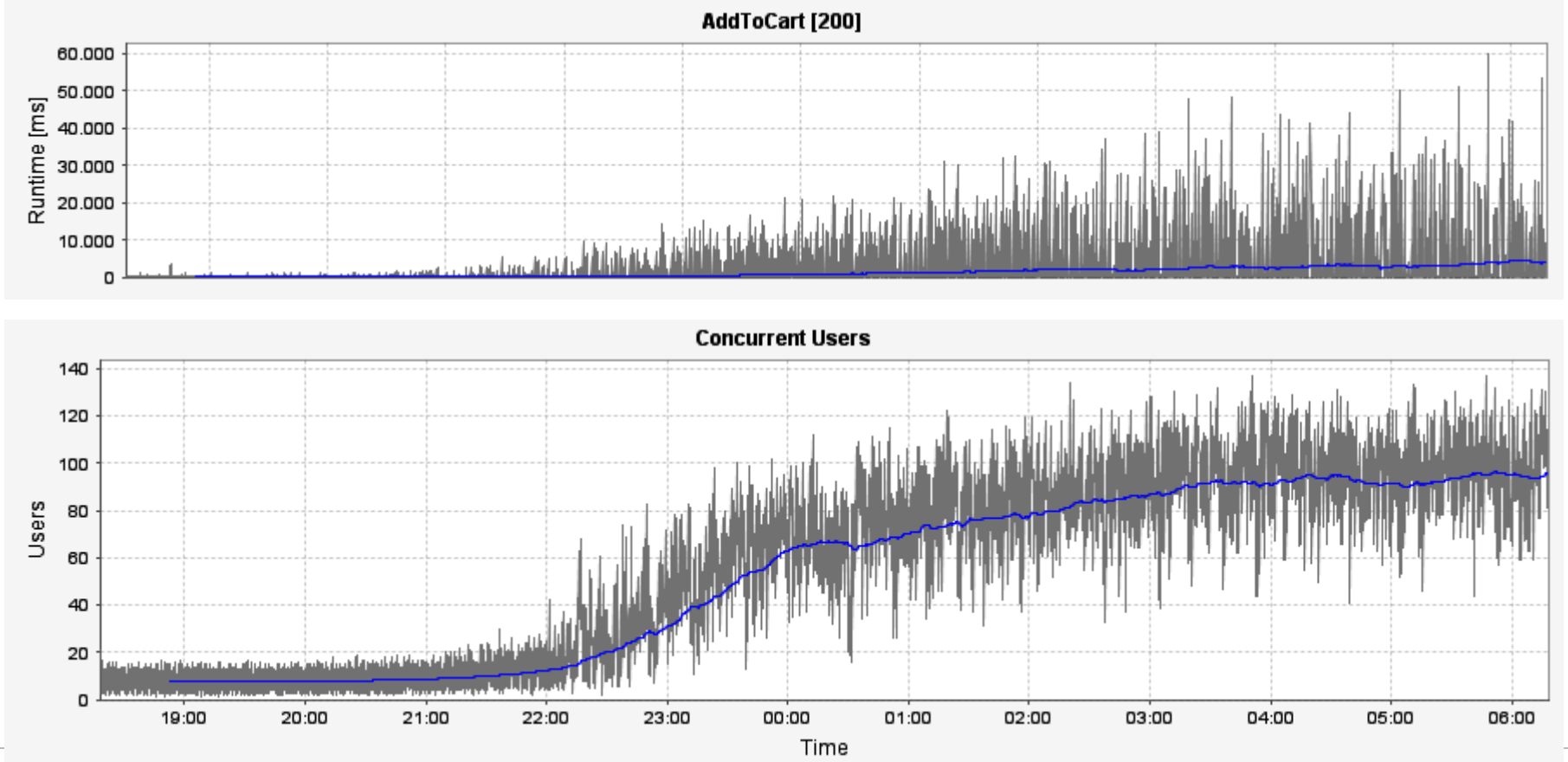
- die Anzahl der pro Zeiteinheit eintreffenden neuen Nutzer ist festgelegt
- die Anzahl parallel ausgeführter Nutzer (paralleler Sessions) ergibt sich aus
  - dem Antwortverhalten des SUT
  - den Testszenarien und konfigurierten Denkzeiten zwischen den Requests
- die Anzahl paralleler Nutzer kann je nach Verhalten des SUT stetig steigen
  ➔ Maximum konfigurieren!
- oft mit einem Mix aus Nutzerszenarien und mit realen Denkzeiten angewendet
- realistischer, aber schwerer vergleichbar und interpretierbar

Einsatzbeispiele

- Test auf Erfüllung vorgegebener Performance-Ziele (Performance-Test)
- Dauerlasttest
- Fail-Over-Test

Testplanung – Lastmodelle (3)

Beispiel: Anzahl paralleler Nutzer bei vorgegebener Ankunftsrate



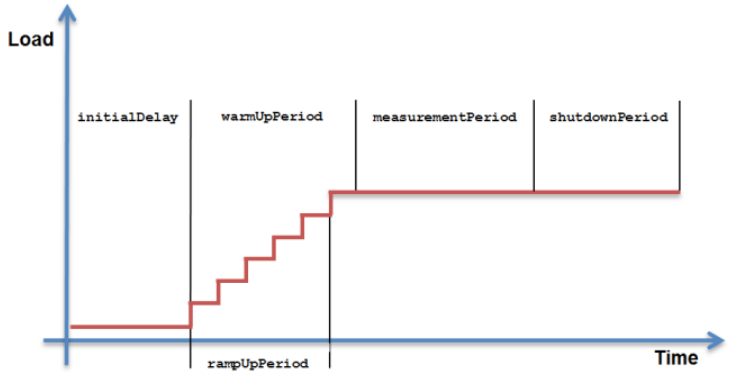
Testplanung – Lastprofile (1)

Konstante Last

- Nutzerzahl oder Ankunftsrate bleiben über Testlaufzeit gleich
- leicht zu konfigurieren, übersichtlich, einfacher auszuwerten

Steigende Last (Ramp-Up)

