Verteilte Systeme

VS Praktikum SoSe 2025

Manh-An David Dao, Philipp Patt, Jannik Schön, Marc Siekmann

4. Mai 2025

Inhaltsverzeichnis

1 Einführung und Ziele

1.1 Aufgabenstellung und Anforderungen

Es soll ein verteiltes Steuerungssystem gemäß den Prinzipien verteilter Systeme nach Tanenbaum & van Steen entworfen und implementiert werden. Über ein ITS-Board (bspw. STM32F4) soll ein autonomer Roboter innerhalb eines kontrollierten Areals (z.B. BT7 R7.65 – als realer Testbereich) sicher und effizient gesteuert werden. Die verteilte Architektur soll dabei explizit gewährleisten, dass durch Fehlverhalten der Software oder Architektur keine Gefahr für anwesende entstehen kann.

Funktionale Beschreibung

- **Betreiber - System**
  - Nutzer wählt genau einen Roboterarm aus
  - Nutzer gibt Steuerbefehl an genau einen Roboterarm
  - System zeigt an, ob Roboterarm verfügbar
  - Notstop löst sofortigen Stopp des ausgewählten Roboters aus
- **System**
  - Roboterarme stoppen, sobald nicht mehr von aussen kontrollierbar
  - automatisches Erkennen von Hinzufügen und Entfernen von Roboterarmen

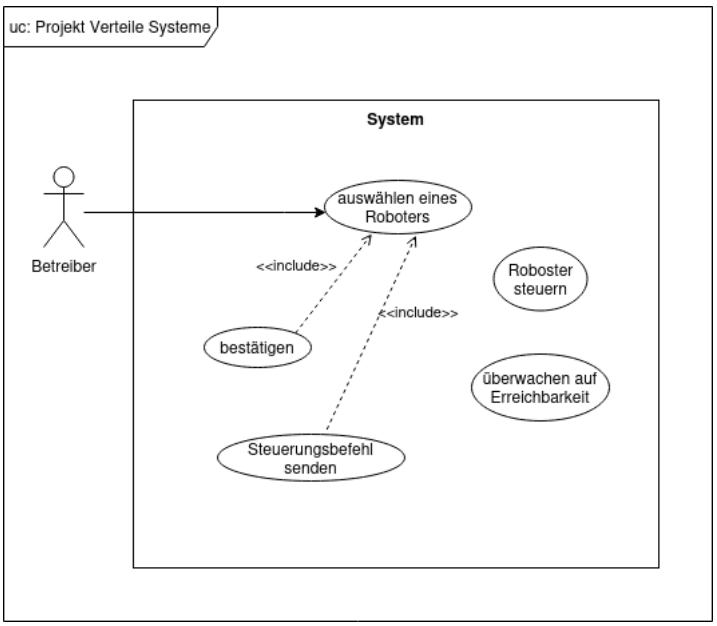

Abbildung 1.1: Funktionales Use Case Diagramm der Aufgabenstellung

1.2 Qualitätsziele

Das verteilte Steuerungssystem soll eine Reihe von nicht-funktionalen Anforderungen erfüllen, um einen sicheren, wartbaren und erweiterbaren Betrieb im Einsatzumfeld zu gewährleisten.

1.2.1 Ziele für Software Engineering

Ziel	**Beschreibung**	**Metrik**
Funktionalität		Alle Abnahmetests werden erfolgreich bestanden
Zuverlässigkeit		Das System ist über dem gesamten Abnahmezeitraum stabil (ca. 1,5 h)
Skalierbarkeit	Zusätzliche Roboter oder Komponenten sollen ohne Änderungen an der bestehenden Architektur integrierbar sein.	Es können beliebig viele Roboter hinzugefügt und entfernt werden (0 - N)
Leistung		Reaktionszeit max. 250 ms
Sicherheit (Safety)	Es bewegt sich immer genau ein Roboterarm. Sollte das System nicht wie gewünscht reagieren, wird ein sicherer Zustand erreicht	Reaktionszeit max. 250 ms, bis Roboterarm stoppt
Benutzerfreundlichkeit		Keine Einweisung erforderlich
Anpassbarkeit		
Kompatibilität		

