Lernzettel

Nicht-funktionale Eigenschaften und Qualitätsbewertung: Skalierbarkeit, Verfügbarkeit, Sicherheit, Konsistenz, Zuverlässigkeit, Wartbarkeit, Kosten

Universität: Technische Universität Berlin
Kurs/Modul: Architektur von Anwendungssystemen
Erstellungsdatum: September 19, 2025

Lernzettel: Nicht-funktionale Eigenschaften und Qualitätsbewertung: Skalierbarkeit, Verfügbarkeit, Sicherheit, Konsistenz, Zuverlässigkeit, Wartbarkeit, Kosten

(1) Einordnung. Nicht-funktionale Eigenschaften beschreiben, wie gut eine Anwendung arbeitet, unabhängig von der eigentlichen Funktionalität. In verteilten Architekturen beeinflussen sie Entwurf, Technologieauswahl und Betrieb. Ziel ist eine systematische Bewertung und Optimierung dieser Eigenschaften im Gesamtsystem.

(2) Skalierbarkeit. Definition: Die Fähigkeit eines Systems, steigende Lasten ohne signifikanten Leistungsabfall zu bewältigen.

- Typen
  - Horizontal skalieren (Scale out): mehr Instanzen, Lastverteilung.
  - Vertikal skalieren (Scale up): stärkere Ressourcen pro Instanz.
  - Elastische Skalierung: automatische Anpassung an Last.
- Maßnahmen
  - Lastverteilung (Load Balancer)
  - Statelessness von Diensten
  - Caching und Content Delivery Networks
  - Partitionierung/Sharding
- Metriken
  - Throughput T (Requests/s)
  - Latenz L (z.B. p95)
  - Skalierbarkeitsreserve
  - Kosten pro Lasteinheit
- Designprinzipien
  - Lose Kopplung, klare Schnittstellen
  - Statelessness, idempotente Operationen
  - Datenpartitionierung, konsistente Caches

(3) Verfügbarkeit. Definition: Anteil der Zeit, in der das System funktionsfähig ist.

- Kennzahlen
  - MTTF (Mean Time To Failure)
  - MTTR (Mean Time To Repair)
  - Availability $A = \frac{MTTF}{MTTF + MTTR}$ (oft in neunen Neunen, z.B. 99.9
- Designstrategien
  - Redundanz (Multi-Region, Active/Passive, Active/Active)

  - Failover, Health Checks, Heartbeats
  - Quorum-S Mechanismen, Replikation
  - Graceful Degradation
  - Disaster-Recovery-Strategien

(4) Sicherheit. Schutzziele: Vertraulichkeit, Integrität, Verfügbarkeit.

- Maßnahmen
  - Authentifizierung, Autorisierung (RBAC, OAuth, JWT)
  - Verschlüsselung (TLS, At-Rest, In-Transit)
  - Geheimnisverwaltung (Secret-Management)
  - Audit Monitoring
  - Prinzip der geringsten Berechtigungen
  - Patch- und Patch-Management
- Bedrohungen und Best Practices
  - OWASP Top 10
  - Secure-by-Default, Sicherheits-Tests im SDLC

(5) Konsistenz. Verteilte Systeme und Konsistenzmodelle.

- CAP-Theorem: Consistency, Availability, Partition tolerance — nur zwei davon können in einem verteilten System gleichzeitig garantiert werden.
- Konsistenzmodelle
  - Strong Consistency
  - eventual Consistency
  - causal/monotone Reads
  - Read-Your-Writes, Monotonic Reads
- Anwendungen
  - Strong Consistency bei Transaktionen (Bankwesen)
  - Eventual Consistency bei Social- oder Content-Anwendungen
- Mechanismen
  - Verteilte Transaktionen, Konfliktlösung
  - Idempotente Operationen, Vector Clocks

(6) Zuverlässigkeit. Zuverlässigkeit als Fähigkeit, erwartungsgemäß zu funktionieren, auch bei Fehlern.

- Techniken
  - Redundanz, Wiederholungslogik, Retries mit Backoff
  - Idempotente API-Aufrufe
  - Circuit Breaker Muster
  - Fehlertolerante Designprinzipien
  - Recovery-Strategien (Checkpoints, Snapshots)
  - Chaos Engineering zur Resilienzprüfung

(7) Wartbarkeit. Wartbarkeit und Änderbarkeit des Systems.

- Faktoren
  - Modulare Architektur, klare Schnittstellen
  - Lesbarer Code, umfangreiche Tests
  - Logging, Monitoring, Observability
  - Automatisierte Builds, CI/CD, Blue/Green Deployments
  - Dokumentation und Conventionen
- Metriken
  - MTTR für Defekte, Change-Failure-Rate
  - Testabdeckung, Code-Komplexität

(8) Kosten. Kosten-Aspekte der Qualitätsbewertung.

- Kostenarten
  - CAPEX (Investitionen in Hardware/Software)
  - OPEX (Betriebskosten, Lizenzgebühren, Support)
  - Betriebskosten durch Skalierung (Hosting, Netzwerk, Speicher)
- Kostentreiber
  - Anzahl Instanzen, Speicherkapazität, Datenverkehr
  - Verfügbarkeit/downtime, Failover-Überdimensionierung
- Optimierung
  - Right-sizing, Autoscaling, Reserve-/Spot-Modelle
  - Caching, CDNs, effiziente Algorithmen
- Vereinfachtes Kostenmodell

$$C_{\text{total}} \approx \sum_r price_r \cdot usage_r + fixed\_costs$$

  (Schätzung je nach Architektur)

(9) Beurteilung und Praxis. Vorgehen zur Qualitätsbewertung verteil ter Architekturen:

- Anforderungen ableiten und relevante Attribute identifizieren
- Metriken ableiten (z.B. Verfügbarkeit, Latenz, Kosten)
- Architekturen gegen diese Metriken bewerten (Skalierungsszenarien, Redundanz)
- Tests, Simulationen und Chaos-Experimenten durchführen

Beispiel-Skizze Kurze Skizze einer verteilten Architektur mit Microservices, Statelessness, reinforced durch Caching, Logging/Monitoring und mehrregionale Replikation. Dieses Beispiel illustriert die Balance zwischen Skalierbarkeit, Verfügbarkeit, Sicherheit, Kosten und Wartbarkeit.