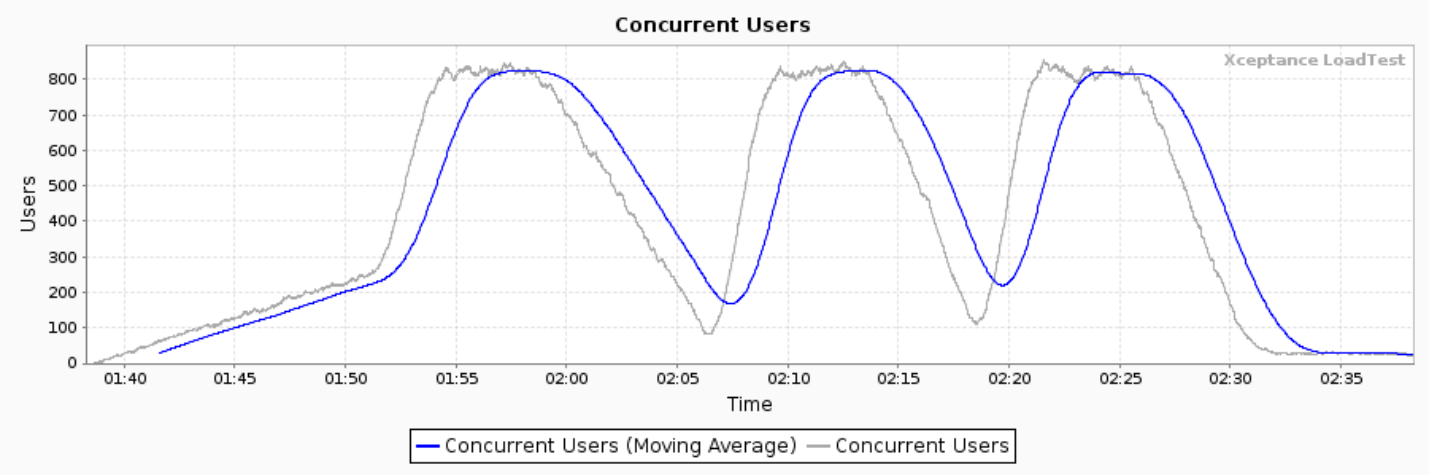
- Nutzerzahl oder Ankunftsrate steigen schrittweise auf einen Maximalwert
  - kontinuierlich in kleinen Schritten (Gerade) oder
  - in größeren Schritten mit längeren Plateaus (Treppe) ➔ besser auswertbar
- Anstieg langsam genug, damit sich SUT jeweils auf neue Last einschwingen kann
- Einsatzbeispiele
  - sanftes Anlaufen bei sehr hoher Last
  - Stresstests, Ermittlung von Grenzen
  - Sizing-Tests, Skalierbarkeitstests



Testplanung – Lastprofile (2)

Freie Profile

- Nutzerzahl oder Ankunftsrate frei gestaltbar
- komplexere Konfiguration und Auswertung
- für spezielle Zwecke sehr hilfreich
- Einsatzbeispiele
  - Tagesprofile
  - Flash-Sale-Szenarien



Testvorbereitung – Testdesign (1)

Auf Robustheit gegenüber Testdaten und Code-Änderungen achten

- möglichst keine fest eingebauten/fest konfigurierten Testdaten nutzen
  ➔ vorhandene Daten auf Website durch Testskripte “entdecken”
- robuste Erkennung von Elementen unter Nutzung von IDs, CSS-Klassen, ...
- keine Endlosschleifen mit unsicherer Abbruchbedingung nutzen
  ➔ Nicht: “Suche in Kategorie, bis Produkt mit Bestand > 0 gefunden wird”
  ➔ Besser: definierten Abbruch einbauen, z.B. maximal drei Versuche
- bei im SUT veränderlichen Daten Robustheitsfunktionen implementieren, z.B. nicht mehr verfügbare Produkte vor Bestellung aus Warenkorb entfernen

Spezielle Testdaten über alle Nutzer/Last-Agenten hinweg verwalten

- Einmal-Ressourcen, z.B. Gutschein-Codes
- exklusive Ressourcen, z.B. Login-Daten

Testvorbereitung – Testdesign (2)

Realistische Testdaten

- ausreichende Menge und realitätsnahe Verteilung
- unterschiedliche Fälle berücksichtigen, zum Beispiel für Suchwörter
  - ohne Treffer
  - ein Treffer
  - viele Treffer
- Gültigkeitszeiträume beachten, z.B. bei Aktionen oder Gutscheinen

Zufall in Skripte programmieren

- genutzte Links ausreichend zufällig auswählen (Kategorien, Produkte, ...)
  ➔ breitere Testabdeckung, Vermeidung von Caching-Effekten
- Verteilung beachten (gut sichtbare Webseiten versus tiefe Webseiten, …)
- verschiedene Anzahlen von Produkten im Warenkorb
- verschiedene Namen, Adressen, ...

Ergebnisse validieren

Testvorbereitung – Testdesign (3)