Tabelle 1.1: Qualitätsziele der Software Engineering

1.2.2 Ziele der Verteilte Systeme

Ziel	**Beschreibung**	**Metrik**
Ressourcenteilung	...	
Offenheit	...	
Skalierbarkeit	...	siehe Tabelle 10.3
Verteilung Transparenz	...	siehe Tabelle 10.4

Tabelle 1.2: Qualitätsziele der Verteilten Systeme

Skalierbarkeit

Ziel	**Metrik**	**Metrik**
Vertikale Skalierung	...	
Horizontale Skalierung	...	
Räumliche Skalierbarkeit		1
Funktionale Skalierbarkeit	...	
Administrative-Skalierbarkeit		1

Tabelle 1.3: Skalierbarkeit von verteilten Systemen

Verteilungs-Transparenzen

Ziel	**Beschreibung**	**Metrik**
Zugriffstransparenz	...	
Lokalitäts-Transparenz	...	
Migrationstransparenz	...	
Replikationstransparenz	...	
Fehlertransparenz	...	
Ortstransparenz	..	
Skalierbarkeits-Transparenz	...	

Tabelle 1.4: Verteilungs-Transparenzen

1.3 Stakeholder

Die folgenden Gruppen sind direkt oder indirekt vom System betroffen:

- **Endnutzer:** Personen, die mit dem Roboter interagieren (z.B. Lehrpersonal, Kinder im Testbereich)
- **Entwicklerteam:** Zuständig für Entwurf, Umsetzung, Tests und Wartung des Systems
- **Betreiber:** Verantwortlich für die Überwachung und den sicheren Betrieb des Systems im realen Umfeld
- **Professor:** Person, die für die Bewertung des Systems verantwortlich ist und Rahmenbedingungen vorgibt.

2 Randbedingungen

2.1 Technische Randbedingungen

Das verteilte Steuerungssystem unterliegt mehreren festgelegten technischen Rahmenbedingungen, die den Entwicklungs- und Implementierungsspielraum einschränken. Diese Bedingungen sind im Folgenden aufgeführt:

- **Hardwareplattform:** Das System basiert auf ein Raspberry Pi der den jeweiligen Roboterarm steuert. Eine Software auf dem Raspberry Pi stellt eine API zur Verfügung, sodass die Roboter sich ohne Einschränkungen bewegen können. Ein ITS-BRD dient als Steuerung/Administration der Roboterarme.
- **Programmiersprachen:**
  - **C:** Das ITS-Board wird mit C Programmiert.
  - **Java:** Die Roboterarme werden mit einem Javaprogramm angesteuert.
- **Netzwerkumgebung:** Es steht ein Adressenraum von 255 Adressen zur verfügung. Daraus folgt, dass nicht mehr als 254 Roboterarme eingesetzt werden können. Das ITS-Board ist ebenfalls ein Teilnehmer des Netzes.

2.2 Organisatorische Randbedingungen

- **Umgebung:** Der Abnahme bereich befindet sich im Raum BT7 R7.65. Die Steuerung und Navigation des Roboters müssen innerhalb der räumlich definierten Grenzen erfolgen. Die dort vorhandene LAN-Infrastruktur kann zur Netzwerkkommunikation verwendet werden.
- **Zeit:** Entwicklungszeitraum beträgt 12 Wochen.
- **Vorwissen:** Einige Konzepte und Herangehensweisen werden erst im Laufe der 12 Wochen gelernt.
- **Budget:** Es steht kein Budget zur Verfügung.

3 Kontextabgrenzung

Ziel dieses Kapitels ist es, das zu entwickelnde System innerhalb seines fachlichen und technischen Umfelds klar einzugrenzen. Dazu wird das System in Bezug auf seine Aufgaben (fachlicher Kontext), seine Einbettung in die bestehende technische Infrastruktur (technischer Kontext) sowie die definierten externen Schnittstellen beschrieben.