Hochparallele Ausführung beachten

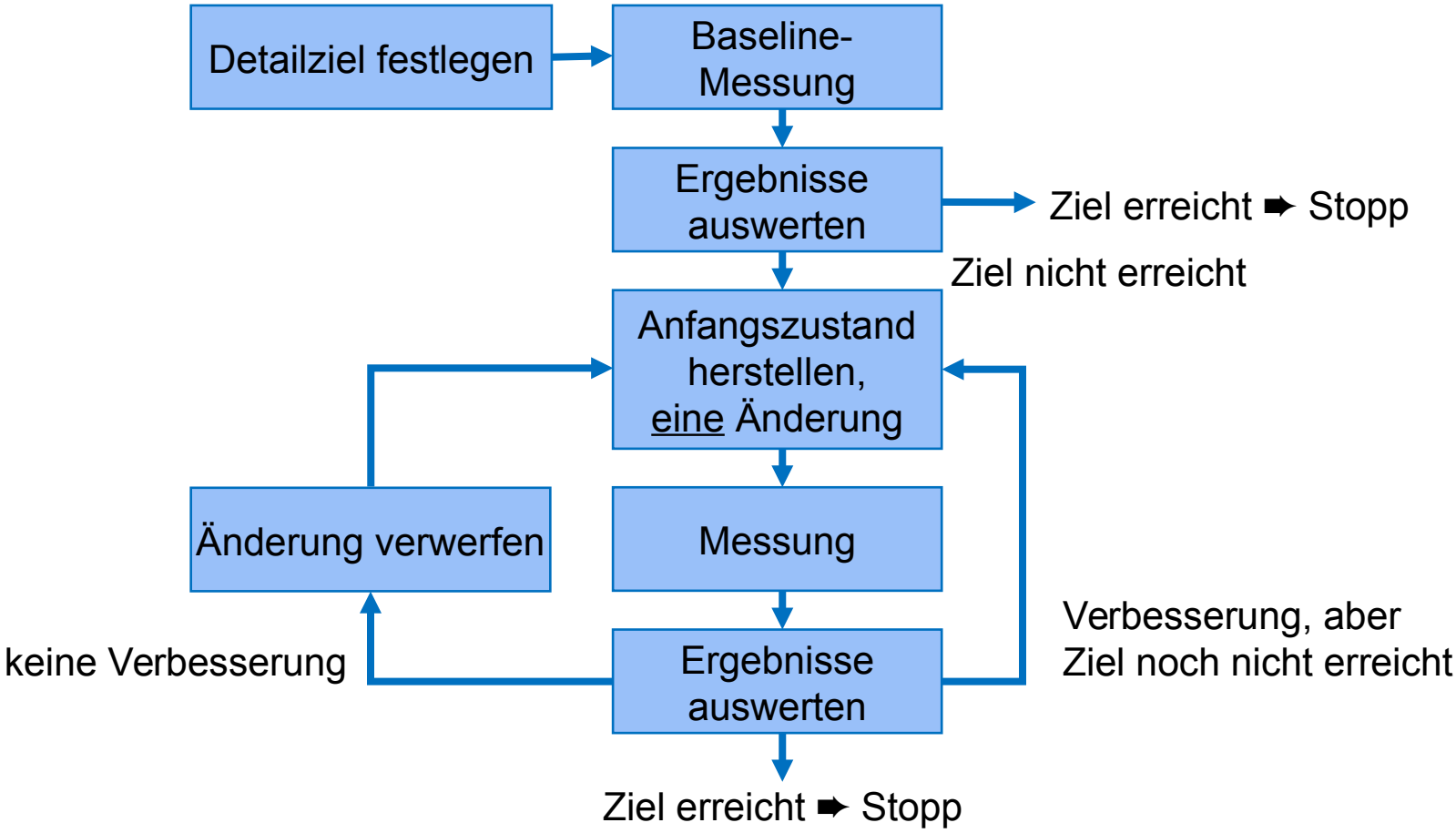
- vermeiden Sie Abhängigkeiten zwischen simulierten Nutzern
- vermeiden Sie globale Datenobjekte, soweit möglich

Kein eigenes Multi-Threading, das Testwerkzeug sorgt für parallele Ausführung!

Jede Operation benötigt Client-Performance (wichtig bei hohen Lasten)

- achten Sie auch bei der Entwicklung der Testfälle auf Effizienz
- entscheiden Sie früh, welcher Testmodus im Lasttestwerkzeug genutzt wird
  - mit DOM-Aufbau oder auf reiner HTTP-Protokollebene
  - mit oder ohne Ausführung von JavaScript
- Validierung soweit nötig, aber nicht zu extensiv

Testdurchführung - Iterationen



Testdurchführung – Testläufe (1)

1. Manueller Test der geplanten Szenarien

- grundsätzliche Funktion sicherstellen
- was mit einem Nutzer lange dauert, wird unter paralleler Last nicht schneller

2. Frühe Probeläufe

- Proof-Of-Concept für das Scripting
- eventuell Wechsel des Scripting-Modus (DOM/HTTP, mit/ohne JavaScript)
- Finden lastabhängiger Fehler, Erkennen von Abhängigkeiten
- minimales Testsystem ausreichend, eventuell zunächst ohne Drittsysteme
- Zeit für Fehlerbehebung und Tuning einplanen!

3. Probeläufe auf dem zu testenden System

- Konfiguration des Testsystems prüfen und optimieren
- Testläufe mit Drittsystemen

Testdurchführung – Testläufe (2)

4. Performance-Test mit der geforderten Last

- Prüfung auf Einhaltung der Anforderungen mit realem Nutzermix
- eventuell mit konstanter Ankunftsrate

5. Ermitteln der Systemgrenzen

- Test mit steigender Last

6. Langzeittest, zum Beispiel 24h

- Prüfung von Stabilität und Ressourcenverhalten

7. Eventuell Test eines Tages-Profils mit Lastspitzen

- Beispiel: Anmeldung aller Mitarbeiter am System am Morgen, Flashsale
- Gesamtverhalten mit Einbeziehung aller Prozesse (nächtliches Backup, …)

Testauswertung und Testreports

Basis

- Rohdaten und Reports des Lasttest-Werkzeugs
- Logdateien der getesteten Systeme, z.B. Webserver, Applikationsserver
- Monitoring-Daten des System- und End-Zu-End-Monitorings

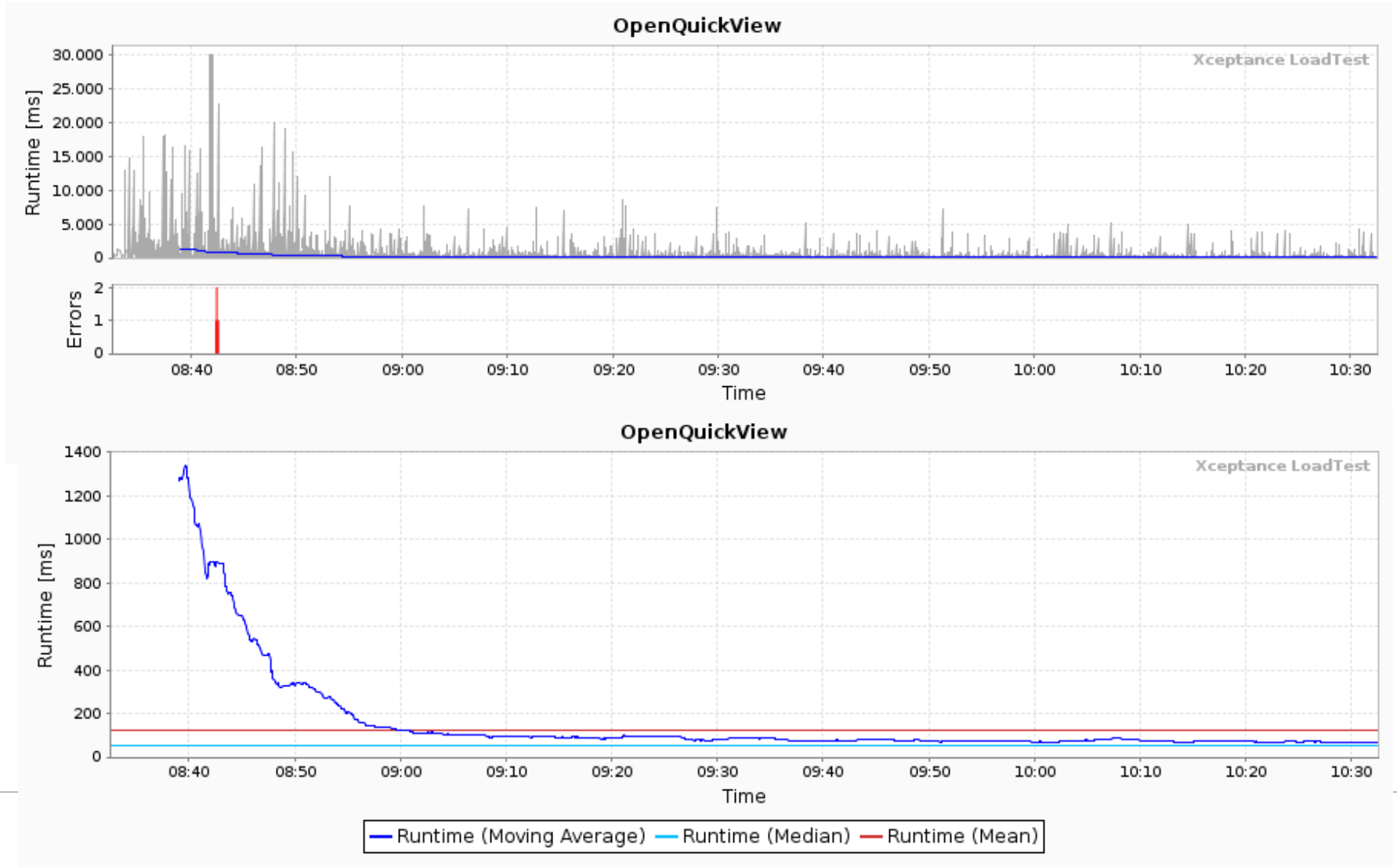
Auswertungen

- quantitative Auswertung des Lastverhaltens (Mittelwerte, Verteilung, Maxima)
- Korrelation von Server- und Clientdaten
- Hinweise auf mögliche Bottlenecks
- Dokumentation von durchgeführten Fixing- und Tuning-Maßnahmen
- Auswertung aufgetretener Fehler und Probleme
- Bewertung der Ergebnisse

Agenda

- Was ist ein Lasttest?
- Simulation von Web-Last
- Vorgehensweise bei Lasttests
- Interpretation der Ergebnisse - Patterns

Patterns (1) – Warmup



Patterns (2) – Warmup

Merkmale

- zu Beginn stark erhöhte Antwortzeiten, eventuell auch Fehler
- Antwortzeiten relativ schnell sinkend, dann Übergang in gleichmäßige Phase
- gut sichtbar am gleitenden Durchschnitt
- anderer Indikator: durchschnittliche Antwortzeit sinkt bei langlaufenden Tests