3.1 Fachlicher Kontext

Das verteilte System soll es ermöglichen beliebig viele (1 - 254) Roboterarme in einem Raum zu steuern. Die Kommunikation zum Nutzer und die Steuerung wird durch ein ITS-Board realisiert.

3.2 Technischer Kontext

Die Realisierung wird in einem /24 Netzwerk durchgeführt. In diesem wird das ITS-Board liegen. Ebenfalls sind in den Netzwerk die Roboterarme mit den vorgeschalteten Raspberry Pi, der jeweils einen Roboterarm ansteuert. Weitere Hilfmittel wie eine ICC-Cloud können ebenfalls genutzt werden.

3.3 Externe Schnittstellen

- System: Der Benutzer kann über ein Bildschirm einzelne erreichbare Roboterarme erkennen. Die Auswahl und Steuern der einzelnen Roboterarme wird durch IO-Buttons realisiert.
- ITS-Board: Das ITS-Board kommuniziert über einen TCP-Server mit einer eigenen IP-Adresse mit dem System.
- Roboterarm: Der Roboterarm wird mit einer IP-Adresse angesprochen. Dafür ist der vorgeschaltete Raspberry Pi mit einem eigenem TCP-Server zuständig. Dieser stellt ebenfalls eine API für die Steuerung zur Verfügung.
- ICC Cloud: Die ICC Cloud kommuniziert über eine IP-Adresse.

4 Lösungsstrategie

Grundlage ist ein bi-direktionales Client-Server-Modell, das auf die Anforderungen (KAPITEL) zugeschnitten ist.

- Client: Dieser besteht aus einer GUI-Software auf dem ITS-Board, das über ein integriertes Display eine grafische Benutzeroberfläche (GUI) zur Verfügung stellt. Die Interaktion erfolgt über Buttons, die Steuerbefehle auslösen. Das ITS-Board kommuniziert mit der Roboter-Software, die auf dem Raspberry Pi des zu steuernden Roboterarms, liegt.
- Server: Die Roboter-Software liegt auf dem Raspberry Pi der Roboterarme. Sie verwaltet den Verbindungszustand, koordiniert die einkommenden Steuerbefehle und leitet/-übersetzt Anfragen gezielt an den Roboterarm weiter. Zusätzlich fungiert sie als Watchdog: Sie überwacht eine aktive Verbindungen (zur GUI-Software) und sorgt im Fehlerfall (z.B. Verbindungsabbruch) für einen sicheren Stopp der Roboteraktivität. Die Roboter-Software führt einen Registrierung durch mit einem eindeutigen Bezeichner (Name oder ID) , und einem status beim ITS-Board durch Der Server umfasst auch die Roboter selbst sowie deren lokale Steuereinheit und Sensorik. Jeder Roboter betreibt eine eigene API zur Steuerung und stellt Statusinformationen bereit.

Die Kommunikation zwischen dem Client und Servern erfolgt über TCP/IP. Die Wahl von TCP garantiert eine zuverlässige, geordnete und fehlerfreie Datenübertragung. Um jedoch Verbindungsabbrüche rechtzeitig zu erkennen, wird zusätzlich ein Heartbeat-Mechanismus (Watchdog) eingesetzt, der regelmäßig geprüft wird (siehe Metriken, Kapitel). Jede Roboter-Software muss sich nach dem Systemstart aktiv bei dem ITS-Board registrieren. Die GUI-Software verwaltet eine zentrale Teilnehmerliste in Form einer Tabelle mit Tupeln der Form `(Name, IP-Adresse, Status)`. Diese wird zur Adressierung und gezielten Kommunikation verwendet.