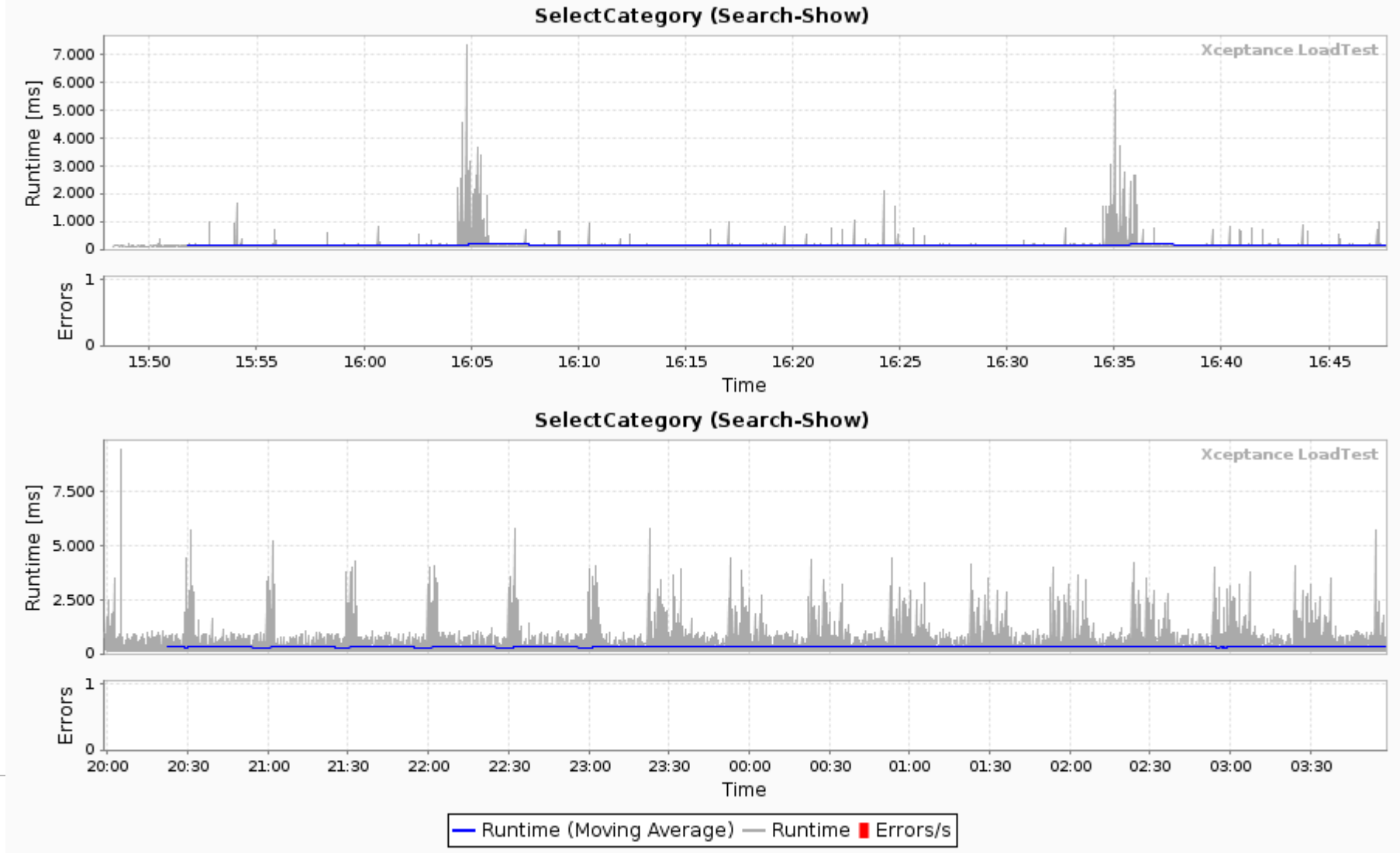
Ursachen

- Initialisierung von Prozessen, Laden von Code und Daten
- Aufbau von Verbindungen (Application Server - Datenbankserver, …)
- Vorkompilieren von Code, Templates, …
- erstes Füllen leerer Caches (DB, ORM-Layer, Dateisystem, …)

➔ Ob das Verhalten akzeptabel ist, hängt vom Kontext ab.

➔ Wenn akzeptabel, Messwerte oft um Warmup-Time bereinigt.

Patterns (3) – Caching



Patterns (4) – Caching

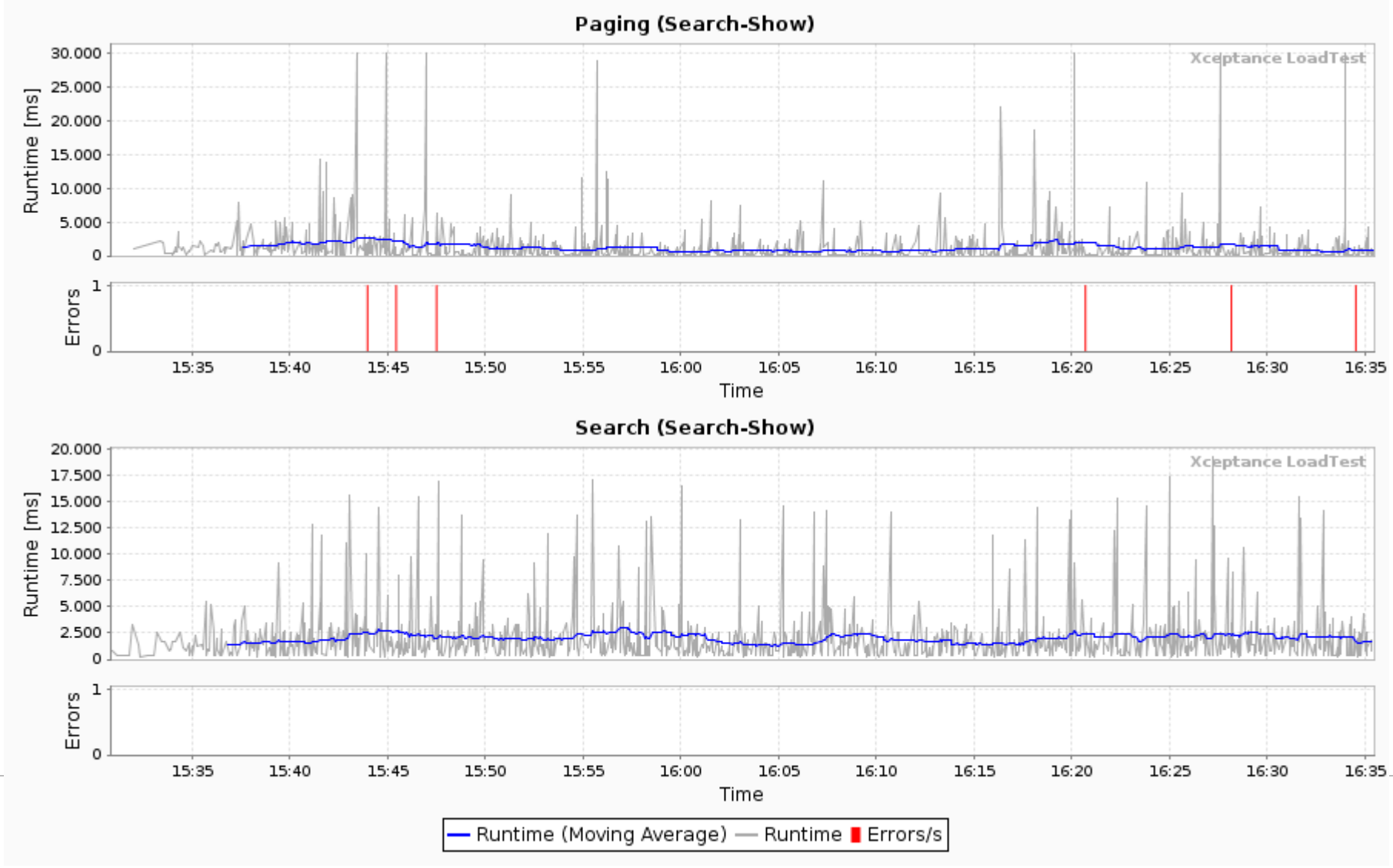
Merkmale

- Ablauf der Caching-Dauer einzelner Objekte oder Invalidierung eines gesamten Caches
- längere Testläufe durchführen, um Regelmäßigkeit besser zu erfassen
- wenn nicht regelmäßig, eventuell andere Ursachen

Hinweise

- Grad der regelmäßigen Verschlechterung kann akzeptabel sein oder nicht
- explizites Leeren von Caches während der Tests einplanen und durchführen