In der GUI werden nur die erreichbaren Roboterarme aufgeführt. So kann sich der Benutzer einen der Roboterarme aussuchen zu diesem eine TCP Verbindung aufbauen und diesen anschließend per Buttons steuern. Durch den Abbruch der vorherigen Verbindung werden evtl. vorherige bewegte Roboterarme gestoppt. Dadurch ist gesichert, dass immer nur ein Roboterarm gesteuert wird. Sollte ein Roboterarm ausfallen bzw. vom Netz getrennt werden, ist dieser Roboterarm mit dem status öffline"gekennzeichnet. Alle Bewegungen dieses Roboters werden sofort beendet. Ebenfalls wird das ITS-Board mit einem Not-Aus Button ausgestattet, sodass der ausgewählte Roboter sofort jede Bewegung abbricht. Die Ansteuerung des Roboters erfolgt mit dem RPC-Protokoll. Dadurch wird eine Funktion auf der Roboter-Software aufgerufen, anschliessend übersetzt und die Bewegung am Roboterarm durchgeführt. Das Statusupdate des jeweiligen Roboterarms kann dann per RPC von der Roboter-Software zum ITS-Board erfolgen.

Überprüfung der Lösungsstrategie

Ziele für Software Engineering

- **Client-Server**: Es existiert genau ein Client und mehrere Server, die einzeln angesprochen werden können. Die Server haben alle die gleiche API und Funktionaliät (SE-Ziele Funktionalität, Skalierbarkeit, Wartbarkeit, Anpassbarkeit, Kompatibilität)
- **Watchdog**: Roboter-Software und Roboter überwachen gegenseitig den Verbindungsstatus, um bei Abbruch in einen sicheren Zustand zu wechseln. (SE-Ziel Safety)
- **Kommunikationsprotokoll:** TCP/IP wurde gewählt, um eine zuverlässige, verbindungsorientierte Kommunikation zu gewährleisten. RPC wurde gewählt, da so eine funktionale Umsetzung konsequent durchgeführt wird.
- **Adressierung:** Die Teilnehmer befinden sich im selben /24-Netzwerk und kommunizieren über IPv4. Jeder Roboter benötigt eine eindeutige ID oder Namen.

Ziele für Verteilte Systeme

- **Ressourcenteilung:** Alle Roboter können einheitlich über eine GUI-Software des ITS-Boards gemeinsam verwendet werden.
- **Offenheit:** Neue Roboter können sich dynamisch am ITS-Board registrieren. Das System bleibt erweiterbar ohne strukturelle Änderungen.
- **Skalierbarkeit:** Das Netzwerk begrenzt die Anzahl der Roboterarme. Diese registrieren sich mit aktualisierenden Status bei der GUI-Software des Clients
- **Verteilungstransparenzen:** Durch das Client-Server Modell und das MMI über die GUI werden die Verteilungstransparenzen teilweise eingehalten. Die Architektur versteckt die konkrete Lage und Ansteuerung der Roboter (Ortstransparenz, Zugriffstransparenz) vor dem Nutzer. Die GUI zeigt nur logische Namen an, nicht IP-Adressen oder spezifische Schnittstellen.

Schwächen

- Netzwerk kann durch UDP-Anfragen "lahmgelegt werden", Watchdog an Roboter-Software und am Roboterarm darf also nicht auf Antwort warten
- Ziel Leistung: Leistung des Systems durch ITS-Board Hardware abhängig. (Befehle pro Sekunde). Das ITS-Board ist aber sehr eingeschränkt in Speicher und Leistungsfähigkeit.
- Fehlertransparenz: GUI zeigt nicht verfügbaren Roboter nicht mehr an.
- Skalierbarkeitstransparenz: Neue Roboter könnten durch zu Leistungseinbrüchen des Systems durch ITS-Board führen
- Lokalitätstransparenz: Ist durch den eindeutigen Namen nicht gegeben und durch Hinzufügen bzw. Verschwinden auf der GUI nicht gegeben.

Alternative Überlegung:
Um das ITS-Board zu entlasten.