Patterns (5) – Spiking



Patterns (6) – Spiking

Merkmale

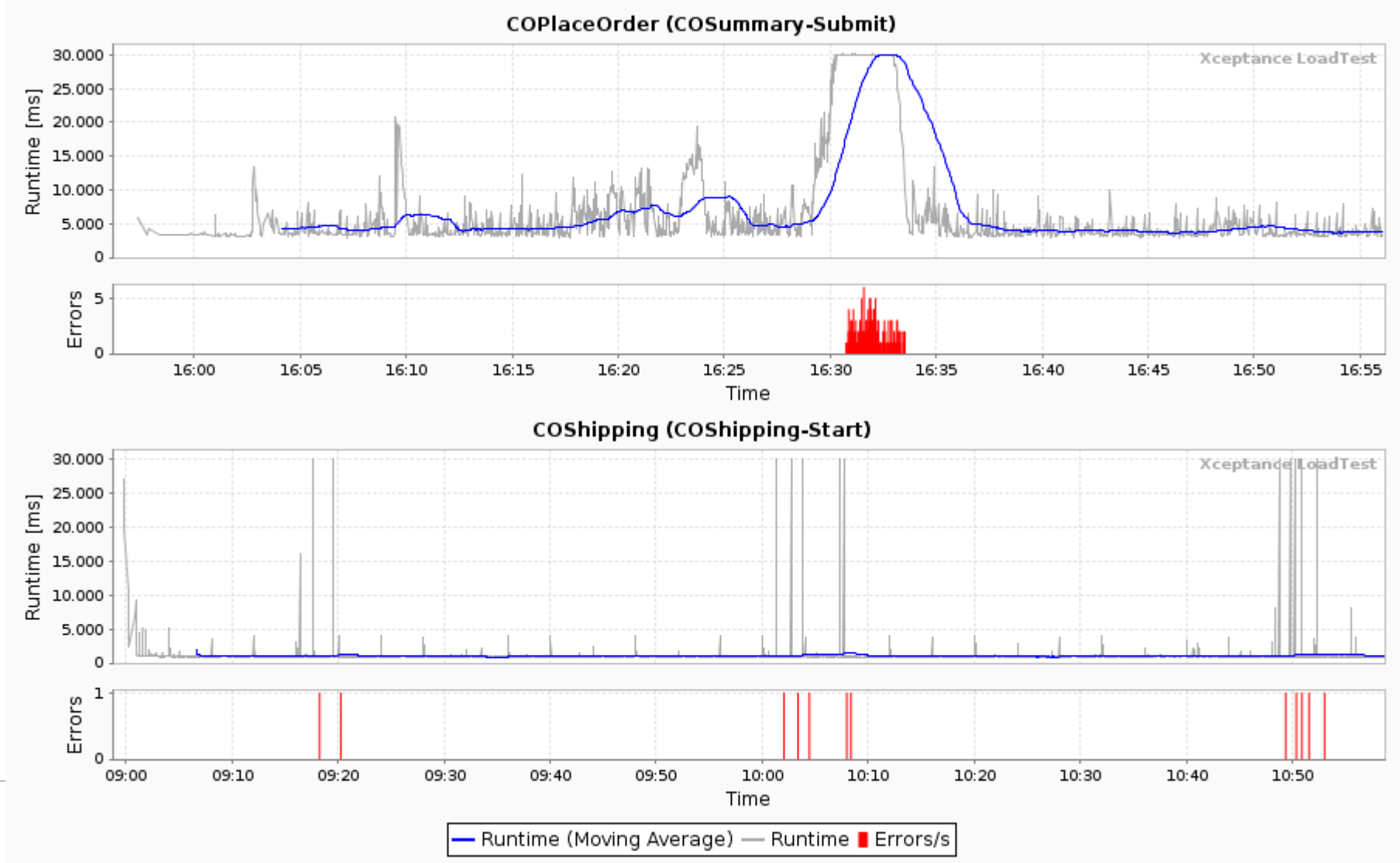
- Spitzen in den Antwortzeiten, die keinem erkennbaren Muster folgen
- mehrere gleiche, lange Antwortzeiten deuten auf Timeouts hin (hier: 30s)
- eventuell durch bestimmte URLs oder Datenzugriffe ausgelöst

Hinweise

- Details in der Tiefe betrachten
- Gemeinsamkeiten suchen
  - Gleiche URLs?
  - Gleiche Suchanfragen, Kataloge, Produkte, …?
  - Gleiche Operationen?
- Server-Logs usw. prüfen