- Die Roboter melden sich bei einem Service an und diese verwaltet und überprüft die Verfügbarkeit der Roboter und des ITS-Boards. Die Liste der verfügbaren Roboter wird dem ITS-Board übermittelt. Dieser Service läuft auf der ICC Cloud, dafür wird auf einem Rechner im Netzwerk ein Proxy eingerichtet, der die Anfragen weiterleitet.
  - Vorteil: Rechenlast ausgelagert
  - Nachteil: Single Point of Failure durch Proxy
- Die Raspberry Pi tauschen sich untereinander aus, wer die Liste an das ITS-Board sendet. So ist das Gesamtsystem ausfallsicher. Die Raspberry Pi wissen alle voneinander.
  - Vorteil: Kein Single Point of Failure, Rechenlast auf Rapberry Pi, der schon im System vorhanden ist und somit keine neue Fehlerquelle.
  - Nachteil: Erhöhter Verwaltungsaufwand

5 Bausteinsicht

Welche Dekompistionsstrategie soll gewählt werden(funktional, technisch, Schichten)? Erstes Diagramm zeigt Gesamtystem, wobei System als Blackbox dargestellt wird und der Benutzer auf das System zugreifen kann. Zweites Diagramm zeigt innere Struktur von Blackboxen,

5.1 <insert overview diagram of overall system>

<describe motivation/reasoning for overall system decomposition>

<describe contained building blocks (blackboxes)>

<(optionally) describe important interfaces>

5.2 5.2.1 <insert white box template of building block 1> 5.2.2 ... 5.3 5.3.1 < insert white box template of building block x.1 >

Level 1

User auf System

System besteht aus ITS-BOARD, ... , Roboterarm

Level 2

ITS Board besteht aus GUI, IO, Heartbeat

Roboterarm besteht aus Steuerung, (Watchdog), Hardware-API

Level 3

Steuerung, Watchdog, Heartbeat, GUI, IO

6 Laufzeitsicht

Betrachtet das dynamische Verhalten des Systems waehrend der Ausfuehrung/Laufzeit. Wie die jeweiligen Bausteinen sich untereinander verhalten und in beziehung stehen.

6.1 Einleitung - Was?

Wie fuehren die jeweiligen Bausteine Ihre Funktion aus?
Wie interagieren Komponente miteinander
–> Schnittstellen? Wie ist der fliessende Datenverkehr der jeweiligen nachrichte
Sequenz Diagramm zur visualisierung (bsp. Waehlen des Roboterarms)
–> Wie sieht es aus, wenn ein Fehler geschieht
–> Wie sieht der Start des Systems aus

Tipp:: Ein Szenario waehlen, das vieles abdeckt!

Motivation? Sehr gut für Stakeholder gedacht - anschaulicher
Hilft zur Beurteilung das System
Hilft es Use-Case zu visualisieren/Realisierung
Hilft die Performance zu verbessern
Troubleshooter Problems (Analysieren und beheben)
Ueberwachung und Wartung des Systems

Form
Sequenzdiagramme zu verschiedenen Szenarien um abläufe zu verdeutlichen (Funktional)

7 Verteilungssicht

7.1 Infrastructure Level 1

< insert infrastructure overview diagram >

Motivation < insert description of motivation or explanation in text form>

(optional) Quality and/or Performance Features < optionally insert description quality or performance features >

Mapping < insert description of mapping of building blocks >

7.2 Infrastructure Level 2

7.2.1 < Infrastructure element 1> < insert diagram + explanation >

7.2.2 < Infrastructure element 1> < insert diagram + explanation >

... < insert diagram + explanation >

7.2.n < Infrastructure element 1> < insert diagram + explanation >

8 Konzepte

8.1 Offenheit

8.2 Verteilungstranzparenzen

8.3 Kohärenz

Kohärenz wird dadurch sichergestellt, dass die Steuerung alle Roboter ausschliesslich über das ITS-Board zulässig ist.

8.4 Sicherheit (Safety)

Watchdogs auf dem Raspberry Pi des Roboterarms überwacht die eigene Verfügbarkeit. Wenn sich der Roboterarm durch einen vorherigen Aufruf vom ITS-Board bewegt, muss eine Verbindung zum ITS-Board bestehen. Bricht diese ab, oder das Netzwerk ist zu stark ausgelastet, muss der Roboterarm sofort anhalten. Wenn das ITS-Board einen Abbruch der Verbindung feststellt ist der Roboterarm als "nicht verfügbarërkennbar. Security ist keine Anforderung an das System.

8.5 Bedienoberfläche

Die GUI wird textuell auf dem Display des ITS-Boards dargestellt. Es wird auf Einfachheit und Eindeutigkeit geachtet.

8.6 Ablaufsteuerung

Ablaufsteuerung von IT-Systemen bezieht sich sowohl auf die an der (grafischen) Oberfläche sichtbaren Abläufe als auch auf die Steuerung der Hintergrundaktivitäten. Zur Ablaufsteuerung gehört daher unter anderem die Steuerung der Benutzungsoberfläche, die Workflow- oder Geschäftsprozessteuerung sowie Steuerung von Batchabläufen.

8.7 Ausnahme- und Fehlerbehandlung

8.8 Kommunikation

Für die Kommunikation wird das TCP-Protokoll genutzt. Die Steuerung der Roboterarme wird RPC durchgeführt.

8.9 Konfiguration

Die Raspberry Pi's und das ITS-Board bekommen eine feste IP-Adresse in einer bestehenden Netzwerkumgebung. Es existiert kein DHCP. Jeder Roboterarm hat eine eigene Hardgecodete Konfiguration, die dem ITS-Board mitgeteilt wird.

8.10 Logging, Protokollierung

Das Logging findet auf dem Raspberry Pi statt.

8.11 Plausibilisierung und Validierung

Wo und wie plausibilisieren und validieren Sie (Eingabe-)daten, etwa Benutzereingaben?

8.12 Sessionbehandlung

Sofern ein Roboterarm sich beim ITS-Board erfolgreich registriert hat, kann dieser, wenn erreichbar, angesteuert werden. Dafür wird eine TCP-Verbindung aufgebaut und danach mit RPC kommuniziert. Ein Watchdog auf beiden Seiten überwacht die Verbindung.

8.13 Skalierung

Dies Skalierung ist durch das /24 Netz begrenzt. Ansonsten können sich die Roboterarme beliebig mit ihren Kenndaten bei dem ITS-Board anmelden.

8.14 Verteilung

Das ITS-Board ruft per RPC die API der Middleware auf, diese wird dann in eine Bewegung des Roboterarms übersetzt. Die Nachrichten werden asynchron ausgetauscht. Die Anmeldung des Roboters erfolgt ebenfalls durch einen RPC aufruf. am ITS-Board.

9 Entwurfsentscheidungen

Entscheidung	**Alternativen**	**Begründung**	**Woche**
Client-Server	Schichten-Modell, Service-Orientiert, Ereignisgesteuert,Microservice, P2P, Serverless	Bi-Direktionale Kommunikation via RPC. Komplexität vermeiden (Microservices, Serverless)	2
RPC Kommunikation	MQTT, REST...	Funktionaler Ansatz; Es muss nur eine API bereitgestellt werden, sher effizient, kleiner Overhead, asynchrone Kommunikation möglich	2
Roboter-Software		Möglichkeit eine API bereitzustellen; Flexibilität, um Befehle zu verarbeiten (Sprache etc.); Erweiterbare Funktionalität	2
Logik liegt auf Raspberry Pi des zu steuernden R-Arms	Auf ITS-Board, Externer Server	Leistungsstark, kein Single Point Of Failure, Selbstverwaltung, erweiterte Skalierbarkeit	2
TCP	UDP	verbindungsorientiert, damit Daten vollständig sind	2
direkte Kopplung	Indirekte Kopplung,Losgekoppelte Kopplung, Strukturelle Kopplung	Verbindung ausschliesslich zu gesteuertem Roboter	2