XCEPTANCE
Software Technologies

Patterns (7) – Unregelmäßige Effekte

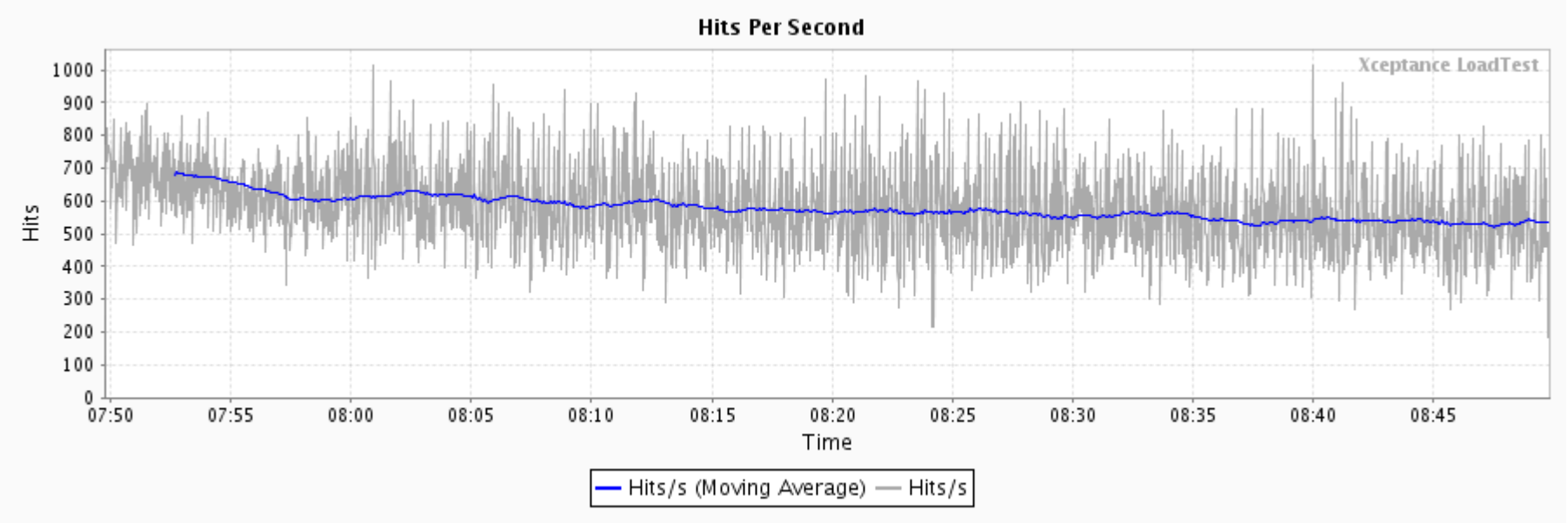


Patterns (8) – Unregelmäßige Effekte

Hinweise

- Reproduzierbarkeit prüfen, oft durch längere Testläufe
- menschliche Aktivitäten und andere Störungen ausschließen
- mehrere gleiche, lange Antwortzeiten deuten auf Timeouts hin (30s)
- Logs aller Komponenten des SUT prüfen
- oft durch Drittsysteme, die vom Server aus angesprochen werden
- zweites Diagramm ähnelt Caching-Pattern, aber hier handelt es sich um eine dynamische, nicht cache-bare Seite

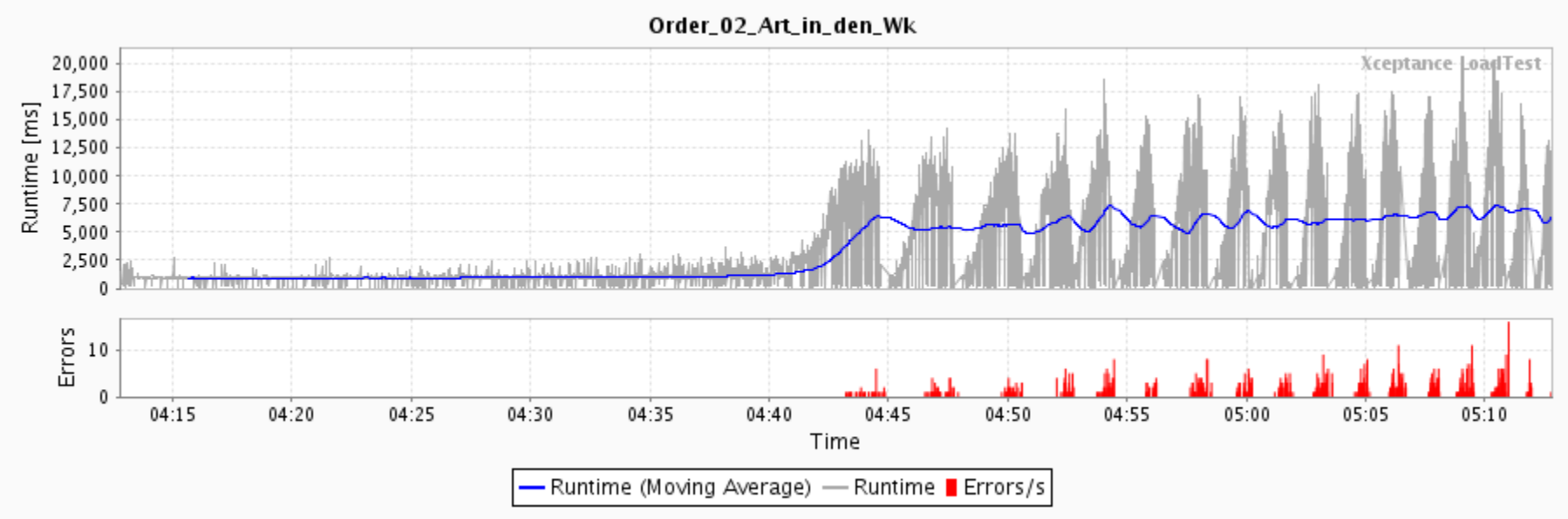
Patterns (9) – Sinkende Leistung



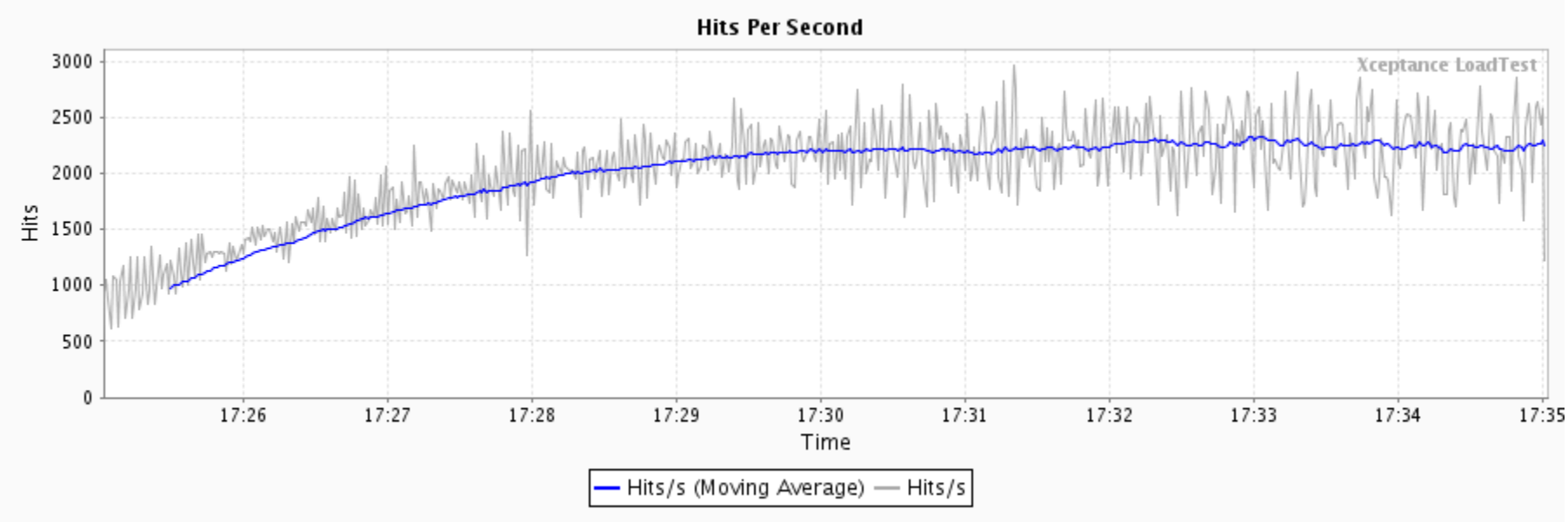
Patterns (10) – Sinkende Leistung

Hinweise

- eventuell durch steigenden Ressourcenverbrauch verursacht („Leaks“: Speicher, Threads, Prozesse, …)
- eventuell durch steigende Datenmengen in DB
- eventuell „Aufschaukeln“ bis in einen instabilen Systemzustand, siehe unten