Tabelle 9.1: Zentrale Entwurfsentscheidungen

10 Qualitätsszenarien

- Bezug auf section 1.2 - weniger wichtige Requirements müssen genannt werden

10.1 Quality Requirements Overview

-

10.1.1 Ziele für Software Engineering

Ziel	**Beschreibung**	**Metrik**
Funktionalität		Alle Abnahmetests werden erfolgreich bestanden
Zuverlässigkeit	Teilausfälle dürfen den Gesamtsystembetrieb nicht gefährden. Fehlererkennung und -toleranz müssen integriert sein.	Das System ist über dem gesamten Abnahmezeitraum stabil (ca. 1,5 h)
Skalierbarkeit	Zusätzliche Roboter oder Komponenten sollen ohne Änderungen an der bestehenden Architektur integrierbar sein.	Es können beliebig viele Roboter hinzugefügt und entfernt werden (0 - N)
Leistung	Reaktionszeiten auf Steuerbefehle und Ereignisse müssen innerhalb definierter Zeitgrenzen liegen.	Reaktionszeit max. 250 ms
Sicherheit (Safety)	Es bewegt sich immer genau ein Roboterarm. Sollte das System nicht wie gewünscht reagieren, wird ein sicherer Zustand erreicht Das System darf unter keinen Umständen eine Gefährdung für Personen/Gegenstände darstellen. Bei Fehlern muss sofort ein sicherer Zustand erreicht werden (z.B. Notstopp).	Reaktionszeit max. 250 ms, bis Roboterarm stoppt
Wartbarkeit	Der Code muss modular, gut dokumentiert und testbar sein. Fehlerdiagnose und Protokollierung sollen integriert sein.	
Portabilität	Die Software soll ohne großen Aufwand auf vergleichbaren Embedded-Systemen lauffähig sein.	
Benutzerfreundlichkeit	Konfiguration und Überwachung müssen intuitiv bedienbar und gut visualisiert sein.	Keine Einweisung erforderlich
Anpassbarkeit	Neue Funktionen, Sensoren oder Regeln sollen ohne tiefgreifende Änderungen am System integrierbar sein.	
Kompatibilität	Das System soll mit bestehenden Standards und Protokollen kommunizieren können.	

Tabelle 10.1: Qualitätsziele der Software Engineering

10.1.2 Ziele der Verteilte Systeme

Ziel	**Beschreibung**	**Metrik**
Ressourcenteilung	...	
Offenheit	...	
Skalierbarkeit	...	siehe Tabelle 10.3
Verteilung Transparenz	...	siehe Tabelle 10.4

Tabelle 10.2: Qualitätsziele der Verteilten Systeme

Skalierbarkeit

Ziel	**Metrik**	**Metrik**
Vertikale Skalierung	...	
Horizontale Skalierung	...	
Räumliche Skalierbarkeit		1
Funktionale Skalierbarkeit	...	
Administrative-Skalierbarkeit		1

Tabelle 10.3: Skalierbarkeit von verteilten Systemen

Verteilungs-Transparenzen

Ziel	Beschreibung	Metrik
Zugriffstransparenz	...	
Lokalitäts-Transparenz	...	
Migrationstransparenz	...	
Replikationstransparenz	...	
Fehlertransparenz	...	
Ortstransparenz	..	
Skalierbarkeits-Transparenz	...	

Tabelle 10.4: Verteilungs-Transparenzen

10.2 Bewertungsszenarien

10.3 Qualitätsbaum

11 Risiken

Beispieleintrag für ein Glossarverweis: Architekturdokumentation.

Eine Liste identifizierter technischer Risiken oder technischer Schulden, nach Priorität geordnet!

Ziel des chapters

Frühzeitige Identifikation und Dokumentation technischer Risiken
Aufzeigen von bewusst eingegangenen technischen Schulden
Unterstützung bei Risikomanagement und fundierter Entscheidungsfindung

Technische Risiken Fehlende Erfahrungen, Komplexität, Externe Abhängigkeiten

Technische Schulden Hardcodierung, Fehlende Test

Form Eine Liste der jeweiligen Risiken und Schulden. Zusätzlich sollten vorransgehensweise und lösungswege hinzugefügt werden. Wie ist man mit den Risiken umgegangen? Warum nimmt man die Schuld in kauf?