Patterns (11) – Test mit steigender Last

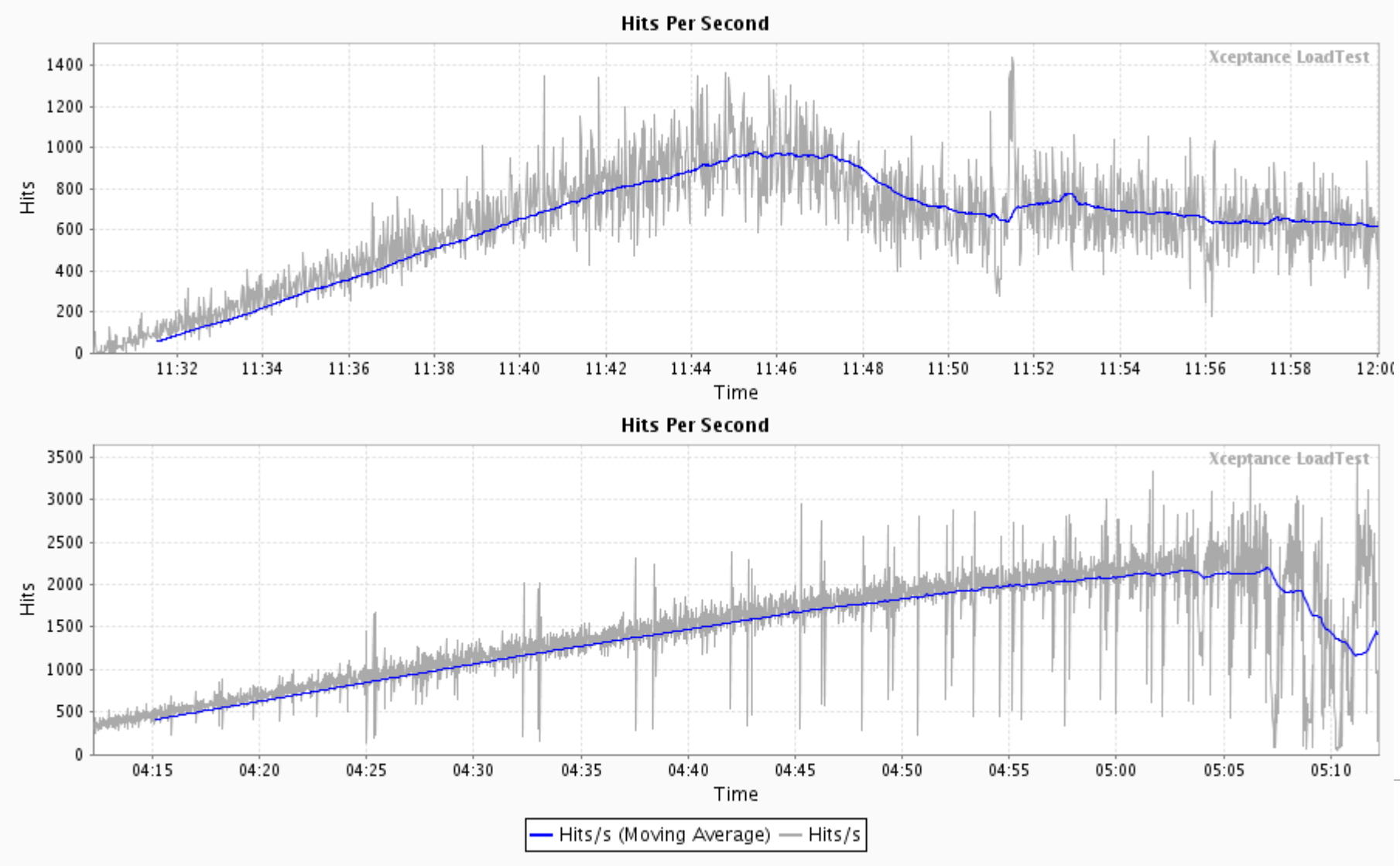


➔ **Sättigung, aber stabiles Verhalten**

- Durchsatz steigt nicht mehr, aber sinkt auch nicht wieder; robustes System
- Antwortzeiten gegenüberstellen

➔ **Ab einer bestimmten Last Kollaps, siehe nächstes Slide**

Patterns (12) – Test mit steigender Last



Patterns (13)

Zusammenfassung

- client-seitige Testergebnisse allein zeigen keine eindeutigen Ursachen
- sie sind aber wichtiger Ausgangspunkt für qualifizierte Untersuchungen
- absolute Werte beachten - eventuell sind alle Antwortzeiten akzeptabel
- längere Testlaufzeiten machen Muster sichtbar
- bei langlaufenden oder fehlerhaften Requests Gemeinsamkeiten suchen
- beachten, ob betroffene Requests cache-bar/nicht cache-bar sind
- Vorsicht bei Mittelwerten, sie verstecken Informationen
- Test mit Einzelszenarien zur Eingrenzung von Problemen
- als mögliche Ursache immer auch prüfen:
  - Client-Seite (HW/SW-Umgebung, Netzwerk, …)
  - Testskripte, Last-Agenten, Testdaten
- in jedem Einzelfall sind genauere Untersuchungen nötig

Vielen Dank für Ihre Aufmerksamkeit!

Haben Sie Fragen?

Kontakt

Xceptance Software Technologies GmbH
Leutragraben 2-4
07743 Jena
Germany

Telefon: +49 3641 55944 0
Telefax: +49 3641 376122

E-Mail: kontakt@xceptance.de

Web: http://www.xceptance.de
http://www.xceptance.de/produkte/xlt/was-ist-xlt.html

Copyrights